UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

BROADCOM CORPORATION, *et al.*,

Plaintiffs,

v.

NETFLIX, INC.

Defendants.

Case No. 3:20–cv–04677–JD

JOINT CLAIM CONSTRUCTION AND PREHEARING STATEMENT

In accordance with Patent L.R. 4-3 and the Court's Standing Order for Claim Construction in Patent Cases Before Judge James Donato, Plaintiffs Broadcom Corporation and Avago Technologies International Sales PTE Ltd. and Defendant Netflix, Inc. (collectively the "Parties") jointly submit this Joint Claim Construction and Prehearing Statement (the "Statement") for U.S. Patent Nos. 7,266,079 (the "'079 Patent"); 8,259,121 ("the '121 Patent"); 8,959,245 ("the '245 Patent"); 8,270,992 ("the '992 Patent"); 6,341,375 ("the '375 Patent"); 8,572,138 ("the '138 Patent"); 6,744,387 ("the '387 Patent"); 6,982,663 ("the '663 Patent"); 9,332,283 ("the '283 Patent"), 8,548,976 ("the '976 Patent"); 7,457,722 ("the '722 Patent"); and 8,365,183 ("the '183 Patent"). This Statement contains information on claim construction for these patents under Patent L.R. 4-3 as follows.

I. STIPULATED CLAIM CONSTRUCTIONS UNDER PATENT L.R. 4–3(a)

To narrow their disputes, the Parties have met and conferred and have reached agreement on claim constructions for the following terms:

Term(s)	Patent	Claim(s)	Stipulated Construction
"flow control module"	'121	1, 35	"valve to control data flow"

"decoder devices" / "decoders"	'375	15	"decoders" in Steps (C) and (D) of Claim 15 refers to the "decoder devices" included in Steps (B) and (D) of Claim 15
"said exp-Golomb binarization comprises means for"	'387	4	Governed by 35 U.S.C. § 112(6). Function: "a) forming . . . b) determining . . . c) appending . . . d) determining . . . and e) appending . . ." Structure: binarization module 62 configured to perform at least Steps 4(a)-4(e).
"determining the least significant bits, of v−(N=2)−2**γ"	'387	4, 6	"determining the least significant bits, of v−(N−2)−2**γ"
"determining the number of bits, γ+1, required to represent v−(N−2) where γ= ⌊log2(v−(N−2))⌋ , v is the code symbol index value, and N is the threshold value, and then transforming y into a unary representation to create a result"	'387	4	"determining the number of bits, γ+1, required to represent v−(N−2) where γ= ⌊log2(v−(N−2))⌋ , v is the code symbol index value, and N is the threshold value, and thereafter transforming γ into a unary representation to create a result" No further construction is necessary regarding order of limitations.
"determining the number of bits, γ+1, required to represent v−(N=2) where γ= ⌊log2(v−(N−2))⌋ , v is the code symbol index value, and N is the threshold value, and then transforming γ into a unary representation to create a result"	'387	6	"determining the number of bits, γ+1, required to represent v−(N−2) where γ= ⌊log2(v−(N−2))⌋ , v is the code symbol index value, and N is the threshold value, and thereafter transforming γ into a unary representation to create a result" No further construction is necessary regarding order of limitations.
Claim 12's Preamble	'663	12	The preamble of Claim 12 is limiting.
"to generate to generate"	'283	1	"to generate"
"syntax element"	'283	1, 14, 16	"a value that is encoded to form part of the output bitstream"
"output bitstream"	'283	1, 3, 14, 16, 18	"a sequence of bits representing an encoded video signal"
"virtual machine"	'138	1, 9, 19, 22, 23	"software that emulates a physical computing platform"
"image instance"	'138	1, 19	"a snapshot, or copy, of installed and configured software"

"operational characteristics"	'183	1, 11, 16	"hardware and/or software characteristics"
"utilization criteria"	'183	1, 4, 5, 8, 9, 11–13, 15–18, 20	"criteria specifying a level of hardware and/or software utilization"
"wherein said codeword in compatible with at least one of an International Organization for Standardization/ International Electrotechnical Commission 14496-10 standard and an International Telecommunication Union-Telecommunications Standardization Sector Recommendation H.264"	'663	20	"wherein said codeword is compatible with a binarization scheme of an International Organization for Standardization/ International Electrotechnical Commission [ISO/IEC] 14496-10 standard or of an International Telecommunication Union-Telecommunications Standardization Sector [ITU-T] Recommendation H.264"
"employ [a/the] single binary tree"	'283	1, 14, 16	"employ [a/the] common binary tree [when processing at least one P slice and at least one B slice to generate the output bitstream]."
"a compute program"	'079	38	"a computer program"

II. DISPUTED TERMS, PROPOSED CONSTRUCTIONS, AND SUPPORTING EVIDENCE UNDER PATENT L.R. 4–3(b)

In accordance with Patent L.R. 4–3(b) and the Court's Standing Order on Claim Construction, the Parties identify the following ten terms for construction[1]:

Claim Terms	Plaintiffs' Proposed Construction and Support	Defendant's Proposed Construction and Support
"display pipeline" ('121 Patent, Claims 1-4, 33, 35, 36)	**Proposed Construction:** No construction necessary.	**Proposed Construction:** "a series of video processing modules"

[1] Copies of the patents for which the parties have requested the Court to construe terms are attached to this Joint Statement — Ex. 1: '121 Patent; Ex. 2: '387 Patent; Ex. 3: '663 Patent; Ex. 4: '283 Patent; Ex. 5: '138 Patent.

	Plain and ordinary meaning which, in the context of the patent, is "chain of multiple nodes" **Intrinsic Evidence:** '121 Patent at 6:42–50, 7:36–38, 8:46–54, Fig. 4. '121 Patent File History Response (Dec. 9, 2008). **Extrinsic Evidence:** Declaration of Dr. Nathaniel Polish	**Intrinsic Evidence:** 1:41-49 1:66-2:4 2:5-9 2:55-59 3:23-25 4:45-47 4:54-58 5:41-44 5:50-53 6:15-17 6:42-53 7:1-4 7:13-19 7:36-44 8:6-8 8:25-33 8:46-54 9:45-64 10:23-26 11:20-24 Figs. 4, 5, 6, 8 U.S. Prosecution History: pp. 535-36 (Dec. 9, 2008 Response to Final Office Action, pp. 15-16); p. 392 (Office Action of June 23, 2010, p. 3); IPR 2020-01647: Patent Owner's Preliminary Response; Decision Denying Institution of *Inter Partes* Review; U.S. Pat. App. No. 10/300,370 at: Figs. 11A, 14 Pg. 14, ¶ [57]; Pg. 9, ¶ [39] Pg. 22, ¶ [89]; U.S. Pat. App. No. 10/114,798 at Pgs. 21-22, ¶¶ [66 & 68]; U.S. Prov. App. No. 60/420,347 at Pg. 18, ¶ [66] **Extrinsic Evidence:** IPR 2020-01647: Declaration of James A. Storer;

		IBM Dictionary of Computing (1994), p. 512 ("pipeline"); U.S. Patent No. 4,187,539 to Eaton at 1:8-22; Computer Desktop Encyclopedia (9th ed. 2001) ("pipeline processing"); "Pipe (Pipelined Image-Processing Engine)," Journal of Parallel and Distributed Computing 2, pp. 50-78 (1995); Computer Graphics (2nd ed. 1992) (18.4.1 "Pipelining"); IEEE Standard Glossary of Computer Hardware Terminology (1994), p.69 ("pipeline processing," "pipeline processor," "pipelining"); Microsoft Press Computer Dictionary (1991) ("pipelining"); The Computer Glossary (7th ed. 1995), p. 302 ("pipeline processing"); Webster's New World Dictionary of Computer Terms (1983), p.193 ("pipelining"); U.S. Patent No. 5,798,770 to Baldwin at 5:15-25; U.S. Patent No. 6,628,243 to Lyons at 6:12-55 & Figs. 2-6; U.S. Patent No. 7,054,867 to Bosley at 4:49-55, 5:22-27; U.S. Patent No. 6,489,953 to Chen at 3:16-33 Declaration of Prof. K.K. Ramakrishnan
"means for determining if a code symbol index value is less than a threshold value" ('387 Patent, Claim 3)	**Proposed Construction** Governed by 35 U.S.C. § 112(6). Function: determining if a code symbol index value is less than a threshold value Structure is disclosed in Fig. 2,	**Proposed Construction** Function: determining if a code symbol index value is less than a threshold value Structure: binarization module 62 and step 102 of Figure 4 as described at 7:46 and 7:63-66

	which describes an encoder, and in particular binarization block 62 in conjunction with Fig. 4, and Fig. 4, step 102, Col. 4, lines 1-13, Tables 3 and 4, Col. 6, line 26 through Col. 8, line 23; and Claim 1. **Intrinsic and Extrinsic Evidence** See e.g., '387 patent at: 4:1-13; 6:26–8:23; Tables 3-4; Figs. 2, 4; Claims 1-2. Dr. Richardson may provide expert testimony regarding the level of skill of a person having ordinary skill in the art for the field of the '387 Patent, the meaning of this term, binarization modules within encoders, as well as rebuttal testimony (if appropriate).	**Intrinsic and Extrinsic Evidence** 1:6-11 2:15-34 4:01-05 6:26-8:10 Fig. 2 Fig. 4. Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181. Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181; Broadcom Corp v. Amazon (16-cv-1774) (C.D. Cal); Broadcom Corp v. Sony Corp (16-cv-1052) (C.D. Cal.); Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al. (No. 3:15-cv-04525) (N.D. Cal.) Expert testimony of Prof. Michael Orchard regarding the understanding of a person of ordinary skill regarding corresponding structure and the intrinsic evidence, technical background, and rebuttal of any incorrect positions or expert testimony asserted by Broadcom.
"means for constructing a codeword using a unary binarization if said code symbol index value is less than a threshold value" ('387 Patent, Claim 3)	**Proposed Construction** Governed by 35 U.S.C. § 112(6). Function: constructing a codeword using a unary binarization if said code symbol index value is less than said threshold value	**Proposed Construction** Function: constructing a codeword using a unary binarization if said code symbol index value is less than said threshold value

	Structure is disclosed in Fig. 2, which describes an encoder, and in particular binarization block 62 in conjunction with Fig. 4, and Fig. 4, steps 102, 104 and 112, Col. 4, lines 1-13, Tables 3 and 4, Col. 6, line 26 through Col. 8, line 23; and Claim 1. **Intrinsic and Extrinsic Evidence** See e.g., '387 patent at: 4:1-13; 4:52–54; 6:26–8:23; Tables 3-4; Figs. 2, 4; Claims 1-2. Dr. Richardson may provide expert testimony regarding the level of skill of a person having ordinary skill in the art for the field of the '387 Patent, the meaning of this term, binarization modules within encoders, as well as rebuttal testimony (if appropriate).	Structure: binarization module 62 and steps 102 and 104 of Figure 4 as described at 7:46-47 and 7:61-8:1 **Intrinsic and Extrinsic Evidence** 1:6-11 2:15-34 4:01-05 6:26-8:10 Fig. 2 Fig. 4. Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181. Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181; Broadcom Corp v. Amazon (16-cv-1774) (C.D. Cal); Broadcom Corp v. Sony Corp (16-cv-1052) (C.D. Cal.); Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al. (No. 3:15-cv-04525) (N.D. Cal.) Expert testimony of Prof. Michael Orchard regarding the understanding of a person of ordinary skill regarding corresponding structure and the intrinsic evidence, technical background, and rebuttal of any incorrect positions or expert testimony asserted by Broadcom. A. Moffat et al., Compression and Coding Algorithms, Kluwer Academic Publishers (Mar. 31, 2002) at Ch. 3

"means for constructing a codeword using a exp-Golomb binarization if said code symbol index value is [not] less than a threshold value" ('387 Patent, Claim 3)	**Proposed Construction** Governed by 35 U.S.C. § 112(6). Function: constructing a codeword using a exp-Golomb binarization if said code symbol index value is not less than said threshold value Structure is disclosed in Fig. 2, which describes an encoder, and in particular binarization block 62 in conjunction with Fig. 4, and Fig. 4, steps 102, 106, 108, 110, 112; Col. 4, lines 1-13, Tables 3 and 4, Col. 6, line 26 through Col. 8, line 23; and Claims 1-2. **Intrinsic and Extrinsic Evidence** See e.g., '387 patent at: 4:1–13; 4:52–54; 6:26–8:23; Tables 3-4; Figs. 2, 4; Claims 1-2. Dr. Richardson may provide expert testimony regarding the level of skill of a person having ordinary skill in the art for the field of the '387 Patent, the meaning of this term, binarization modules within encoders, as well as rebuttal testimony (if appropriate).	**Proposed Construction** Function: constructing a codeword using a exp-Golomb binarization if said code symbol index value is [not] less than said threshold value Structure: binarization module 62 and steps 106, 108, and 110 of Figure 4 as described at 7:49-60 and 8:1-8:10. **Intrinsic and Extrinsic Evidence** 1:6-11 2:15-34 4:01-05 5:54-67 6:26-8:10 Fig. 2 Fig. 4 Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181. Prior claim construction briefing (including extrinsic evidence cited therein) from IPR 2017-00963; IPR 2017-01181; Broadcom Corp v. Amazon (16-cv-1774) (C.D. Cal); Broadcom Corp v. Sony Corp (16-cv-1052) (C.D. Cal.); Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al. (No. 3:15-cv-04525) (N.D. Cal.) Potential expert testimony of Prof. Michael Orchard regarding

		the understanding of a person of ordinary skill regarding corresponding structure and the intrinsic evidence, technical background, and rebuttal of any incorrect positions or expert testimony asserted by Broadcom. Teuhola, A Compression Method for Clustered Bit-Vectors, Information Processing Letters, Vol. 7, No. 6 (Oct. 1978) at 1–4; D. Marpe et al., Improved CABAC, VCEG-O18 (Dec. 4, 2001) at 1–2; U.S. Patent No. 6,304,607 to Talluri et al. at 4:43–60, 5:08–6:14, Table 1, Table 2; P. Howard, Interleaving Entropy Codes, IEEE (June 11, 1997) at 45–46, 48; A. Moffat et al., Compression and Coding Algorithms, Kluwer Academic Publishers (Mar. 31, 2002) at Ch. 3.
’663 Patent, Claim 12 (Entire Claim)	**Proposed Construction:** The claim should be given its plain and ordinary meaning, in the context of the patent. No further construction is necessary regarding order of limitations. **Intrinsic Evidence:** ’663 patent at: 1:38–62, 3:21–39, 4:42–50, 5:41–54, 6:24–28, 6:44–7:13; Fig. 4; Tables 1 and 2; Claims 1, 11–13, 18–19, 21.	**Proposed Construction:** Steps recited within this claim must be performed in order **Intrinsic Evidence:** ’663 Patent at Abstract, Figs. 4-6, 1:37-2:33, 3:10-20, 4:46-7:12, 7:30-10:3; ’387 Patent at Abstract, Fig. 4, 1:36-2:34, 3:7-18, 4:52-8:10; ’663 File History: Non-final Rejection dated Dec. 2, 2004 at 2-5; ’663 File History: Amendment dated Mar. 4, 2005 at 4, 6-7, 9, 11-19. Prior claim construction briefing (including extrinsic evidence

		cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182. **Extrinsic Evidence:** Prior claim construction briefing (including extrinsic evidence cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182, *Barnes & Noble, Inc. v. LSI Corp.*, (No. 3:11-cv-2709) (N.D. Cal.); *Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al.* (No. 3:15-cv-04525) (N.D. Cal.), *Broadcom Corp. et al v. Amazon.com, Inc. et al.* (No. 8:16-cv-01774-JVS-JCG) (C.D. Cal.), *Broadcom Corp. et al v. Sony Corp. et al.* (No. 8:16-cv-01052-JVS-JCG) (C.D. Cal.) Lowell Winger, JVT-C162 (May 2, 2002); "Draft ITU-T Recommendation H.264," Joint Video Team, JVT-C167 (May 10, 2002); U.S. Patent No. 6,236,960 at Figs. 5 & 7, 11:11–12:30, 13:15–54; Expert testimony of Prof. Michael Orchard regarding the understanding of a person of ordinary skill regarding the intrinsic evidence, technical background, and rebuttal of any incorrect positions or expert testimony asserted by Broadcom.
"a fourth pattern in said first portion based on said index value in response to said index value	**Proposed Construction:** Plain and ordinary meaning. No construction of "[generating] a fourth pattern in said first portion based on said index value in	**Proposed Construction:** Indefinite **Intrinsic Evidence:**

being below said threshold" ('663 Patent, Claim 13)	response to said index value being below said threshold" needed. **Intrinsic Evidence:** '663 patent at: Tables 3–4, 6:50–7:13. **Extrinsic Evidence:** Declaration of Dr. Iain Richardson	'663 Patent at Abstract, Figs. 3-6, 2:15-33, 3:10-20, 4:1-12; 4:13-33; 4:46-7:12; 7:30-10:3 Prior claim construction briefing (including extrinsic evidence cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182 **Extrinsic Evidence:** Prior claim construction briefing (including extrinsic evidence cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182, *Barnes & Noble, Inc. v. LSI Corp*., (No. 3:11-cv-2709) (N.D. Cal.); *Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al*. (No. 3:15-cv-04525) (N.D. Cal.), *Broadcom Corp. et al v. Amazon.com, Inc. et al*. (No. 8:16-cv-01774-JVS-JCG) (C.D. Cal.), *Broadcom Corp. et al v. Sony Corp. et al*. (No. 8:16-cv-01052-JVS-JCG) (C.D. Cal.). Lowell Winger, JVT-C162 (May 2, 2002); "Draft ITU-T Recommendation H.264," Joint Video Team, JVT-C167 (May 10, 2002) Declaration of Prof. Michael Orchard
"wherein said [second/third] portion is void in response to said index value being below said threshold"	**Proposed Construction:** Plain and ordinary meaning, which in the context of the patent is "wherein said [second/third] portion does not contribute any bins to the codeword in response	**Proposed Construction:** Indefinite **Intrinsic Evidence:**

('663 Patent, Claims 18, 19)	to said index value being below said threshold." **Intrinsic Evidence:** '663 patent at: Tables 3–4, 6:50–7:13. **Extrinsic Evidence:** *Void*, MERRIAM-WEBSTER'S COLLEGIATE DICTIONARY (10th ed. 1993). *Void*, THE AMERICAN HERITAGE DICTIONARY OF THE ENGLISH LANGUAGE (4th ed. 2000). Declaration of Dr. Iain Richardson	'663 Patent at Abstract, Figs. 3-6, 2:15-33, 3:10-20, 4:1-12; 4:13-33; 4:46-7:12; 7:30-10:3 Prior claim construction briefing (including extrinsic evidence cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182. **Extrinsic Evidence:** Prior claim construction briefing (including extrinsic evidence cited therein) from ITC No. 337-TA-837, IPR 2021-00468, IPR 2017-00964, IPR 2017-01182, *Barnes & Noble, Inc. v. LSI Corp.*, (No. 3:11-cv-2709) (N.D. Cal.); *Avago Technologies General IP (Singapore) Pte. Ltd. v. ASUSTeK Computer, Inc. et al.* (No. 3:15-cv-04525) (N.D. Cal.), *Broadcom Corp. et al v. Amazon.com, Inc. et al.* (No. 8:16-cv-01774-JVS-JCG) (C.D. Cal.), *Broadcom Corp. et al v. Sony Corp. et al.* (No. 8:16-cv-01052-JVS-JCG) (C.D. Cal.). Lowell Winger, JVT-C162 (May 2, 2002); "Draft ITU-T Recommendation H.264," Joint Video Team, JVT-C167 (May 10, 2002); Declaration of Prof. Michael Orchard
"wherein the second syntax element specifies the PU partition mode for the selected CU, wherein the PU partition mode is based on a	**Proposed Construction:** Plaintiffs object to Defendant's request to construe this entire phrase without specifying which portion(s)/element(s) it contends require construction.	**Proposed Construction:** Indefinite **Intrinsic Evidence:** 17:19-23

size N×N PU when the selected CU is a smallest CU (SCU) of the plurality of CUs and is based on a different size PU than the size N×N PU when the selected CU is another CU than the SCU of the plurality of CUs" ('283 Patent, Claims 1, 14)	No construction necessary. Plain and ordinary meaning. **Intrinsic Evidence:** '283 patent at: 16:41–19:12, Figs. 11–13. **Extrinsic Evidence:** Declaration of Dr. Iain Richardson	18:12-54 19:62-20:18 Figs. 12, 13, 14 & 15 Claims 2 &17 **Extrinsic Evidence:** Declaration of Dr. Vivienne Sze
"virtual machine manager" ('138 patent, claims 1, 19)	**Proposed Construction:** No construction necessary. Plain and ordinary meaning. **Intrinsic Evidence:** '138 Patent at 32:60-62. **Extrinsic Evidence:** MICROSOFT COMPUTER DICTIONARY 327 (5th ed. 2002). INTRODUCTION TO VMWARE INFRASTRUCTURE 44 (2006-2007). MERRIAM-WEBSTER DICTIONARY [706, 1993]	**Proposed Construction:** "software program that manages one or more virtual machines" **Intrinsic Evidence:** Claim 1 32:59-67 33:21-23 33:39-41 Fig. 24
"rules engine" ('138 patent, claims 11, 14, 19)	**Proposed Construction:** No construction necessary. Plain and ordinary meaning. **Intrinsic Evidence:**	**Proposed Construction:** "software that performs logical reasoning using 'if/then' rules" **Intrinsic Evidence:** 22:20-24

	'138 Patent at 22:56-58; 25:60-65; 2 **Extrinsic Evidence:** MICROSOFT COMPUTER DICTIONARY 193 (5th ed. 2002).	22:38-49 22:54-62 25:60-66 27:15-34 27:56-28:11 Fig. 20 **Extrinsic Evidence** Computer Dictionary, Microsoft Press, 2d Ed. (1994) (cross-referencing "rule-based system" with "expert system") Computer Dictionary, Microsoft Press, 5th Ed. (2002) (cross-referencing "rule-based system" with "expert system")

III. IDENTIFICATION OF CASE-DISPOSITIVE TERMS UNDER PATENT L.R. 4–3(a)

A. Plaintiffs' Position:

None of the disputed terms will be case dispositive. Plaintiffs disagree with Defendant's statements in Subparagraph B below regarding the alleged dispositive nature of each of the terms listed with respect to Defendant's alleged non-infringement. Regarding Defendant's statements regarding invalidity, Plaintiffs state only that a finding of indefiniteness as to a claim term applies only to claims including such term and are not dispositive of the case or all claims of a particular patent.

Plaintiffs object to Defendant's attempt to reserve the right to later seek construction of "one or more underutilized computer devices" in the '183 Patent or any other term not set forth in this Joint Statement. Neither party identified "one or more underutilized computer devices" in their Patent L.R. 4-1 or 4-2 disclosures.

B. Defendant's Position:

The following disputed claim terms are dispositive of non-infringement:

- "display pipeline"

- "means for determining if a code symbol index value is less than a threshold value"
- "means for constructing a codeword using a unary binarization if said code symbol index value is less than a threshold value"
- "means for constructing a codeword using a exp-Golomb binarization if said code symbol index value is [not] less than a threshold value"
- Claim 12 (Entire Claim)
- "virtual machine manager"
- "rules engine"

The following disputed claim terms are dispositive of invalidity:

- "wherein the second syntax element specifies the PU partition mode for the selected CU, wherein the PU partition mode is based on a size N×N PU when the selected CU is a smallest CU (SCU) of the plurality of CUs and is based on a different size PU than the size N×N PU when the selected CU is another CU than the SCU of the plurality of CUs"
- "a fourth pattern in said first portion based on said index value in response to said index value being below said threshold"
- "wherein said [second/third] portion is void in response to said index value being below said threshold"

On May 17, 2021, the Patent Trial and Appeals Board issued an institution decision for the '183 patent. The PTAB construed "one or more underutilized computer devices" to mean "one or more computer devices that have a utilization level below, or less than, a specified utilization level." Accordingly, it found that Claim 1 "requires both a determination that 'one or more underutilized computer devices exist on the list of suitable computer devices' and that the 'one or more underutilized computer devices hav[e] a suitable level of utilization for performing the job.'" IPR2021-00098. This decision is not yet final. Netflix reserves the right to move for leave to supplement its proposed terms for claim construction in light of a final decision.

IV. ANTICIPATED LENGTH OF CLAIM CONSTRUCTION HEARING

The Parties request that the Court provide three hours for the hearing.

V. PROPOSED CLAIM-CONSTRUCTION WITNESSES

The parties plan to submit expert declarations with their claim-construction briefs as outlined in Section II above. The parties do not intend to call live witnesses at the claim-construction hearing.

Dated: May 18, 2021

Respectfully submitted,

/s/ *Richard L. Wynne, Jr.*

Bruce S. Sostek, admitted pro hac vice
bruce.sostek@tklaw.com
Richard L. Wynne, Jr., admitted pro hac vice
richard.wynne@tklaw.com
Adrienne E. Dominguez, admitted pro hac vice
adrienne.dominguez@tklaw.com
THOMPSON & KNIGHT LLP
One Arts Plaza
1722 Routh Street, Suite 1500
Dallas, Texas 75201
Telephone:(214) 969-1700
Facsimile: (214) 969-1751

John V. Picone III, Bar No. 187226
jpicone@hopkinscarley.com
HOPKINS & CARLEY
A Law Corporation
The Letitia Building
70 South First Street
San Jose, CA 95113-2406

mailing address:
P.O. Box 1469
San Jose, CA 95109-1469
Telephone:(408) 286-9800
Facsimile: (408) 998-4790
Attorneys for Plaintiffs

COUNSEL FOR
BROADCOM CORPORATION and AVAGO TECHNOLOGIES INTERNATIONAL SALES PTE. LIMITED.

Dated: May 18, 2021

KEKER, VAN NEST & PETERS LLP

By: */s/ Sharif E. Jacob*
ROBERT A. VAN NEST
MATTHIAS A. KAMBER
PAVEN MALHOTRA
SHARIF E. JACOB
THOMAS E. GORMAN
EDWARD A. BAYLEY

Attorneys for Defendant
NETFLIX, INC.

Exhibit 1

U.S. Patent No. 8,259,121

US008259121B2

(12) **United States Patent**
Law et al.

(10) **Patent No.: US 8,259,121 B2**
(45) **Date of Patent: Sep. 4, 2012**

(54) **SYSTEM AND METHOD FOR PROCESSING DATA USING A NETWORK**

(75) Inventors: **Patrick Law**, Milpitas, CA (US); **Darren Neuman**, San Jose, CA (US); **David Baer**, San Jose, CA (US)

(73) Assignee: **Broadcom Corporation**, Irvine, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1509 days.

(21) Appl. No.: **10/314,525**

(22) Filed: **Dec. 9, 2002**

(65) **Prior Publication Data**

US 2004/0078418 A1 Apr. 22, 2004

Related U.S. Application Data

(60) Provisional application No. 60/420,151, filed on Oct. 22, 2002.

(51) **Int. Cl.**

G06T 1/20	(2006.01)
G06F 12/02	(2006.01)
G06F 13/28	(2006.01)
G06F 3/00	(2006.01)
G06K 9/54	(2006.01)

(52) **U.S. Cl.** **345/506**; 345/567; 382/303; 382/304

(58) **Field of Classification Search** 345/505, 345/501, 502, 506, 567; 382/303, 304; 710/22, 710/29

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,638,533	A *	6/1997	Law	711/157
5,646,693	A *	7/1997	Cismas	348/441
5,822,779	A *	10/1998	Intrater et al.	711/168
5,828,903	A *	10/1998	Sethuram et al.	710/53
5,841,439	A *	11/1998	Pose et al.	345/418
5,987,246	A *	11/1999	Thomsen et al.	717/109
6,101,591	A *	8/2000	Foster et al.	711/219
6,412,061	B1	6/2002	Dye	
6,429,902	B1 *	8/2002	Har-Chen et al.	348/518
6,489,953	B1 *	12/2002	Chen	345/213
6,525,738	B1 *	2/2003	Devins et al.	345/553
6,628,243	B1 *	9/2003	Lyons et al.	345/1.1
6,636,214	B1 *	10/2003	Leather et al.	345/422

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1091318	A2	4/2001

(Continued)

OTHER PUBLICATIONS

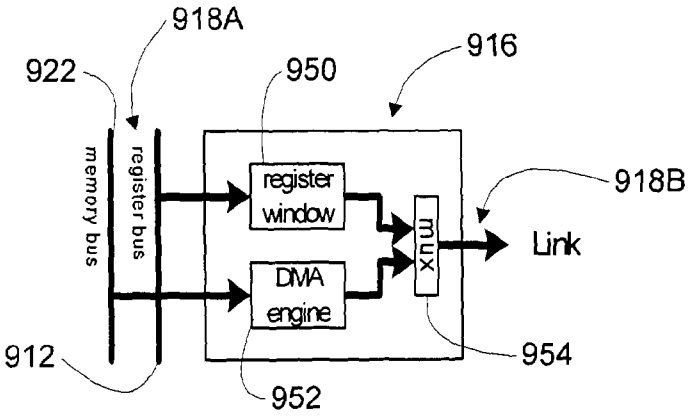
European Patent Office, Communication with European Search Report, in Application No. 03023968.5, dated Oct. 1, 2010, pp. 1-3.

Primary Examiner — Hoang-Vu A Nguyen-Ba
(74) *Attorney, Agent, or Firm* — Thomas, Kayden, Horstemeyer & Risley LLP.

(57) **ABSTRACT**

Systems and methods are disclosed for video processing modules. More specifically a network is disclosed for processing data. The network comprises a register DMA controller adapted to support register access and at least one node adapted to the data. At least one link communicates with the node, and is adapted to transmit data and at least one network module communicates with at least the link, and is adapted to route data to at least the link.

36 Claims, 11 Drawing Sheets



U.S. PATENT DOCUMENTS

6,700,588	B1 *	3/2004	MacInnis et al.	345/629
6,801,591	B1 *	10/2004	Frencken	375/373
6,919,896	B2 *	7/2005	Sasaki et al.	345/505
7,034,828	B1 *	4/2006	Drebin et al.	345/426
7,054,867	B2 *	5/2006	Bosley et al.	707/10
7,218,676	B2 *	5/2007	Kono et al.	375/240.25
7,230,651	B2 *	6/2007	Schoner et al.	348/500
2002/0066007	A1 *	5/2002	Wise et al.	712/300
2003/0080963	A1 *	5/2003	Van Hook et al.	345/501
2003/0146857	A9 *	8/2003	Koike	341/55
2005/0114569	A1 *	5/2005	Bogin et al.	710/52

FOREIGN PATENT DOCUMENTS

WO	WO99/13637	A2	3/1999
WO	WO99/52277	A1	10/1999
WO	WO-01/22736	*	3/2001

* cited by examiner

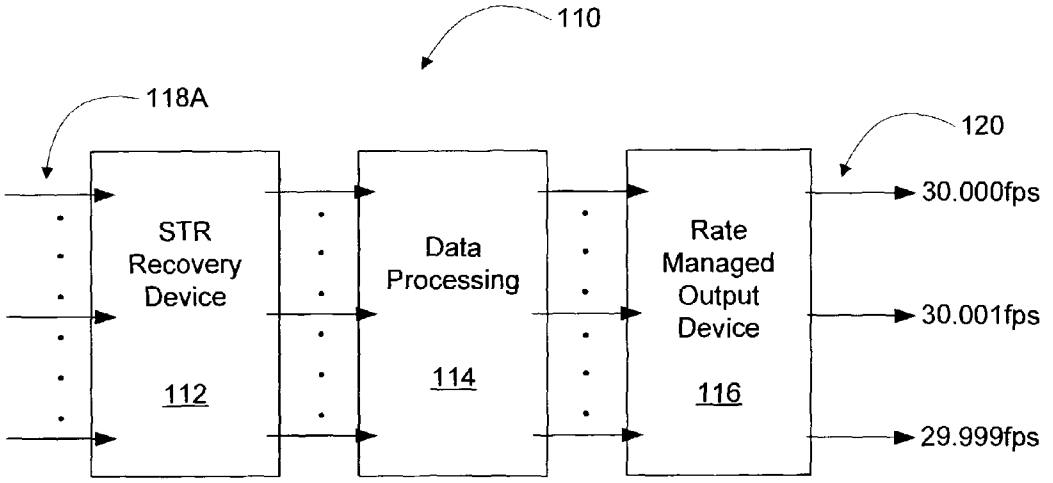

110
118A
120
STR Recovery Device
112
Data Processing
114
Rate Managed Output Device
116
30.000fps
30.001fps
29.999fps

Fig. 1

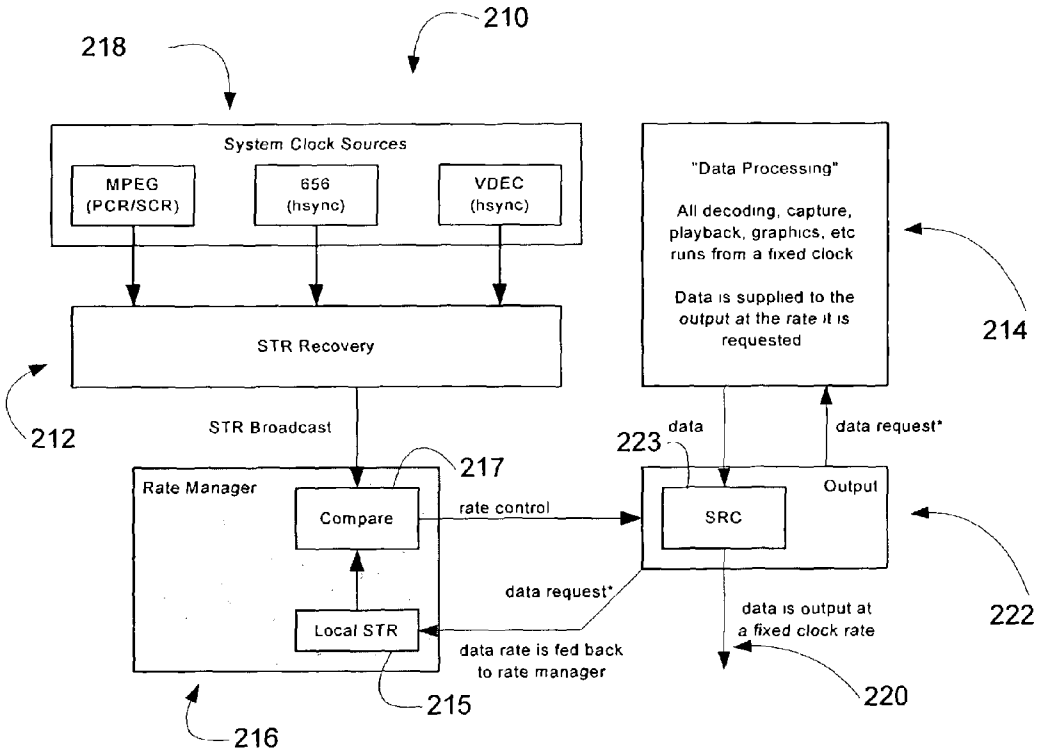

210
218
System Clock Sources
MPEG (PCR/SCR)
656 (hsync)
VDEC (hsync)
STR Recovery
212
STR Broadcast
"Data Processing"
All decoding, capture, playback, graphics, etc runs from a fixed clock
Data is supplied to the output at the rate it is requested
214
data
223
data request*
217
Rate Manager
Compare
rate control
Output
SRC
data request*
Local STR
data is output at a fixed clock rate
222
data rate is fed back to rate manager
215
220
216

Fig. 2

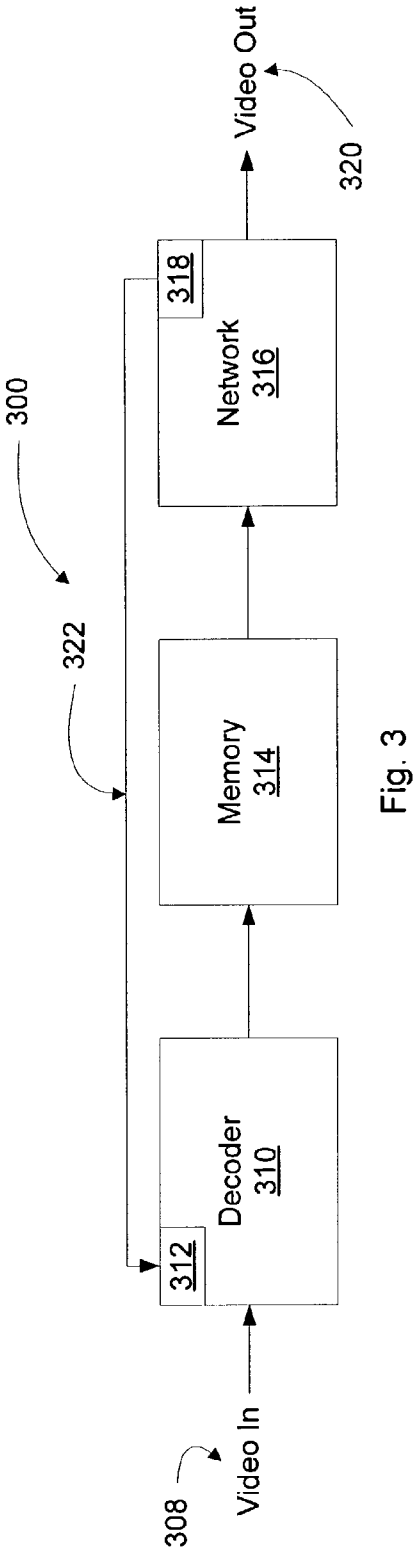

300
322
Video Out
320
318
Network
316
Memory
314
Decoder
310
312
Video In
308

Fig. 3

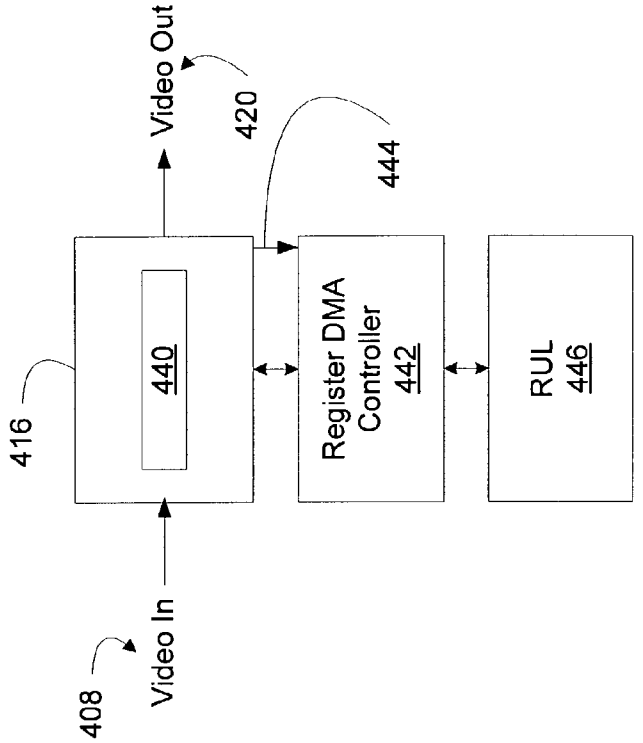

Video Out
420
444
416
440
Register DMA
Controller
442
RUL
446
Video In
408

Fig. 4

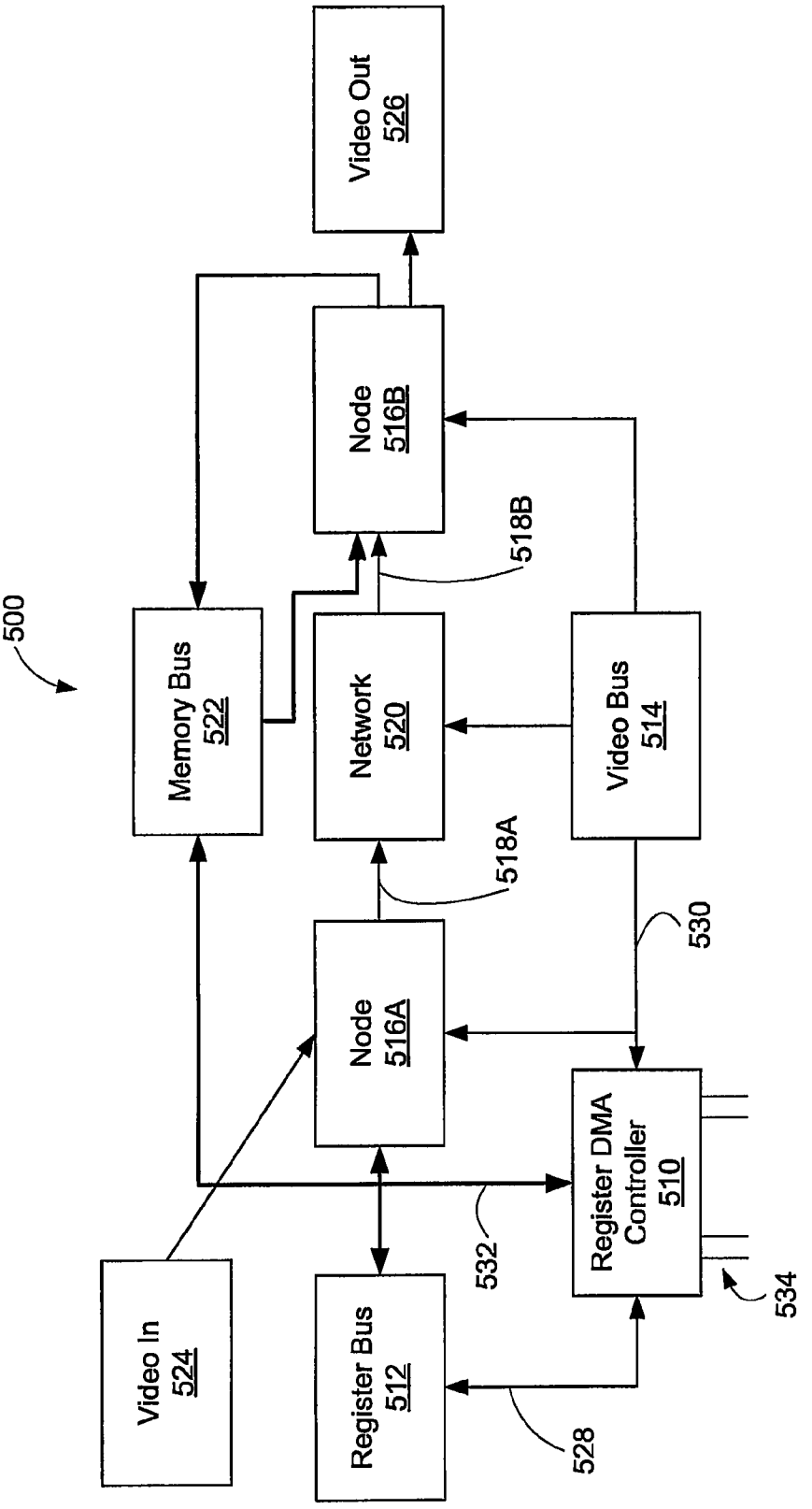

500
Video Out 526
Node 516B
518B
Memory Bus 522
Network 520
Video Bus 514
518A
530
Node 516A
Register DMA Controller 510
532
534
Video In 524
Register Bus 512
528

Fig. 5

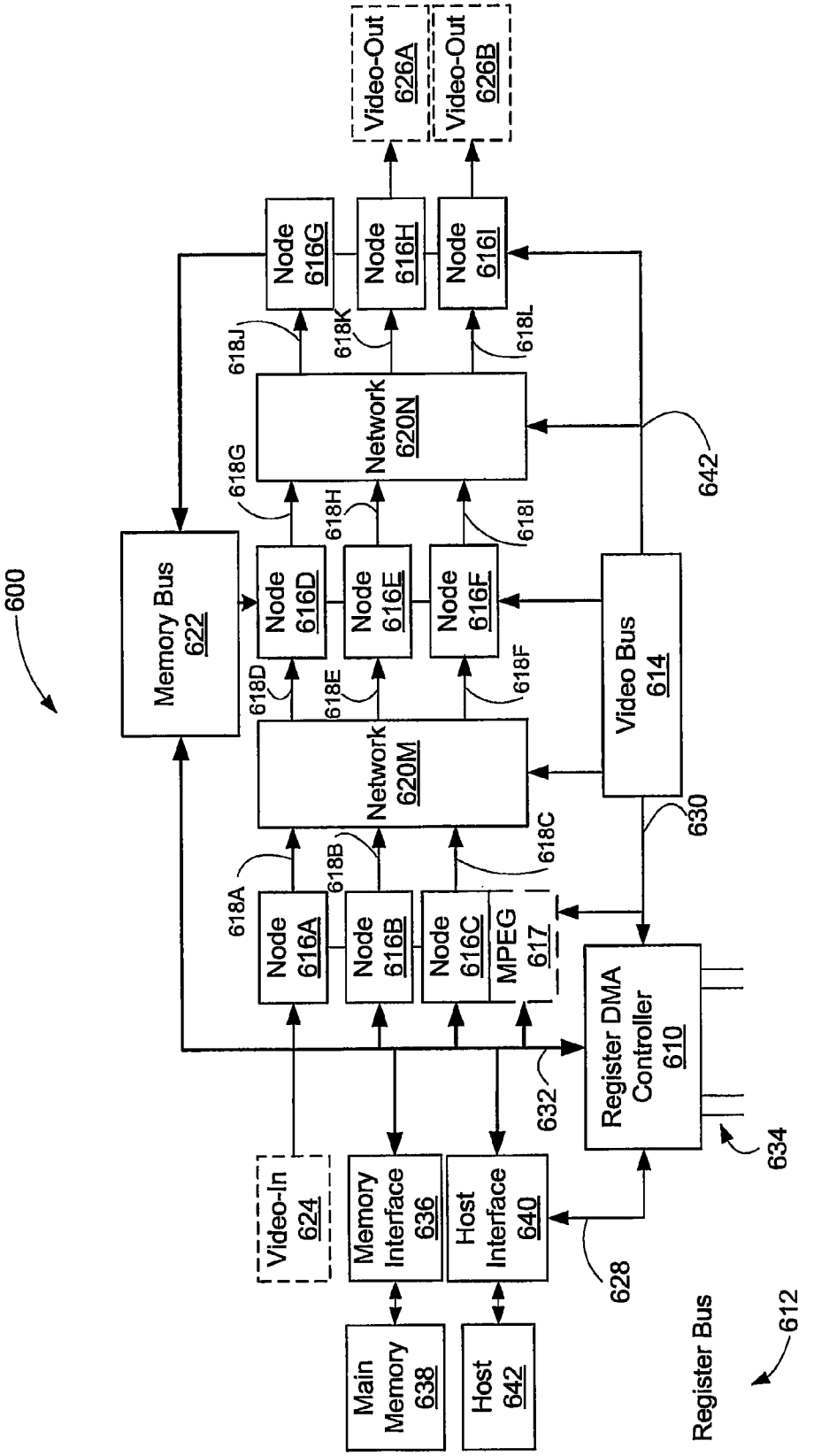

600
Memory Bus 622
Video-Out 626A
Video-Out 626B
Node 616G
Node 616H
Node 616I
Node 616D
Node 616E
Node 616F
Node 616A
Node 616B
Node 616C
MPEG 617
Network 620N
Network 620M
Video Bus 614
Register DMA Controller 610
Video-In 624
Memory Interface 636
Host Interface 640
Main Memory 638
Host 642
Register Bus 612
618A
618B
618C
618D
618E
618F
618G
618H
618I
618J
618K
618L
630
632
634
628
642

Fig. 6

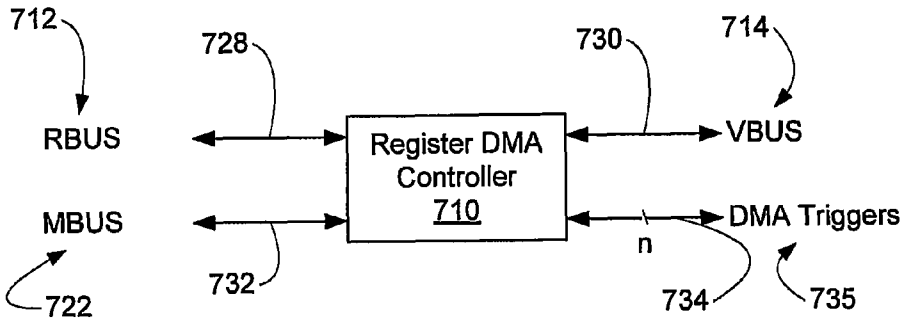

712
728
730
714
RBUS
Register DMA Controller 710
VBUS
MBUS
DMA Triggers
n
732
734
735
722

Fig. 7

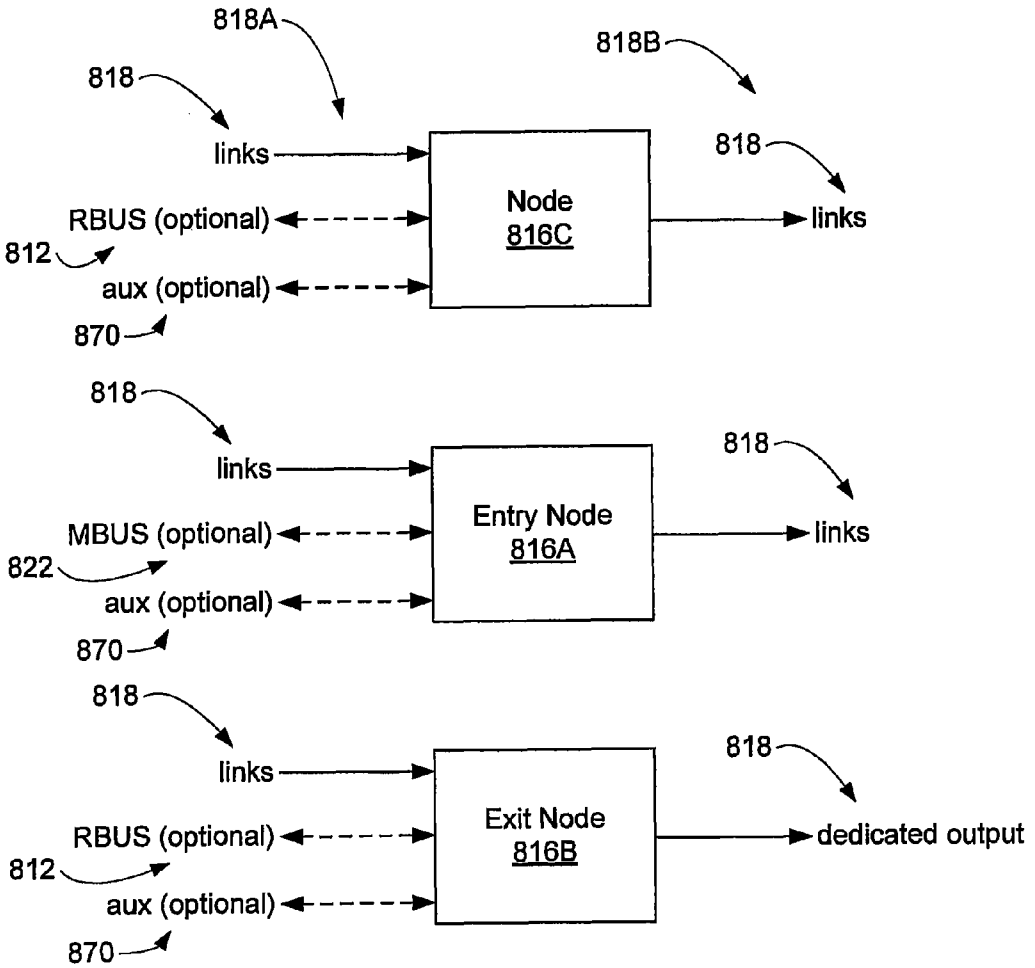

818A
818B
818
links
Node 816C
818
links
RBUS (optional)
812
aux (optional)
870
818
links
Entry Node 816A
818
links
MBUS (optional)
822
aux (optional)
870
818
links
Exit Node 816B
818
dedicated output
RBUS (optional)
812
aux (optional)
870

Fig. 8

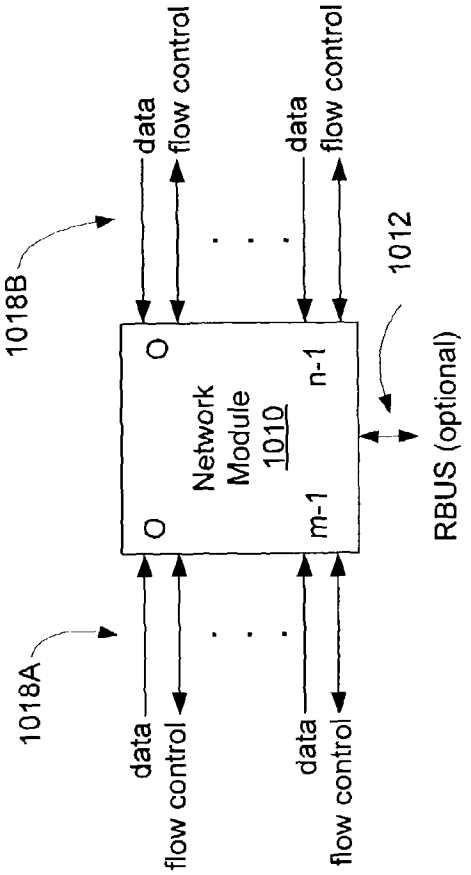

1018B
data
flow control
data
flow control
1012
0
Network Module 1010
n-1
0
m-1
RBUS (optional)
1018A
data
flow control
data
flow control

Fig. 10

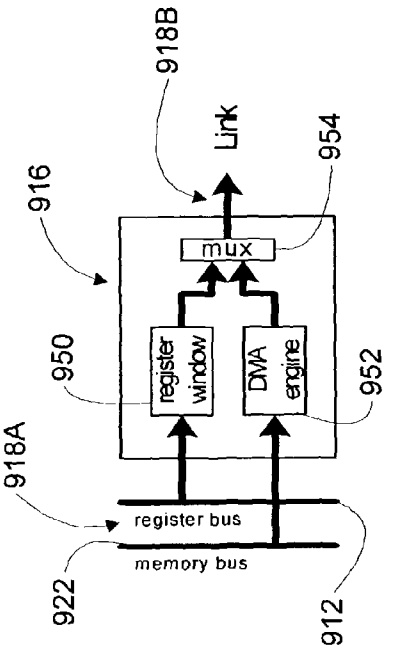

918B
Link
954
916
mux
950
register window
DMA engine
952
918A
register bus
memory bus
922
912

Fig. 9

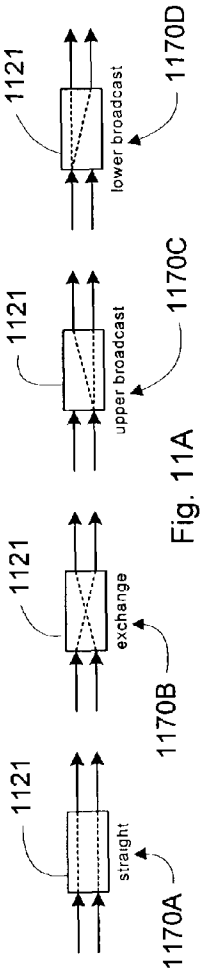


Fig. 11A

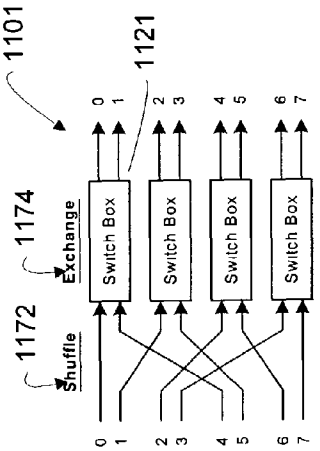


Fig. 11B

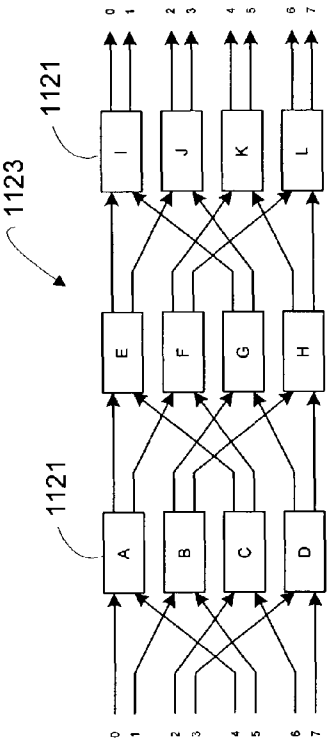


Fig. 11C

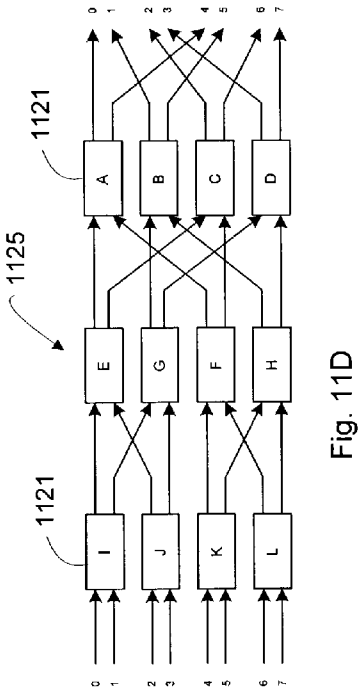


Fig. 11D

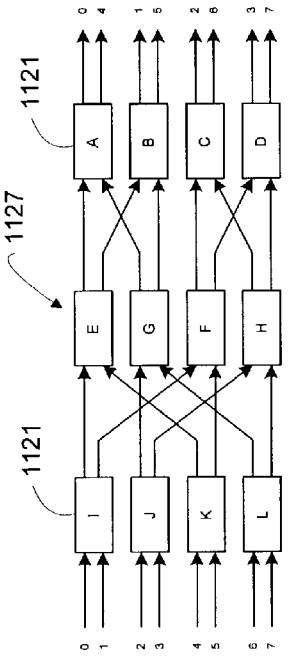


Fig. 11E

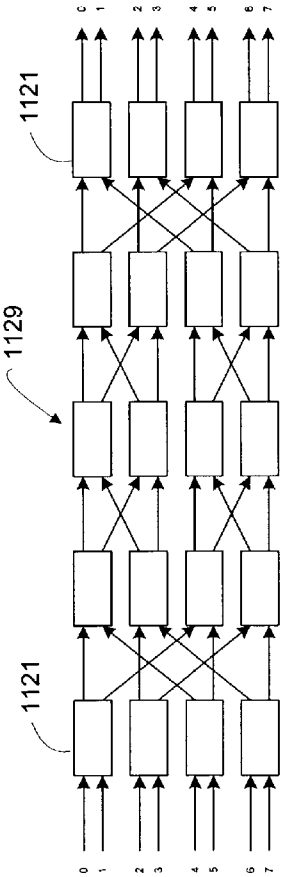


Fig. 11F

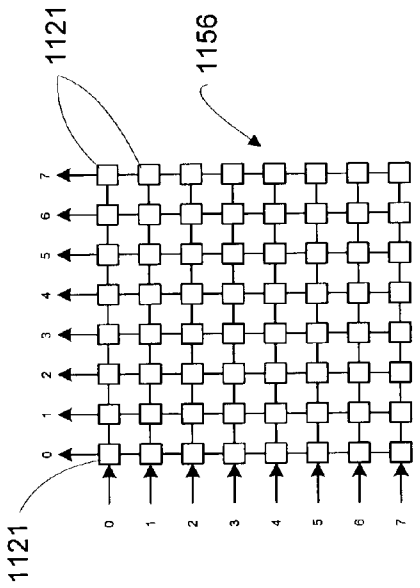


Fig. 11G

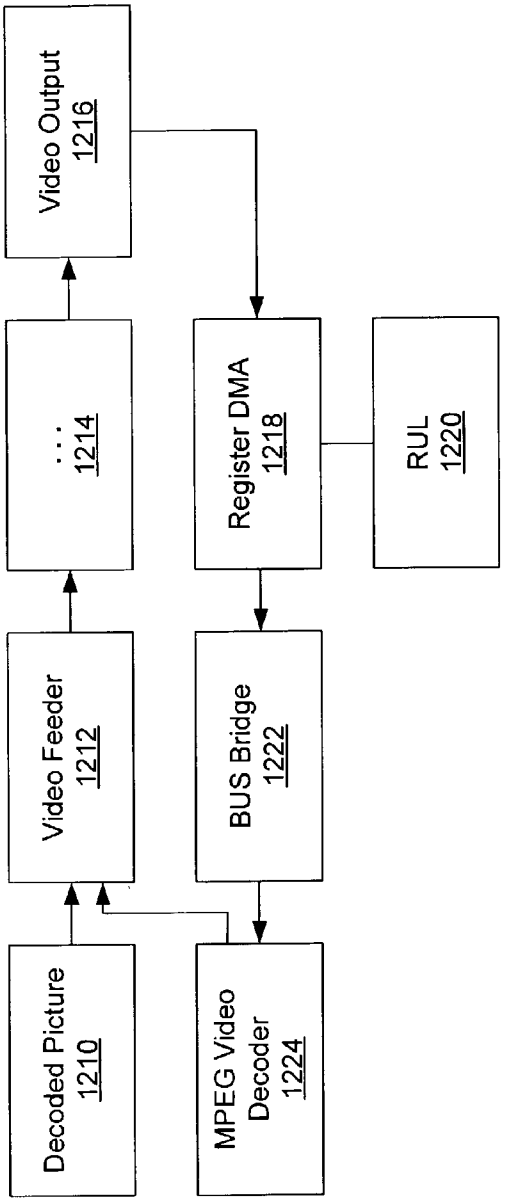


Fig. 12

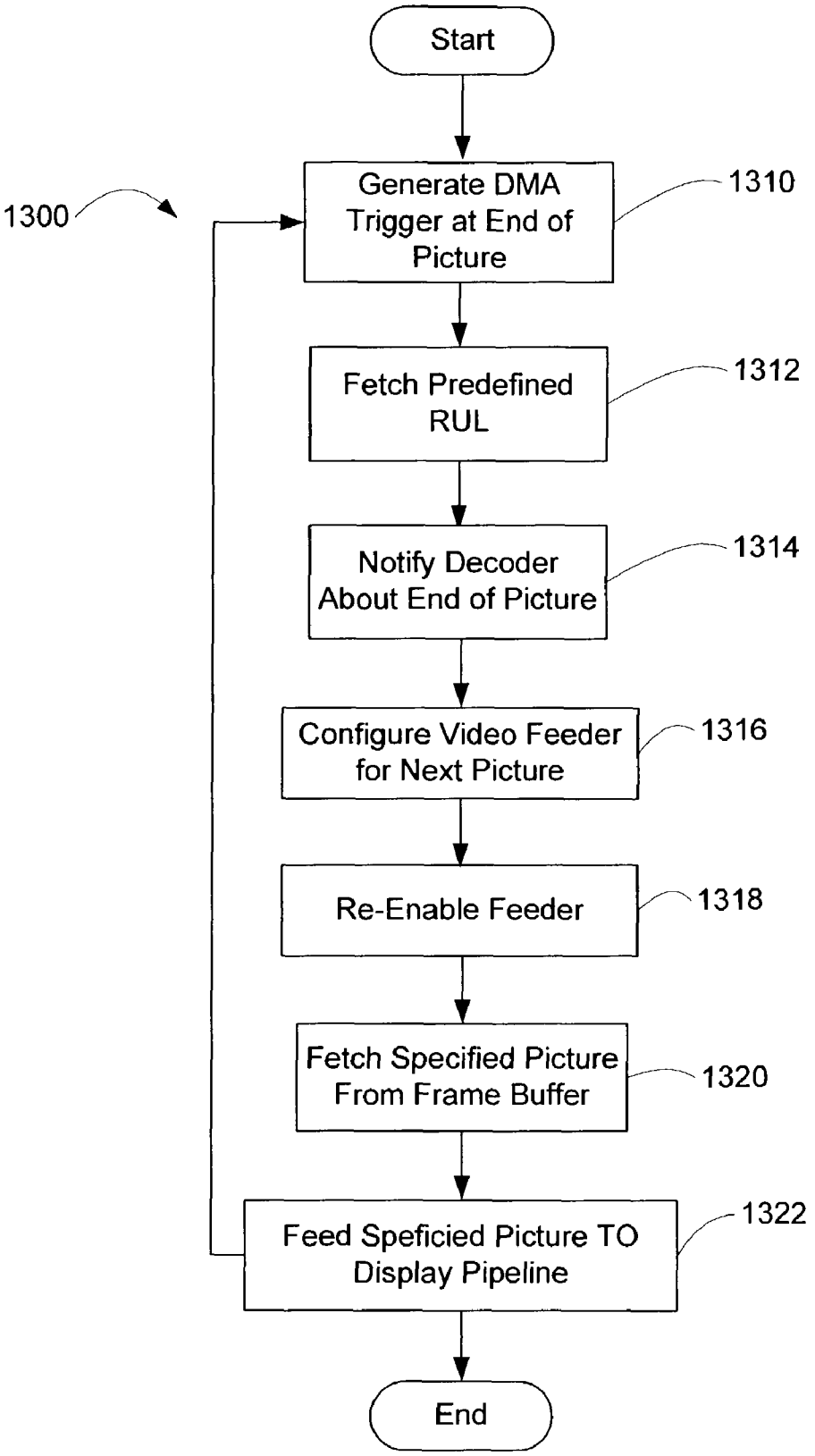

Start
1300
Generate DMA Trigger at End of Picture
1310
Fetch Predefined RUL
1312
Notify Decoder About End of Picture
1314
Configure Video Feeder for Next Picture
1316
Re-Enable Feeder
1318
Fetch Specified Picture From Frame Buffer
1320
Feed Speficied Picture TO Display Pipeline
1322
End

Fig. 13

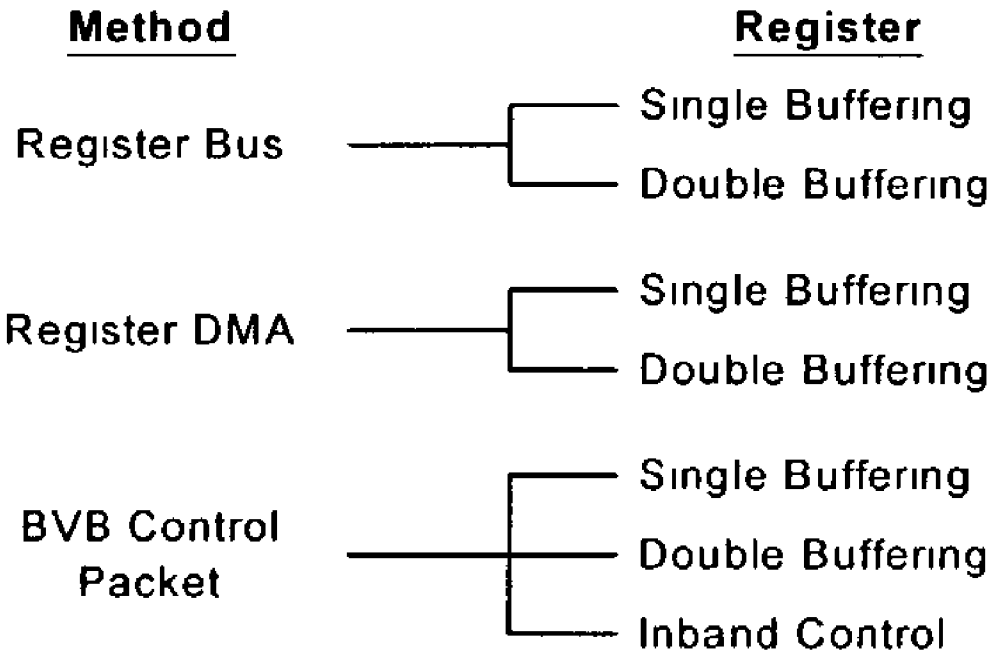

Method
Register
Register Bus
Single Buffering
Double Buffering
Register DMA
Single Buffering
Double Buffering
BVB Control Packet
Single Buffering
Double Buffering
Inband Control

Fig. 14

SYSTEM AND METHOD FOR PROCESSING DATA USING A NETWORK

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to, and claims benefit of and priority from, Provisional Application No. 60/420,151 dated Oct. 22, 2002, titled "Network Environment for Video Processing Modules", the complete subject matter of which is incorporated herein by reference in its entirety. This application is also related to the following applications, each of which is incorporated herein by reference in its entirety for all purposes: U.S. patent application Ser. No. 10/300,371, filed Nov. 20, 2002, titled "A/V Decoder Having A Clocking Scheme That Is Independent Of Input Data Streams"; U.S. Provisional Application No. 60/420,347, filed Oct. 22, 2002, titled "Video Bus For a Video Decoding System"; U.S. patent application Ser. No. 10/300,370, filed Nov. 20, 2002, titled "Hardware Assisted Format Change Mechanism in a Display Controller"; U.S. patent application Ser. No. 10/114,798, filed Apr. 1, 2002, titled "Video Decoding System Supporting Multiple Standards"; and U.S. Provisional Application No. 60/420,308, filed Oct. 22, 2002, titled "Multi-Pass System and Method Supporting Multiple Streams of Video".

FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[Not Applicable]

SEQUENCE LISTING

[Not Applicable]

MICROFICHE/COPYRIGHT REFERENCE

[Not Applicable]

BACKGROUND OF THE INVENTION

The present invention relates to a network adapted to process data. More specifically, the present invention relates to a network environment in an A/V system using "A/V decoders", where the A/V decoders are adapted to process, decode or decompress one or more input data streams (alternatively referred to as "input data", "input data streams" or "data streams").

There is currently no known methodological way to connect video processing modules in A/V systems. Most video processing modules are connected together in an ad-hoc manner. As a result, such ad-hoc designs may become difficult to verify, maintain and reuse. Furthermore, as more features are added to the A/V systems (i.e., incorporating more video processing modules for example) it becomes more difficult to design and integrate such features properly. This may result in long development cycles, poor design reuse and an unreliable product.

Further limitations and disadvantages of conventional and traditional approaches will become apparent to one of skill in the art, through comparison of such systems with the present invention as set forth in the remainder of the present application with reference to the drawings.

BRIEF SUMMARY OF THE INVENTION

There is a need for an architecture or network that provides a general model illustrating how various video processing modules behaves in a network environment. Further, an exemplary embodiment of such network should reduce the number of clock domains, ease design reuse and perform format changes in a robust manner.

Features of the present invention may be found in a network environment in an A/V system and method supporting a pull data flow scheme for an A/V decoder. The network is adapted to video process modules using a pull data flow (an output rate driven by data flow for example).

One embodiment of the present invention relates to a network for processing data to form at least one display pipeline therein by selecting and concatenating at least two nodes from a plurality of nodes in the network together. It is contemplated that this selection and concatenation happens on the fly (i.e., in real time). In this embodiment, the network is further adapted to form a plurality of the same or different display pipelines using at least the two nodes. It is contemplated that the network may change the functionality of the display pipeline by concatenating more than two nodes together. In one embodiment, the network is adapted to form at least two display pipelines having different and/or independent data rates (using a flow control valve or module for example). It is further contemplated that such network is adapted to form at least two of the display pipelines using a handshaking or ready/accept protocol.

In another embodiment, the network comprises at least a register DMA controller adapted to support register access. The register DMA controller is further adapted to obtain at least one instruction from a register update list and provide that instruction to the display pipeline. It is further contemplated that the register DMA controller may obtain the instruction in response to a trigger event.

Yet another embodiment of the present invention relates to a network for processing data. In this embodiment, the network comprises a register DMA controller adapted to support register access and a plurality of nodes adapted to process the data. The network further comprises at least one link communicating with the nodes and adapted to transmit the data between the nodes, and at least one network module communicating with at least the link and adapted to route the data thereto, wherein the network is adapted to form at least one display pipeline therein by selecting and concatenating at least two nodes from the plurality of nodes.

Another embodiment of the present invention relates to a method of processing data using a network. In this embodiment, the network comprises forming a first display pipeline using at least one node in the network and processing the data using the first display pipeline. The method further comprises forming a second display pipeline using at least one node in the network and processing the data using the second display pipeline, where the first and second display pipelines are different.

Still another embodiment of the present invention relates to a method of processing data using a network. In this embodiment, the network comprises forming a display pipeline by selecting and concatenating at least two nodes from a plurality of nodes in the network on the fly (i.e., in real time) and processing the data using the display pipeline.

Another embodiment of the present invention relates to a method of programming an A/V system using a network. In this embodiment, the network comprises generating at least one trigger at an end of a first picture and obtaining at least one register update list from a main memory. The network notifies a decoder about the end of the first picture and configures at least one node in the network for a second picture. The network enables the at least one node, obtains the second picture

from a frame buffer, and provides the second picture to a display pipeline in the network.

These and other advantages and novel features of the present invention, as well as details of an illustrated embodiment thereof, will be more fully understood from the following description and drawings.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

FIG. **1** illustrates one embodiment of a block diagram of an A/V decoder in accordance with the present invention;

FIG. **2** illustrates another embodiment of a block diagram of an A/V decoder in accordance with the present invention;

FIG. **3** illustrates one embodiment of a block diagram of an A/V system having a network in accordance with the present invention;

FIG. **4** illustrates another embodiment of a block diagram of an A/V system having a network in accordance with the present invention;

FIG. **5** illustrates one embodiment of a block diagram of a network environment for videoprocessing modules;

FIG. **6** illustrates another embodiment of a block diagram of a network environment in accordance with the present invention;

FIG. **7** illustrates one embodiment of a register DMA controller in accordance with one embodiment of the present invention;

FIG. **8** illustrates embodiments of block diagrams of nodes in accordance with the present invention;

FIG. **9** illustrates one embodiment of an entry node in accordance with one embodiment of the present invention;

FIG. **10** illustrates one embodiment of a network module in accordance with one embodiment of the present invention;

FIGS. **11**A, **11**B, **11**C, **11**D, **11**E, **11**F and **11**G illustrate embodiments of switched used in a network module in accordance with one embodiment of the present invention;

FIG. **12** illustrates one embodiment of a programming model in accordance with one embodiment of the present invention;

FIG. **13** illustrates one embodiment of a high level flow chart of a programmable method using at least one node in accordance with one embodiment of the present invention; and

FIG. **14** illustrates three methods used to write or implement control registers in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The following description is made with reference to the appended figures.

One embodiment of the present invention relates to a network environment. More specifically, one embodiment relates to a network environment in an A/V decoder device that decodes one or more input data streams with multiple output rates using a single clock reference. This embodiment enables video processing modules having multiple time bases to be implemented using a single clock reference (alternatively referred to as a "system clock"). FIGS. **1** and **2** illustrate block diagrams of embodiments of an A/V decoders in accordance with the present invention.

FIG. **1** illustrates one embodiment of a high level block diagram of an embodiment of an A/V decoder, generally designated **110**. More detail about the A/V decoder is provided in U.S. patent application Ser. No. 10/300,371 filed Nov. 20, 2002, titled "A/V Decoder Having A Clocking Scheme That Is Independent Of Input Data Streams", the complete subject matter of which is incorporated herein by reference in its entirety. In the illustrated embodiment, the decoder **110** comprises a system time reference recovery device **112** (alternatively referred to as an "STR recovery device") having one or more input data streams **118**.

The STR recovery device **112** is illustrated communicating with an A/V data processing device **114**. In one embodiment of the invention, STR refers to a reference time value. It is anticipated that different or more complex systems are also possible and within the scope of the present invention. For example if the A/V decoder **110** has more than one data source, the decoder may include more than one STR recovery device, where the number of STR recovery devices may or may not correspond to the number of data sources.

As an alternative to the MPEG scheme, an A/V system incorporating an A/V decoder may accept analog television signals as inputs. In this embodiment, the analog video input goes through, and is processed or decoded by, the A/V data processing device **114**, which may comprise a video decoder or VDEC. Likewise, analog audio goes through, and is processed or decoded by, the A/V data processing device **114** which may further comprise a BTSC audio decoder (alternatively referred to as a "ADEC" or "BTSC").

One embodiment of the present invention uses a system clock (a fixed system clock for example) to control the data processing. More specifically, the system clock may be used to control the data process in a network in accordance with the present invention. It is contemplated that the STR recovery device **112** may be locked to the analog video line rate. The analog hysncs are converted into a pseudo-STR using a simple counter in one embodiment. The STR recovery device **112** locks to this pseudo-STR and broadcasts the recovered STR to the rest of the decoder **110**. The broadcast STR is used to control the output rates as provided previously.

FIG. **1** further illustrates a rate managed output device **116**, which is illustrated as communicating with the data processing device **114**. In the illustrated embodiment, the rate managed output device **116** has one or more A/V outputs **120**, which are output at the same or different rates. In FIG. **1**, three A/V outputs, generally designated **120**, are illustrated. For example, one A/V output is output at 29.999 frames per second (alternatively referred to as "fps"), one is output at 30.001 fps and one is output at 30.000 fps.

In one embodiment, the A/V data processing device **114** includes a network environment for video processing modules. The data processing device **114** bases audio and video processing on multiples of a single, fixed clock, a 27 MHz crystal clock for example. It is contemplated that, as a single fixed clock is used, the processing is not constrained by clock boundaries. Video and audio may be muxed between modules. It is further contemplated that such architecture may be made orthogonal, and easy to control.

In accordance with one embodiment, all data, including all audio and video data, is processed by a network environment and transferred using a "pull" model or mode, even though typical A/V streams (e.g., MPEG) are adapted to operate according to a push model or mode. The outputs request data as needed. Each module in the A/V decoder **110** may supply data to its outputs at the rate it is requested. Because a pull model or mode is used, the data processing clock (i.e., the system clock) is not tied to the input data rate. For example, the audio decoder may be clocked at 243 MHz, 133 MHz, or any other reasonable rate. The audio decoder clock does not need to "track" the input data rate.

Conventional A/V decoders use a VCXO or VCXO-PLL to lock the chip clock to the input data rate. However, one

embodiment of the present invention uses rate managed output devices **116** and the associated SRC devices to change or adjust the video and audio output rates.

It is contemplated that, in one embodiment of the present invention, the output data rate tracks the STR. If the A/V decoder decodes multiple video streams, there may be multiple STRs. Each output data rate tracks an associated STR. The process of controlling the output rates may be called "rate management." In one embodiment, the rate managed output device **116** (alternatively referred to as a "output rate manager" or "output rate manager PLL"), comprising for example a digital PLL, is used to compare the output rate with the STR, and adjust the output rate accordingly, such that the output data rate matches the STR and the input data rate. In one embodiment, the A/V decoder may include several output rate managers, one for each output of the A/V decoder. More detail about rate managers is provided in U.S. Provisional Application No. 60/420,344 filed Oct. 22, 2002, titled "Data Rate Management System and Method for A/V Decoder".

FIG. **2** illustrates another embodiment of a block diagram of an A/V decoder, generally designated **210**, in accordance with one embodiment of the present invention. In the illustrated embodiment, the decoder **210** comprises an STR recovery device **212** having one or more input data streams **218** and a STR broadcast output.

In the illustrated embodiment, the input data streams (alternatively referred to as "system clock sources" or "system reference sources") **218** comprise an MPEG (PCR/SCR) stream, a 656 (hysnc) stream and a VDEC (hysnc) stream. While three input streams are illustrated, more complex systems, having more or different input data streams are contemplated. In the illustrated embodiment, the input time references are MPEG PCR/SCR values. However, for analog video or ITU656 video inputs, the hsync timing may be used as the time reference or a fixed timing reference may be used for PVR playback.

The STR recovery device **212** is illustrated as communicating (indirectly in this embodiment) with a data processing device **214**. In one embodiment, the SRT recovery device **212** controls the output data rates (in conjunction with a rate managed output and SRC devices). The data processing device **214** is adapted to decode, capture, play back and produce graphics, etc. from the data inputs (i.e., the input data streams **218**) using a fixed clock or timing reference. That is the data processing devices may decode, capture, play back and produce graphics, etc. using a fixed clock (i.e., the system clock for example). In one embodiment, the data is supplied to an output device or buffer **222** as requested (i.e., the output device requests data from the data processing device or the data is "pulled"). It is contemplated that, in one embodiment, the data processing device **214** comprises or includes a network environment for video processing modules in accordance with the present invention.

A rate managed output device **216** is illustrated as communicating (indirectly in this embodiment) with at least the data processing device **214**. More specifically, the rate managed output device **216** communicates with the STR recovery device **212** and the output device **222**. In the illustrated embodiment, the rate managed output device **216** comprises at least local STR and compare devices **215** and **217** respectively, while the output device **222** comprises at least an SRC device **223**.

In one embodiment, the output device **222** outputs data **220** at a fixed clock rate (i.e., the system clock rate) as it is requested. The output device **222** submits data requests to the data processing device **214**, and thus pulls the data. The data request is also submitted or mirrored to the rate managed output device **216**, where it is compared with the STR broadcast in the compare module **217**. A rate control signal is communicated to the output device **222** (specifically the SRC device **223**), ensuring that the data **220** is output at the fixed clock rate, and the output data rate matches the input data rate. The digital sample rate converter converts data from an input sample rate to an output sample rate. In one embodiment, the output sample rate may differ from the input sample rate. By adjusting the SRC parameters, the rate managed output device **216**B changes the rate of the sample rate at the input of the SRC device **223**B. This change to the sample rate changes the rate the data is requested from the data processing device **214**B.

FIG. **3** illustrates one embodiment of a block diagram of an A/V system, generally designated **300**, having a network in accordance with the present invention. It is contemplated that the illustrated A/V system may be similar to those A/V systems provided previously. It is also contemplated that the network may be used in different systems. In this embodiment, system **300** includes a decoder **310** (an MPEG decoder for example) adapted to receive video inputs or data **308**. In this embodiment, the decoder **310** includes one or more STR recovery devices **312**, used, with the system clock (a fixed system clock for example) to control the data processing similar to that provided previously. However, other decoders, with or without STR recovery devices are contemplated.

A memory or frame buffer **314** is illustrated coupled to the decoder **310** and receives data therefrom. The memory **314** is shown coupled to network **316** as illustrated, which is adapted to transport and process video or data, outputting video out or data **320**. In one embodiment, the network **316** is adapted to support a pull data flow. The network **316** includes one or more counters **318** (coupled to the STR recovery device via feedback loop **322**) that, along with the rate managed output device (not shown) control the data rate of the output.

FIG. **4** illustrates one embodiment of a block diagram of a network, similar to the network **316** of FIG. **3** in accordance with the present invention. In this embodiment, the network **416** is adapted to receive video-in **408** (from a memory for example) and output video out **420**.

FIG. **4** further illustrates at least one display pipeline **440** inside the network **416**. In one embodiment of the present invention, the display pipeline **440** is changeably formed by chaining, coupling or concatenating one or more network nodes together, depending on the network requirements, on the fly (i.e., in real time). It is contemplated that the nodes may be re-configured, so that a plurality of display pipelines **440** may be formed, each pipeline having different functionality depending on the nodes that are concatenated together. Moreover, in one embodiment, it is contemplated that the network **440** may change the display pipeline **440** every $1/60^{th}$ of a second for example.

In this embodiment, a register DMA controller **442** (alternatively referred to as an "RDC") is illustrated coupled to the network **416** and one or more register update lists **446** (alternatively referred to as an "RUL"). The RDC **442** is adapted to support multiple, configurable pipelines **440** by accessing and fetching (i.e., obtaining) one or more instructions from the RUL **446** and providing such instructions to the display pipeline **440**. In one embodiment, the RDC **442** accesses the RUL **446** (fetching the instructions) in response to the one or more trigger signals **444** (real time DMA trigger signals or events generated by the last node in the pipeline **440** for example). It is contemplated that, if the network **416** did not have an RDC **442** associated therewith, the network **416** would have to reconfigure the pipeline one register at a time.

FIG. **5** illustrates one embodiment of a block diagram of a network environment (alternatively referred to as a "display engine") for video processing modules in accordance with the present invention. The network, generally designated **500**, is adapted to support a pull data scheme and comprises at least a register DMA controller, one or more nodes, one or more links, and one or more network modules. In this embodiment, the register DMA controller **510** (or register DMA controller) is responsible for register access within the system **500**. The register DMA controller **510** connects the register bus **512** (alternatively referred to as "RBUS") with the video register bus **514** (alternatively referred to as "VBUS").

The system **500**, in one embodiment, further comprises one or more nodes **516** (two nodes **516**A & **516**B are illustrated). Nodes **516** are modules that process video information (nodes **516**A & **516**B are illustrated having video-in signals **514** and video-out signals **526** respectively). Some examples of nodes comprise video scalers, 2D graphics compositors, video encoders, etc.

FIG. **5** further illustrates one or more links **518** (links **518**A & **518**B are illustrated). In this embodiment, the links **518** comprise a set of signals or buses that tie or connect at least two nodes together (link **518**A is illustrated coupling node **516**A to network module **520** while link **518**B is illustrated coupling network module **520** to node **516**B). The links **518** are adapted to transfer information using a predefined protocol. More detail about the links is provided in U.S. Provisional Application No. 60/420,347 filed Oct. 22, 2002, titled "Video Bus For a Video Decoding System", the complete subject matter of which is incorporated herein by reference in its entirety.

Additionally, system **500** comprises one or more network modules **520** that, in this embodiment, are specialized nodes that don't perform video processing functions. Rather, the network module **520** connects at least two or more links **518** together, routing information between them. In general, the system **500** may include a number of pipelines (i.e., display pipelines) formed by chaining multiple nodes together. Each pipeline starts at one or more nodes **516**, where it is contemplated that each node has a memory interface to a frame buffer (not shown in FIG. **5**). Functions are added to the pipeline by cascading more nodes to the pipelines. Finally, a pipeline ends at one or more nodes, where each such node is a desired output channel.

In accordance with the present invention, the register bus or RBUS **512** is connected to the video register bus or VBUS **514** through the register DMA controller **510**. In this embodiment, both buses use identical signaling and protocols. The register DMA controller **510** acts as a slave to the RBUS **512** and forwards all the transactions to VBUS **514**. In addition, register DMA controller **510** may perform one or more Register DMA operations, which comprises decoupling a host from video timing by automating mode changes.

In one embodiment, register DMA controller **510** includes four interfaces. There are two register bus interfaces, one interface **528** coupling the register DMA controller **510** to RBUS **512** and the other interface **530** coupling the register DMA controller **510** to VBUS **514**. The third interface is a memory bus interface **532** coupling the register DMA controller **510** to the memory bus **522** (alternatively referred to as "MBUS"). The memory bus **522** is used to access register writes from an external memory. Finally the last interface **534** comprises an array of signals coming from at least one of the nodes **516**, which are used as DMA triggers.

In accordance with one embodiment, display modes are configured or changed using control registers. Instead of updating the display modes one at a time, the host uses the register DMA controller, feature or operation (alternatively referred to as the register DMA controller in FIG. **5**) to automate the process. In this embodiment, the Register DMA comprises three entities: a register update list, a DMA descriptor and a DMA trigger as provided below.

FIG. **6** illustrates another embodiment of a block diagram of a network or display engine according to the present invention. In this embodiment, the network, generally designated **600**, video processes modules and is further adapted to support a pull data scheme. Register DMA controller **610** is responsible for register accesses within the network **600** (i.e., the register DMA controller **610** is a register DMA). The register DMA controller **610** connects the register bus or RBUS **612** with the video register bus or VBUS **614**.

In this embodiment, the RBUS **612** comprises at least one video-in module **624** coupled to and communicating with at least one node (Node **616**A for example). Further the RBUS **612** may comprise a memory interface **636** coupled to and communicating with at least the memory bus **622** (using memory bus interface **632** for example) and main memory **638**; and a host interface **640** communicating with at least the memory bus **622** (using memory bus interface **632** for example), host **642** and register DMA controller (using interface **628** for example).

The network **600**, in this embodiment, comprises a plurality of nodes **616** (nine nodes **616**A-**616**I are illustrated) adapted to process video information. While only nine nodes are illustrated, more (or less) nodes are contemplated. Again, the nodes **616** process video information (node **616**A is illustrated having video-in signals **624** communicating therewith, while nodes **616**H and **616**I are illustrated having video-out signals **626**A and **626**B respectively communicating therewith). In this embodiment an optional MPEG decoder **617** is illustrated coupled to node **616**C, and communicating with video bus **614**, register DMA controller **610** and memory bus **622**.

FIG. **6** further illustrates a plurality of links **618** (12 links **618**A-**618**L are illustrated). Again, while **12** links **618** are shown, a different number is contemplated. In this embodiment, the links **618** comprise a set of signals or buses that tie at least two nodes **616** together and transfer information using a predefined protocol.

Additionally, network **600** comprises a plurality of specialized nodes or network modules **620** that, in this embodiment, connect at least two or more links **618** together, routing information therebetween. It is again contemplated that, in general, the network **600** may include a number of display pipelines formed by chaining multiple nodes together using the network modules **620** to switch between the nodes **616**, thus varying or changing the pipeline. Each pipeline starts and ends at one or more nodes **616**, where it is contemplated that each node has a memory interface **636** to a frame buffer. Functions are added to the pipelines by cascading that pipeline with more nodes.

In accordance with the present invention, the RBUS **612** is connected to the VBUS **614** through the register DMA controller **610**. In this embodiment, both buses use identical signaling and protocols. The register DMA controller **610** acts as a slave to the RBUS **612** and forwards all the transactions to VBUS **614**. In addition, register DMA controller **610** is a Register DMA, decoupling the host from video timing using automating mode changes.

In accordance with one embodiment, one or more modules (nodes for example) process pixels or other data as fast as possible, responding to an incoming accept signal transmitted via the links to stall pixel processing at the current cycle. The modules communicate using a ready-accept protocol trans-

mitted via the links (i.e., a protocol using ready and accept signals alternatively referred to as a handshake protocol). More fully described in U.S. Provisional Application No. 60/420,347 as provided above.

It is contemplated that, in one embodiment, the links contain information that may be used to delineate the start of a line of video information, and the start of a field or frame of video information. StartLine information is active only during the first beat of the first pixel of a line. StartField information indicates the start of a field/frame, or the end of a field or frame. This signal is active only during the first beat of the first pixel of the first line of a field or frame or the first beat of the last pixel of the last line of the field or frame (i.e., end frame). It is contemplated that in this embodiment, unlike other video standards such as Rec656, StartLine and StartField information is not separated by blanking lines or blanking pixels. All blanking information is removed from the data structure of the bus or link.

Essentially, the field of data is sent as a contiguous array of data on the bus, without blank pixels. This removes the strict timing relationship between the arrival time of the StartField on the bus, and the Vertical Sync information defined by NTSC or SMPTE standards. The output module inserts the correct timing information which governs the pull-rate of the data flow across the bus. Further, all modules supply pixel data to the output module at or ahead of the time the pixels are needed. This is governed by the flow control ready/accept signals (i.e., ready-accept protocol).

FIG. **7** illustrates one embodiment of register DMA controller **710** including four interfaces similar to that provided previously. There are two register bus interfaces, one interface **728** coupling the register DMA controller **710** to RBUS **712** and the other interface **730** coupling the register DMA controller **710** to VBUS **714**. The third interface is a memory bus interface **732** coupling the register DMA controller **710** to the memory bus **722**. Finally, interface **734** comprises an array of signals (**0**-n) coupled to at least one of the nodes **716**, which are used as DMA triggers, and generally designated **735**. More detail about the register DMA controller is provided in U.S. patent application Ser. No. 10/300,370 filed Nov. 20, 2002, titled "Hardware Assisted Format Change Mechanism in a Display Controller", the complete subject matter of which is incorporated herein by reference in its entirety.

FIG. **8** illustrates different embodiments of the nodes, generally designated **816**, used in one embodiment of the network. The network, in accordance with the present invention, is adapted to perform video processing functions similar to a display engine, including video playback, scaling, encoding, etc. It is contemplated that each node **816** in the network may be generally divided into three categories according to its position in a display pipeline: entry, exit, and intermediate. Video data enters a display pipeline at an "entry node" designated **816**A and leaves at an "exit node" designated **816**B. All the nodes in-between are referred to as "intermediate nodes" or "nodes" designated **816**C. Examples of entry nodes **816**A include MPEG display feeders, playback engines, etc. Examples of exit nodes **816**B include video encoders, capture engines, etc. Examples of intermediate nodes **816**C include scalers, compositors, etc. It is further contemplated that the position of each node in the pipeline configuration is not fixed; rather its position varies depending on the display pipeline (i.e., an entry node in one pipeline may be an intermediate node in another display pipeline).

As illustrated, the nodes **816** each generally include at least one input and output interface or link **818** communicating therewith. It is contemplated however that each node **816** is adapted to have multiple input or output links **818**A & **818**B coupled thereto and communicating therewith (a compositor for example has multiple input links). Furthermore, each node **816** may also have an optional RBUS **814**, MBUS **822** or some other optional auxiliary interface **880** (a DMA trigger for the register DMA controller for example) communicating therewith. If the node **816** is an entry node **816**A, it is contemplated that the input link is an MBUS interface **822** as illustrated. For exit nodes **816**B, the output is replaced by a dedicated output **850** (e.g., a memory interface for a capture engine or an analog video output for a video encoder).

As provided previously, a display pipeline in the network starts or begins at one or more entry nodes **816**A. The entry node **816**A is responsible for feeding video to the downstream nodes **816** and includes, for example, MPEG display feeders and playback engines. In one embodiment, the input to an entry node **816**A may comprise RBUS and memory interfaces. Its output may comprise one or more output links **818**B. In addition, the entry node **816**A may include one or more auxiliary interfaces **870** such as a DMA trigger for the register DMA controller.

The intermediate node **816**C, in one embodiment, may have specific functions comprising scaling, compositing, etc. One or more nodes are added to a display pipeline as its features are used to satisfy certain output requirements. In general, the input and output of an intermediate node **816**C comprises one or more links **818**A & **818**B as provided previously. In addition, the intermediate node **816**C may have an optional register bus interface or some other auxiliary interface **870** coupled thereto and communicating therewith.

As provided previously, the display pipeline ends at exit node **816**B, which may comprise a video interface such as a composite signal encoder or capture engine for example. In general, the inputs to an exit node **816**B consist of an input link **818**, an optional register bus **812**, and a video output or a memory bus interface **870**.

In addition to the functions described previously, the exit nodes **816**B may include some debugging functions. For example, a checkpoint register may be written into control packets and read by the register bus **812**. This register is programmed in every field to a field dependent number. At the same time, a host may check the progress of the video packets by monitoring this register through the register bus **812**.

It is contemplated that exemplary embodiments of the nodes **812** should meet certain requirements in order to maintain intra- and inter-packet synchronization. For example, nodes should be adapted to forward incoming control packets without being modified. If the node is a multi-input node, one particular input may be designated as the primary link, such that the control packets of the primary links are forwarded, while control packets from other inputs are terminated.

It is contemplated that exemplary nodes **816** process and output packets in their arriving order. If the node is a multi-input node, it may only operate on packets corresponding to the same field in time. For example, if the node **816** is a graphics compositor, the i-th field of one input may be combined with the i-th field of another input. If the active input is not receiving any data, other inputs and the outputs may be stalled.

If the exemplary node **816** is a multi-output node, control and video packets may be forwarded to all the output links. Stalling by one of the output links stalls the inputs as well as the other outputs. Unused input or output links of such exemplary nodes **816** may be disabled using RBUS **812** and the control register. The disabled link may be excluded from

controlling other inputs or outputs. For a pipelined node, the next field's control packet should not have any effect on current field's video packet.

Another embodiment of an entry node, generally designated **916**, is illustrated in FIG. **9**. It is contemplated that the entry node **916**, in addition to having input links **918**A (comprising RBUS **912** and MBUS **922**) and output link **918**B, may include an optional register referred to as a "register window" generally designated **950**. In one embodiment, the register window **950** is adapted to insert control packets into the output link **918**B (using DMA engine **952** and mux **954**). In this embodiment, a write to a specific location outputs a 32-bit control word.

FIG. **10** illustrates one embodiment of a network module **1020** in accordance with the present invention. In this embodiment, the network module **1020** comprises a plurality of network interfaces or links generally designated **1018** and switches, described in greater detail below. In this invention, one or more network modules are used to connect one or more nodes, forming a display pipeline. Since the nodes may be re-configured, it is contemplated that display pipelines having different functionality may be implemented for different applications. In other words, the display pipelines are dynamic and not static.

The network interfaces **1018**, in this embodiment, comprise input and output links **1018**A & **1018**B respectively, and an optional register bus **1012**. In this embodiment, m input links **1018**A and n output links **1018**B are illustrated, where m and n may be the same or different. It is contemplated that m may be greater than, equal to or less than n (i.e., the number of input links **1018**A may be greater than, equal to or less than the number of output links **1018**B).

It is contemplated that different types of network modules may be used within the register DMA controller or display engine. The network module **1020**, in accordance with the present invention, is comprised of an array of switches coupled together using predefined topology. This topology determines the network module's routing capabilities, as well as the implementation cost.

In accordance with the present invention, a multi-stage network module may comprises at least one 2×2 switch box **1121** as illustrated in FIG. **11**A. Although a 2×2 switchbox is discussed, other switches are contemplated. Each switch box **1121** is, in this embodiment, a two-input two-output interchange device. The switch box has four functions as illustrated: straight, designated **1170**A; exchange, designated **1170**B; upper broadcast, designated **1170**C; and lower broadcast, designated **1170**D. For bijections interchanges (i.e., one-to-one connections) such broadcast functions are not used.

It is contemplated that, in the present invention, multiple switch boxes may be coupled together to form a subset of multi-stage network modules. As illustrated in FIG. **11**B, a single stage shuffle-exchange network module, generally designated **1101**, may be formed by connecting or coupling N/2 switch boxes **1121** (where N equals the number of inputs) after a hardwired shuffle function **1172**. The shuffle function **1172** is, in this embodiment, a single bit rotation of a network address. The switch boxes **1121** perform an exchange function **1174**, which is a single bit negation of a network address.

FIG. **11**C illustrates an example of an N×N Omega network module, generally designated **1123**, formed by cascading log(N) stages of shuffle-exchange network modules **1101** as illustrated in FIG. **11**B. As a result, such network module **1123** has a complexity of O(N log(N)).

Other networks having topologies similar to the network module **1123** of FIG. **11**C are illustrated in FIGS. **11**D-**11**F. For example, an n-cube network **1125** illustrated in FIG. **11**D may be formed from an network module **1123** by reversing the signal direction and swapping the middle two switch boxes **1121** (switch boxes **1121**F and **1121**G for example) in the second stage. In one embodiment, the n-cube network **1125** uses only two-function switch boxes instead of four-function switch boxes used in the network module **1123**.

The n-cube network **1125** of FIG. **11**D may be converted to another type of network module referred to as a butterfly network module generally designated **1127** and illustrated in FIG. **11**E. This butterfly network module **1127** may be formed from an n-cube network **1125** by swapping the first two shuffle functions and replacing the last shuffle function by re-mapping the network addresses.

It is contemplated that the multi-stage shuffle exchange network modules provided previously are adapted to provide connections from any input to any output at a very low cost. However, such multi-stage shuffle network modules are considered blocking networks. Simultaneous connections of more than one input/output pair may result in conflicts in the links and switches. For example, in the network module **1123** illustrated in FIG. **11**C, connections from 5 to 0 and 7 to 1 may not be established simultaneously.

FIG. **11**F illustrates a variation of the butterfly network **1127** of **11**E and referred to as the Beneš network module, generally designated **1129**. In the illustrated embodiment, the N×N Beneš network module **1129** comprises 2 log(N)−1 levels. The first and last log(N) levels comprise two butterfly network modules, where the middle level is shared between the two butterflies. The Beneš network module **1129** is a rearrangeable network module. However, it is contemplated that any new connections in the Beneš network module may require a complete reordering of the internal connections.

FIG. **11**G illustrates a crossbar network module **1156**, comprising a plurality of switches **1121**, similar to the crossbar switches provided previously. In this embodiment, the crossbar network module **1156** is a non-blocking network, adapted to handle all possible connections without blocking. This enables the network to map any input to any output. Furthermore, a connection may be set up or torn down dynamically without affecting the existing connections. In one embodiment, the switch boxes **1121** in the crossbar network module **1156** are different from those provided previously, representing a tap from the horizontal data bus to the vertical data bus.

It is contemplated that one or more embodiments of the present invention are adapted to provide at least one display pipeline of a plurality of display pipelines having a data rate different from at least one other display pipeline of the plurality of display pipelines. It is also contemplated that at least one display pipeline of a plurality of display pipelines may have a data rate that is independent of at least one other display pipeline of the plurality of display pipelines (using a flow control valve for example). FIG. **12** illustrates one embodiment of a block diagram of a programming model using an entry node **1212** (a video feeder for example) similar to the entry nodes provided previously. The video feeder or entry node **1212** is adapted to fetch or capture a decoded picture **1210** from a frame pipeline and feed it to the display pipeline. In this embodiment, it is contemplated that the MPEG video decoder **1224** is a TITAN decoder, although other decoders are contemplated (an MVP examples of which are described in U.S. patent application Ser. No. 10/114,798 filed Apr. 1, 2002, titled "Video Decoding System Supporting Multiple Standards" incorporated herein by reference in its entirety.

As illustrated, the register DMA unit **1218** is connected to at least some registers shared with the TITAN decoder or

other video decoder **1224** through bus register DMA controller **1222**. The register DMA **1218** is adapted to fetch one or more predefined RULs **1220** from the main memory. One of these entries may be written to the video decoder or TITAN decoder's share register, which is used to notify the MPEG decoder **1224** about the end of the picture. In this embodiment, the video output **1216**, coupled to the display pipeline **1214**, may comprise a video encoder or capture engine for example. The video output **1216** is adapted to generate one or more DMA trigger signals tied to the register DMA **1218**.

FIG. **13** illustrates one embodiment of a high level flowchart of a method of programming an A/V system (an A/V system having a network for example) using at least one node (entry node or video feeder for example) in accordance with the present invention. In this embodiment, during a frame or field time, the video output generates at least one DMA trigger at the end of a first picture as illustrated by block **1310**. In this embodiment the DMA trigger is communicated to the register DMA unit. The register DMA unit fetches or obtains at least one predefined RUL from the main memory (not shown) as illustrated by block **1312**. One of the RUL entries is written to the video decoder share register, which is used to notify the video decoder about the end of the picture as illustrated by block **1314**.

After the video feeder is configured for the next picture as illustrated by block **1316**, the video decoder re-enables the video feeder as illustrated by block **1318**. It is contemplated that, if the video feeder is double buffered, the video decoder may re-enable the video feeder before the end of the picture. The video feeder fetches or obtains a second or next picture from a frame buffer as illustrated by block **1320**. The video feeder **1312** feeds at least the second picture to the display pipeline as illustrated by block **1322**. It is contemplated that this programming method may end or repeat for one or more pictures.

In accordance with the present invention, control registers are utilized to set up the network module routing. Two types of control structures (i.e., individual stage control and individual box control) are discussed with respect to setting up or establishing such network module routing, although other control structures are contemplated. In individual stage control, the same register is used to set up all switch boxes within the same stage. In other words, all the switch boxes assume the same state. This simplifies the control design but may be considered inflexible. In individual box control, each switch box may be configured independently. This independent configuration generally requires more hardware when compared to the individual stage control, but it offers greater flexibility.

In addition to the two types of control structures, three methods for configuring network modules are discussed, although other methods are contemplated. One method to configure a network module comprises using an asynchronous control scheme, which is considered the simplest of the three. The switch boxes of the network module may be configured directly using the register bus by packing their control signals into a number of registers. The host may set up or tear down connections by programming different values into these registers. However, as the register writes are asynchronous to video timing, such register writes have to be handled carefully to avoid interrupting the display. In a non-blocking network module, this may be accomplished using a Register DMA. In a blocking or rearrangeable network module, additional buffering may be used at the network modules' outputs in order to accommodate the pipeline bubbles created during the reconfiguration.

Another method for configuring network modules comprises semi-synchronous control, which is an extension of the asynchronous control scheme discussed previously. This extension may be accomplished using double buffering and a trigger mask. Firstly, semi-synchronous control double buffers all the switch box control registers. The front registers control the switch boxes while the back registers are programmed by the host. The front registers are updated by the back registers when a force update bit is set or a trigger signal is generated by the trigger mask.

Secondly, the semi-synchronous control method uses a trigger mask. In this embodiment, the trigger mask contains an array of bits, each bit corresponding to an input port of the network. A trigger is generated at the end of a video stream for which the mask bit is set. During initialization, the host uses a force update bit to program the network module. Afterward, the host reconfigures the network module by programming the back registers and setting a mask bit accordingly. At the end of the video stream corresponding to the mask bit, the network is automatically reconfigured. One benefit associated with such exemplary semi-synchronous control method is that reconfiguration may be automatically synchronized to video timing.

Another method for configuring network modules comprises synchronous control. This method requires that the network connections be changed synchronously with video streams. Such synchronization may be achieved using control packets to configure the network modules. The network module creates a connection using the control packets, forwarding subsequence packets according to the resulting route. If a packet is forwarded to an occupied output link, the packet is stalled until that link is free.

In accordance with one embodiment of the present invention, the network carefully accommodates format changes for the display engine, as even a slight mistake may be noticeable on a display. In accordance with one embodiment of the present invention, control registers are used to set one or more nodes in the network. Three methods for implementing the control registers are discussed, although other methods are contemplated. One method, referred to as "single buffering", relies on the fact that the values of some control registers are designated "don't care" during certain periods of time during the transmission (e.g., vertical blanking). These registers may be modified freely during such period without any damaging effect.

Another method for implementing the control registers comprises using double buffering, which may be implemented using a pair of front (i.e., "current") and back (i.e., "next") registers. The front register provides the current control information while the back register may be updated in the background. A properly timed signal is used to copy the content of the back register to the front register. This method may be used in situations where the window for register updating is small or the control doesn't allow any slack for a format change.

Yet another method for implementing control registers comprises an inband control method, wherein control information is embedded within the data stream, such that the control information and the data stream share a single path. This method r utilizes synchronization between the control information and the data stream. It is contemplated that, in this method, format changes may be performed rapidly, even in a heavily pipelined design. This method is well suited for high performance designs such as 3D graphics processors.

FIG. **14** illustrates the three methods (i.e., Register bus, Register DMA, and control packets) used to write or implement control registers in accordance with the present invention. Each of these method supports certain types of control

register. While only three methods are discussed and illustrated, other methods are contemplated.

One method for writing or implementing control registers comprises using the register bus and supports single and double buffering. The host uses the register bus to directly program the control registers. The host further controls the write timing and ordering. In one embodiment, double buffering may be used to decouple the host from the video timing. However, since the registers are written one at a time using a relatively slow interface (i.e., the register bus), the process may be considered time consuming in comparison to the other methods.

Another method for writing or implementing control registers comprises using the register DMA and supports single and double buffering. The register DMA automates the register programming. The register DMA controller is used to stream predefined lists of register write into the display engine through the register bus. The write timing is controlled by the triggering signals generated by various nodes, thus the real-time requirement on the host is relaxed. In addition, this method may potentially eliminate most double buffering.

Yet another method for writing or implementing control registers comprises using control packets and supports all three control register types. A control packet may be fed into an entry node's register window using a register DMA. Using the control packets with single and double buffered control registers provides benefits similar to those provided by the Register DMA. The control packet may enable rapid format changes for inband control. However, such rapid format changes require extensive control register staging. Furthermore, such rapid format changes aren't used in video processing applications, as a format changes occur, at most, once per field.

It is contemplated that the Register DMA in accordance with the present invention may be an exemplary method used to implement format change. However, it is contemplated that the register bus may be used to handle simpler or ad hoc control register accesses, while control packets may be used as a complement to these methods in limited situations.

A flow control valve is used, in one embodiment of the invention, as a device to control data flow in a display engine. It is contemplated that such flow control valve or module may provide for independent data flow rates on one, two more display pipelines, and enable one or more display pipelines having different and/or independent data rates. The flow control valve sequences video data and controls information inside the display engine. Such valve acts primarily by stalling and restarting the flow control signals of at least one link. An exemplary flow control valve maintains synchronization between video and control with minimum effort. Four flow control valve modes (i.e., Manual On Manual Off, Manual On Auto Off, Auto On Manual Off and Auto On Auto Off) are discussed, although other modes are contemplated.

The Manual On Manual Off type of flow control valve may be turned on and off by writing to the valve's control register. The Manual On Auto Off type of flow control valve is turned on manually. However, the type of flow valve senses a trigger signal to shut itself off, where the signal may be an external signal or a bit from the content of a link (e.g., an end of field signal).

The Auto On Manual Off type of flow control valve is the opposite of the Manual On Auto Off type of flow control valve. However, in this embodiment, the Auto On Manual Off type of flow control valve uses an external trigger signal. The Auto On Auto Off type of flow control valve uses two trigger signal inputs: trigger on and trigger off.

In general, the front-end of a video decoder is responsible for producing pictures while the display engine consumes them. A frame buffer may be placed between the video decoder and the display engine as an intermediate storage.

However, it is contemplated that modern display engines may incorporate one or more front-end like features (compositing, graphics overlaying, windowing for example). These features are included in the display engine to eliminate the memory bandwidth required for handling the intermediate results. In accordance with one embodiment of the present invention, it is possible to perform multi-pass operations using a display engine by capturing its output in a frame buffer (for example to down scale a picture for PIP displaying or for non real-time compositing of a complicated graphics background). In addition, using multi-pass operation on a network (taking advantage of the flow control architecture of the network) in accordance with one embodiment of the present invention enables a data throughput greater than the video rate. As a result, some functions may be shared or reused between different video streams. More detail about the multi-pass operations is disclosed is provided in U.S. Provisional Application No. 60/420,308 filed Oct. 22, 2002, titled "Multi-Pass System and Method Supporting Multiple Streams of Video", incorporated herein by reference in its entirety.

Many modifications and variations of the present invention are possible in light of the above teachings. Thus, it is to be understood that, within the scope of the appended claims, the invention may be practiced otherwise than as described hereinabove.

The invention claimed is:

1. A network for processing data configured by a controller to form at least one display pipeline therein by dynamically selecting use of at least two selectable nodes from a plurality of selectable nodes and dynamically concatenating the selected at least two selectable nodes in the network together, wherein said at least one display pipeline has an independent data rate and a flow control module enables said independent data rate.

2. The network of claim **1**, wherein the network is further configured by said controller to form a plurality of display pipelines.

3. The network of claim **1**, further comprising at least two display pipelines having different data rates.

4. The network of claim **1**, further comprising a handshaking protocol configured to generate at least two display pipelines.

5. The network of claim **1**, wherein the network is further configured by said controller to change a functionality of said display pipeline by dynamically concatenating more than two nodes together.

6. The network of claim **1**, wherein said controller is a register DMA controller configured to support register access.

7. The network of claim **6**, wherein said register DMA controller is further configured to provide at least one instruction to form said display pipeline.

8. The network of claim **6**, wherein said register DMA controller is further configured to obtain at least one instruction from a register update list.

9. The network of claim **8**, wherein said register DMA controller is further configured to obtain said instruction in response to a trigger event.

10. A network for processing data comprising:

at least one display pipeline formed by dynamically selecting and concatenating at least two selectable nodes from

a plurality of nodes, wherein the plurality of nodes are configured to process the data;

at least one network module configured to route the data by connecting the at least two selectable nodes using at least one link communicating between each of the at least two selectable nodes and the at least one network module; and

a register DMA controller for dynamically configuring the at least one display pipeline.

11. The network of claim **10**, wherein the network is further configured to form a plurality of display pipelines.

12. The network of claim **10**, comprising at least two display pipelines having different data rates.

13. The network of claim **10**, comprising at least one display pipeline having an independent data rate.

14. The network of claim **13**, further comprising a flow control module configured to enable said independent data rate.

15. The network of claim **10**, further comprising a handshaking protocol configured to generate at least two display pipelines.

16. The network of claim **10**, further comprising a plurality of network modules connecting at least two links and further configured to route information there between.

17. The network of claim **10**, further comprising a video register bus coupled to said register DMA controller.

18. The network of claim **17** further comprising a register bus connected to at least said register DMA controller, wherein said video register bus and register bus use identical protocol and signals.

19. The network of claim **18**, wherein said register DMA controller forwards all transactions to said video register bus.

20. The network of claim **10**, wherein said register DMA controller is configured to perform register DMA operations.

21. A method of processing data using a network comprising:

forming a first display pipeline using a register DMA controller to dynamically select at least two selectable nodes in the network;

processing the data using said first display pipeline;

forming a second display pipeline using said register DMA controller to dynamically select at least two selectable nodes in the network; and

processing the data using said second display pipeline, wherein said first and second display pipelines are different.

22. A method of processing data using a network comprising:

forming a display pipeline by using a register DMA controller to dynamically select and concatenate at least two selectable nodes from a plurality of nodes in the network, wherein at least one network module connects the at least two selectable nodes using at least one link between each of the at least two selectable nodes and the at least one network module; and

processing the data using said display pipeline.

23. The method of claim **22**, wherein forming said display pipeline comprises dynamically coupling a plurality of nodes together.

24. The method of claim **22**, further comprising starting said display pipeline at an entrance node.

25. The method of claim **22** further comprising ending said display pipeline at an exit node.

26. The method of claim **22**, further comprising adding functionality to said display pipeline by dynamically cascading at least one additional node to said at least two nodes.

27. A method of programming an A/V system using a network comprising:

(a) generating, at a video output, at least one trigger at an end of a first picture;

(b) obtaining at least one register update list from a main memory in response to receiving the at least one trigger;

(c) writing at least one register update entry from said at least one register update list to at least one register of a decoder to notify said decoder about said end of said first picture;

(d) configuring at least one node in the network for a second picture;

(e) enabling said at least one node;

(f) obtaining said second picture from a frame buffer; and

(g) providing said second picture to a display pipeline in the network.

28. The programming method of claim **27** comprising repeating steps (a)-(g).

29. The method of claim **27**, wherein said at least one node is an entry node.

30. The method of claim **27**, wherein said at least one node is a video feeder.

31. The network of claim **27**, comprising at least two display pipelines having different data rates.

32. The network of claim **27**, comprising at least one display pipeline having an independent data rate.

33. A network for processing data configured by a controller to form at least one display pipeline therein by dynamically selecting use of at least two selectable nodes from a plurality of selectable nodes and dynamically concatenating the selected at least two selectable nodes in the network together, wherein a handshaking protocol is configured to generate at least two display pipelines.

34. A network for processing data configured by a register DMA controller to form at least one display pipeline therein by dynamically selecting use of at least two selectable nodes from a plurality of selectable nodes and dynamically concatenating the selected at least two selectable nodes in the network together, wherein said register DMA controller is configured to support register access and to obtain at least one instruction from a register update list in response to a trigger event.

35. A system, comprising:

a controller that is operable to configure two or more nodes in a data processing network to form two or more display pipelines within said data processing network; and

a flow control module that is operable to configure each of said two or more display pipelines such that a corresponding data rate of each of said two or more display pipelines is independent of each other.

36. The system according to claim **35**, wherein said controller dynamically concatenates said two or more nodes in said data processing network to form said two or more display pipelines.

* * * * *

Exhibit 2

U.S. Patent No. 6,744,387

US006744387B2

(12) **United States Patent**
Winger

(10) **Patent No.: US 6,744,387 B2**
(45) **Date of Patent: Jun. 1, 2004**

(54) **METHOD AND SYSTEM FOR SYMBOL BINARIZATION**

(75) Inventor: **Lowell Winger**, Waterloo (CA)

(73) Assignee: **LSI Logic Corporation**, Milpitas, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 7 days.

(21) Appl. No.: **10/191,596**

(22) Filed: **Jul. 10, 2002**

(65) **Prior Publication Data**

US 2004/0008769 A1 Jan. 15, 2004

(51) **Int. Cl.**[7] **H03M 7/00**
(52) **U.S. Cl.** **341/50**; 341/51
(58) **Field of Search** 341/50, 51, 107, 341/106

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,471,207 A * 11/1995 Zandi 341/107

OTHER PUBLICATIONS

Article–Predictive Quantizing Systems by J.B. O'Neil, Jr—Manuscript received Dec. 27, 1965.
Article—A method for the construction of Minimum–redundancy Codes by David A. Huffman, no date.
Article—A Compressioin Method for Clustered Bit–Vectors—Oct. 1978.
Article—pp. 399–401 of the publication entitled IEEE Transactions on Information Theory Published 1966.

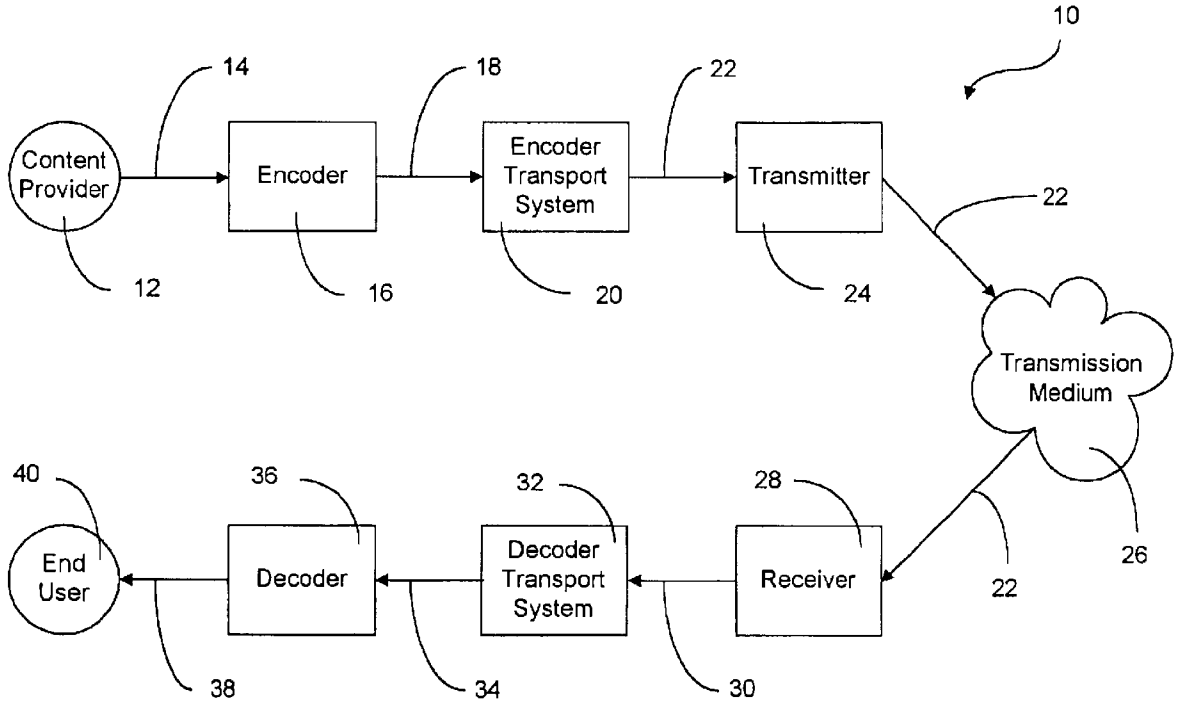
* cited by examiner

Primary Examiner—Brian Young
(74) *Attorney, Agent, or Firm*—Christopher P. Maiorana PC

(57) **ABSTRACT**

The present invention is directed to an improved method for the binarization of data in an MPEG data stream. The invention makes use of unary binarization to create codewords up until an index threshold. Once the threshold has been met, succeeding code symbols have appended to them an exp-Golomb suffix. This hybrid binarization scheme reduces the number of binary codewords to be processed by a Binary Arithmetic Coder (BAC), thus reducing the computation required by the BAC.

6 Claims, 4 Drawing Sheets



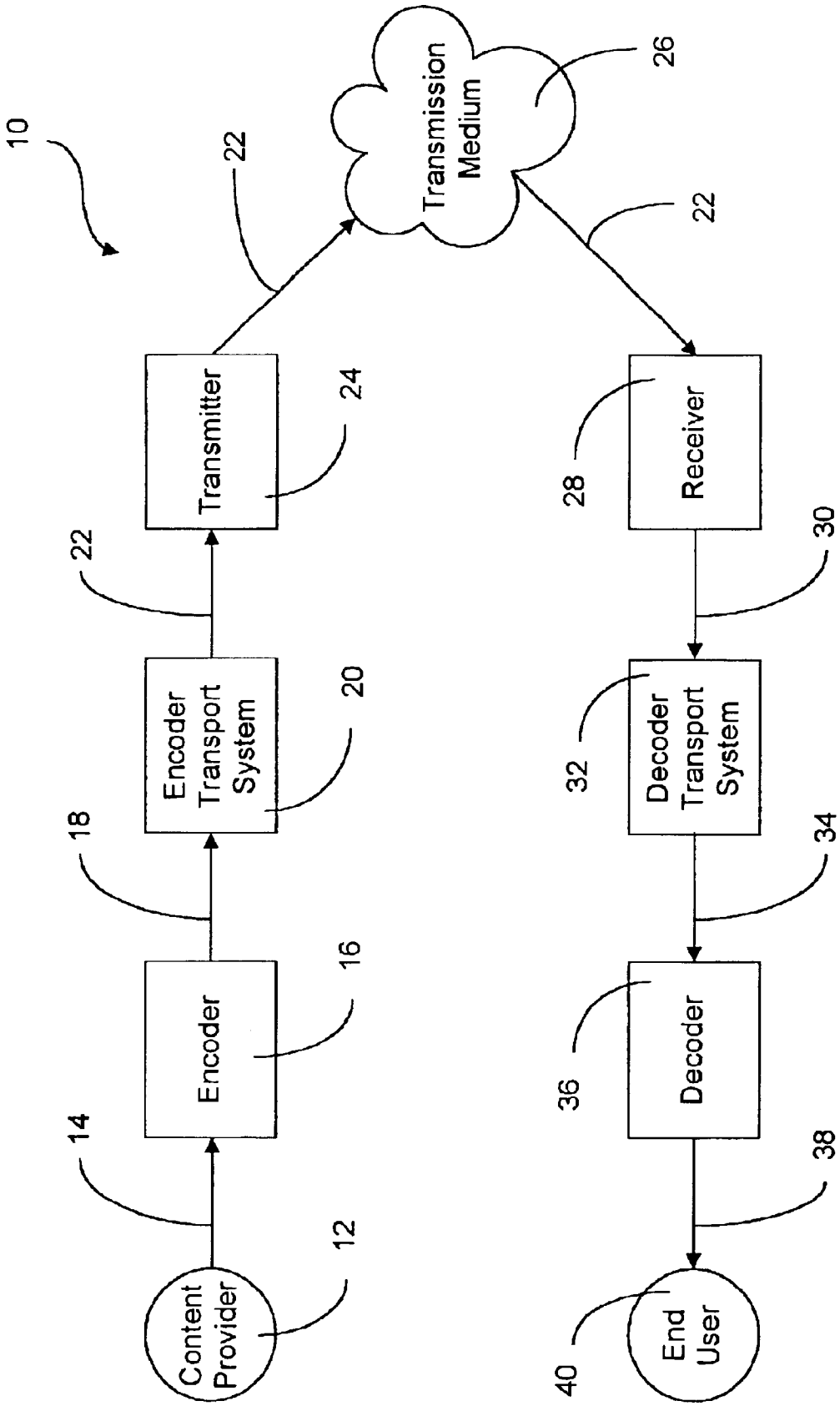
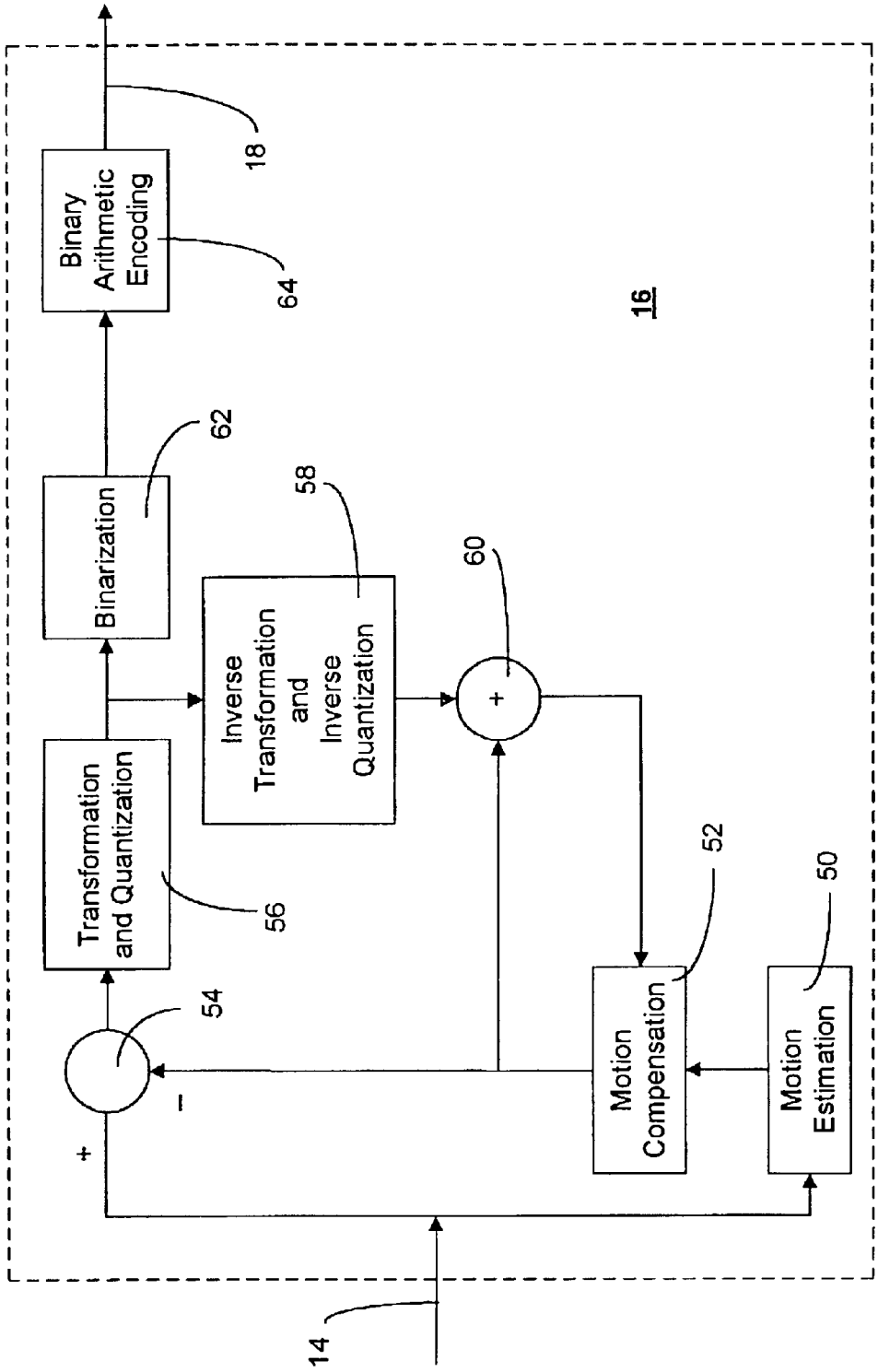

10
22
Transmission Medium
26
22
Transmitter
24
Receiver
28
22
30
Encoder Transport System
20
Decoder Transport System
32
18
34
Encoder
16
Decoder
36
14
38
Content Provider
12
End User
40

FIG. 1

FIG. 2


16
18
Binary Arithmetic Encoding
64
62
Binarization
58
Inverse Transformation and Inverse Quantization
60
+
Transformation and Quantization
56
54
−
+
52
Motion Compensation
50
Motion Estimation
14

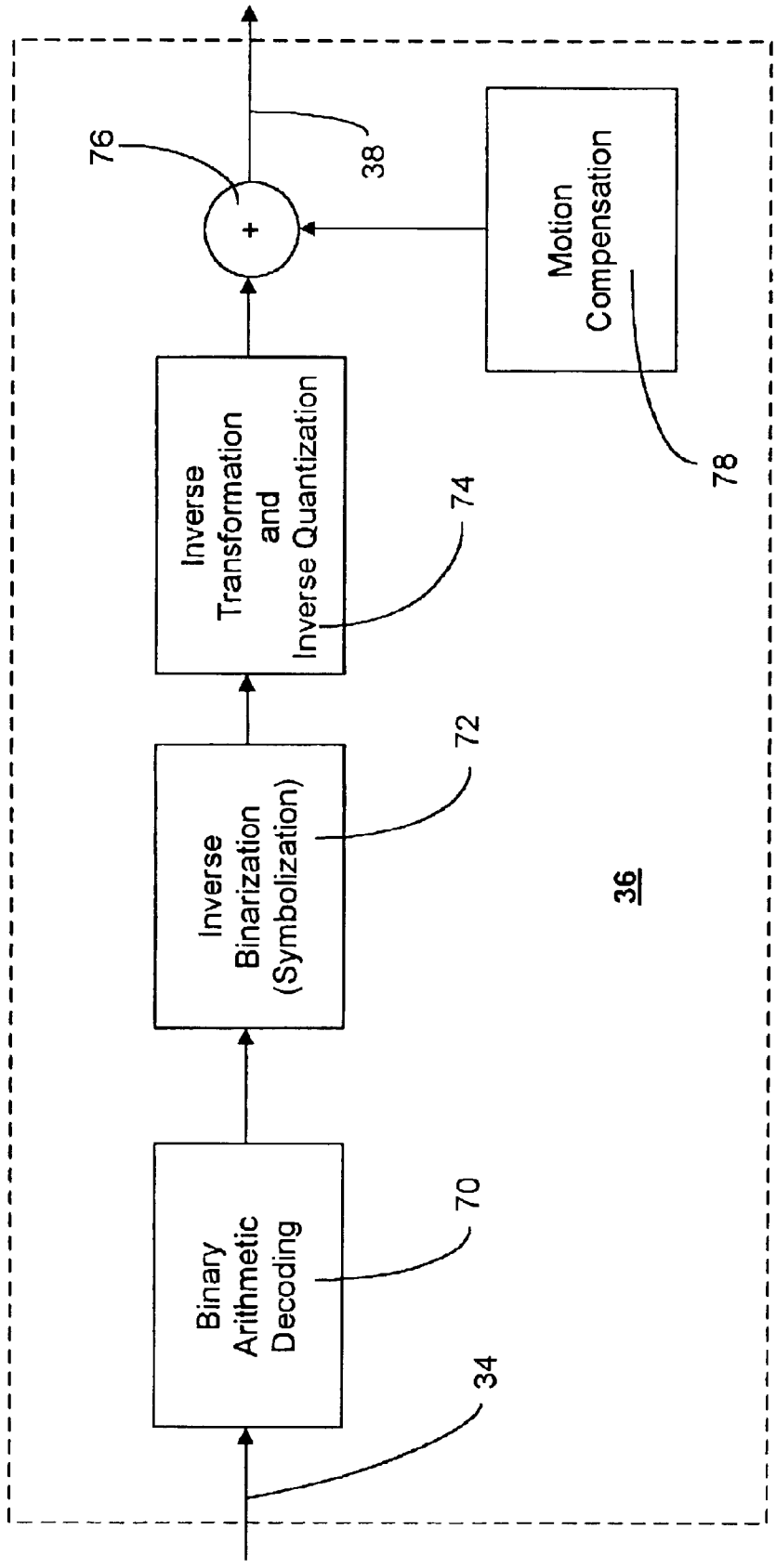
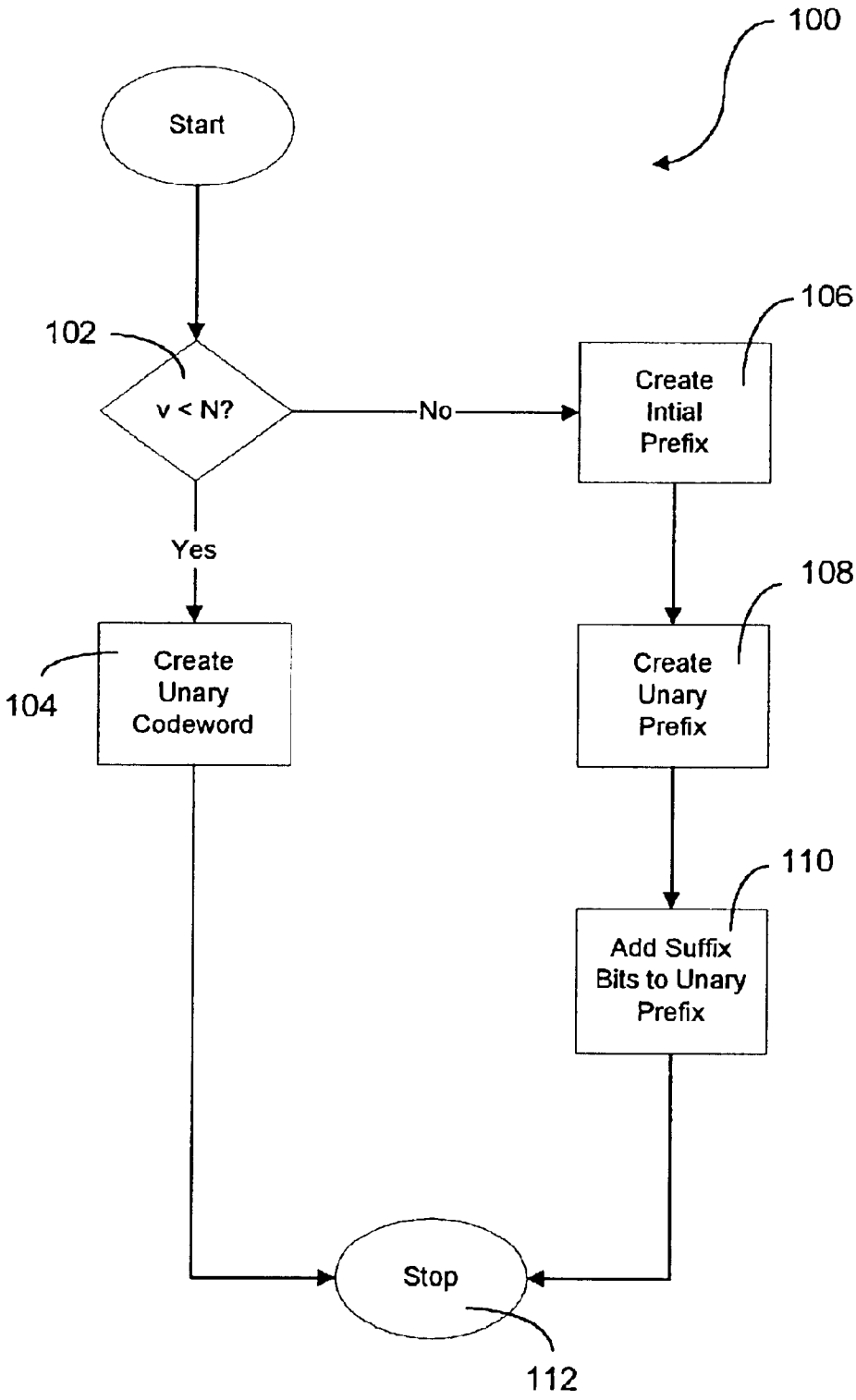

76
38
+
Motion Compensation
78
Inverse Transformation and Inverse Quantization
74
Inverse Binarization (Symbolization)
72
36
Binary Arithmetic Decoding
70
34

FIG. 3

FIG. 4


100
Start
102
v < N?
No
106
Create Intial Prefix
Yes
108
Create Unary Prefix
104
Create Unary Codeword
110
Add Suffix Bits to Unary Prefix
Stop
112

METHOD AND SYSTEM FOR SYMBOL BINARIZATION

FIELD OF THE INVENTION

The present invention relates generally to a system and method for the compression of digital signals. More specifically, the present invention relates to reducing the file size or the bit rate required by a system using binary arithmetic encoding to entropy encode a digital signal such as a digital image or digital video.

BACKGROUND OF THE INVENTION

Throughout this specification we will be using the term MPEG as a generic reference to a family of international standards set by the Motion Picture Expert Group. MPEG reports to sub-committee 29 (SC29) of the Joint Technical Committee (JTC1) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Throughout this specification the term H26x will be used as a generic reference to a closely related group of international recommendations by the Video Coding Experts Group (VCEG). VCEG addresses Question 6 (Q.6) of Study Group 16 (SG16) of the International Telecommunications Union Telecommunication Standardization Sector (ITU-T). These standards/recommendations specify exactly how to represent visual and audio information in a compressed digital format. They are used in a wide variety of applications, including DVD (Digital Video Discs), DVB (Digital Video Broadcasting), Digital cinema, and videoconferencing.

Throughout this specification the term MPEG/H.26x will refer to the superset of MPEG and H.26x standards and recommendations.

A feature of MPEG/H.26x is that these standards are often capable of representing a video signal with data roughly $1/50^{th}$ the size of the original uncompressed video, while still maintaining good visual quality. Although this compression ratio varies greatly depending on the nature of the detail and motion of the source video, it serves to illustrate that compressing digital images is an area of interest to those who provide digital transmission. MPEG/H.26x achieves high compression of video through the successive application of four basic mechanisms:

1) Storing the luminance (black & white) detail of the video signal with more horizontal and vertical resolution than the two chrominance (colour) components of the video.
2) Storing only the changes from one video frame to another, instead of the entire frame. Thus, often storing motion vector symbols indicating spatial correspondence between frames.
3) Storing these changes with reduced fidelity, as quantized transform coefficient symbols, to trade-off a reduced number of bits per symbol with increased video distortion.
4) Storing all the symbols representing the compressed video with entropy encoding, which exploits the statistics of the symbols, to reduce the number of bits per symbol without introducing any additional video signal distortion.

With regard to point 4), the symbols may be encoded as codewords in a variety of ways. One such encoding is binarization. Small codewords are well handled by unary or exp-Golomb binarizations while large codewords are best represented with the binarization limited to a reasonable length. Thus there is a need for a method and system binarization system that retains the most valuable properties of the unary and exp-Golomb binarizations. That is, small codewords should be distinguishable as with a unary binarization, while large codewords should have their binarization limited to a reasonable length. A binarization that simultaneously satisfies these two requirements will reduce the complexity and the bitrate/size for compressing and decompressing video, images, and signals that are compressed using binary arithmetic encoding for entropy encoding. The present invention addresses this need.

SUMMARY OF THE INVENTION

The present invention is directed to a method of binarization, the method comprising the step of determining if a code symbol index value is less than a threshold value, if so then constructing a codeword using a first binarization model; else constructing a codeword using a second binarization model.

The present invention is also directed to a binarization system, the system comprising means for determining if a code symbol index value is less than a threshold value, if so then utilizing means for constructing a codeword using a first binarization model; else utilizing means for constructing a codeword using a second binarization model.

The present invention is further directed to a computer readable medium containing instructions for binarization, comprising instructions for determining if a code symbol index value is less than a threshold value, if so then constructing a codeword using a first binarization model; else constructing a codeword using a second binarization model.

BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention, and to show more clearly how it may be carried into effect, reference will now be made, by way of example, to the accompanying drawings which aid in understanding an embodiment of the present invention and in which:

FIG. **1** is a block diagram of a video transmission and receiving system;

FIG. **2** is a block diagram of an encoder;

FIG. **3** is a block diagram of a decoder; and

FIG. **4** is a flowchart of a process for codeword construction.

DETAILED DESCRIPTION OF THE INVENTION

By way of introduction we refer first to FIG. **1**, a video transmission and receiving system, is shown generally as **10**. A content provider **12** provides a video source **14** to an encoder **16**. A content provider may be anyone of a number of sources but for the purpose of simplicity one may view video source **14** as originating from a television transmission, be it analog or digital. Encoder **16** receives video source **14** and utilizes a number of compression algorithms to reduce the size of video source **14** and passes an encoded stream **18** to encoder transport system **20**. Encoder transport system **20** receives stream **18** and restructures it into a transport stream **22** acceptable to transmitter **24**. Transmitter **24** then distributes transport stream **22** through a transport medium **26** such as the Internet or any form of network enabled for the transmission of MPEG data streams. Receiver **28** receives transport stream **22** and passes

it as received stream **30** to decoder transport system **32**. In a perfect world, steams **22** and **30** would be identical. Decoder transport system **32** processes stream **30** to create a decoded stream **34**. Once again, in a perfect world streams **18** and **34** would be identical. Decoder **36** then reverses the steps applied by encoder **16** to create output stream **38** that is delivered to the user **40**.

There are several existing major MPEG/H.26x standards: H.261, MPEG-1, MPEG-2/H.262, MPEG-4/H.263. Among these, MPEG-2/H.262 is clearly most commercially significant, being sufficient for all the major TV standards, including NTSC (National Standards Television Committee) and HDTV (High Definition Television). Of the series of MPEG standards that describe and define the syntax for video broadcasting, the standard of relevance to the present invention is the draft standard ITU-T Recommendation H.264, ISO/IEC 14496-10 AVC, which is incorporated herein by reference and is hereinafter referred to as "MPEG-AVC/H.264".

An MPEG video transmission is essentially a series of pictures taken at closely spaced time intervals. In the MPEG/H.26x standards a picture is referred to as a "frame", and a "frame" is completely divided into rectangular sub-partitions known as "picture blocks", with associated "motion vectors". Often a picture may be quite similar to the one that precedes it or the one that follows it. For example, a video of waves washing up on a beach would change little from picture to picture. Except for the motion of the waves, the beach and sky would be largely the same. Once the scene changes, however, some or all similarity may be lost. The concept of compressing the data in each picture relies upon the fact that many images often do not change significantly from picture to picture, and that if they do the changes are often simple, such as image pans or horizontal and vertical block translations. Thus, transmitting only block translations (known as "motion vectors") and differences between picture blocks, as opposed to the entire picture, can result in considerable savings in data transmission.

Usually motion vectors are predicted, such that they are represented as a difference from their predictor, known as a predicted motion vector residual. In practice, the pixel differences between picture blocks are transformed into frequency coefficients, and then quantized to further reduce the data transmission. Quantization allows the frequency coefficients to be represented using only a discrete number of levels, and is the mechanism by which the compressed video becomes a "lossy" representation of the original video. This process of transformation and quantization is performed by an encoder.

Referring now to FIG. **2** a block diagram of an encoder is shown generally as **16**. Encoder **16** accepts as input video source **14**. Video source **14** is passed to motion estimation module **50**, which determines the motion difference between frames. The output of motion estimate module **50** is passed to motion compensation module **52**. At combination module **54**, the output of motion compensation module **52** is subtracted from the input video source **14** to create input to transformation and quantization module **56**. Output from motion compensation module **52** is also provided to module **60**. Module **56** transforms and quantizes output from module **54**. The output of module **56** may have to be recalculated based upon prediction error, thus the loop comprising modules **52**, **54**, **56**, **58** and **60**. The output of module **56** becomes the input to inverse transformation module **58**. Module **58** applies an inverse transformation and an inverse quantization to the output of module **56** and provides that to module **60** where it is combined with the output of module **52** to provide feedback to module **52**.

Binarization module **62** is where the present invention resides. Module **62** accepts as input, symbols created by module **56** and creates a binary representation of each one in of the form of a codeword. The codewords are passed to binary arithmetic encoding module **64** where the frequency of each codeword is determined and the most frequently occurring codewords are assigned the lowest values. The output of module **64** is encoded stream **18**.

With regard to the above description of FIG. **2**, as those skilled in the art will appreciate that the functionality of the modules illustrated are well defined in the MPEG family of standards. Further, numerous variations of modules of FIG. **2** have been published and are readily available.

Referring now to FIG. **3** a block diagram of a decoder is shown. Decoder **36** accepts as input decoded stream **34** to binary arithmetic decoding module **70**. Module **70** decodes the binary arithmetic encoding performed by module **64** (see FIG. **2**) and passes the output to inverse binarization module **72**. Module **72** reverses the binarization of module **62** (see FIG. **2**) and passes its output to inverse transformation and inverse quantization module **74**, which reverses the effects of module **56** (see FIG. **2**). At combination module **76** the output from module **74** is combined with the output of motion compensation module **78** to create output stream **38**.

The MPEG/H.26x standards define precisely the syntax that is used for specifying how quantized coefficients, motion vectors, and other associated information such as block modes are to be represented, as well as the semantics for reconstructing video source **14** from the syntax of encoded stream **18**. In particular, codewords such as transformed-quantized picture differences and predicted motion vector residuals are entropy coded with such schemes as variable length codes (e.g. Huffman codes) or arithmetic encoding to become the syntax elements that form encoded bitstream **18**.

Several types of arithmetic codecs (encoder/decoder pairs) exist. One of the most efficient is the family of binary arithmetic coders (BACs). Well-known members of this family include, among others, the Q-coder, QM-coder, MQ-coder, and Qx-coder. A BAC accepts as input a binary representation of a codeword and by recursively examining the codewords it receives, is able to compress the codewords based upon the probability of their frequency.

Since BACs operate only on binary valued data, a signal compression standard such as MPEG-AVC/H.264 maps codewords such as transformed-quantized picture differences and motion vector residuals to binarized symbol representations prior to binary arithmetic encoding.

Among the commonly used binarization methods are the following: unary, binary, Golomb, and exp-Golomb.

Unary binarization consists of a number of binary 1's equal to an index for a symbol followed by a zero as shown in Table 1.

TABLE 1

Binarization by means of the unary code tree

Symbol Index	Codeword						
0	0						
1	1	0					
2	1	1	0				
3	1	1	1	0			
4	1	1	1	1	0		
5	1	1	1	1	1	0	
6	1	1	1	1	1	1	0

TABLE 1-continued

Binarization by means of the unary code tree

Symbol Index	Codeword							
. . .	.	.	.	.	.	.	.	.
bin_no.	1	2	3	4	5	6	7	. . .

The first column of Table 1 contains an index for a symbol. The corresponding row for each index contains a binarization of the symbol represented by the index into a codeword. Thus symbol index "0" results in a codeword of a single bit, namely "0". Symbol "1" results in a codeword of "10', which comprises two bits, and so on. The row labeled bin_no at the bottom of Table 1 contains the frequency of each bit for a codeword. For example bin_no "1" will contain the number of 0's and 1's that occur as the first bit of a codeword. Similarly bin_no "2" contains the number of 0's and 1's in the second bit of a codeword, and so on.

A BAC by examining each bin_no can determine the length and frequency of a codeword by determining if there is a zero in the bin. For example bin_no "5" will contain a zero for each codeword having a length of five, thus allowing the BAC to determine the number or frequency of five bit codewords.

The advantage of the unary binarization is that a bin containing a value 0 distinguishes a particular codeword from all codewords with a larger symbol index. Therefore, the BAC can be constructed to separately account for the statistical frequency of every individual codeword by maintaining separate statistics on the frequency of zeroes and ones for every bin. A disadvantage of unary binarization is that in practice the statistics of high bins are lumped together since the smaller, more frequent codewords account for the majority of the bitstream, and infrequently occurring codewords do not occur often enough to enable accurate gathering of statistical data. A major disadvantage of unary binarization is that the number of binary values that must run through the BAC will, in the worst case, be as many bins as the largest symbol index (which may range into the tens of thousands). For example, the encoding of a large symbol index may require tens of thousands of binary bins to be sent through the BAC.

Binary binarization creates a codeword as a fixed length binary representation of the symbol index. Thus symbol index "3" is encoded as "11" with the appropriate number of leading zeros applied. The disadvantage of binary binarization is that a single bin no longer distinguishes each codeword uniquely. This results in a greatly reduced compression ratio.

Golomb and exp-Golomb codewords, which use a unary prefix followed by a binary postfix, may be regarded as compromise positions between unrary and binary binarizations. Golomb codewords with parameter 'k' begin with unary binarizations as shown by the column labeled MSB (Most Significant Bits) in Table 2. Appended to the unary binerization are 'k' binary bits as is shown in the column labeled LSB (Least Significant Bits) in Table 2. This combination produces 2**k distinct binerizations for each MSB. The example illustrated in Table 2 shows an exp-Golob code with k=1, thus the first LSB contains 2**1 values. The next levels contains 2*2 LSB values and so on. Unfortunately, this still permits extremely long binarizations to occur for large symbol indices.

TABLE 2

Exp-Golomb codes.

Index	MSB	LSB
0	0	
1	10	0
2	10	1
3	110	00
4	110	01
5	110	10
6	110	11
7	1110	000
8	1110	001
9	1110	010
10	1110	011
11	1110	100
12	1110	101

The exp-Golomb code does greatly reduce the maximum possible number of bins in the binarization of symbol indices. However, it does not permit codewords with a small symbol index (other than index 0) to be uniquely distinguished from codewords with larger symbol indices. This results in reduced compression ratio for the binarizations, relative to the unary binarization of Table 1.

The present invention provides a binarization that retains the most valuable properties of the unary and exp-Golomb binarizations. That is, small codewords are distinguishable as with a unary binarization, while large codewords have their binarization limited to a reasonable length. By doing so, the present invention provides a binarization that reduces the complexity and the bitrate/size for compressing and decompressing video, images, and signals that are compressed using binary arithmetic encoding for entropy encoding.

The present invention allows for the maintenance of a true prefix code, while switching between a unary binarization for small codeword indices and a modified exp-Golomb binarization for larger codewords. The invention prepends a fixed prefix to an exp-Golomb code that begins at a fixed index value, prior to which unary binarization is used.

Tables 3 and Table 4 demonstrate particular instances of this new class of binary codes: hybrid unary-exp-Golomb codes. Table 3 illustrates a binerization that is particularly appropriate for the binarization of quarter or residual magnitudes of MPEG-AVC/H.264.

TABLE 3

Motion vector magnitude residual binarization.

Index	Unary Prefix	exp-Golomb Suffix
0	0	
1	10	
2	110	
. . .		
63	1 . . . 1 0	
64	1 . . . 1 10	0
65	1 . . . 1 10	1
66	1 . . . 1 110	00
67	1 . . . 1 110	01
68	1 . . . 1 110	10
69	1 . . . 1 110	11
70	1 . . . 1 1110	000
71	1 . . . 1 1110	001
72	1 . . . 1 1110	010
73	1 . . . 1 1110	011

TABLE 3-continued

Motion vector magnitude residual binarization.

Index	Unary Prefix	exp-Golomb Suffix
74	1 . . . 1 1110	100
75	1 . . . 1 1110	101
. . .		

Table 4 is a binarization that is particularly appropriate for the binerization of the quantized frequency-transform coefficient magnitudes (also often called coefficient levels) of MPEG-AVC/H.264.

TABLE 4

Coefficient level binarization.

Index	Unary Prefix	exp-Golomb Suffix
0	0	
1	10	
2	110	
. . .		
15	1 . . . 1 0	
16	1 . . . 1 10	0
17	1 . . . 1 10	1
18	1 . . . 1 110	00
19	1 . . . 1 110	01
20	1 . . . 1 110	10
21	1 . . . 1 110	11
22	1 . . . 1 1110	000
23	1 . . . 1 1110	001
24	1 . . . 1 1110	010
25	1 . . . 1 1110	011
26	1 . . . 1 1110	100
27	1 . . . 1 1110	101

When these binarizations illustrated in Tables 3 and 4, bitrate and complexity are both reduced for MPEG-AVC/H.264 relative to other known

A detailed description of the method for constructing such hybrid binerization follows. Let N be the threshold at which unary to exp-Golomb switching occurs (N=64 for Table 3, N=16 for Table 4). The construction of a codeword of this modified unary binarization table for a given index v is given by the following algorithm:

If v<N

1) use a unary code of v 1's terminated with a 0

If v>=N

1) Form an initial prefix of (N−1) 1's;

2) Determine the number of bits γ+1 required to represent v−(N−2). For example, for N=64, $\gamma=\lfloor \log_2(v-62) \rfloor$, and put it in a unary representation. The unary representation is appended to the initial prefix to form the unary prefix as shown in Tables 3 and 4.

3) Append the γ least significant bits of "g" where g=v−(N−2)−2**γ in its binary representation to the prefix.

4) The corresponding bits obtained at step 3) are shown in the exp-Golomb Suffix column of Tables 3 and 4.

Referring now to FIG. **4**, a flowchart of a process for codeword construction is shown generally as **100**. Process **100** illustrates the steps of the present invention. Process **100** begins at step **102** where a test is made to determine if the value of the code symbol index is less than the value of the threshold. If it is processing moves to step **104** were a unary codeword is constructed comprising a series of v 1's terminated with a 0. Processing then ends at step **112**. Returning to step **102** if the test is negative, processing moves to step **106**, where an initial prefix of N 1's is created. Processing then moves to step **108** where the most significant bits of the value v−(N−2) are extracted and converted to a unary representation. The unary representation is then appended to the initial prefix to create a unary prefix. Process **100** then moves to step **110** where the binary representation of the least significant bits of the value of v−(N−2) are appended to the unary prefix to create the codeword.

Although the description of the present invention describes a binarization scheme for MPEG-AVC/H.264, it is not the intent of the inventors to restrict this binarization scheme solely to the referenced proposed standard. As one skilled in the art can appreciate any system utilizing BAC may make use of the present invention improve binarization.

Although the present invention has been described as being implemented in software, one skilled in the art will recognize that it may be implemented in hardware as well. Further, it is the intent of the inventors to include computer readable forms of the invention. Computer readable forms meaning any stored format that may be read by a computing device.

Although the invention has been described with reference to certain specific embodiments, various modifications thereof will be apparent to those skilled in the art without departing from the spirit and scope of the invention as outlined in the claims appended hereto.

What is claimed is:

1. A method of binarization,

comprising the step of:

determining if a code symbol index value;

if said code symbol index value is less than a threshold value, constructing a codeword using a unary binarization; and

if said code symbol index value is not less than said threshold value, constructing a codeword using a exp-Golomb binarization.

2. The method of claim **1**, wherein said exp-Golomb binarization model comprises the steps of:

a) forming an initial prefix of 1's, equal in number to said threshold value minus one;

b) determining the number of bits, γ+1 required to represent v−(N−2) where $\gamma=\lfloor \log_2(v-(N-2)) \rfloor$, v is the code symbol index value, and N is the threshold value, and then transforming γ into a unary representation to create a result;

c) appending the result of step b) to the result of step a);

d) determining the least significant bits, of v−(N−2)−2**γ, and

e) appending the result of step d) in its binary representation to the result of step c) to create said codeword.

3. A binarization system comprising:

means for determining if a code symbol index value is less than a threshold value

means for constructing a codeword using a unary binarization if said code symbol index value is less than a threshold value; and

means for constructing a codeword using a exp-Golomb binarization if said code symbol index value is less than a threshold value.

4. The system of claim **3**, wherein said exp-Golomb binarization comprises means for:

a) forming an initial prefix of 1's, equal in number to said threshold value minus one;

b) determining the number of bits, γ+1 required to represent v−(N−2) where $\gamma=\lfloor \log_2(v-(N-2)) \rfloor$, v is the code symbol index value, and N is the threshold value, and then transforming y into a unary representation to create a result;

c) appending the result of step b) to the result of step a);

d) determining the least significant bits, of v−(N=2)−2**γ; and

e) appending the result of step d) in its binary representation to the result of step c) to create said codeword.

5. A computer readable medium containing instructions for binarization, comprising instructions for:

determining if a code symbol index value;

if said code symbol index value is less than a threshold value, constructing a codeword using a unary binarization; and

if said symbol index value is not less than said threshold value constructing a codeword using a exp-Golomnb binarization.

6. The computer readable medium of claim **5**, wherein said exp-Golomb binarization comprises instructions for:

a) forming an initial prefix of 1's, equal in number to said threshold value;

b) determining the number of bits, γ+1, required to represent v−(N=2) where $\gamma=\lfloor \log_2(v-(N-2)) \rfloor$, v is the code symbol index value, and N is the threshold value, and then transforming γ into a unary representation to create a result;

c) appending the result of step b) to the result of step a);

d) determining the least significant bits, of v−(N=2)−2**γ; and

e) appending the result of step d) in its binary representation to the result of step c) to create said codeword.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,744,387 B2
APPLICATION NO. : 10/191596
DATED : June 1, 2004
INVENTOR(S) : Lowell Winger

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

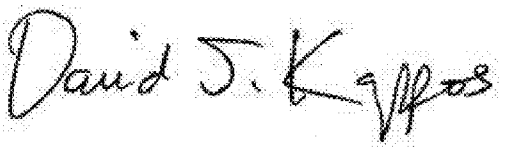
In column 8, line 32, replace “determining if a code symbol” with --determining a code symbol--.

In column 8, line 59, replace “a threshold value” with --said threshold value--.

In column 8, line 62, replace “a threshold value” with --said threshold value--.

In column 9, line 13, replace “determining if a code symbol” with --determining a code symbol--.

Signed and Sealed this
Eighth Day of May, 2012



David J. Kappos
Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,744,387 B2
APPLICATION NO. : 10/191596
DATED : June 1, 2004
INVENTOR(S) : Lowell Winger

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims

In column 8, line 62, replace “is less than” with --is not less than--.

Signed and Sealed this
Tenth Day of March, 2015

Michelle K. Lee

Michelle K. Lee
Deputy Director of the United States Patent and Trademark Office

Exhibit 3

U.S. Patent No. 6,982,663

US006982663B2

(12) **United States Patent**
Winger

(10) **Patent No.: US 6,982,663 B2**
(45) **Date of Patent: Jan. 3, 2006**

(54) **METHOD AND SYSTEM FOR SYMBOL BINARIZATION**

(75) Inventor: **Lowell Winger**, Waterloo (CA)

(73) Assignee: **LSI Logic Corporation**, Milpitas, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 7 days.

(21) Appl. No.: **10/770,213**

(22) Filed: **Feb. 2, 2004**

(65) **Prior Publication Data**

US 2004/0150540 A1 Aug. 5, 2004

Related U.S. Application Data

(63) Continuation of application No. 10/191,596, filed on Jul. 10, 2002, now Pat. No. 6,744,387.

(51) **Int. Cl.**
H03M 7/00 (2006.01)

(52) **U.S. Cl.** **341/107**

(58) **Field of Classification Search** 341/50, 341/51, 107, 106; 704/221, 222, 201, 204, 704/503, 500, 200, 211, 236
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,471,207 A 11/1995 Zandi et al. 341/107
6,236,960 B1 * 5/2001 Peng et al. 704/211
6,636,222 B1 * 10/2003 Valmiki et al. 345/505
6,744,387 B2 * 6/2004 Winger 341/50
6,850,568 B1 * 2/2005 Williams et al. 375/240.16
2004/0114683 A1 * 6/2004 Schwarz et al. 375/240.2

OTHER PUBLICATIONS

Article—pp. 308–311, Predictive Quantizing Systems by J.B. O'Neil, Jr. —Manuscript received Dec. 27, 1965.

Article—pp. 1098–1101, A Method for the Construction of Minimum–Redundancy Codes by David A. Huffman, no date given.

Article—pp. 689–721, A Compression Method for Clustered Bit–Vectors by Jukka Teuhola—Oct. 1978.

Article—pp. 399–401 of the publication entitled IEEE transactions on Information Theory, 1966, Sep.

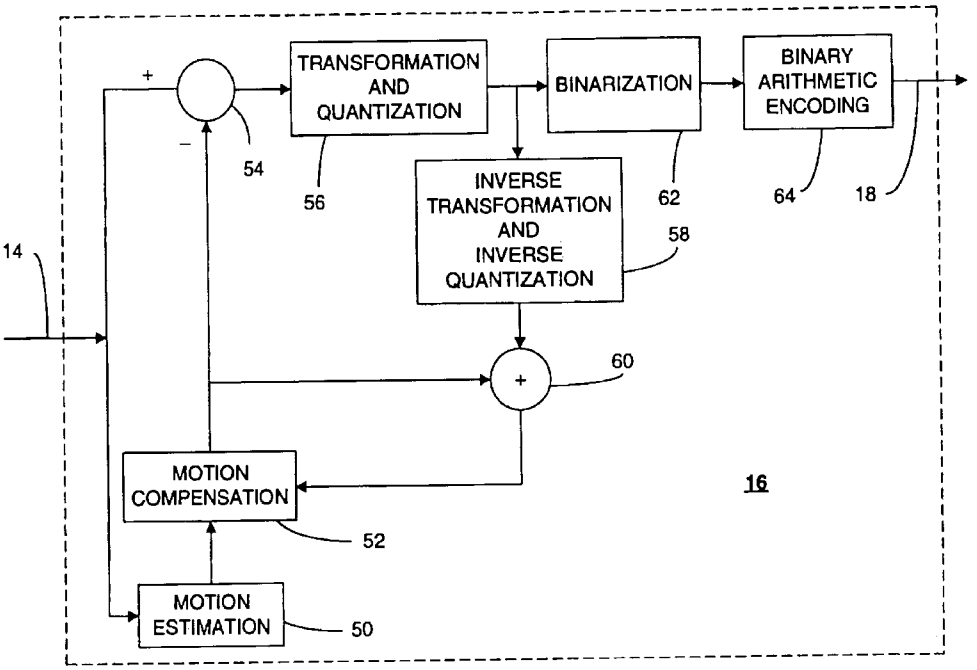
* cited by examiner

Primary Examiner—Brian Young
(74) *Attorney, Agent, or Firm*—Christopher P. Maiorana

(57) **ABSTRACT**

The present invention is directed to an improved method for the binarization of data in an MPEG data stream. The invention makes use of unary binarization to create codewords up until an index threshold. Once the threshold has been met, succeeding code symbols have appended to them an exp-Golomb suffix. This hybrid binarization scheme reduces the number of binary codewords to be processed by a Binary Arithmetic Coder (BAC), thus reducing the computation required by the BAC.

21 Claims, 6 Drawing Sheets



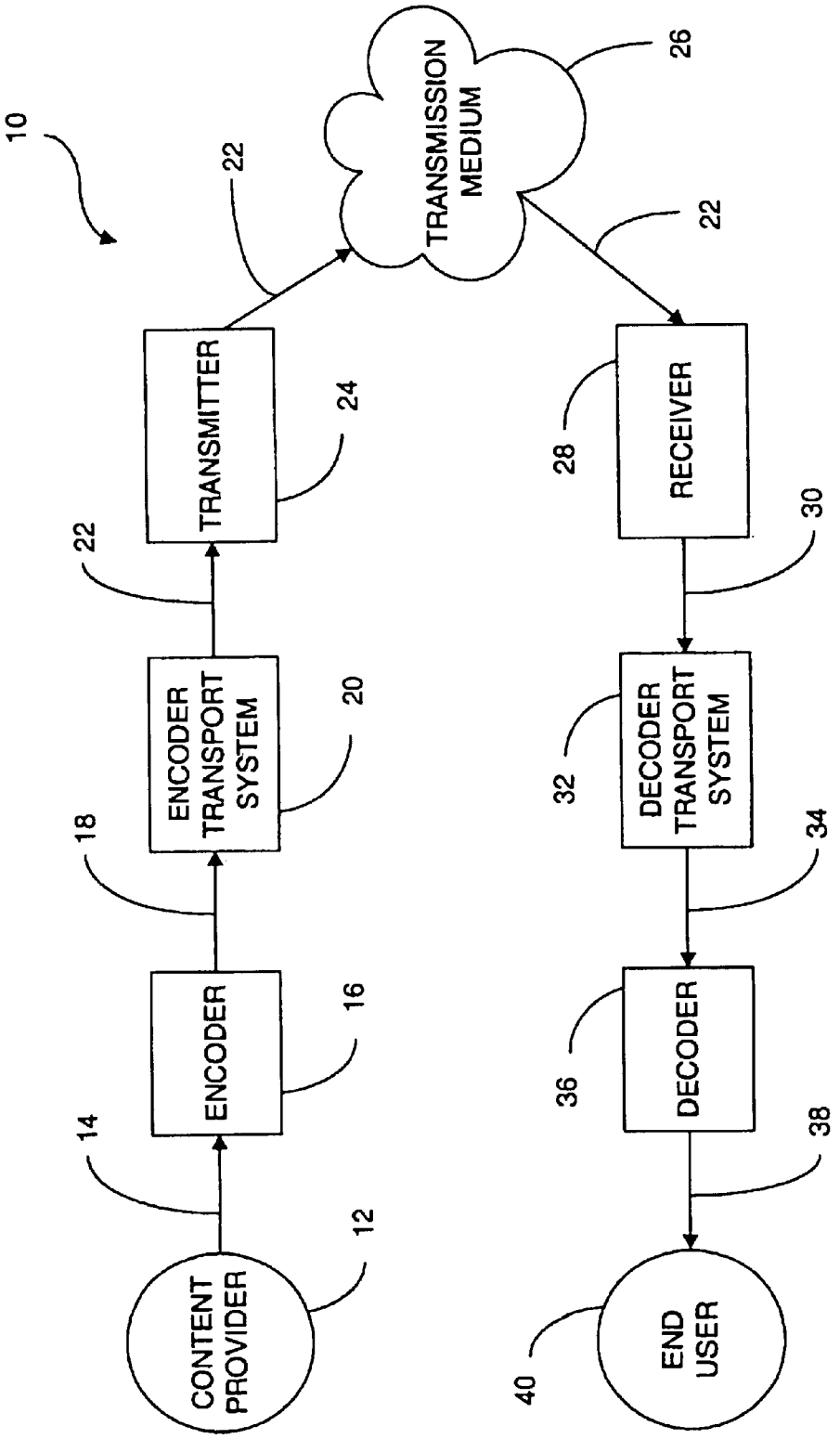

10
TRANSMISSION MEDIUM
26
22
22
TRANSMITTER
24
RECEIVER
28
22
30
ENCODER TRANSPORT SYSTEM
20
DECODER TRANSPORT SYSTEM
32
18
34
ENCODER
16
DECODER
36
14
38
CONTENT PROVIDER
12
END USER
40

FIG. 1

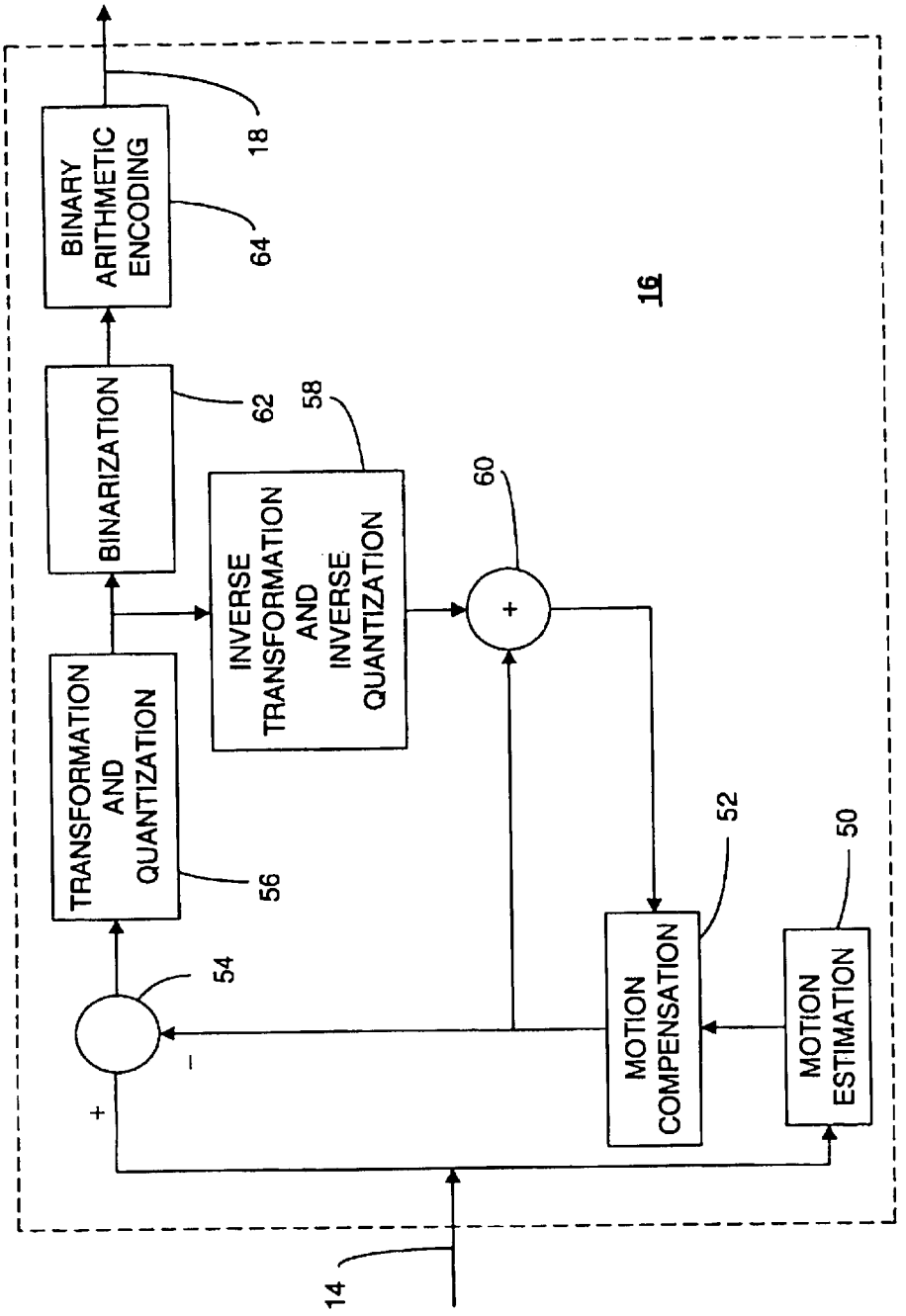

16
BINARY ARITHMETIC ENCODING
64
18
BINARIZATION
62
INVERSE TRANSFORMATION AND INVERSE QUANTIZATION
58
60
TRANSFORMATION AND QUANTIZATION
56
54
MOTION COMPENSATION
52
MOTION ESTIMATION
50
14

FIG. 2

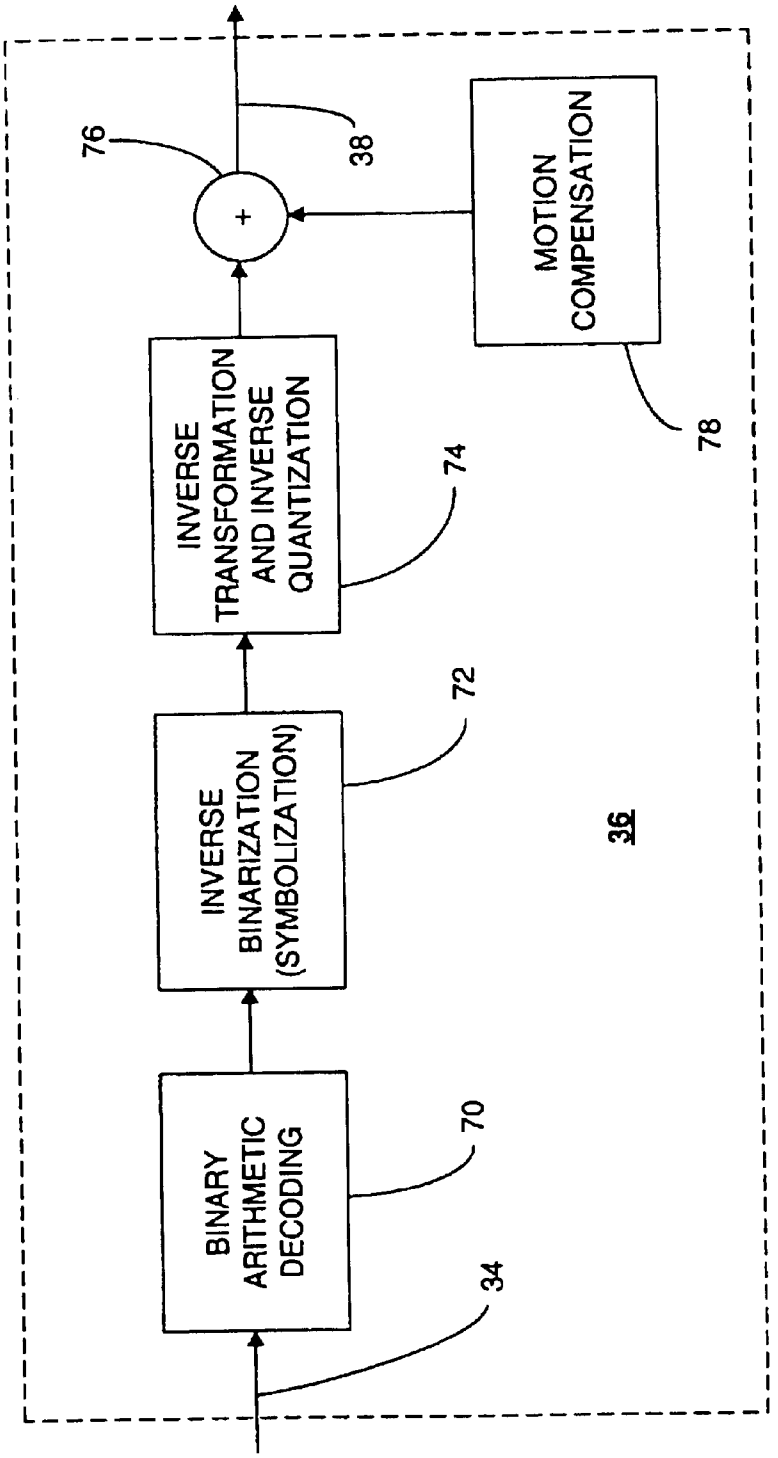

76
38
+
MOTION COMPENSATION
78
INVERSE TRANSFORMATION AND INVERSE QUANTIZATION
74
INVERSE BINARIZATION (SYMBOLIZATION)
72
36
BINARY ARITHMETIC DECODING
70
34

FIG. 3

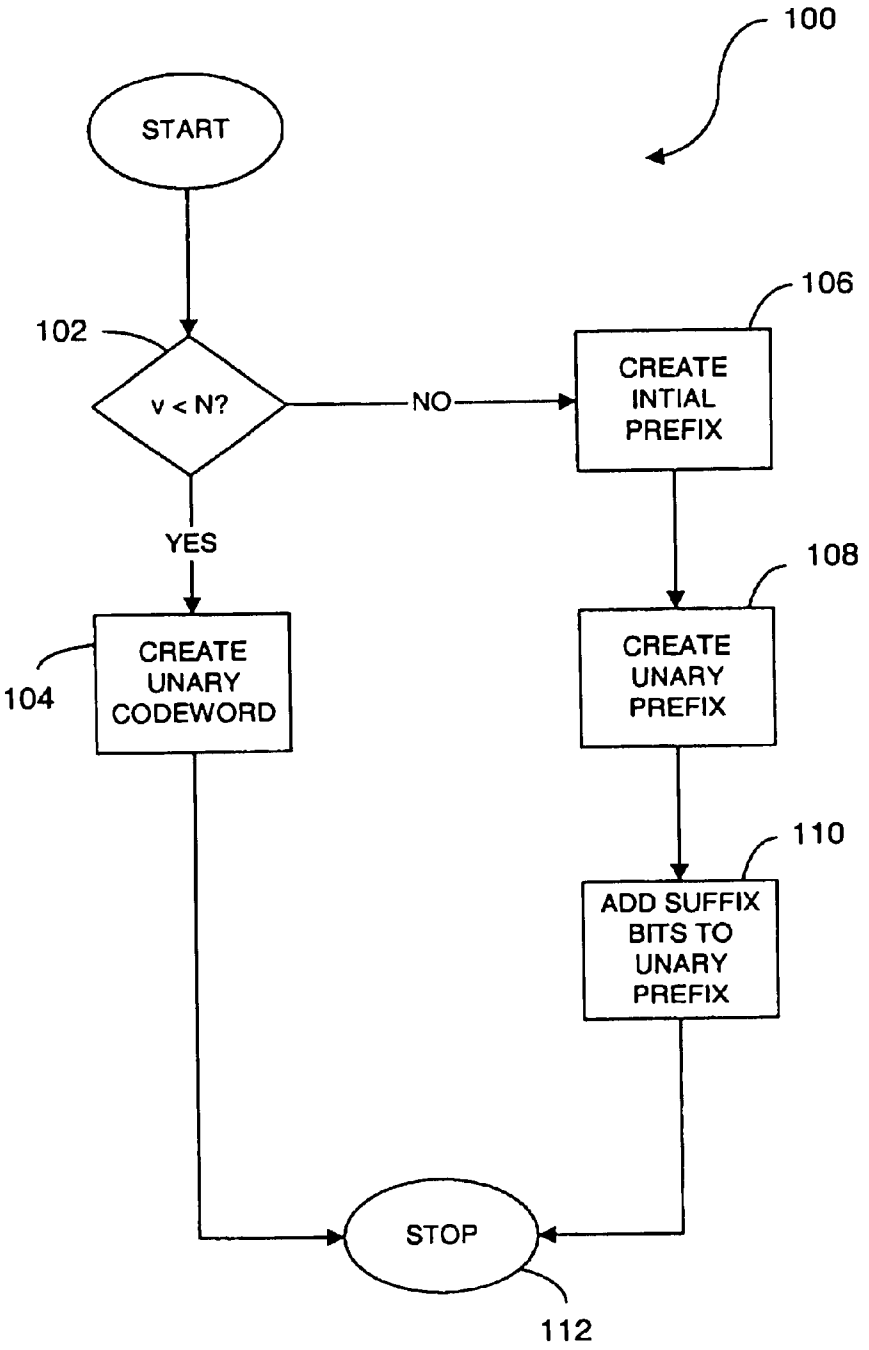

100
START
102
v < N?
NO
YES
106
CREATE INTIAL PREFIX
108
CREATE UNARY PREFIX
110
ADD SUFFIX BITS TO UNARY PREFIX
104
CREATE UNARY CODEWORD
STOP
112

FIG. 4

Table 3 - Motion vector magnitude residual binarization.

Index	Unary Prefix	exp-Golomb Suffix
0	0	
1	10	
2	110	
...		
63	1...1 0	
64	1...1 10	0
65	1...1 10	1
66	1...1 110	00
67	1...1 110	01
68	1...1 110	10
69	1...1 110	11
70	1...1 1110	000
71	1...1 1110	001
72	1...1 1110	010
73	1...1 1110	011
74	1...1 1110	100
75	1...1 1110	101
...		

FIG. 5

Table 4 - Coefficient level binarization.

Index	Unary Prefix	exp-Golomb Suffix
0	0	
1	10	
2	110	
...		
15	1...1 0	
16	1...1 10	0
17	1...1 10	1
18	1...1 110	00
19	1...1 110	01
20	1...1 110	10
21	1...1 110	11
22	1...1 1110	000
23	1...1 1110	001
24	1...1 1110	010
25	1...1 1110	011
26	1...1 1110	100
27	1...1 1110	101
...		

FIG. 6

METHOD AND SYSTEM FOR SYMBOL BINARIZATION

This is a continuation of U.S. Ser. No. 10/191,596, filed Jul. 10, 2002, now U.S. Pat. No. 6,744,387.

FIELD OF THE INVENTION

The present invention relates generally to a system and method for the compression of digital signals. More specifically, the present invention relates to reducing the file size or the bit rate required by a system using binary arithmetic encoding to entropy encode a digital signal such as a digital image or digital video.

BACKGROUND OF THE INVENTION

Throughout this specification we will be using the term MPEG as a generic reference to a family of international standards set by the Motion Picture Expert Group. MPEG reports to sub-committee 29 (SC29) of the Joint Technical Committee (JTC1) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Throughout this specification the term H.26x will be used as a generic reference to a closely related group of international recommendations by the Video Coding Experts Group (VCEG). VCEG addresses Question 6 (Q.6) of Study Group 16 (SG16) of the International Telecommunications Union Telecommunication Standardization Sector (ITU-T). These standards/recommendations specify exactly how to represent visual and audio information in a compressed digital format. They are used in a wide variety of applications, including DVD (Digital Video Discs), DVB (Digital Video Broadcasting), Digital cinema, and videoconferencing.

Throughout this specification the term MPEG/H.26x will refer to the superset of MPEG and H.26x standards and recommendations.

A feature of MPEG/H.26x is that these standards are often capable of representing a video signal with data roughly $1/50^{th}$ the size of the original uncompressed video, while still maintaining good visual quality. Although this compression ratio varies greatly depending on the nature of the detail and motion of the source video, it serves to illustrate that compressing digital images is an area of interest to those who provide digital transmission. MPEG/H.26x achieves high compression of video through the successive application of four basic mechanisms:

1) Storing the luminance (black & white) detail of the video signal with more horizontal and vertical resolution than the two chrominance (colour) components of the video.
2) Storing only the changes from one video frame to another, instead of the entire frame. Thus, often storing motion vector symbols indicating spatial correspondence between frames.
3) Storing these changes with reduced fidelity, as quantized transform coefficient symbols, to trade-off a reduced number of bits per symbol with increased video distortion.
4) Storing all the symbols representing the compressed video with entropy encoding, which exploits the statistics of the symbols, to reduce the number of bits per symbol without introducing any additional video signal distortion.

With regard to point 4), the symbols may be encoded as codewords in a variety of ways. One such encoding is binarization. Small codewords are well handled by unary or exp-Golomb binarizations while large codewords are best represented with the binarization limited to a reasonable length. Thus there is a need for a method and system binarization system that retains the most valuable properties of the unary and exp-Golomb binarizations. That is, small codewords should be distinguishable as with a unary binarization, while large codewords should have their binarization limited to a reasonable length. A binarization that simultaneously satisfies these two requirements will reduce the complexity and the bitrate/size for compressing and decompressing video, images, and signals that are compressed using binary arithmetic encoding for entropy encoding. The present invention addresses this need.

SUMMARY OF THE INVENTION

The present invention is directed to a method of binarization, the method comprising the step of determining if a code symbol index value is less than a threshold value, if so then constructing a codeword using a first binarization model; else constructing a codeword using a second binarization model.

The present invention is also directed to a binarization system, the system comprising means for determining if a code symbol index value is less than a threshold value, if so then utilizing means for constructing a codeword using a first binarization model; else utilizing means for constructing a codeword using a second binarization model.

The present invention is further directed to a computer readable medium containing instructions for binarization, comprising instructions for determining if a code symbol index value is less than a threshold value, if so then constructing a codeword using a first binarization model; else constructing a codeword using a second binarization model.

BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention, and to show more clearly how it may be carried into effect, reference will now be made, by way of example, to the accompanying drawings which aid in understanding an embodiment of the present invention and in which:

FIG. **1** is a block diagram of a video transmission and receiving system;

FIG. **2** is a block diagram of an encoder;

FIG. **3** is a block diagram of a decoder;

FIG. **4** is a flowchart of a process for codeword construction;

FIG. **5** is a table for motion vector magnitude residual binarization; and

FIG. **6** is a table for coefficient level binarization.

DETAILED DESCRIPTION OF THE INVENTION

By way of introduction we refer first to FIG. **1**, a video transmission and receiving system, is shown generally as **10**. A content provider **12** provides a video source **14** to an encoder **16**. A content provider may be anyone of a number of sources but for the purpose of simplicity one may view video source **14** as originating from a television transmission, be it analog or digital. Encoder **16** receives video source **14** and utilizes a number of compression algorithms to reduce the size of video source **14** and passes an encoded stream **18** to encoder transport system **20**. Encoder transport system **20** receives stream **18** and restructures it into a transport stream **22** acceptable to transmitter **24**. Transmitter **24** then distributes transport stream **22**

through a transport medium **26** such as the Internet or any form of network enabled for the transmission of MPEG data streams. Receiver **28** receives transport stream **22** and passes it as received stream **30** to decoder transport system **32**. In a perfect world, steams **22** and **30** would be identical. Decoder transport system **32** processes stream **30** to create a decoded stream **34**. Once again, in a perfect world streams **18** and **34** would be identical. Decoder **36** then reverses the steps applied by encoder **16** to create output stream **38** that is delivered to the user **40**.

There are several existing major MPEG/H.26x standards: H.261, MPEG-1, MPEG-2/H.262, MPEG-4/H.263. Among these, MPEG-2/H.262 is clearly most commercially significant, being sufficient for all the major TV standards, including NTSC (National Standards Television Committee) and HDTV (High Definition Television). Of the series of MPEG standards that describe and define the syntax for video broadcasting, the standard of relevance to the present invention is the draft standard ITU-T Recommendation H.264, ISO/IEC 14496-10 AVC, which is incorporated herein by reference and is hereinafter referred to as "MPEG-AVC/H.264".

An MPEG video transmission is essentially a series of pictures taken at closely spaced time intervals. In the MPEG/H.26x standards a picture is referred to as a "frame", and a "frame" is completely divided into rectangular sub-partitions known as "picture blocks", with associated "motion vectors". Often a picture may be quite similar to the one that precedes it or the one that follows it. For example, a video of waves washing up on a beach would change little from picture to picture. Except for the motion of the waves, the beach and sky would be largely the same. Once the scene changes, however, some or all similarity may be lost. The concept of compressing the data in each picture relies upon the fact that many images often do not change significantly from picture to picture, and that if they do the changes are often simple, such as image pans or horizontal and vertical block translations. Thus, transmitting only block translations (known as "motion vectors") and differences between picture blocks, as opposed to the entire picture, can result in considerable savings in data transmission.

Usually motion vectors are predicted, such that they are represented as a difference from their predictor, known as a predicted motion vector residual. In practice, the pixel differences between picture blocks are transformed into frequency coefficients, and then quantized to further reduce the data transmission. Quantization allows the frequency coefficients to be represented using only a discrete number of levels, and is the mechanism by which the compressed video becomes a "lossy" representation of the original video. This process of transformation and quantization is performed by an encoder.

Referring now to FIG. **2** a block diagram of an encoder is shown generally as **16**. Encoder **16** accepts as input video source **14**. Video source **14** is passed to motion estimation module **50**, which determines the motion difference between frames. The output of motion estimate module **50** is passed to motion compensation module **52**. At combination module **54**, the output of motion compensation module **52** is subtracted from the input video source **14** to create input to transformation and quantization module **56**. Output from motion compensation module **52** is also provided to module **60**. Module **56** transforms and quantizes output from module **54**. The output of module **56** may have to be recalculated based upon prediction error, thus the loop comprising modules **52**, **54**, **56**, **58** and **60**. The output of module **56** becomes the input to inverse transformation module **58**. Module **58** applies an inverse transformation and an inverse quantization to the output of module **56** and provides that to module **60** where it is combined with the output of module **52** to provide feedback to module **52**.

Binarization module **62** is where the present invention resides. Module **62** accepts as input, symbols created by module **56** and creates a binary representation of each one in of the form of a codeword. The codewords are passed to binary arithmetic encoding module **64** where the frequency of each codeword is determined and the most frequently occurring codewords are assigned the lowest values. The output of module **64** is encoded stream **18**.

With regard to the above description of FIG. **2**, as those skilled in the art will appreciate that the functionality of the modules illustrated are well defined in the MPEG family of standards. Further, numerous variations of modules of FIG. **2** have been published and are readily available.

Referring now to FIG. **3** a block diagram of a decoder is shown. Decoder **36** accepts as input decoded stream **34** to binary arithmetic decoding module **70**. Module **70** decodes the binary arithmetic encoding performed by module **64** (see FIG. **2**) and passes the output to inverse binarization module **72**. Module **72** reverses the binarization of module **62** (see FIG. **2**) and passes its output to inverse transformation and inverse quantization module **74**, which reverses the effects of module **56** (see FIG. **2**). At combination module **76** the output from module **74** is combined with the output of motion compensation module **78** to create output stream **38**.

The MPEG/H.26x standards define precisely the syntax that is used for specifying how quantized coefficients, motion vectors, and other associated information such as block modes are to be represented, as well as the semantics for reconstructing video source **14** from the syntax of encoded stream **18**. In particular, codewords such as transformed-quantized picture differences and predicted motion vector residuals are entropy coded with such schemes as variable length codes (e.g. Huffman codes) or arithmetic encoding to become the syntax elements that form encoded bitstream **18**.

Several types of arithmetic codecs (encoder/decoder pairs) exist. One of the most efficient is the family of binary arithmetic coders (BACs). Well-known members of this family include, among others, the Q-coder, QM-coder, MQ-coder, and Qx-coder. A BAC accepts as input a binary representation of a codeword and by recursively examining the codewords it receives, is able to compress the codewords based upon the probability of their frequency.

Since BACs operate only on binary valued data, a signal compression standard such as MPEG-AVC/H.264 maps codewords such as transformed-quantized picture differences and motion vector residuals to binarized symbol representations prior to binary arithmetic encoding.

Among the commonly used binarization methods are the following: unary, binary, Golomb, and exp-Golomb.

Unary binarization consists of a number of binary 1's equal to an index for a symbol followed by a zero as shown in Table 1.

TABLE 1

Binarization by means of the unary code tree

Symbol Index	Codeword							
0	0							
1	1	0						
2	1	1	0					
3	1	1	1	0				
4	1	1	1	1	0			
5	1	1	1	1	1	0		
6	1	1	1	1	1	1	0	
. . .	.	.	.	.	.	.	.	.
bin_no.	1	2	3	4	5	6	7	. . .

The first column of Table 1 contains an index for a symbol. The corresponding row for each index contains a

binarization of the symbol represented by the index into a codeword. Thus symbol index "0" results in a codeword of a single bit, namely "0". Symbol "1" results in a codeword of "10", which comprises two bits, and so on. The row labeled bin_no at the bottom of Table 1 contains the frequency of each bit for a codeword. For example bin_no "1" will contain the number of 0's and 1's that occur as the first bit of a codeword. Similarly bin_no "2" contains the number of 0's and 1's in the second bit of a codeword, and so on.

A BAC by examining each bin_no can determine the length and frequency of a codeword by determining if there is a zero in the bin. For example bin_no "5" will contain a zero for each codeword having a length of five, thus allowing the BAC to determine the number or frequency of five bit codewords.

The advantage of the unary binarization is that a bin containing a value 0 distinguishes a particular codeword from all codewords with a larger symbol index. Therefore, the BAC can be constructed to separately account for the statistical frequency of every individual codeword by maintaining separate statistics on the frequency of zeroes and ones for every bin. A disadvantage of unary binarization is that in practice the statistics of high bins are lumped together since the smaller, more frequent codewords account for the majority of the bitstream, and infrequently occurring codewords do not occur often enough to enable accurate gathering of statistical data. A major disadvantage of unary binarization is that the number of binary values that must run through the BAC will, in the worst case, be as many bins as the largest symbol index (which may range into the tens of thousands). For example, the encoding of a large symbol index may require tens of thousands of binary bins to be sent through the BAC.

Binary binarization creates a codeword as a fixed length binary representation of the symbol index. Thus symbol index "3" is encoded as "11", with the appropriate number of leading zeros applied. The disadvantage of binary binarization is that a single bin no longer distinguishes each codeword uniquely. This results in a greatly reduced compression ratio.

Golomb and exp-Golomb codewords, which use a unary prefix followed by a binary postfix, may be regarded as compromise positions between unary and binary binarizations. Golomb codewords with parameter 'k' begin with unary binarizations as shown by the column labeled MSB (Most Significant Bits) in Table 2. Appended to the unary binarization are 'k' binary bits as is shown in the column labeled LSB (Least Significant Bits) in Table 2. This combination produces 2**k distinct binarizations for each MSB. The example illustrated in Table 2 shows an exp-Golomb code with k=1, thus the first LSB contains 2**1 values. The next level contains 2**2 LSB values and so on. Unfortunately, this still permits extremely long binarizations to occur for large symbol indices.

TABLE 2

Exp-Golomb codes.

Index	MSB	LSB
0	0	
1	10	0
2	10	1
3	110	00
4	110	01
5	110	10
6	110	11
7	1110	000

TABLE 2-continued

Exp-Golomb codes.

Index	MSB	LSB
8	1110	001
9	1110	010
10	1110	011
11	1110	100
12	1110	101

The exp-Golomb code does greatly reduce the maximum possible number of bins in the binarization of symbol indices. However, it does not permit codewords with a small symbol index (other than index 0) to be uniquely distinguished from codewords with larger symbol indices. This results in reduced compression ratio for the binarizations, relative to the unary binarization of Table 1.

The present invention provides a binarization that retains the most valuable properties of the unary and exp-Golomb binarizations. That is, small codewords are distinguishable as with a unary binarization, while large codewords have their binarization limited to a reasonable length. By doing so, the present invention provides a binarization that reduces the complexity and the bitrate/size for compressing and decompressing video, images, and signals that are compressed using binary arithmetic encoding for entropy encoding.

The present invention allows for the maintenance of a true prefix code, while switching between a unary binarization for small codeword indices and a modified exp-Golomb binarization for larger codewords. The invention prepends a fixed prefix to an exp-Golomb code that begins at a fixed index value, prior to which unary binarization is used.

Referring to FIG. **5** and **6**, Tables 3 and Table 4 demonstrate particular instances of this new class of binary codes: hybrid unary-exp-Golomb codes. Table 3 illustrates a binarization that is particularly appropriate for the binarization of quarter pixel motion vector residual magnitudes of MPEG-AVC/H.264.

With these binarizations illustrated in Tables 3 and 4, bitrate and complexity are both reduced for MPEG-AVC/H.264 relative to other known binarizations.

A detailed description of the method for constructing such hybrid binarizations follows. Let N be the threshold at which unary to exp-Golomb switching occurs (N=64 for Table 3, N=16 for Table 4). The construction of a codeword of this modified unary binarization table for a given index v is given by the following algorithm:

If v<N

1) use a unary code of v 1's terminated with a 0

If v>=N

1) Form an initial prefix of (N−1) 1's;

2) Determine the number of bits γ+1 required to represent v−(N−2). For example, for N=64, $\gamma=\lfloor \log_2(v-62) \rfloor$, and put it in a unary representation. The unary representation is appended to the initial prefix to form the unary prefix as shown in Tables 3 and 4.

3) Append the γ least significant bits of "g" where g=v−(N−2)−2**γ in its binary representation to the prefix.

4) The corresponding bits obtained at step 3) are shown in the exp-Golomb Suffix column of Tables 3 and 4.

Referring now to FIG. **4**, a flowchart of a process for codeword construction is shown generally as **100**. Process **100** illustrates the steps of the present invention. Process **100** begins at step **102** where a test is made to determine if the

value of the code symbol index is less than the value of the threshold. If it is processing moves to step **104** were a unary codeword is constructed comprising a series of v 1's terminated with a 0. Processing then ends at step **112**. Returning to step **102** if the test is negative, processing moves to step **106**, where an initial prefix of N 1's is created. Processing then moves to step **108** where the most significant bits of the value v−(N−2) are extracted and converted to a unary representation. The unary representation is then appended to the initial prefix to create a unary prefix. Process **100** then moves to step **110** where the binary representation of the least significant bits of the value of v−(N−2) are appended to the unary prefix to create the codeword.

Although the description of the present invention describes a binarization scheme for MPEG-AVC/H.264, it is not the intent of the inventors to restrict this binarization scheme solely to the referenced proposed standard. As one skilled in the art can appreciate any system utilizing BAC may make use of the present invention improve binarization.

Although the present invention has been described as being implemented in software, one skilled in the art will recognize that it may be implemented in hardware as well. Further, it is the intent of the inventors to include computer readable forms of the invention. Computer readable forms meaning any stored format that may be read by a computing device.

Although the invention has been described with reference to certain specific embodiments, various modifications thereof will be apparent to those skilled in the art without departing from the spirit and scope of the invention as outlined in the claims appended hereto.

What is claimed is:

1. A method for generating an index value from a codeword for digital video decoding, comprising the steps of:

(A) setting said index value to a threshold in response to a first portion of said codeword having a first pattern;

(B) adding an offset to said index value based on a second pattern in a second portion of said codeword following said first portion in response to said first portion having said first pattern; and

(C) adding a value to said index value based on a third pattern in a third portion of said codeword following said second portion in response to said first portion having said first pattern.

2. The method according to claim **1**, further comprising the step of:

generating said index value based on a fourth pattern in said first portion in response to said fourth pattern being other than said first pattern.

3. The method according to claim **2**, wherein said first pattern is a predetermined pattern unique from all possible representations of said fourth pattern.

4. The method according to claim **2**, wherein said fourth pattern comprises (i) between zero and a plurality of first bits having a first state and (ii) a second bit having a second state opposite said first state.

5. The method according to claim **4**, wherein said second bit follows said first bits.

6. The method according to claim **1**, wherein said first pattern comprises a plurality of bits each having a first state.

7. The method according to claim **1**, wherein said second pattern comprises between zero and a plurality of first bits having a first state and (ii) a second bit having a second state opposite said first state.

8. The method according to claim **1**, wherein said third pattern comprises a binary number.

9. The method according to claim **1**, wherein said codeword in compatible with at least one of an International Organization for Standardization/International Electrotechnical Commission 14496-10 standard and an International Telecommunication Union-Telecommunications Standardization Sector Recommendation H.264.

10. The method according to claim **1**, further comprising the step of:

generating said index value based on a fourth pattern in said first portion in response to said fourth pattern being other than said first pattern, wherein (i) said first pattern comprises a plurality of bits each having a first state, (ii) said first pattern is unique from all possible representations of said fourth pattern, (iii) each of said representations of said fourth pattern ends in a bit having a second state opposite said first state and (iv) said second pattern has a same number of bits as said third pattern.

11. A system comprising:

a decoder configured to generate a codeword; and

a circuit configured to (i) set an index value to a threshold in response to a first portion of said codeword having a first pattern, (ii) add an offset to said index value based on a second pattern in a second portion of said codeword following said first portion in response to said first portion having said first pattern and (iii) add a value to said index value based on a third pattern in a third portion of said codeword following said second portion in response to said first portion having said first pattern.

12. A method for generating a codeword from an index value for digital video encoding, comprising the steps of:

(A) generating a first pattern in a first portion of said codeword in response to said index value being at least as great as a threshold;

(B) generating a second pattern in a second portion of said codeword following said first portion representing an offset of said index value above said threshold; and

(C) generating a third pattern in a third portion of said codeword following said second portion representing a value of said index value above said offset.

13. The method according to claim **12**, further comprising the step of:

generating a fourth pattern in said first portion based on said index value in response to said index value being below said threshold.

14. The method according to claim **13**, wherein said first pattern is a predetermined pattern unique from all possible representations of said fourth pattern.

15. The method according to claim **13**, wherein said fourth pattern comprises (i) between zero and a plurality of first bits having a first state and (ii) a second bit having a second state opposite said first state.

16. The method according to claim **12**, wherein (i) said offset has a first representation and (ii) said value has a second representation different than said first representation.

17. The method according to claim **12**, wherein said second pattern has a same number of bits as said third pattern.

18. The method according to claim **12**, wherein said second portion is void in response to said index value being below said threshold.

19. The method according to claim **12**, wherein said third portion is void in response to said index value being below said threshold.

20. The method according to claim **12**, wherein said codeword in compatible with at least one of an International Organization for Standardization/International Electrotechnical Commission 14496-10 standard and an International Telecommunication Union-Telecommunications Standardization Sector Recommendation H.264.

21. A system comprising:

a circuit configured to (i) generate a first pattern in a first portion of a codeword in response to an index value

being at least as great as a threshold, (ii) generate a second pattern in a second portion of said codeword following said first portion representing an offset of said index value above said threshold and (iii) generating a third pattern in a third portion of said codeword following said second portion representing a value of said index value above said offset; and

an encoder configured to encode said codeword.

* * * * *

Exhibit 4

U.S. Patent No. 9,332,283

US009332283B2

(12) **United States Patent**
Chen et al.

(10) **Patent No.: US 9,332,283 B2**
(45) **Date of Patent: May 3, 2016**

(54) **SIGNALING OF PREDICTION SIZE UNIT IN ACCORDANCE WITH VIDEO CODING**

(75) Inventors: **Peisong Chen**, San Diego, CA (US); **Brian Heng**, Irvine, CA (US); **Wade K. Wan**, Orange, CA (US)

(73) Assignee: **BROADCOM CORPORATION**, Irvine, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 860 days.

(21) Appl. No.: **13/523,822**

(22) Filed: **Jun. 14, 2012**

(65) **Prior Publication Data**

US 2013/0077684 A1 Mar. 28, 2013

Related U.S. Application Data

(60) Provisional application No. 61/539,948, filed on Sep. 27, 2011.

(51) **Int. Cl.**

H04N 7/12	(2006.01)
H04N 11/02	(2006.01)
H04N 11/04	(2006.01)
H04N 19/96	(2014.01)
H04N 19/503	(2014.01)

(Continued)

(52) **U.S. Cl.**

CPC ***H04N 19/96*** (2014.11); ***H04N 19/503*** (2014.11); ***H04N 19/70*** (2014.11); *H04N 19/119* (2014.11); *H04N 19/174* (2014.11)

(58) **Field of Classification Search**

CPC . H04N 19/119; H04N 19/115; H04N 19/169; H04N 19/174; H04N 19/177; H04N 19/96

USPC 375/240.01–240.29

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0213469 A1* 10/2004 Apostolopoulos et al. ... 382/239
2005/0038837 A1 2/2005 Marpe et al.

(Continued)

OTHER PUBLICATIONS

Bross, et al.; WD4: Working Draft 4 of High-Efficiency Video Coding; Joint Colaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11; 6th Meeting: Torino, IT; Jul. 14-22, 2011; 216 pgs.

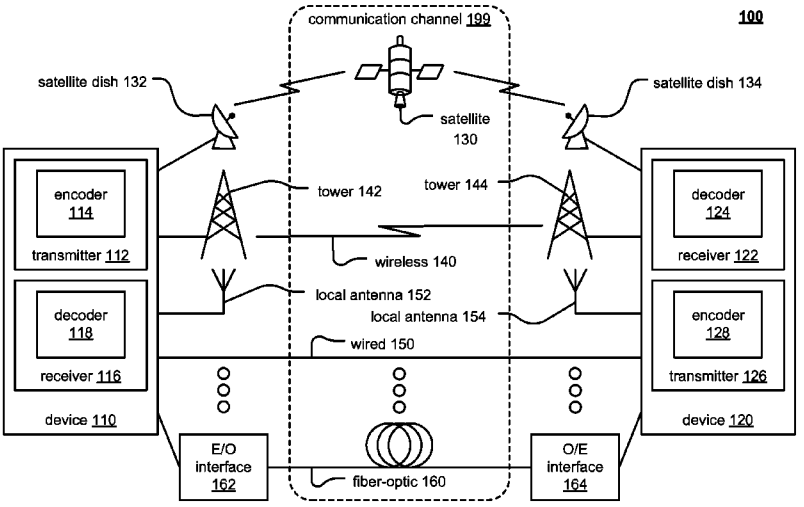
(Continued)

Primary Examiner — Andy Rao
Assistant Examiner — Jared Walker
(74) *Attorney, Agent, or Firm* — Garlick & Markison; Shayne X. Short

(57) **ABSTRACT**

Signaling of prediction size unit in accordance with video coding. In accordance with video coding, various binarization may be performed. In accordance with coding related to different types of slices (e.g., I, P, B slices), one or more binary trees may be employed for performing various respective operations (e.g., coding unit (CU) prediction and prediction unit (PU) partition mode operations). In one implementation, a common or singular binary tree is employed to encode jointly CU prediction and PU partition mode in a single syntax element for both P slices and B slices. That is to say, in such an implementation, instead of employing different respective binary trees for at least these different respective processes/operations, a common or single binary tree may be employed for them both. Appropriate coordination between and encoder/transmitter device and a decoder/receiver device may be performed to ensure appropriate handling of different respective phases of video coding.

20 Claims, 17 Drawing Sheets



(51) **Int. Cl.**

H04N 19/70	(2014.01)
H04N 19/174	(2014.01)
H04N 19/119	(2014.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

2010/0086032	A1*	4/2010	Chen et al.	375/240.12
2010/0098155	A1	4/2010	Demircin et al.	
2011/0206123	A1*	8/2011	Panchal et al.	375/240.15
2011/0228858	A1	9/2011	Budagavi et al.	

OTHER PUBLICATIONS

European Patent Office; European Search Report; EP App No. 12005568.6; Dec. 17, 2012; 5 pgs.

* cited by examiner

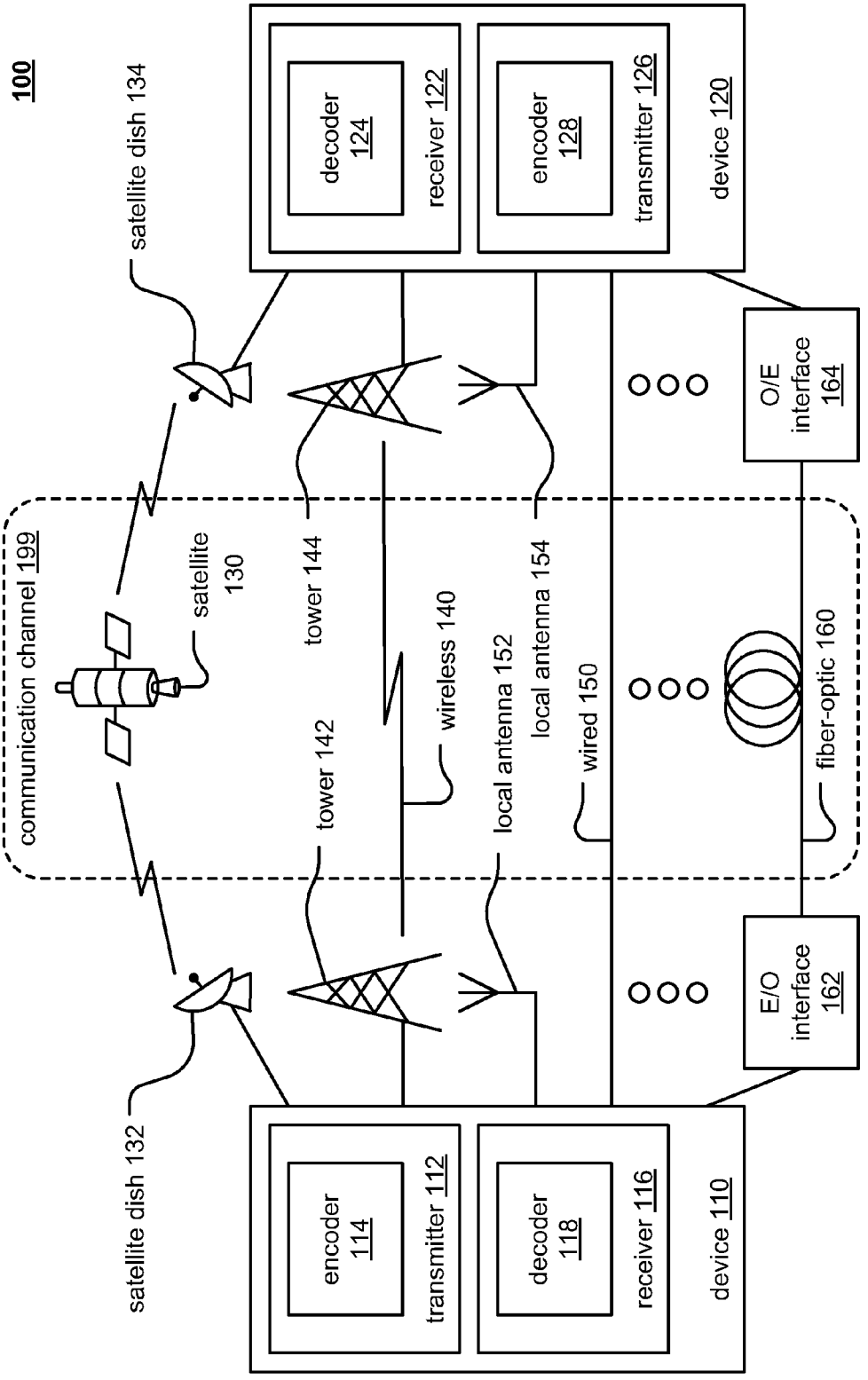

100
satellite dish 134
decoder 124
receiver 122
encoder 128
transmitter 126
device 120
O/E interface 164
communication channel 199
satellite 130
tower 144
wireless 140
local antenna 154
tower 142
local antenna 152
wired 150
fiber-optic 160
E/O interface 162
satellite dish 132
encoder 114
transmitter 112
decoder 118
receiver 116
device 110

FIG. 1

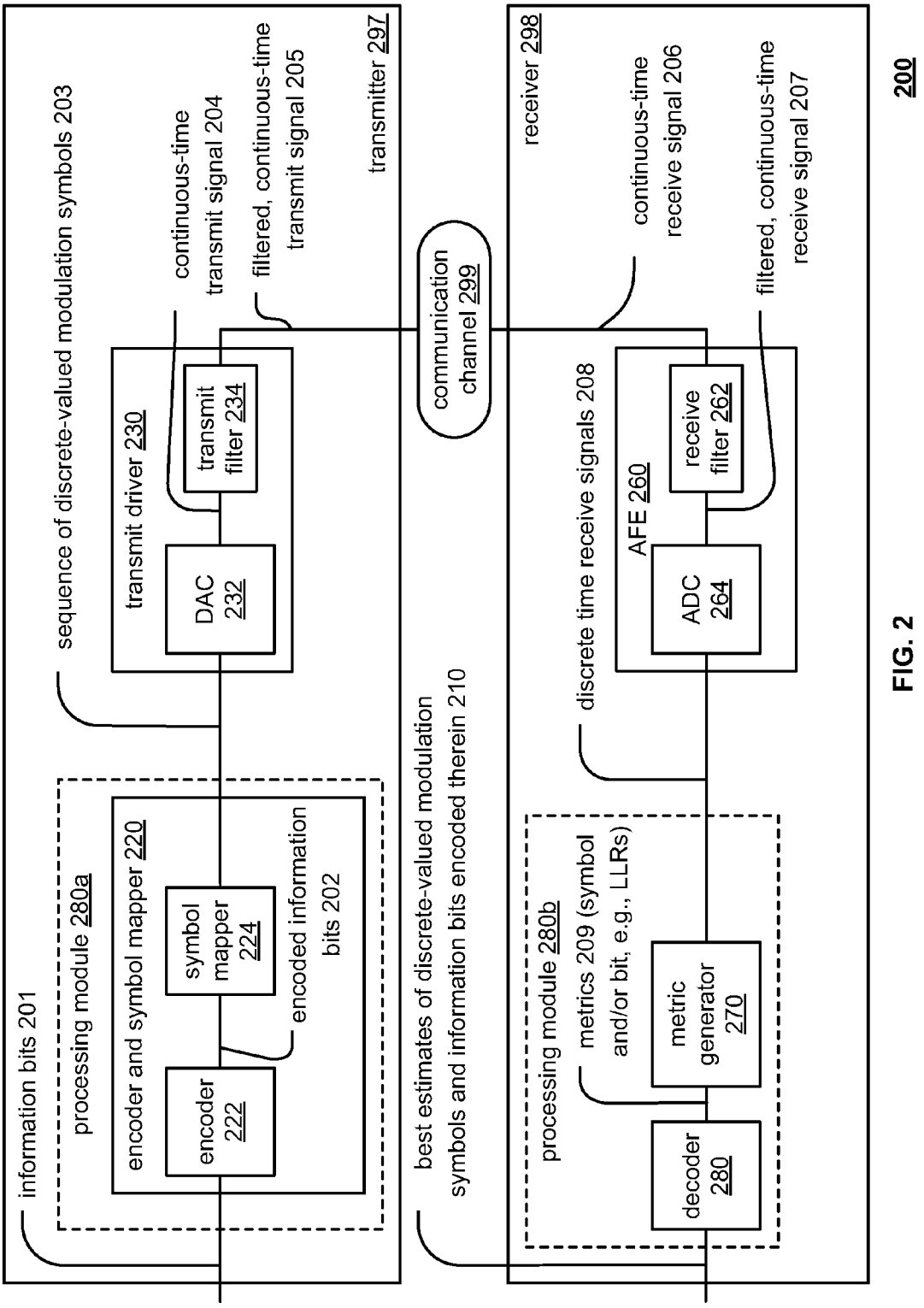
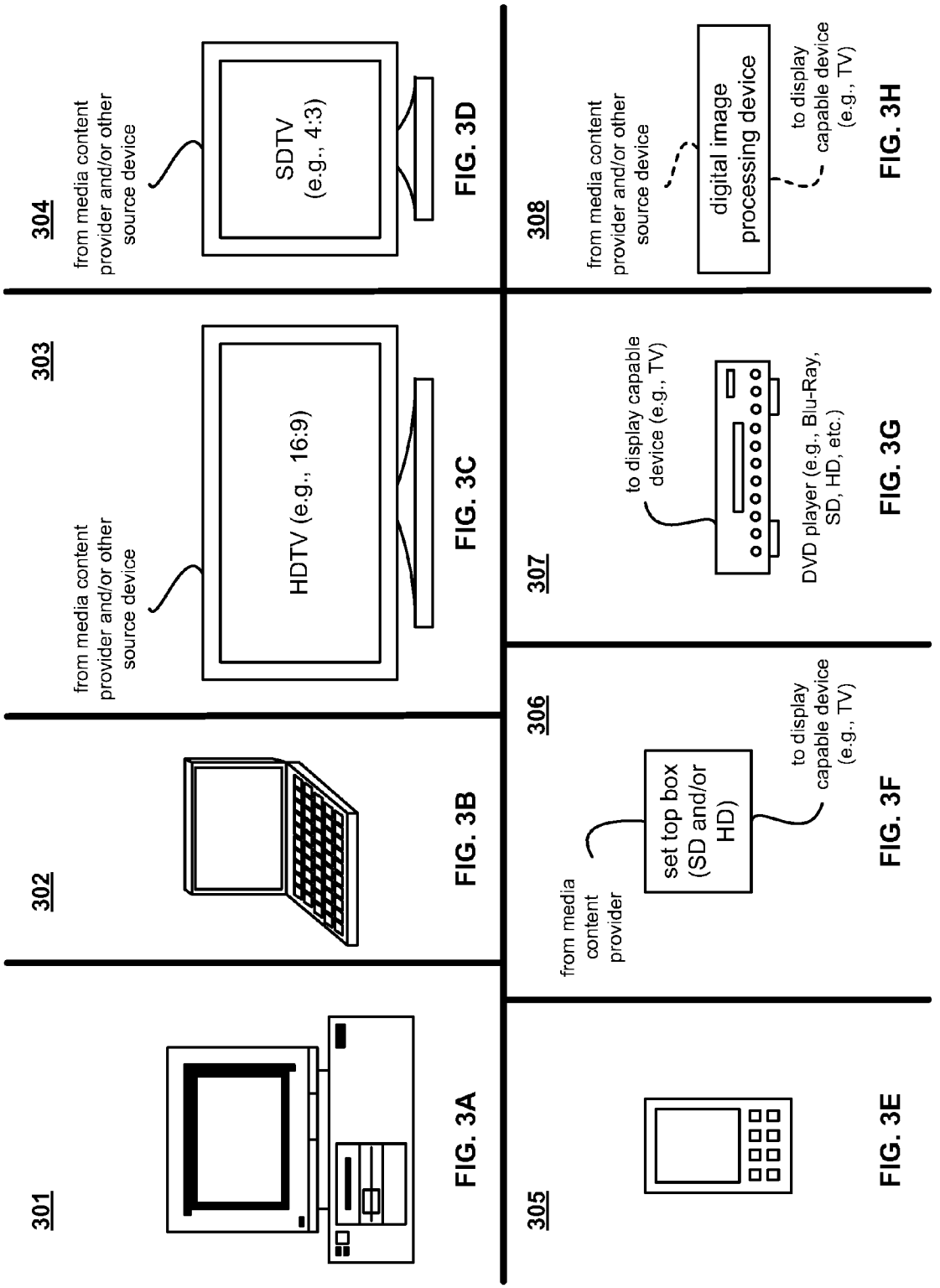

200
transmitter 297
receiver 298
information bits 201
processing module 280a
encoder and symbol mapper 220
encoder 222
symbol mapper 224
encoded information bits 202
sequence of discrete-valued modulation symbols 203
transmit driver 230
DAC 232
transmit filter 234
continuous-time transmit signal 204
filtered, continuous-time transmit signal 205
communication channel 299
continuous-time receive signal 206
filtered, continuous-time receive signal 207
AFE 260
receive filter 262
ADC 264
discrete time receive signals 208
processing module 280b
metrics 209 (symbol and/or bit, e.g., LLRs)
metric generator 270
decoder 280
best estimates of discrete-valued modulation symbols and information bits encoded therein 210

FIG. 2


301
302
303
304
305
306
307
308
from media content provider and/or other source device
HDTV (e.g., 16:9)
SDTV (e.g., 4:3)
from media content provider
set top box (SD and/or HD)
to display capable device (e.g., TV)
to display capable device (e.g., TV)
DVD player (e.g., Blu-Ray, SD, HD, etc.)
digital image processing device
FIG. 3A
FIG. 3B
FIG. 3C
FIG. 3D
FIG. 3E
FIG. 3F
FIG. 3G
FIG. 3H

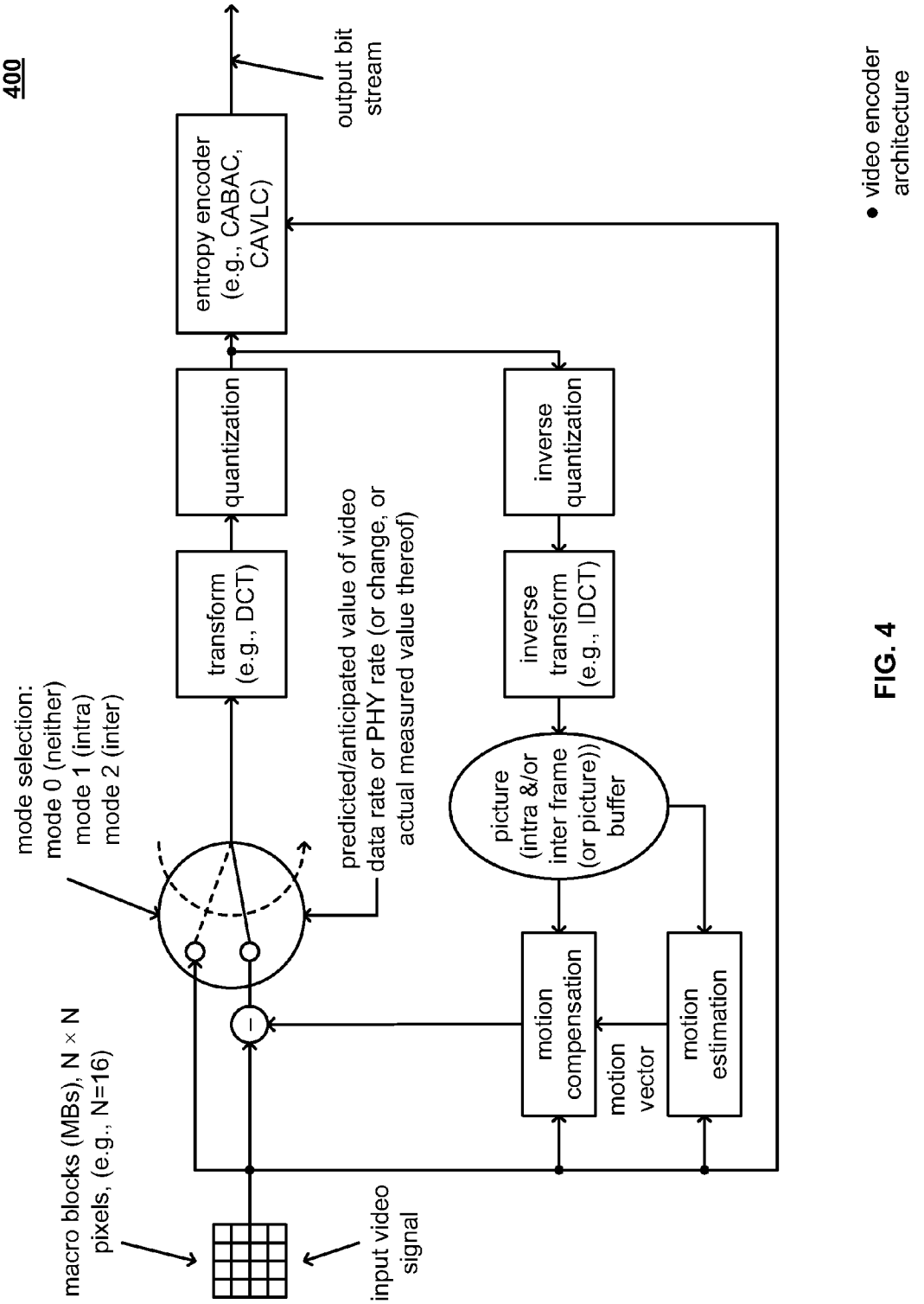

400
output bit stream
entropy encoder (e.g., CABAC, CAVLC)
quantization
transform (e.g., DCT)
inverse quantization
inverse transform (e.g., IDCT)
picture (intra &/or inter frame (or picture)) buffer
motion compensation
motion vector
motion estimation
mode selection: mode 0 (neither) mode 1 (intra) mode 2 (inter)
predicted/anticipated value of video data rate or PHY rate (or change, or actual measured value thereof)
macro blocks (MBs), N × N pixels, (e.g., N=16)
input video signal
• video encoder architecture

FIG. 4

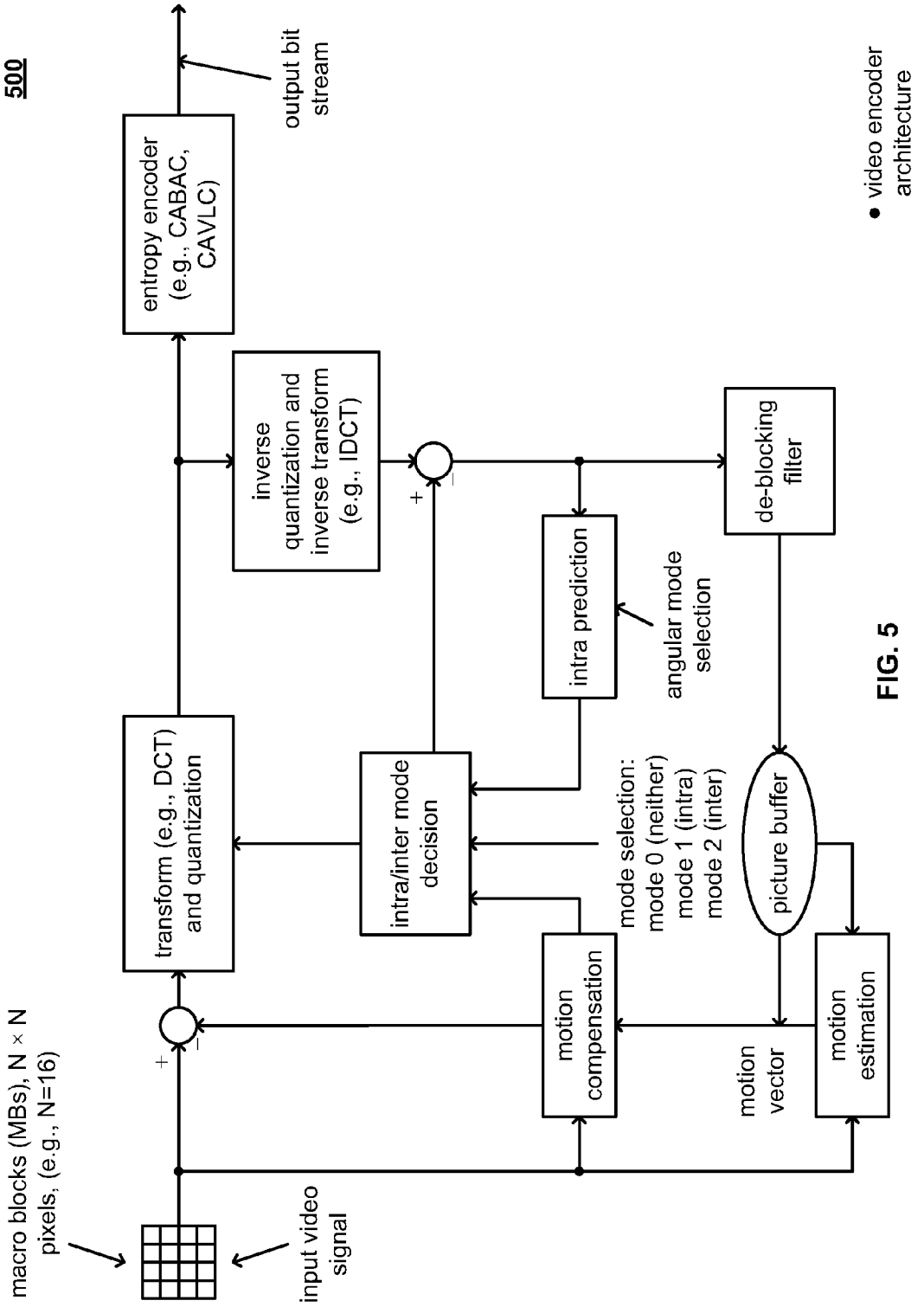

500
output bit stream
entropy encoder (e.g., CABAC, CAVLC)
inverse quantization and inverse transform (e.g., IDCT)
de-blocking filter
intra prediction
angular mode selection
transform (e.g., DCT) and quantization
intra/inter mode decision
mode selection: mode 0 (neither) mode 1 (intra) mode 2 (inter)
picture buffer
motion compensation
motion vector
motion estimation
macro blocks (MBs), N × N pixels, (e.g., N=16)
input video signal
• video encoder architecture

FIG. 5

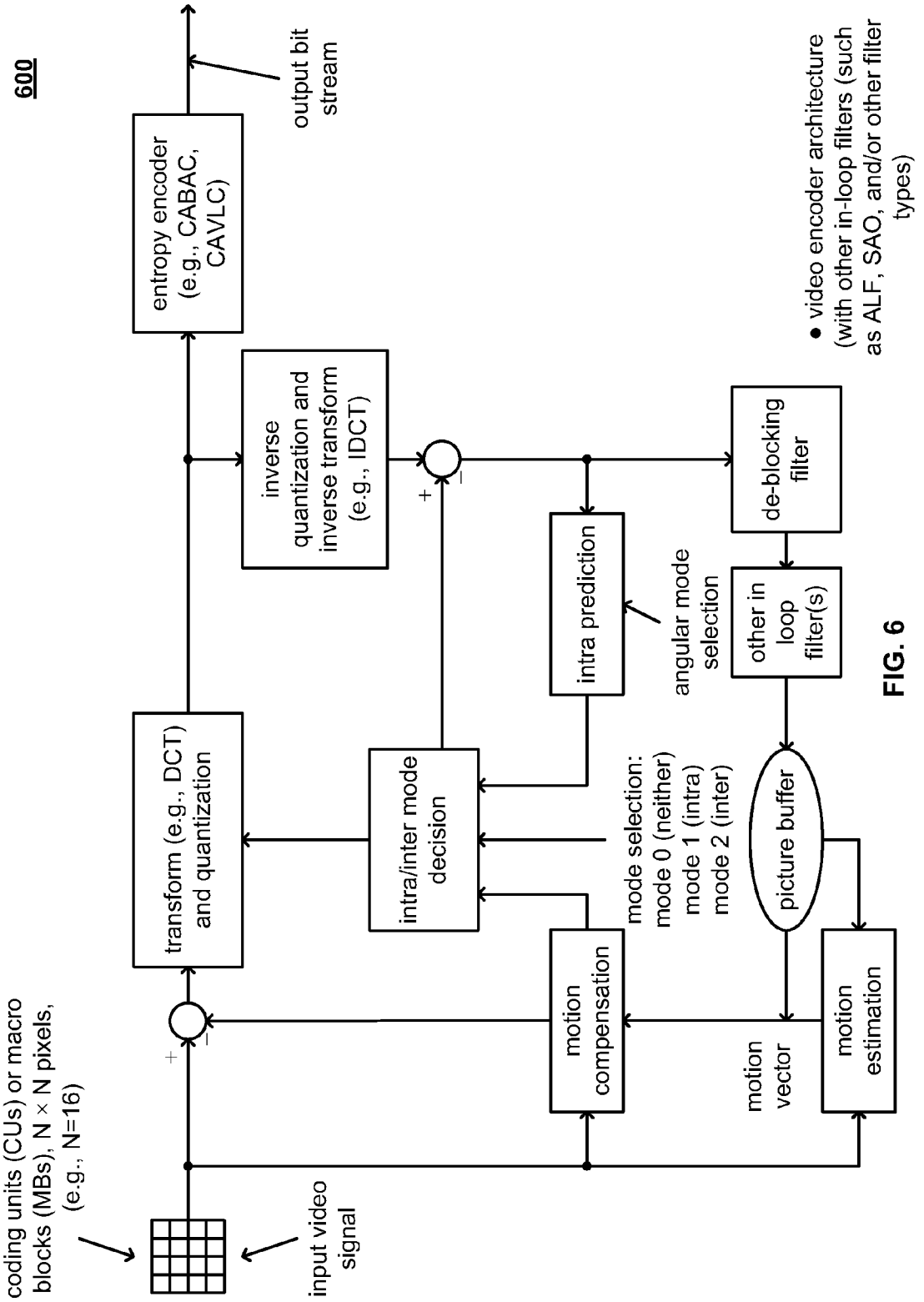

600
output bit stream
entropy encoder (e.g., CABAC, CAVLC)
inverse quantization and inverse transform (e.g., IDCT)
de-blocking filter
• video encoder architecture (with other in-loop filters (such as ALF, SAO, and/or other filter types)
intra prediction
angular mode selection
other in loop filter(s)
transform (e.g., DCT) and quantization
intra/inter mode decision
mode selection: mode 0 (neither) mode 1 (intra) mode 2 (inter)
picture buffer
motion compensation
motion vector
motion estimation
coding units (CUs) or macro blocks (MBs), N × N pixels, (e.g., N=16)
input video signal

FIG. 6

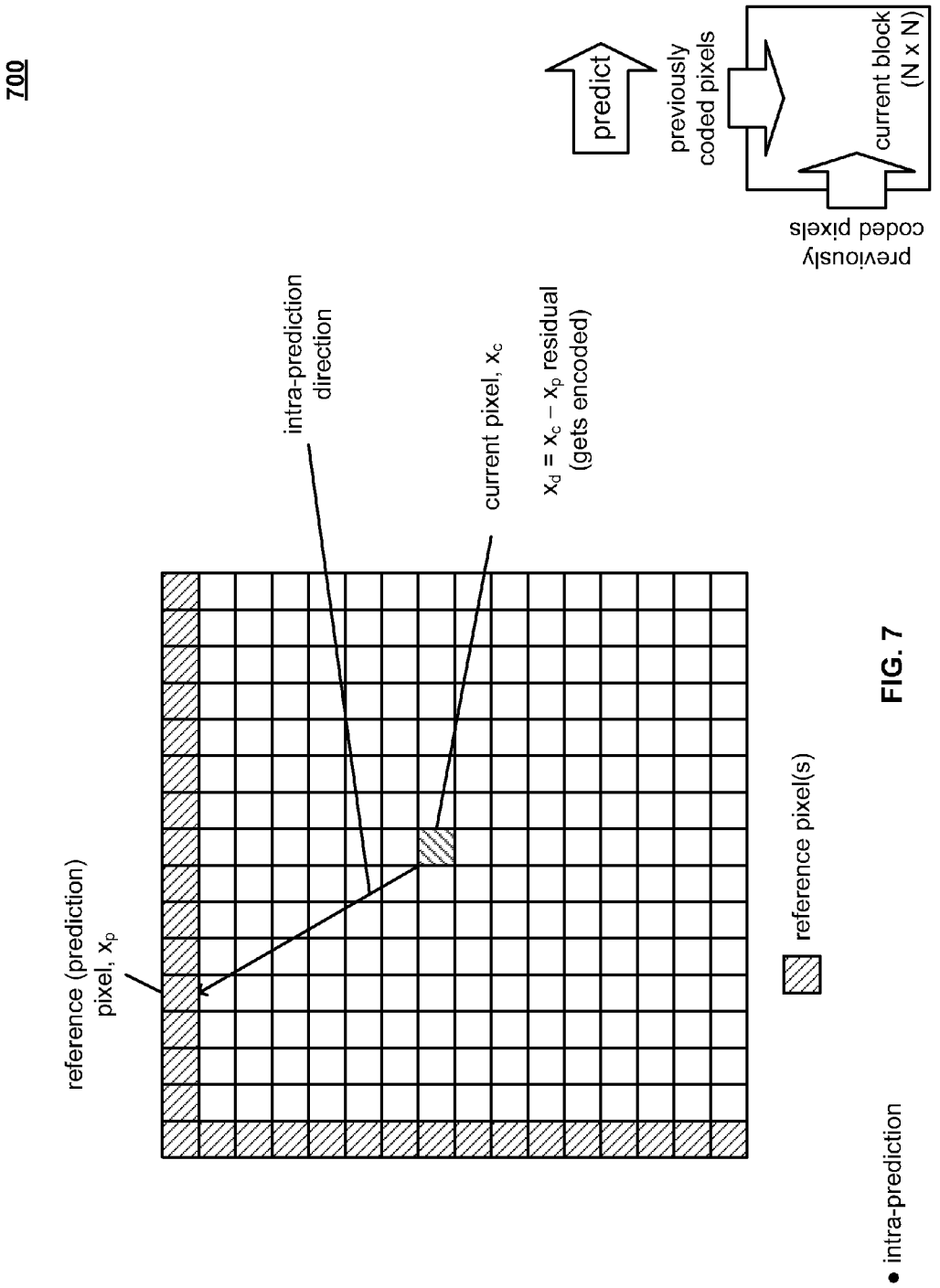

700
predict
previously coded pixels
current block (N x N)
previously coded pixels
intra-prediction direction
current pixel, x_c
$x_d = x_c - x_p$ residual (gets encoded)
reference (prediction) pixel, x_p
reference pixel(s)
• intra-prediction

FIG. 7

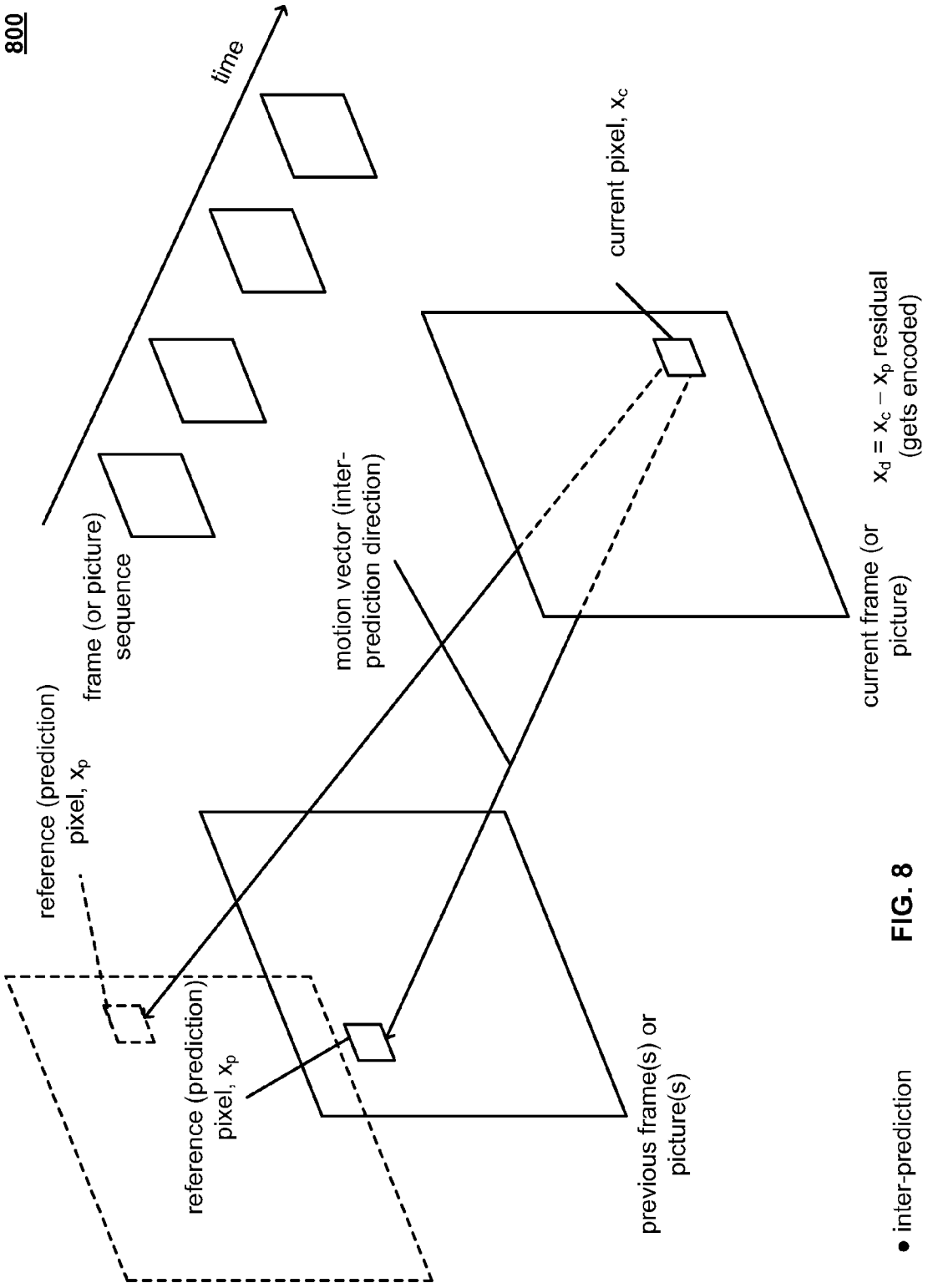

800
time
frame (or picture) sequence
reference (prediction) pixel, x_p
motion vector (inter-prediction direction)
current pixel, x_c
$x_d = x_c - x_p$ residual (gets encoded)
current frame (or picture)
reference (prediction) pixel, x_p
previous frame(s) or picture(s)
• inter-prediction

FIG. 8

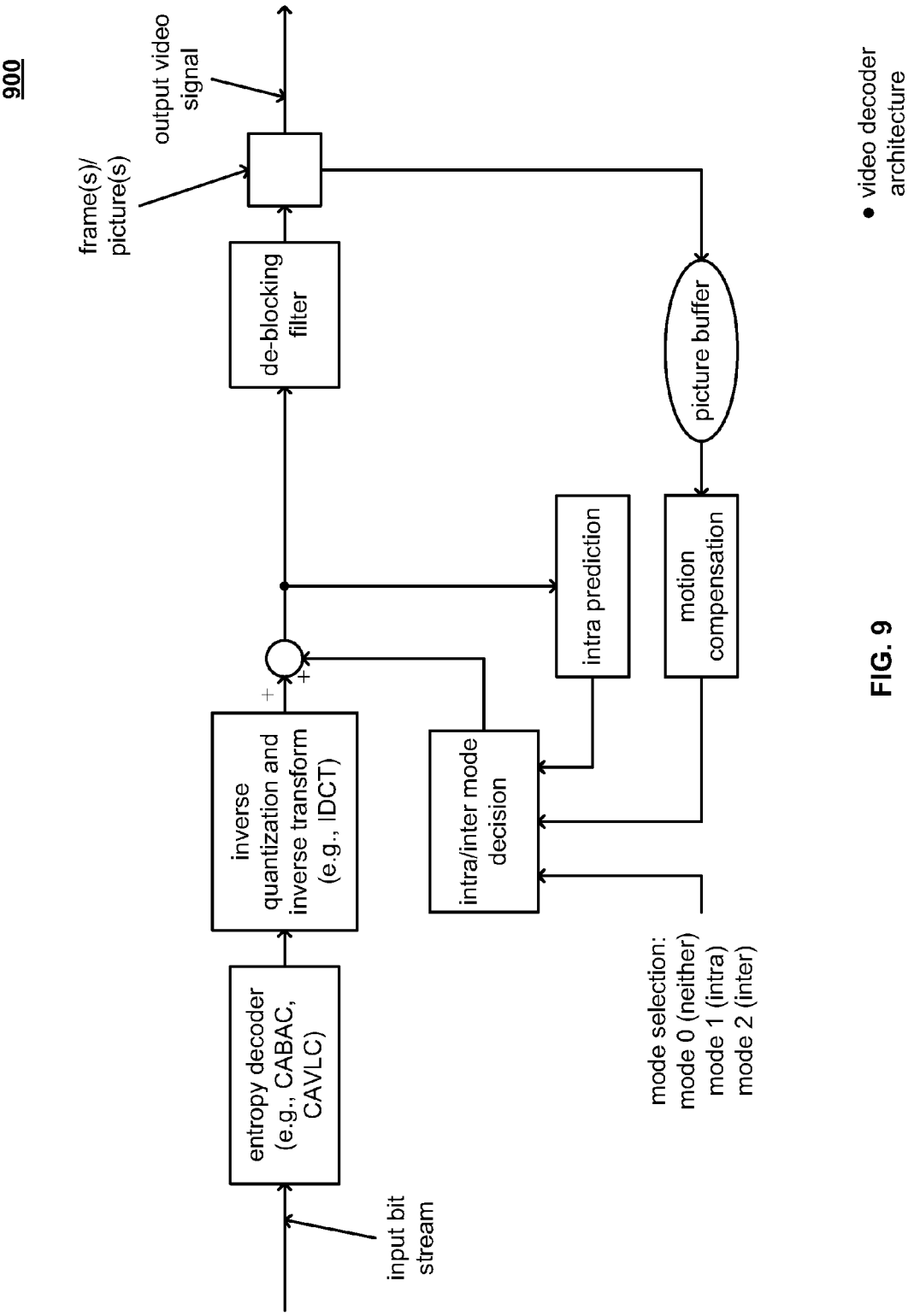

900
output video signal
frame(s)/ picture(s)
de-blocking filter
picture buffer
intra prediction
motion compensation
inverse quantization and inverse transform (e.g., IDCT)
intra/inter mode decision
entropy decoder (e.g., CABAC, CAVLC)
input bit stream
mode selection: mode 0 (neither) mode 1 (intra) mode 2 (inter)
• video decoder architecture

FIG. 9

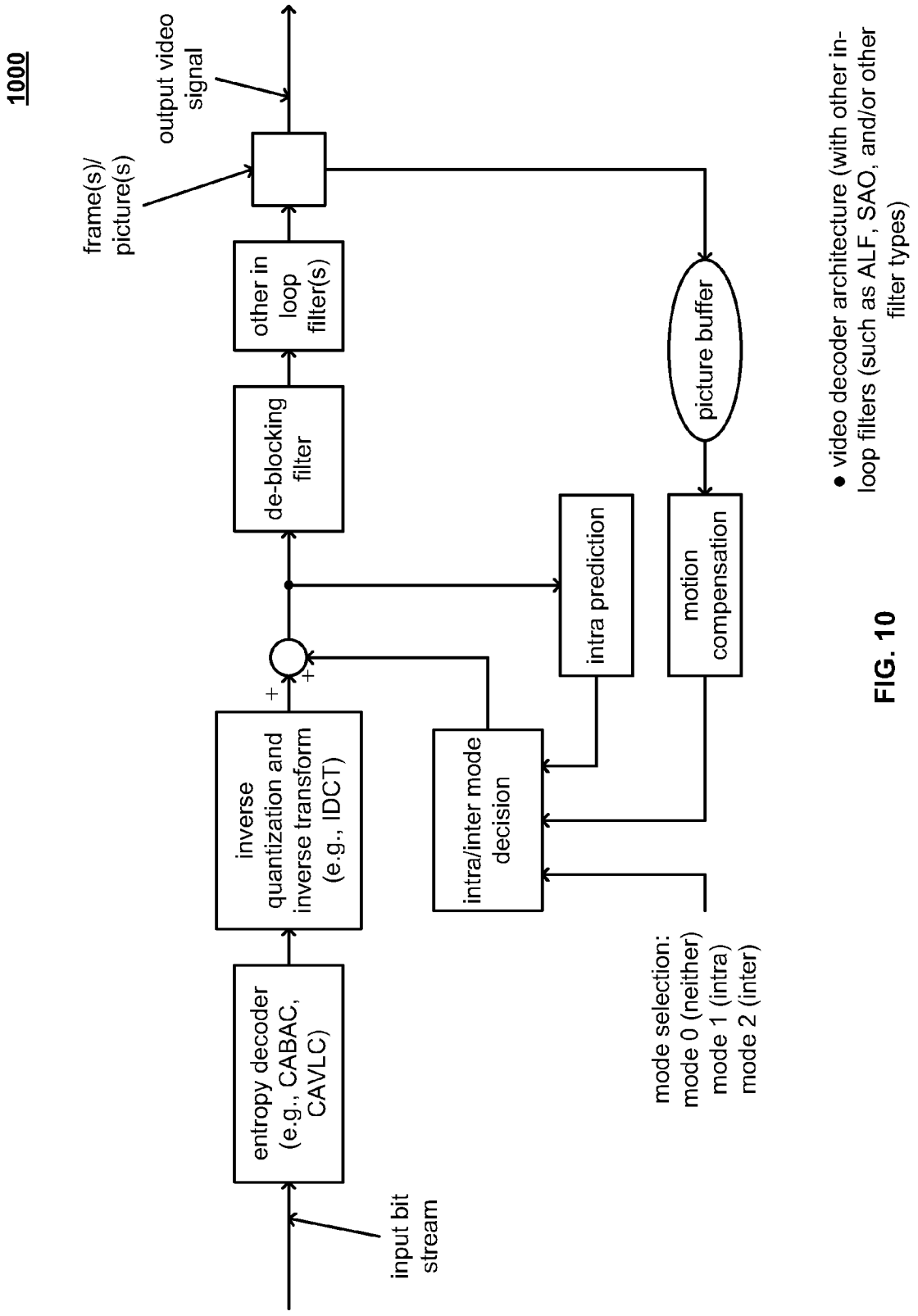

1000
output video signal
frame(s)/ picture(s)
other in loop filter(s)
de-blocking filter
picture buffer
intra prediction
motion compensation
inverse quantization and inverse transform (e.g., IDCT)
intra/inter mode decision
entropy decoder (e.g., CABAC, CAVLC)
input bit stream
mode selection: mode 0 (neither) mode 1 (intra) mode 2 (inter)
• video decoder architecture (with other in-loop filters (such as ALF, SAO, and/or other filter types)

FIG. 10

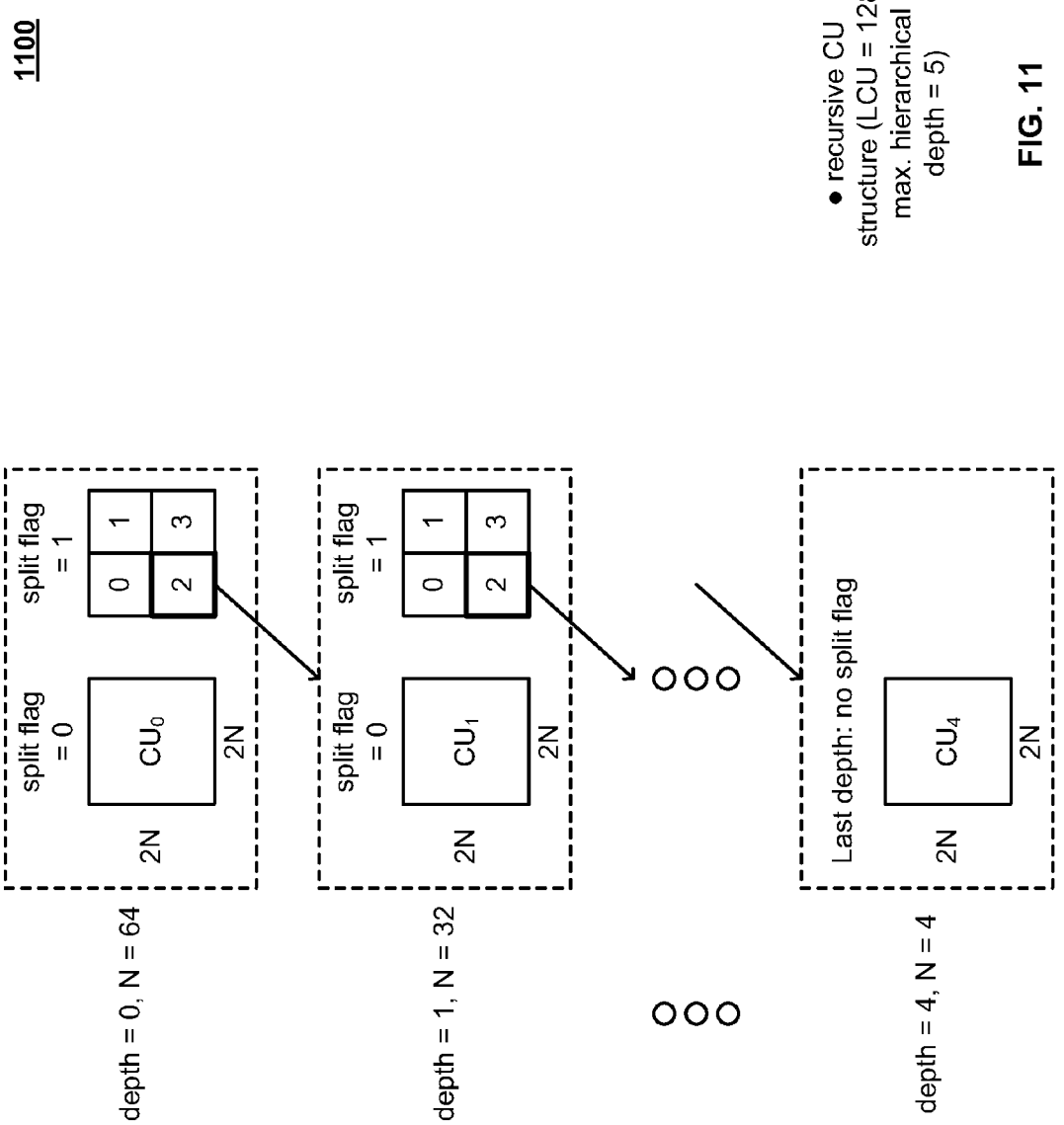
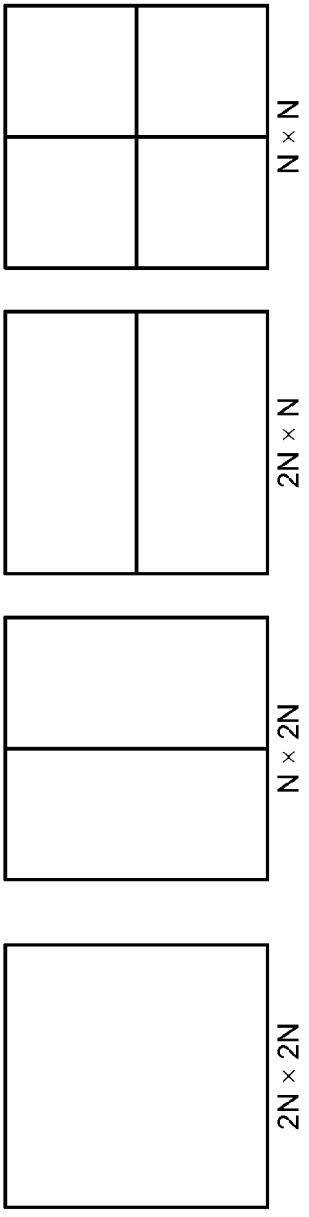

1100
• recursive CU structure (LCU = 128, max. hierarchical depth = 5)
split flag = 1
split flag = 0
CU_0
2N
0
1
2
3
depth = 0, N = 64
CU_1
depth = 1, N = 32
Last depth: no split flag
CU_4
depth = 4, N = 4

FIG. 11

1200



• PU modes

FIG. 12

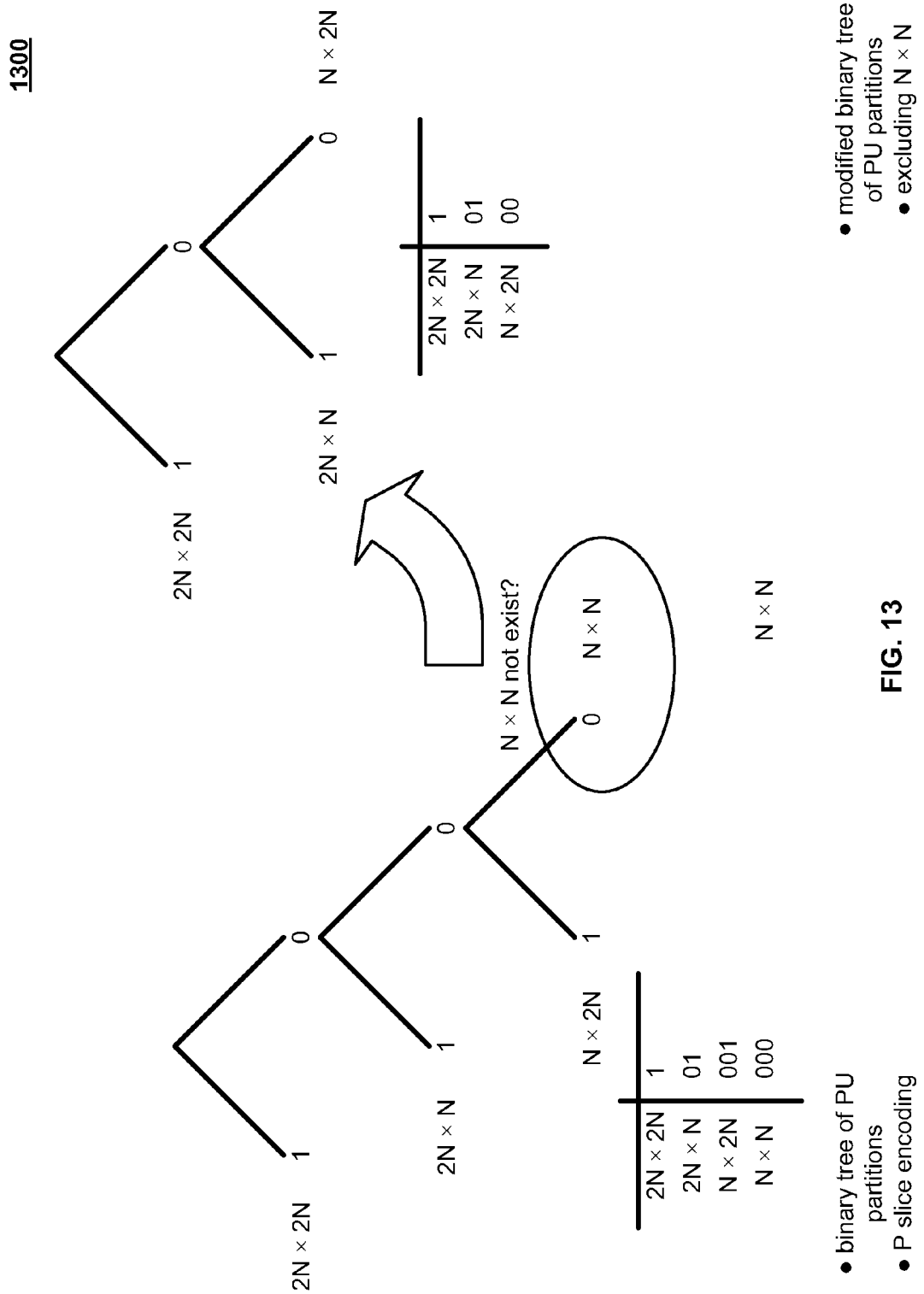

1300
2N × 2N
2N × N
N × 2N
N × N
N × N not exist?
• binary tree of PU partitions
• P slice encoding
• modified binary tree of PU partitions
• excluding N × N

FIG. 13

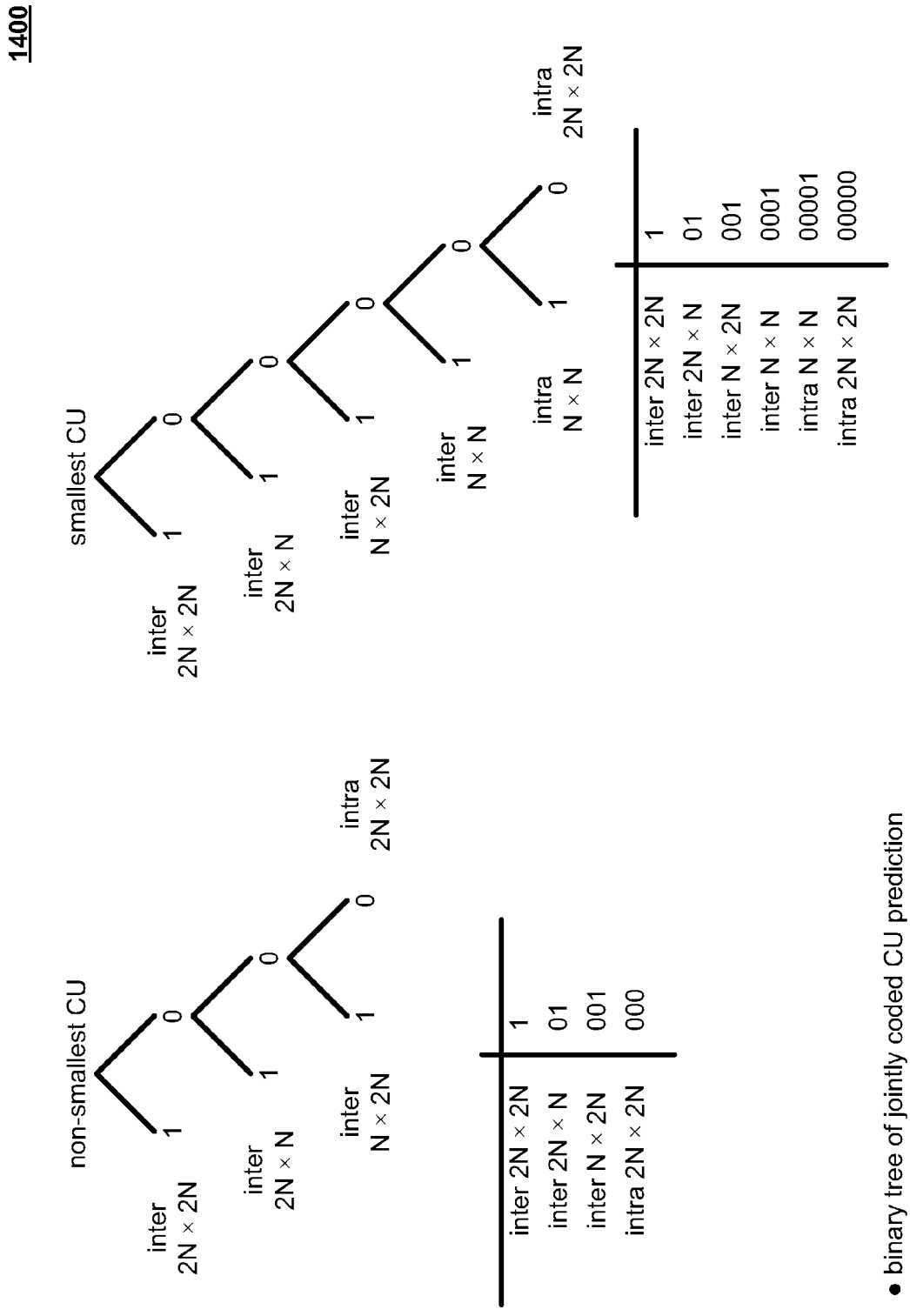

1400
non-smallest CU
inter 2N × 2N
inter 2N × N
inter N × 2N
intra 2N × 2N
1
01
001
000
• binary tree of jointly coded CU prediction mode and PU partition mode for B slices
• B slice encoding
smallest CU
inter 2N × 2N
inter 2N × N
inter N × 2N
inter N × N
intra N × N
intra 2N × 2N
1
01
001
0001
00001
00000
• alternatively, employ same binary tree for both P and B slices

FIG. 14

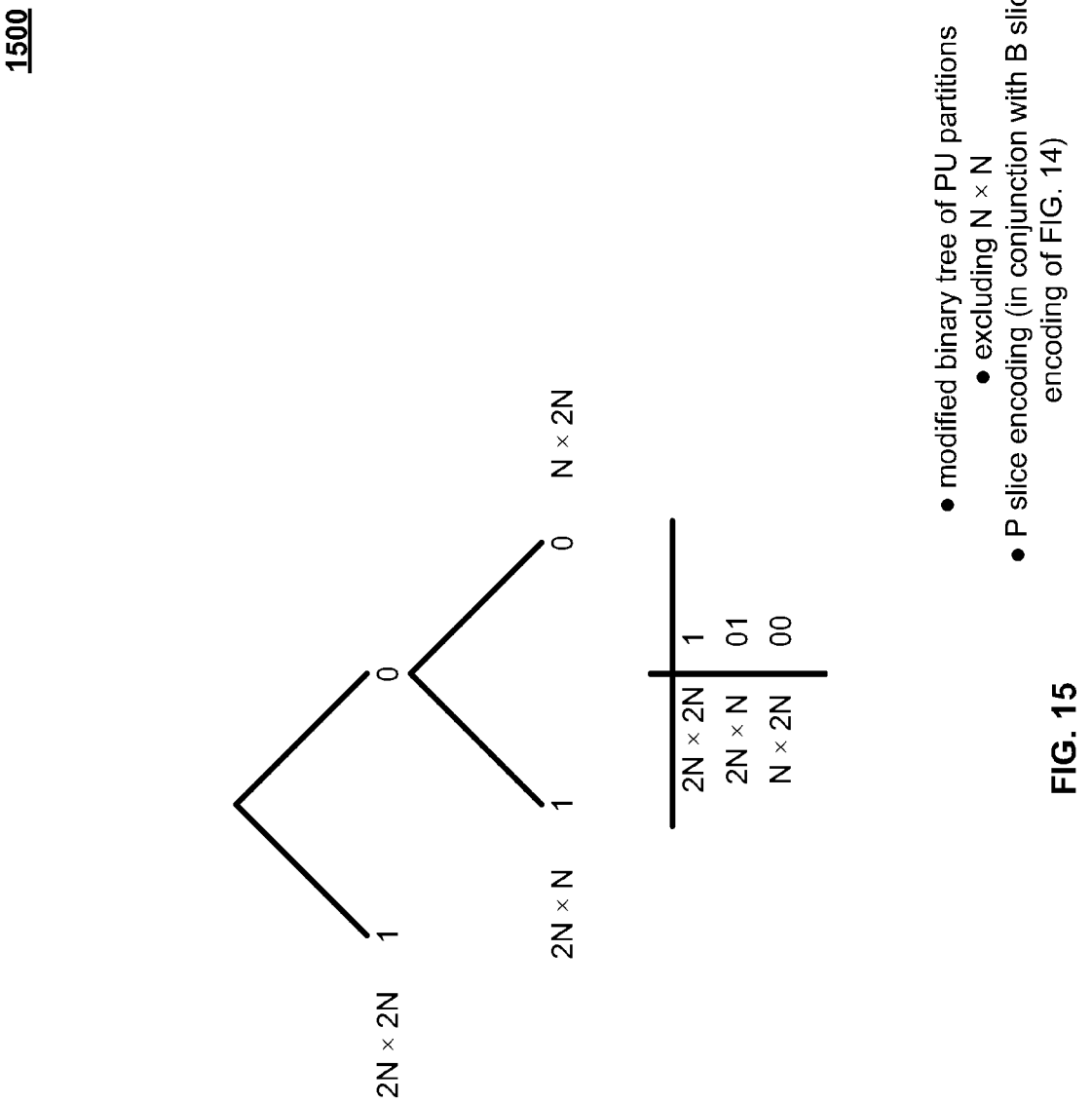

1500
2N × 2N
1
0
2N × N
1
0
N × 2N
2N × 2N 1
2N × N 01
N × 2N 00
• modified binary tree of PU partitions
• excluding N × N
• P slice encoding (in conjunction with B slice encoding of FIG. 14)

FIG. 15

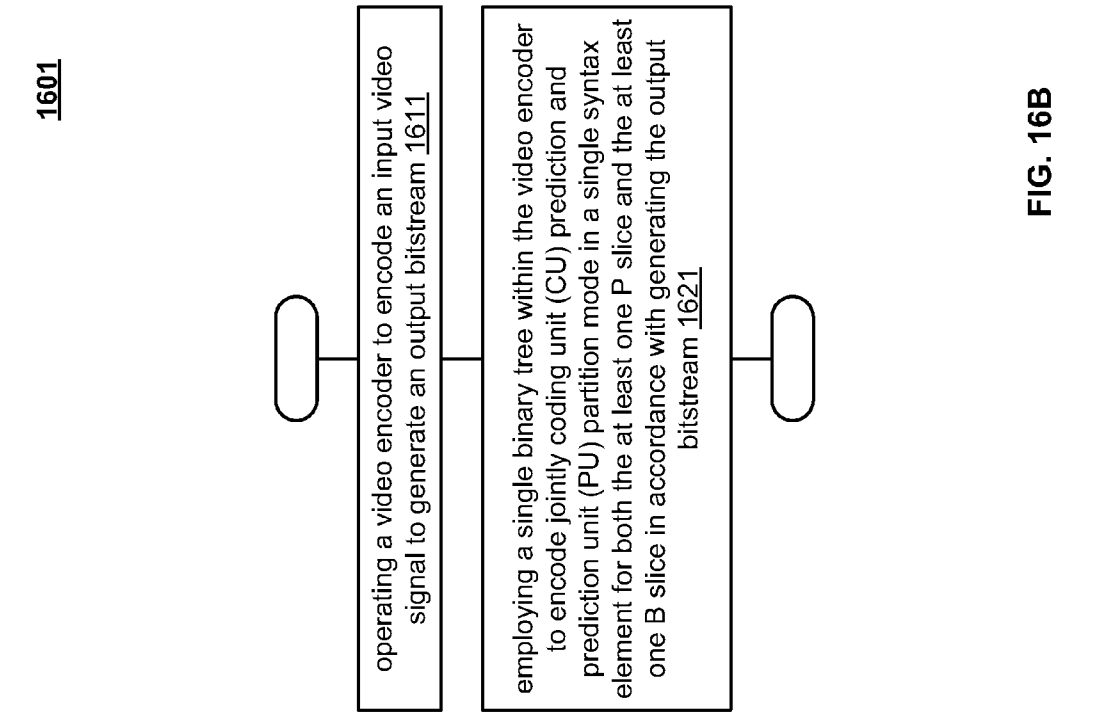


FIG. 16B

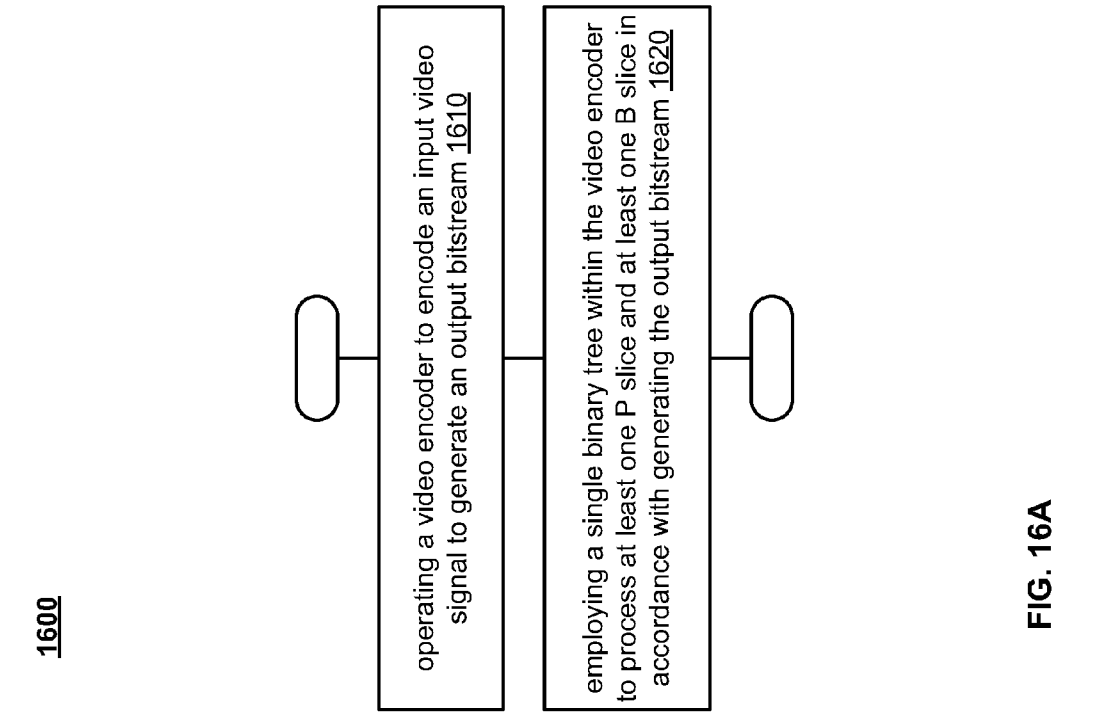


FIG. 16A

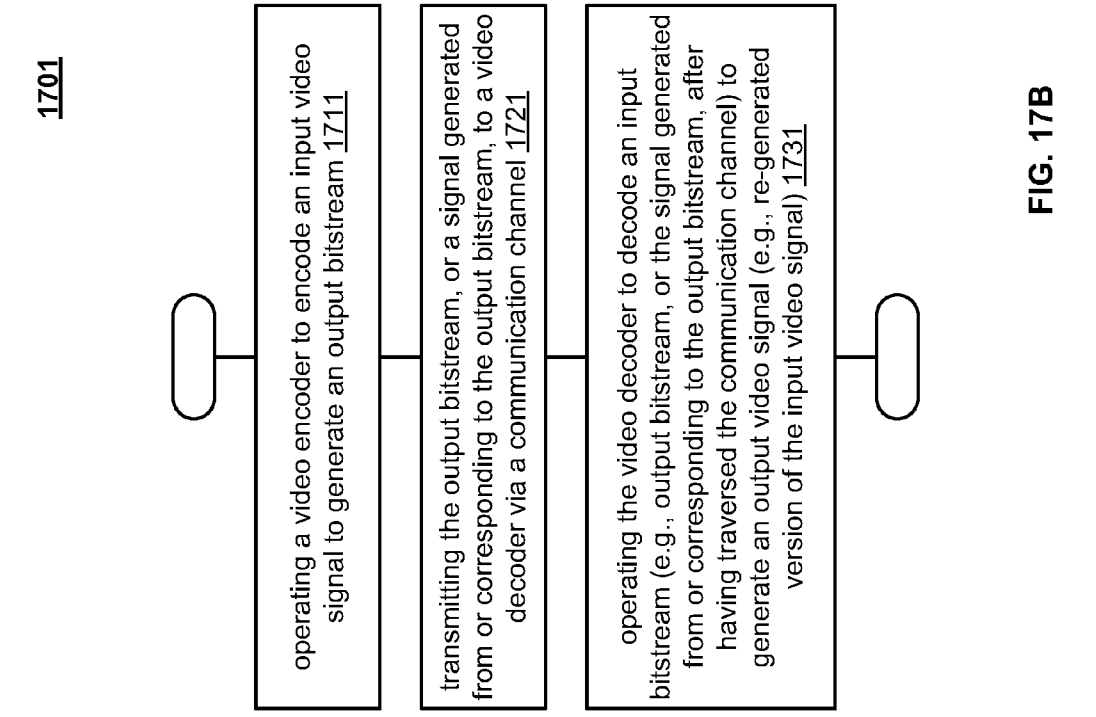


FIG. 17B

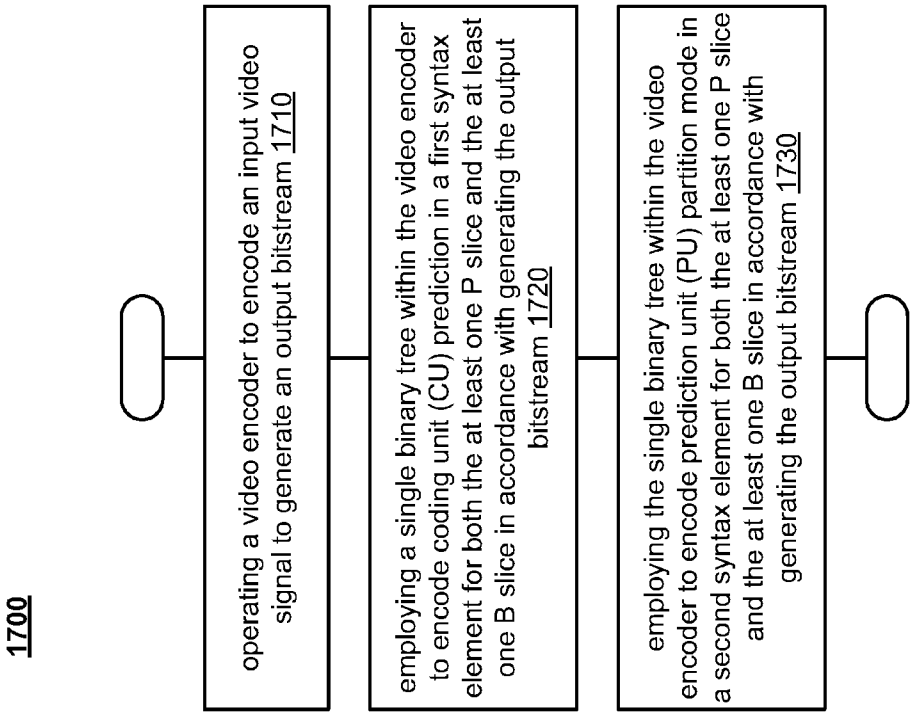


FIG. 17A

SIGNALING OF PREDICTION SIZE UNIT IN ACCORDANCE WITH VIDEO CODING

CROSS REFERENCE TO RELATED PATENTS/PATENT APPLICATIONS

Provisional Priority Claims

The present U.S. Utility Patent Application claims priority pursuant to 35 U.S.C. §119(e) to the following U.S. Provisional Patent Application which is hereby incorporated herein by reference in its entirety and made part of the present U.S. Utility Patent Application for all purposes:

1. U.S. Provisional Patent Application Ser. No. 61/539,948, entitled "Signaling of prediction size unit in accordance with video coding,", filed Sep. 27, 2011.

Incorporation by Reference

The following standards/draft standards are hereby incorporated herein by reference in their entirety and are made part of the present U.S. Utility Patent Application for all purposes:

1. "High efficiency video coding (HEVC) text specification draft 6," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, 7th Meeting: Geneva, CH, 21-30 Nov. 2011, Document: JCTVC-H1003, 259 pages.

2. International Telecommunication Union, ITU-T, TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, H.264 (March 2010), SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services—Coding of moving video, Advanced video coding for generic audiovisual services, Recommendation ITU-T H.264, also alternatively referred to as International Telecomm ISO/IEC 14496-10—MPEG-4 Part 10, AVC (Advanced Video Coding), H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding), ITU H.264/MPEG4-AVC, or equivalent.

BACKGROUND OF THE INVENTION

1. Technical Field of the Invention

The invention relates generally to digital video processing; and, more particularly, it relates to signaling in accordance with such digital video processing.

2. Description of Related Art

Communication systems that operate to communicate digital media (e.g., images, video, data, etc.) have been under continual development for many years. With respect to such communication systems employing some form of video data, a number of digital images are output or displayed at some frame rate (e.g., frames per second) to effectuate a video signal suitable for output and consumption. Within many such communication systems operating using video data, there can be a trade-off between throughput (e.g., number of image frames that may be transmitted from a first location to a second location) and video and/or image quality of the signal eventually to be output or displayed. The present art does not adequately or acceptably provide a means by which video data may be transmitted from a first location to a second location in accordance with providing an adequate or acceptable video and/or image quality, ensuring a relatively low amount of overhead associated with the communications, relatively low complexity of the communication devices at respective ends of communication links, etc.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

FIG. **1** and FIG. **2** illustrate various embodiments of communication systems.

FIG. **3**A illustrates an embodiment of a computer.

FIG. **3**B illustrates an embodiment of a laptop computer.

FIG. **3**C illustrates an embodiment of a high definition (HD) television.

FIG. **3**D illustrates an embodiment of a standard definition (SD) television.

FIG. **3**E illustrates an embodiment of a handheld media unit.

FIG. **3**F illustrates an embodiment of a set top box (STB).

FIG. **3**G illustrates an embodiment of a digital video disc (DVD) player.

FIG. **3**H illustrates an embodiment of a generic digital image and/or video processing device.

FIG. **4**, FIG. **5**, and FIG. **6** are diagrams illustrating various embodiments of video encoding architectures.

FIG. **7** is a diagram illustrating an embodiment of intra-prediction processing.

FIG. **8** is a diagram illustrating an embodiment of inter-prediction processing.

FIG. **9** and FIG. **10** are diagrams illustrating various embodiments of video decoding architectures.

FIG. **11** illustrates an embodiment of recursive coding unit (CU) structure.

FIG. **12** illustrates an embodiment of prediction unit (PU) modes.

FIG. **13** illustrates an embodiment of a binary tree, including a modification thereof, employed for P slice encoding in one implementation and for both P and B slice encoding in another implementation.

FIG. **14** illustrates an embodiment of a binary tree as may be employed for B slice encoding in one implementation and for both P and B slice encoding in another implementation.

FIG. **15** illustrates an embodiment of a binary tree as may be employed for P slice encoding in conjunction with B slice encoding as performed in accordance with FIG. **12**.

FIG. **16**A, FIG. **16**B, FIG. **17**A, and FIG. **17**B, illustrate various embodiments of methods performed in accordance with video coding (e.g., within one or more communication devices).

DETAILED DESCRIPTION OF THE INVENTION

Within many devices that use digital media such as digital video, respective images thereof, being digital in nature, are represented using pixels. Within certain communication systems, digital media can be transmitted from a first location to a second location at which such media can be output or displayed. The goal of digital communications systems, including those that operate to communicate digital video, is to transmit digital data from one location, or subsystem, to another either error free or with an acceptably low error rate. As shown in FIG. **1**, data may be transmitted over a variety of communications channels in a wide variety of communication systems: magnetic media, wired, wireless, fiber, copper, and/or other types of media as well.

FIG. **1** and FIG. **2** are diagrams illustrate various embodiments of communication systems, **100** and **200**, respectively.

Referring to FIG. **1**, this embodiment of a communication system **100** is a communication channel **199** that communicatively couples a communication device **110** (including a transmitter **112** having an encoder **114** and including a receiver **116** having a decoder **118**) situated at one end of the

communication channel **199** to another communication device **120** (including a transmitter **126** having an encoder **128** and including a receiver **122** having a decoder **124**) at the other end of the communication channel **199**. In some embodiments, either of the communication devices **110** and **120** may only include a transmitter or a receiver. There are several different types of media by which the communication channel **199** may be implemented (e.g., a satellite communication channel **130** using satellite dishes **132** and **134**, a wireless communication channel **140** using towers **142** and **144** and/or local antennae **152** and **154**, a wired communication channel **150**, and/or a fiber-optic communication channel **160** using electrical to optical (E/O) interface **162** and optical to electrical (O/E) interface **164**)). In addition, more than one type of media may be implemented and interfaced together thereby forming the communication channel **199**.

It is noted that such communication devices **110** and/or **120** may be stationary or mobile without departing from the scope and spirit of the invention. For example, either one or both of the communication devices **110** and **120** may be implemented in a fixed location or may be a mobile communication device with capability to associate with and/or communicate with more than one network access point (e.g., different respective access points (APs) in the context of a mobile communication system including one or more wireless local area networks (WLANs), different respective satellites in the context of a mobile communication system including one or more satellite, or generally, different respective network access points in the context of a mobile communication system including one or more network access points by which communications may be effectuated with communication devices **110** and/or **120**.

To reduce transmission errors that may undesirably be incurred within a communication system, error correction and channel coding schemes are often employed. Generally, these error correction and channel coding schemes involve the use of an encoder at the transmitter end of the communication channel **199** and a decoder at the receiver end of the communication channel **199**.

Any of various types of ECC codes described can be employed within any such desired communication system (e.g., including those variations described with respect to FIG. **1**), any information storage device (e.g., hard disk drives (HDDs), network information storage devices and/or servers, etc.) or any application in which information encoding and/or decoding is desired.

Generally speaking, when considering a communication system in which video data is communicated from one location, or subsystem, to another, video data encoding may generally be viewed as being performed at a transmitting end of the communication channel **199**, and video data decoding may generally be viewed as being performed at a receiving end of the communication channel **199**.

Also, while the embodiment of this diagram shows bi-directional communication being capable between the communication devices **110** and **120**, it is of course noted that, in some embodiments, the communication device **110** may include only video data encoding capability, and the communication device **120** may include only video data decoding capability, or vice versa (e.g., in a uni-directional communication embodiment such as in accordance with a video broadcast embodiment).

Referring to the communication system **200** of FIG. **2**, at a transmitting end of a communication channel **299**, information bits **201** (e.g., corresponding particularly to video data in one embodiment) are provided to a transmitter **297** that is operable to perform encoding of these information bits **201** using an encoder and symbol mapper **220** (which may be viewed as being distinct functional blocks **222** and **224**, respectively) thereby generating a sequence of discrete-valued modulation symbols **203** that is provided to a transmit driver **230** that uses a DAC (Digital to Analog Converter) **232** to generate a continuous-time transmit signal **204** and a transmit filter **234** to generate a filtered, continuous-time transmit signal **205** that substantially comports with the communication channel **299**. At a receiving end of the communication channel **299**, continuous-time receive signal **206** is provided to an AFE (Analog Front End) **260** that includes a receive filter **262** (that generates a filtered, continuous-time receive signal **207**) and an ADC (Analog to Digital Converter) **264** (that generates discrete-time receive signals **208**). A metric generator **270** calculates metrics **209** (e.g., on either a symbol and/or bit basis) that are employed by a decoder **280** to make best estimates of the discrete-valued modulation symbols and information bits encoded therein **210**.

Within each of the transmitter **297** and the receiver **298**, any desired integration of various components, blocks, functional blocks, circuitries, etc. Therein may be implemented. For example, this diagram shows a processing module **280***a* as including the encoder and symbol mapper **220** and all associated, corresponding components therein, and a processing module **280** is shown as including the metric generator **270** and the decoder **280** and all associated, corresponding components therein. Such processing modules **280***a* and **280***b* may be respective integrated circuits. Of course, other boundaries and groupings may alternatively be performed without departing from the scope and spirit of the invention. For example, all components within the transmitter **297** may be included within a first processing module or integrated circuit, and all components within the receiver **298** may be included within a second processing module or integrated circuit. Alternatively, any other combination of components within each of the transmitter **297** and the receiver **298** may be made in other embodiments.

As with the previous embodiment, such a communication system **200** may be employed for the communication of video data is communicated from one location, or subsystem, to another (e.g., from transmitter **297** to the receiver **298** via the communication channel **299**).

Digital image and/or video processing of digital images and/or media (including the respective images within a digital video signal) may be performed by any of the various devices depicted below in FIG. **3**A-**3**H to allow a user to view such digital images and/or video. These various devices do not include an exhaustive list of devices in which the image and/or video processing described herein may be effectuated, and it is noted that any generic digital image and/or video processing device may be implemented to perform the processing described herein without departing from the scope and spirit of the invention.

FIG. **3**A illustrates an embodiment of a computer **301**. The computer **301** can be a desktop computer, or an enterprise storage devices such a server, of a host computer that is attached to a storage array such as a redundant array of independent disks (RAID) array, storage router, edge router, storage switch and/or storage director. A user is able to view still digital images and/or video (e.g., a sequence of digital images) using the computer **301**. Oftentimes, various image and/or video viewing programs and/or media player programs are included on a computer **301** to allow a user to view such images (including video).

FIG. **3**B illustrates an embodiment of a laptop computer **302**. Such a laptop computer **302** may be found and used in any of a wide variety of contexts. In recent years, with the

ever-increasing processing capability and functionality found within laptop computers, they are being employed in many instances where previously higher-end and more capable desktop computers would be used. As with the computer **301**, the laptop computer **302** may include various image viewing programs and/or media player programs to allow a user to view such images (including video).

FIG. **3**C illustrates an embodiment of a high definition (HD) television **303**. Many HD televisions **303** include an integrated tuner to allow the receipt, processing, and decoding of media content (e.g., television broadcast signals) thereon. Alternatively, sometimes an HD television **303** receives media content from another source such as a digital video disc (DVD) player, set top box (STB) that receives, processes, and decodes a cable and/or satellite television broadcast signal. Regardless of the particular implementation, the HD television **303** may be implemented to perform image and/or video processing as described herein. Generally speaking, an HD television **303** has capability to display HD media content and oftentimes is implemented having a 16:9 widescreen aspect ratio.

FIG. **3**D illustrates an embodiment of a standard definition (SD) television **304**. Of course, an SD television **304** is somewhat analogous to an HD television **303**, with at least one difference being that the SD television **304** does not include capability to display HD media content, and an SD television **304** oftentimes is implemented having a 4:3 full screen aspect ratio. Nonetheless, even an SD television **304** may be implemented to perform image and/or video processing as described herein.

FIG. **3**E illustrates an embodiment of a handheld media unit **305**. A handheld media unit **305** may operate to provide general storage or storage of image/video content information such as joint photographic experts group (JPEG) files, tagged image file format (TIFF), bitmap, motion picture experts group (MPEG) files, Windows Media (WMA/WMV) files, other types of video content such as MPEG4 files, etc. for playback to a user, and/or any other type of information that may be stored in a digital format. Historically, such handheld media units were primarily employed for storage and playback of audio media; however, such a handheld media unit **305** may be employed for storage and playback of virtual any media (e.g., audio media, video media, photographic media, etc.). Moreover, such a handheld media unit **305** may also include other functionality such as integrated communication circuitry for wired and wireless communications. Such a handheld media unit **305** may be implemented to perform image and/or video processing as described herein.

FIG. **3**F illustrates an embodiment of a set top box (STB) **306**. As mentioned above, sometimes a STB **306** may be implemented to receive, process, and decode a cable and/or satellite television broadcast signal to be provided to any appropriate display capable device such as SD television **304** and/or HD television **303**. Such an STB **306** may operate independently or cooperatively with such a display capable device to perform image and/or video processing as described herein.

FIG. **3**G illustrates an embodiment of a digital video disc (DVD) player **307**. Such a DVD player may be a Blu-Ray DVD player, an HD capable DVD player, an SD capable DVD player, an up-sampling capable DVD player (e.g., from SD to HD, etc.) without departing from the scope and spirit of the invention. The DVD player may provide a signal to any appropriate display capable device such as SD television **304** and/or HD television **303**. The DVD player **305** may be implemented to perform image and/or video processing as described herein.

FIG. **3**H illustrates an embodiment of a generic digital image and/or video processing device **308**. Again, as mentioned above, these various devices described above do not include an exhaustive list of devices in which the image and/or video processing described herein may be effectuated, and it is noted that any generic digital image and/or video processing device **308** may be implemented to perform the image and/or video processing described herein without departing from the scope and spirit of the invention.

FIG. **4**, FIG. **5**, and FIG. **6** are diagrams illustrating various embodiments **400** and **500**, and **600**, respectively, of video encoding architectures.

Referring to embodiment **400** of FIG. **4**, as may be seen with respect to this diagram, an input video signal is received by a video encoder. In certain embodiments, the input video signal is composed of coding units (CUs) or macro-blocks (MBs). The size of such coding units or macro-blocks may be varied and can include a number of pixels typically arranged in a square shape. In one embodiment, such coding units or macro-blocks have a size of 16×16 pixels. However, it is generally noted that a macro-block may have any desired size such as N×N pixels, where N is an integer. Of course, some implementations may include non-square shaped coding units or macro-blocks, although square shaped coding units or macro-blocks are employed in a preferred embodiment.

The input video signal may generally be referred to as corresponding to raw frame (or picture) image data. For example, raw frame (or picture) image data may undergo processing to generate luma and chroma samples. In some embodiments, the set of luma samples in a macro-block is of one particular arrangement (e.g., 16×16), and set of the chroma samples is of a different particular arrangement (e.g., 8×8). In accordance with the embodiment depicted herein, a video encoder processes such samples on a block by block basis.

The input video signal then undergoes mode selection by which the input video signal selectively undergoes intra and/or inter-prediction processing. Generally speaking, the input video signal undergoes compression along a compression pathway. When operating with no feedback (e.g., in accordance with neither inter-prediction nor intra-prediction), the input video signal is provided via the compression pathway to undergo transform operations (e.g., in accordance with discrete cosine transform (DCT)). Of course, other transforms may be employed in alternative embodiments. In this mode of operation, the input video signal itself is that which is compressed. The compression pathway may take advantage of the lack of high frequency sensitivity of human eyes in performing the compression.

However, feedback may be employed along the compression pathway by selectively using inter- or intra-prediction video encoding. In accordance with a feedback or predictive mode of operation, the compression pathway operates on a (relatively low energy) residual (e.g., a difference) resulting from subtraction of a predicted value of a current macro-block from the current macro-block. Depending upon which form of prediction is employed in a given instance, a residual or difference between a current macro-block and a predicted value of that macro-block based on at least a portion of that same frame (or picture) or on at least a portion of at least one other frame (or picture) is generated.

The resulting modified video signal then undergoes transform operations along the compression pathway. In one embodiment, a discrete cosine transform (DCT) operates on a set of video samples (e.g., luma, chroma, residual, etc.) to compute respective coefficient values for each of a predetermined number of basis patterns. For example, one embodi-

ment includes 64 basis functions (e.g., such as for an 8×8 sample). Generally speaking, different embodiments may employ different numbers of basis functions (e.g., different transforms). Any combination of those respective basis functions, including appropriate and selective weighting thereof, may be used to represent a given set of video samples. Additional details related to various ways of performing transform operations are described in the technical literature associated with video encoding including those standards/draft standards that have been incorporated by reference as indicated above. The output from the transform processing includes such respective coefficient values. This output is provided to a quantizer.

Generally, most image blocks will typically yield coefficients (e.g., DCT coefficients in an embodiment operating in accordance with discrete cosine transform (DCT)) such that the most relevant DCT coefficients are of lower frequencies. Because of this and of the human eyes' relatively poor sensitivity to high frequency visual effects, a quantizer may be operable to convert most of the less relevant coefficients to a value of zero. That is to say, those coefficients whose relative contribution is below some predetermined value (e.g., some threshold) may be eliminated in accordance with the quantization process. A quantizer may also be operable to convert the significant coefficients into values that can be coded more efficiently than those that result from the transform process. For example, the quantization process may operate by dividing each respective coefficient by an integer value and discarding any remainder. Such a process, when operating on typical coding units or macro-blocks, typically yields a relatively low number of non-zero coefficients which are then delivered to an entropy encoder for lossless encoding and for use in accordance with a feedback path which may select intra-prediction and/or inter-prediction processing in accordance with video encoding.

An entropy encoder operates in accordance with a lossless compression encoding process. In comparison, the quantization operations are generally lossy. The entropy encoding process operates on the coefficients provided from the quantization process. Those coefficients may represent various characteristics (e.g., luma, chroma, residual, etc.). Various types of encoding may be employed by an entropy encoder. For example, context-adaptive binary arithmetic coding (CABAC) and/or context-adaptive variable-length coding (CAVLC) may be performed by the entropy encoder. For example, in accordance with at least one part of an entropy coding scheme, the data is converted to a (run, level) pairing (e.g., data 14, 3, 0, 4, 0, 0, −3 would be converted to the respective (run, level) pairs of (0, 14), (0, 3), (1, 4), (2, −3)). In advance, a table may be prepared that assigns variable length codes for value pairs, such that relatively shorter length codes are assigned to relatively common value pairs, and relatively longer length codes are assigned for relatively less common value pairs.

As the reader will understand, the operations of inverse quantization and inverse transform correspond to those of quantization and transform, respectively. For example, in an embodiment in which a DCT is employed within the transform operations, then an inverse DCT (IDCT) is that employed within the inverse transform operations.

A picture buffer, alternatively referred to as a digital picture buffer or a DPB, receives the signal from the IDCT module; the picture buffer is operative to store the current frame (or picture) and/or one or more other frames (or pictures) such as may be used in accordance with intra-prediction and/or inter-prediction operations as may be performed in accordance with video encoding. It is noted that in accordance with intra-prediction, a relatively small amount of storage may be sufficient, in that, it may not be necessary to store the current frame (or picture) or any other frame (or picture) within the frame (or picture) sequence. Such stored information may be employed for performing motion compensation and/or motion estimation in the case of performing inter-prediction in accordance with video encoding.

In one possible embodiment, for motion estimation, a respective set of luma samples (e.g., 16×16) from a current frame (or picture) are compared to respective buffered counterparts in other frames (or pictures) within the frame (or picture) sequence (e.g., in accordance with inter-prediction). In one possible implementation, a closest matching area is located (e.g., prediction reference) and a vector offset (e.g., motion vector) is produced. In a single frame (or picture), a number of motion vectors may be found and not all will necessarily point in the same direction. One or more operations as performed in accordance with motion estimation are operative to generate one or more motion vectors.

Motion compensation is operative to employ one or more motion vectors as may be generated in accordance with motion estimation. A prediction reference set of samples is identified and delivered for subtraction from the original input video signal in an effort hopefully to yield a relatively (e.g., ideally, much) lower energy residual. If such operations do not result in a yielded lower energy residual, motion compensation need not necessarily be performed and the transform operations may merely operate on the original input video signal instead of on a residual (e.g., in accordance with an operational mode in which the input video signal is provided straight through to the transform operation, such that neither intra-prediction nor inter-prediction are performed), or intra-prediction may be utilized and transform operations performed on the residual resulting from intra-prediction. Also, if the motion estimation and/or motion compensation operations are successful, the motion vector may also be sent to the entropy encoder along with the corresponding residual's coefficients for use in undergoing lossless entropy encoding.

The output from the overall video encoding operation is an output bit stream. It is noted that such an output bit stream may of course undergo certain processing in accordance with generating a continuous time signal which may be transmitted via a communication channel. For example, certain embodiments operate within wireless communication systems. In such an instance, an output bitstream may undergo appropriate digital to analog conversion, frequency conversion, scaling, filtering, modulation, symbol mapping, and/or any other operations within a wireless communication device that operate to generate a continuous time signal capable of being transmitted via a communication channel, etc.

Referring to embodiment **500** of FIG. **5**, as may be seen with respect to this diagram, an input video signal is received by a video encoder. In certain embodiments, the input video signal is composed of coding units or macro-blocks (and/or may be partitioned into coding units (CUs)). The size of such coding units or macro-blocks may be varied and can include a number of pixels typically arranged in a square shape. In one embodiment, such coding units or macro-blocks have a size of 16×16 pixels. However, it is generally noted that a macro-block may have any desired size such as N×N pixels, where N is an integer. Of course, some implementations may include non-square shaped coding units or macro-blocks, although square shaped coding units or macro-blocks are employed in a preferred embodiment.

The input video signal may generally be referred to as corresponding to raw frame (or picture) image data. For

example, raw frame (or picture) image data may undergo processing to generate luma and chroma samples. In some embodiments, the set of luma samples in a macro-block is of one particular arrangement (e.g., 16×16), and set of the chroma samples is of a different particular arrangement (e.g., 8×8). In accordance with the embodiment depicted herein, a video encoder processes such samples on a block by block basis.

The input video signal then undergoes mode selection by which the input video signal selectively undergoes intra and/or inter-prediction processing. Generally speaking, the input video signal undergoes compression along a compression pathway. When operating with no feedback (e.g., in accordance with neither inter-prediction nor intra-prediction), the input video signal is provided via the compression pathway to undergo transform operations (e.g., in accordance with discrete cosine transform (DCT)). Of course, other transforms may be employed in alternative embodiments. In this mode of operation, the input video signal itself is that which is compressed. The compression pathway may take advantage of the lack of high frequency sensitivity of human eyes in performing the compression.

However, feedback may be employed along the compression pathway by selectively using inter- or intra-prediction video encoding. In accordance with a feedback or predictive mode of operation, the compression pathway operates on a (relatively low energy) residual (e.g., a difference) resulting from subtraction of a predicted value of a current macro-block from the current macro-block. Depending upon which form of prediction is employed in a given instance, a residual or difference between a current macro-block and a predicted value of that macro-block based on at least a portion of that same frame (or picture) or on at least a portion of at least one other frame (or picture) is generated.

The resulting modified video signal then undergoes transform operations along the compression pathway. In one embodiment, a discrete cosine transform (DCT) operates on a set of video samples (e.g., luma, chroma, residual, etc.) to compute respective coefficient values for each of a predetermined number of basis patterns. For example, one embodiment includes 64 basis functions (e.g., such as for an 8×8 sample). Generally speaking, different embodiments may employ different numbers of basis functions (e.g., different transforms). Any combination of those respective basis functions, including appropriate and selective weighting thereof, may be used to represent a given set of video samples. Additional details related to various ways of performing transform operations are described in the technical literature associated with video encoding including those standards/draft standards that have been incorporated by reference as indicated above. The output from the transform processing includes such respective coefficient values. This output is provided to a quantizer.

Generally, most image blocks will typically yield coefficients (e.g., DCT coefficients in an embodiment operating in accordance with discrete cosine transform (DCT)) such that the most relevant DCT coefficients are of lower frequencies. Because of this and of the human eyes' relatively poor sensitivity to high frequency visual effects, a quantizer may be operable to convert most of the less relevant coefficients to a value of zero. That is to say, those coefficients whose relative contribution is below some predetermined value (e.g., some threshold) may be eliminated in accordance with the quantization process. A quantizer may also be operable to convert the significant coefficients into values that can be coded more efficiently than those that result from the transform process. For example, the quantization process may operate by dividing each respective coefficient by an integer value and discarding any remainder. Such a process, when operating on typical coding units or macro-blocks, typically yields a relatively low number of non-zero coefficients which are then delivered to an entropy encoder for lossless encoding and for use in accordance with a feedback path which may select intra-prediction and/or inter-prediction processing in accordance with video encoding.

An entropy encoder operates in accordance with a lossless compression encoding process. In comparison, the quantization operations are generally lossy. The entropy encoding process operates on the coefficients provided from the quantization process. Those coefficients may represent various characteristics (e.g., luma, chroma, residual, etc.). Various types of encoding may be employed by an entropy encoder. For example, context-adaptive binary arithmetic coding (CABAC) and/or context-adaptive variable-length coding (CAVLC) may be performed by the entropy encoder. For example, in accordance with at least one part of an entropy coding scheme, the data is converted to a (run, level) pairing (e.g., data 14, 3, 0, 4, 0, 0, −3 would be converted to the respective (run, level) pairs of (0, 14), (0, 3), (1, 4), (2, −3)). In advance, a table may be prepared that assigns variable length codes for value pairs, such that relatively shorter length codes are assigned to relatively common value pairs, and relatively longer length codes are assigned for relatively less common value pairs.

As the reader will understand, the operations of inverse quantization and inverse transform correspond to those of quantization and transform, respectively. For example, in an embodiment in which a DCT is employed within the transform operations, then an inverse DCT (IDCT) is that employed within the inverse transform operations.

In certain optional embodiments, the output from the deblocking filter is provided to one or more other in-loop filters (e.g., implemented in accordance with adaptive loop filter (ALF), sample adaptive offset (SAO) filter, and/or any other filter type) implemented to process the output from the inverse transform block.

For example, such an adaptive loop filter (ALF) may be implemented to process the output from the inverse transform block. Such an adaptive loop filter (ALF) is applied to the decoded picture before it is stored in a picture buffer (sometimes referred to as a DPB, digital picture buffer). The adaptive loop filter (ALF) is implemented to reduce coding noise of the decoded picture, and the filtering thereof may be selectively applied on a slice by slice basis, respectively, for luminance and chrominance whether or not the adaptive loop filter (ALF) is applied either at slice level or at block level. Two-dimensional 2-D finite impulse response (FIR) filtering may be used in application of the adaptive loop filter (ALF). The coefficients of the filters may be designed slice by slice at the encoder, and such information is then signaled to the decoder (e.g., signaled from a transmitter communication device including a video encoder [alternatively referred to as encoder] to a receiver communication device including a video decoder [alternatively referred to as decoder]).

One embodiment operates by generating the coefficients in accordance with Wiener filtering design. In addition, it may be applied on a block by block based at the encoder whether the filtering is performed and such a decision is then signaled to the decoder (e.g., signaled from a transmitter communication device including a video encoder [alternatively referred to as encoder] to a receiver communication device including a video decoder [alternatively referred to as decoder]) based on quadtree structure, where the block size is decided according to the rate-distortion optimization. It is noted that the

implementation of using such 2-D filtering may introduce a degree of complexity in accordance with both encoding and decoding. For example, by using 2-D filtering in accordance and implementation of an adaptive loop filter (ALF), there may be some increasing complexity within an encoder implemented within the transmitter communication device as well as within a decoder implemented within a receiver communication device.

With respect to one type of an in-loop filter, the use of an adaptive loop filter (ALF) can provide any of a number of improvements in accordance with such video processing, including an improvement on the objective quality measure by the peak to signal noise ratio (PSNR) that comes from performing random quantization noise removal. In addition, the subjective quality of a subsequently encoded video signal may be achieved from illumination compensation, which may be introduced in accordance with performing offset processing and scaling processing (e.g., in accordance with applying a gain) in accordance with adaptive loop filter (ALF) processing.

Receiving the signal output from the ALF is a picture buffer, alternatively referred to as a digital picture buffer or a DPB; the picture buffer is operative to store the current frame (or picture) and/or one or more other frames (or pictures) such as may be used in accordance with intra-prediction and/or inter-prediction operations as may be performed in accordance with video encoding. It is noted that in accordance with intra-prediction, a relatively small amount of storage may be sufficient, in that, it may not be necessary to store the current frame (or picture) or any other frame (or picture) within the frame (or picture) sequence. Such stored information may be employed for performing motion compensation and/or motion estimation in the case of performing inter-prediction in accordance with video encoding.

In one possible embodiment, for motion estimation, a respective set of luma samples (e.g., 16×16) from a current frame (or picture) are compared to respective buffered counterparts in other frames (or pictures) within the frame (or picture) sequence (e.g., in accordance with inter-prediction). In one possible implementation, a closest matching area is located (e.g., prediction reference) and a vector offset (e.g., motion vector) is produced. In a single frame (or picture), a number of motion vectors may be found and not all will necessarily point in the same direction. One or more operations as performed in accordance with motion estimation are operative to generate one or more motion vectors.

Motion compensation is operative to employ one or more motion vectors as may be generated in accordance with motion estimation. A prediction reference set of samples is identified and delivered for subtraction from the original input video signal in an effort hopefully to yield a relatively (e.g., ideally, much) lower energy residual. If such operations do not result in a yielded lower energy residual, motion compensation need not necessarily be performed and the transform operations may merely operate on the original input video signal instead of on a residual (e.g., in accordance with an operational mode in which the input video signal is provided straight through to the transform operation, such that neither intra-prediction nor inter-prediction are performed), or intra-prediction may be utilized and transform operations performed on the residual resulting from intra-prediction. Also, if the motion estimation and/or motion compensation operations are successful, the motion vector may also be sent to the entropy encoder along with the corresponding residual's coefficients for use in undergoing lossless entropy encoding.

The output from the overall video encoding operation is an output bit stream. It is noted that such an output bit stream may of course undergo certain processing in accordance with generating a continuous time signal which may be transmitted via a communication channel. For example, certain embodiments operate within wireless communication systems. In such an instance, an output bitstream may undergo appropriate digital to analog conversion, frequency conversion, scaling, filtering, modulation, symbol mapping, and/or any other operations within a wireless communication device that operate to generate a continuous time signal capable of being transmitted via a communication channel, etc.

Referring to embodiment **600** of FIG. **6**, with respect to this diagram depicting an alternative embodiment of a video encoder, such a video encoder carries out prediction, transform, and encoding processes to produce a compressed output bit stream. Such a video encoder may operate in accordance with and be compliant with one or more video encoding protocols, standards, and/or recommended practices such as ISO/IEC 14496-10—MPEG-4 Part 10, AVC (Advanced Video Coding), alternatively referred to as H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding), ITU H.264/MPEG4-AVC.

It is noted that a corresponding video decoder, such as located within a device at another end of a communication channel, is operative to perform the complementary processes of decoding, inverse transform, and reconstruction to produce a respective decoded video sequence that is (ideally) representative of the input video signal.

As may be seen with respect to this diagram, alternative arrangements and architectures may be employed for effectuating video encoding. Generally speaking, an encoder processes an input video signal (e.g., typically composed in units of coding units or macro-blocks, often times being square in shape and including N×N pixels therein). The video encoding determines a prediction of the current macro-block based on previously coded data. That previously coded data may come from the current frame (or picture) itself (e.g., such as in accordance with intra-prediction) or from one or more other frames (or pictures) that have already been coded (e.g., such as in accordance with inter-prediction). The video encoder subtracts the prediction of the current macro-block to form a residual.

Generally speaking, intra-prediction is operative to employ block sizes of one or more particular sizes (e.g., 16×16, 8×8, or 4×4) to predict a current macro-block from surrounding, previously coded pixels within the same frame (or picture). Generally speaking, inter-prediction is operative to employ a range of block sizes (e.g., 16×16 down to 4×4) to predict pixels in the current frame (or picture) from regions that are selected from within one or more previously coded frames (or pictures).

With respect to the transform and quantization operations, a block of residual samples may undergo transformation using a particular transform (e.g., 4×4 or 8×8). One possible embodiment of such a transform operates in accordance with discrete cosine transform (DCT). The transform operation outputs a group of coefficients such that each respective coefficient corresponds to a respective weighting value of one or more basis functions associated with a transform. After undergoing transformation, a block of transform coefficients is quantized (e.g., each respective coefficient may be divided by an integer value and any associated remainder may be discarded, or they may be multiplied by an integer value). The quantization process is generally inherently lossy, and it can reduce the precision of the transform coefficients according to a quantization parameter (QP). Typically, many of the coef-

ficients associated with a given macro-block are zero, and only some nonzero coefficients remain. Generally, a relatively high QP setting is operative to result in a greater proportion of zero-valued coefficients and smaller magnitudes of non-zero coefficients, resulting in relatively high compression (e.g., relatively lower coded bit rate) at the expense of relatively poorly decoded image quality; a relatively low QP setting is operative to allow more nonzero coefficients to remain after quantization and larger magnitudes of non-zero coefficients, resulting in relatively lower compression (e.g., relatively higher coded bit rate) with relatively better decoded image quality.

The video encoding process produces a number of values that are encoded to form the compressed bit stream. Examples of such values include the quantized transform coefficients, information to be employed by a decoder to re-create the appropriate prediction, information regarding the structure of the compressed data and compression tools employed during encoding, information regarding a complete video sequence, etc. Such values and/or parameters (e.g., syntax elements) may undergo encoding within an entropy encoder operating in accordance with CABAC, CAVLC, or some other entropy coding scheme, to produce an output bit stream that may be stored, transmitted (e.g., after undergoing appropriate processing to generate a continuous time signal that comports with a communication channel), etc.

In an embodiment operating using a feedback path, the output of the transform and quantization undergoes inverse quantization and inverse transform. One or both of intra-prediction and inter-prediction may be performed in accordance with video encoding. Also, motion compensation and/or motion estimation may be performed in accordance with such video encoding.

The signal path output from the inverse quantization and inverse transform (e.g., IDCT) block, which is provided to the intra-prediction block, is also provided to a de-blocking filter. The output from the de-blocking filter is provided to one or more other in-loop filters (e.g., implemented in accordance with adaptive loop filter (ALF), sample adaptive offset (SAO) filter, and/or any other filter type) implemented to process the output from the inverse transform block. For example, in one possible embodiment, an ALF is applied to the decoded picture before it is stored in a picture buffer (again, sometimes alternatively referred to as a DPB, digital picture buffer). The ALF is implemented to reduce coding noise of the decoded picture, and the filtering thereof may be selectively applied on a slice by slice basis, respectively, for luminance and chrominance whether or not the ALF is applied either at slice level or at block level. Two-dimensional 2-D finite impulse response (FIR) filtering may be used in application of the ALF. The coefficients of the filters may be designed slice by slice at the encoder, and such information is then signaled to the decoder (e.g., signaled from a transmitter communication device including a video encoder [alternatively referred to as encoder] to a receiver communication device including a video decoder [alternatively referred to as decoder]).

One embodiment generated the coefficients in accordance with Wiener filtering design. In addition, it may be applied on a block by block based at the encoder whether the filtering is performed and such a decision is then signaled to the decoder (e.g., signaled from a transmitter communication device including a video encoder [alternatively referred to as encoder] to a receiver communication device including a video decoder [alternatively referred to as decoder]) based on quadtree structure, where the block size is decided according to the rate-distortion optimization. It is noted that the implementation of using such 2-D filtering may introduce a degree of complexity in accordance with both encoding and decoding. For example, by using 2-D filtering in accordance and implementation of an ALF, there may be some increasing complexity within encoder implemented within the transmitter communication device as well as within a decoder implemented within a receiver communication device.

As mentioned with respect to other embodiments, the use of an ALF can provide any of a number of improvements in accordance with such video processing, including an improvement on the objective quality measure by the peak to signal noise ratio (PSNR) that comes from performing random quantization noise removal. In addition, the subjective quality of a subsequently encoded video signal may be achieved from illumination compensation, which may be introduced in accordance with performing offset processing and scaling processing (e.g., in accordance with applying a gain) in accordance with ALF processing.

With respect to any video encoder architecture implemented to generate an output bitstream, it is noted that such architectures may be implemented within any of a variety of communication devices. The output bitstream may undergo additional processing including error correction code (ECC), forward error correction (FEC), etc. thereby generating a modified output bitstream having additional redundancy deal therein. Also, as may be understood with respect to such a digital signal, it may undergo any appropriate processing in accordance with generating a continuous time signal suitable for or appropriate for transmission via a communication channel That is to say, such a video encoder architecture may be implemented within a communication device operative to perform transmission of one or more signals via one or more communication channels. Additional processing may be made on an output bitstream generated by such a video encoder architecture thereby generating a continuous time signal that may be launched into a communication channel.

FIG. **7** is a diagram illustrating an embodiment **700** of intra-prediction processing. As can be seen with respect to this diagram, a current block of video data (e.g., often times being square in shape and including generally N×N pixels) undergoes processing to estimate the respective pixels therein. Previously coded pixels located above and to the left of the current block are employed in accordance with such intra-prediction. From certain perspectives, an intra-prediction direction may be viewed as corresponding to a vector extending from a current pixel to a reference pixel located above or to the left of the current pixel. Details of intra-prediction as applied to coding in accordance with H.264/AVC are specified within the corresponding standard (e.g., International Telecommunication Union, ITU-T, TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, H.264 (March 2010), SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services—Coding of moving video, Advanced video coding for generic audiovisual services, Recommendation ITU-T H.264, also alternatively referred to as International Telecomm ISO/IEC 14496-10—MPEG-4 Part 10, AVC (Advanced Video Coding), H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding), ITU H.264/MPEG4-AVC, or equivalent) that is incorporated by reference above.

The residual, which is the difference between the current pixel and the reference or prediction pixel, is that which gets encoded. As can be seen with respect to this diagram, intra-prediction operates using pixels within a common frame (or picture). It is of course noted that a given pixel may have different respective components associated therewith, and there may be different respective sets of samples for each respective component.

FIG. **8** is a diagram illustrating an embodiment **800** of inter-prediction processing. In contradistinction to intra-prediction, inter-prediction is operative to identify a motion vector (e.g., an inter-prediction direction) based on a current set of pixels within a current frame (or picture) and one or more sets of reference or prediction pixels located within one or more other frames (or pictures) within a frame (or picture) sequence. As can be seen, the motion vector extends from the current frame (or picture) to another frame (or picture) within the frame (or picture) sequence. Inter-prediction may utilize sub-pixel interpolation, such that a prediction pixel value corresponds to a function of a plurality of pixels in a reference frame or picture.

A residual may be calculated in accordance with inter-prediction processing, though such a residual is different from the residual calculated in accordance with intra-prediction processing. In accordance with inter-prediction processing, the residual at each pixel again corresponds to the difference between a current pixel and a predicted pixel value. However, in accordance with inter-prediction processing, the current pixel and the reference or prediction pixel are not located within the same frame (or picture). While this diagram shows inter-prediction as being employed with respect to one or more previous frames or pictures, it is also noted that alternative embodiments may operate using references corresponding to frames before and/or after a current frame. For example, in accordance with appropriate buffering and/or memory management, a number of frames may be stored. When operating on a given frame, references may be generated from other frames that precede and/or follow that given frame.

Coupled with the CU, a basic unit may be employed for the prediction partition mode, namely, the prediction unit, or PU. It is also noted that the PU is defined only for the last depth CU, and its respective size is limited to that of the CU.

FIG. **9** and FIG. **10** are diagrams illustrating various embodiments **900** and **1000**, respectively, of video decoding architectures.

Generally speaking, such video decoding architectures operate on an input bitstream. It is of course noted that such an input bitstream may be generated from a signal that is received by a communication device from a communication channel. Various operations may be performed on a continuous time signal received from the communication channel, including digital sampling, demodulation, scaling, filtering, etc. such as may be appropriate in accordance with generating the input bitstream. Moreover, certain embodiments, in which one or more types of error correction code (ECC), forward error correction (FEC), etc. may be implemented, may perform appropriate decoding in accordance with such ECC, FEC, etc. thereby generating the input bitstream. That is to say, in certain embodiments in which additional redundancy may have been made in accordance with generating a corresponding output bitstream (e.g., such as may be launched from a transmitter communication device or from the transmitter portion of a transceiver communication device), appropriate processing may be performed in accordance with generating the input bitstream. Overall, such a video decoding architectures and lamented to process the input bitstream thereby generating an output video signal corresponding to the original input video signal, as closely as possible and perfectly in an ideal case, for use in being output to one or more video display capable devices.

Referring to the embodiment **900** of FIG. **9**, generally speaking, a decoder such as an entropy decoder (e.g., which may be implemented in accordance with CABAC, CAVLC, etc.) processes the input bitstream in accordance with performing the complementary process of encoding as performed within a video encoder architecture. The input bitstream may be viewed as being, as closely as possible and perfectly in an ideal case, the compressed output bitstream generated by a video encoder architecture. Of course, in a real-life application, it is possible that some errors may have been incurred in a signal transmitted via one or more communication links. The entropy decoder processes the input bitstream and extracts the appropriate coefficients, such as the DCT coefficients (e.g., such as representing chroma, luma, etc. information) and provides such coefficients to an inverse quantization and inverse transform block. In the event that a DCT transform is employed, the inverse quantization and inverse transform block may be implemented to perform an inverse DCT (IDCT) operation. Subsequently, A/D blocking filter is implemented to generate the respective frames and/or pictures corresponding to an output video signal. These frames and/or pictures may be provided into a picture buffer, or a digital picture buffer (DPB) for use in performing other operations including motion compensation. Generally speaking, such motion compensation operations may be viewed as corresponding to inter-prediction associated with video encoding. Also, intra-prediction may also be performed on the signal output from the inverse quantization and inverse transform block. Analogously as with respect to video encoding, such a video decoder architecture may be implemented to perform mode selection between performing it neither intra-prediction nor inter-prediction, inter-prediction, or intra-prediction in accordance with decoding an input bitstream thereby generating an output video signal.

Referring to the embodiment **1000** of FIG. **10**, in certain optional embodiments, one or more in-loop filters (e.g., implemented in accordance with adaptive loop filter (ALF), sample adaptive offset (SAO) filter, and/or any other filter type) such as may be implemented in accordance with video encoding as employed to generate an output bitstream, a corresponding one or more in-loop filters may be implemented within a video decoder architecture. In one embodiment, an appropriate implementation of one or more such in-loop filters is after the de-blocking filter.

FIG. **11** illustrates an embodiment **1100** of recursive coding unit (CU) structure. High efficiency video coding (HEVC) is a next-generation video compression standard currently under development. In the opinion of some in the industry, the HEVC standard looks to be a successor to H.264/MPEG4-AVC (alternatively referred to as AVC), as referred to and also incorporated by reference above. In accordance with video coding in accordance with the currently developing HEVC standard operates using a coding unit, or CU, as a basic unit which typically has a square shape and whose counterpart in accordance with AVC is a macro block (MB). From certain perspectives, the CU has a similar operational purpose and role to the MB, and the corresponding sub macro block (SMB) in AVC. However, at least one difference between the two respective components employed in accordance with video coding is that a CU may have any one of a number of various sizes, with no particular distinction corresponding to its respective size. In accordance with video coding terminology performed in accordance with the currently developing HEVC standard, to particular terms are defined with respect to CU, namely, the largest coding unit (LCU) and the smallest content unit (SCU).

In accordance with video coding, a picture may be represented using a number of non-overlapped LCUs. Since the CU is assumed to be is restricted to be square in shape, the CU structure within an LCU can be expressed in a recursive tree representation as depicted within FIG. **11**. That is to say, a CU

may be characterized by a corresponding LCU size and the hierarchical depth in the LCU to which that LCU belongs.

Once the tree splitting process is done as detected within FIG. **11**, different respective prediction encoding approaches are specified for every CU which is not further split. In other words, the CU hierarchical tree may be viewed as including a number of respective leaf nodes each corresponding to the respective tree splitting process as applied to the respective CU's corresponding to a picture. In accordance with video coding and specifically with respect to P and B slices, each respective CU may use either intra-prediction or inter-prediction in accordance with video encoding. For example, as is described in respect to the subsequent paragraphs and the respective FIG. **7** and FIG. **8**, inter-prediction coding is operative to employ references from neighboring frames, while intra-prediction coding is operative to employ references from the spatial neighboring pixels or co-located color components.

FIG. **12** illustrates an embodiment **1200** of prediction unit (PU) modes. FIG. **12** particularly shows the PU partition modes respectively for a CU having size of 2N×2N. The N×N PU is different from other PU modes, and it only exists with respect to the SCU, or the smallest CU.

In accordance with the currently developing HEVC standard, there are different respective slice types employed, namely, I slice(s), P slice(s), and B slice(s). Generally speaking, I frames operate such that processing, such as in accordance with prediction, is only with respect to that particular frame (e.g., and I frame only performs prediction with respect to self).

P slices operate by using only a single reference list rather than to respective reference lists. Generally speaking, with respect to P slices, prediction, such as in accordance with motion compensation, is performed in accordance with only one direction or unidirectional with respect to the frame sequence. Such prediction in accordance with P slices may be in either direction but only one direction is employed at a given time. Also, prediction accordance with P slices is performed with respect to only one frame at a time, again, in either direction of the frame sequence.

Generally speaking, with respect to B slices, prediction may be performed in accordance with both directions along the frame sequence. For example, prediction, such as in accordance of motion compensation, can be employed in both directions of the frame sequence simultaneously such that information from at least two respective other frames, besides the current frame, may be used in accordance with such motion compensation. For example, consider a current frame **2** interposed between a preceding frame **1** and a following frame **3**: the preceding frame **1** may be viewed as being in the past, whereas the following frame **3** may be viewed as being in the future or before the current frame **2** (e.g., with respect to the frame sequence). Information may be taken from either that preceding frame **1** and/or the following frame **3** for processing the current frame **2**. Also, certain interpolated information or blended information may be realized from those two other respective frames in accordance with processing the current frame **2** (e.g., information from the preceding frame **1** may be interpolated or blended with information from the following frame **3**). Also, with respect to B slices, B slices may employ to respective reference lists. In accordance with operation using B slices, CU prediction mode in PU partition mode are encoded jointly by using a single syntax element (e.g., such as shown with respect to FIG. **15**). The binarization employed as a function of whether or not the current CU is the smallest CU, SCU, or not. For example, if the smallest CU, or SCU, is not employed, then the binarization and may be employed using the modified binary tree described below with respect to FIG. **13**.

With respect to the PU mode being employed, that information is signaled to a decoder so that it may appropriately know how a signal block has been partitioned and to perform appropriate processing of a video signal encoded in accordance with that particular PU mode. For example, depending upon the particular PU mode employed in accordance with encoding of a video signal, that information is provided to a decoder so that the decoder may appropriately process and decode the received video signal.

FIG. **13** illustrates an embodiment **1300** of a binary tree, including a modification thereof, employed for P slice encoding in one implementation and for both P and B slice encoding in another implementation. As can be seen with respect to this diagram, a binary tree representing that binarization and of PU partition symbols is illustrated. For example, a partition corresponding to 2N×2N uses a codeword of the single digit, namely, 1. A partition corresponding to 2N×N uses a codeword of two respective digits, namely, 01. A partition corresponding to 2N×N uses a codeword of three respective digits, namely, 001, and a partition corresponding to N×N uses a codeword of three respective digits, namely, 000. It is noted that such encoding may be viewed as being implemented in accordance with Huffman coding, such that those values having correspondingly lower probability are provided relatively longer codewords, while those values having correspondingly higher probability are provided relatively shorter codewords. As may be understood, a system operating in accordance with such Huffman coding may provide for efficiency in terms of employing relatively shorter codewords for those values occurring more frequently and with greater predominance.

In certain implementations, the partition corresponding to N×N may not be specifically implemented, and as such, the codebook associated with representing Ms. binary tree may be modified as including only three respective entries. For example, in such a modified binary tree, a partition corresponding to 2N×2N would use a codeword of the single digit, namely, 1. Also, a partition corresponding to 2N×N would use a codeword of two respective digits, namely, 01. A partition corresponding to 2N×N would use a codeword of two respective digits, 00. As may be seen, by employing such a modified binary tree, a more efficiently implemented Huffman codebook may be realized. Again, as described also above, with respect to B slices, B slices may employ to respective reference lists. In accordance with operation using B slices, CU prediction mode in PU partition mode are encoded jointly by using a single syntax element (e.g., such as shown with respect to FIG. **11**). The binarization employed as a function of whether or not the current CU is the smallest CU, SCU, or not. For example, if the smallest CU, or SCU, is not employed, then the binarization and may be employed using the modified binary tree described with respect to FIG. **13**.

In an alternative embodiment, the binary tree as pictorially illustrated with respect to FIG. **13** is employed for both B slices and P slices. That is to say, the very same binary tree is employed for both B slices and P slices in such an implementation. As may be understood with respect to such an embodiment, a very efficient implementation may be achieved by using the very same binary tree for both B slices and P slices, in that, only a singular codebook may be employed for both processing of the B slices and P slices. In other words, B slice encoding and P slice encoding may both be performed using the binary tree as pictorially illustrated with respect to FIG. **13** (e.g., the same binary tree is employed for both B slices and P slices) in certain embodiments. Stated another way, the

binary tree of FIG. **13**, which may be viewed as being intended primarily for use in P slice encoding in one embodiment may, in an alternative embodiment, be employed for both B slice encoding and P slice encoding.

In such an embodiment, separate respective syntax elements may be employed for the CU prediction mode (whether performing inter-prediction or intra-prediction) and the PU partition mode. For example, a first syntax element may be employed for indicating intra-prediction processing or inter-prediction processing, and a second syntax element may be employed for indicating CU partition shape (e.g., indicating how to perform CU splitting and partitioning).

FIG. **14** illustrates an embodiment **1400** of a binary tree as may be employed for B slice encoding in one implementation and for both P and B slice encoding in another implementation. As also described above, with respect to B slices, B slices may employ to respective reference lists. In accordance with operation using B slices, CU prediction mode in PU partition mode are encoded jointly by using a single syntax element (e.g., as shown within FIG. **14**).

In one embodiment, the binary tree as pictorially illustrated with respect to FIG. **14** is employed for B slices, and the modified binary tree pictorially illustrated in the right-hand portion of FIG. **13** as well as in FIG. **15** is employed for P slices. As may be understood, within such an embodiment, by utilizing the modified binary tree (e.g., right-hand portion of FIG. **13** and in FIG. **15**) being a relatively more efficiently implemented binary tree when compared to the binary tree of PU partitions as illustrated on the left-hand side of FIG. **13**, a more efficiently implemented video coding may be achieved with relatively less overhead. As may be understood, with respect to this embodiment, two different and respective codebooks are employed respectively for B slices (e.g., right-hand portion of FIG. **13** and in FIG. **15**) and P slices (e.g., FIG. **14**).

In an alternative embodiment, the binary tree as pictorially illustrated with respect to FIG. **14** is employed for both B slices and P slices. That is to say, the very same binary tree is employed for both B slices and P slices in such an implementation. As may be understood with respect to such an embodiment, a very efficient implementation may be achieved by using the very same binary tree for both B slices and P slices, in that, only a singular codebook may be employed for both processing of the B slices and P slices. In other words, B slice encoding and P slice encoding may both be performed using the binary tree as pictorially illustrated with respect to FIG. **14** (e.g., the same binary tree is employed for both B slices and P slices) in certain embodiments. Stated another way, the binary tree of FIG. **14**, which may be viewed as being intended primarily for use in B slice encoding in one embodiment may, in an alternative embodiment, be employed for both B slice encoding and P slice encoding.

FIG. **15** illustrates an embodiment **1500** of a binary tree as may be employed for P slice encoding in conjunction with B slice encoding as performed in accordance with FIG. **9**. For ease of illustration to the reader, the modified binary tree as depicted in the right-hand portion of FIG. **13** is again pictorially illustrated with respect to this FIG. **15**. Such a modified binary tree may be employed with respect to an embodiment that uses two different and respective codebooks are employed respectively for B slices (e.g., right-hand portion of FIG. **13** and in FIG. **15**) and P slices (e.g., FIG. **14**).

As may be understood with respect to the various embodiments and/or diagrams depicted herein, since the partition corresponding to N×N is allowed only for the smallest CU, or SCU's, that respective partition is not needed for those particular CU's at the root level and at the intermediate levels, respectively. With respect to P slices, the codeword assignment (or binary tree architecture/design) employed for that particular partition corresponding to N×N and the partition sizes that fall into that same branch as N×N (e.g., as may be seen with respect to the partition corresponding to N×2N, within FIG. **13**) should be level dependent. If the current CU is not specifically the smallest CU, or SCU, then the modified binary tree as pictorially illustrated in the right-hand portion of FIG. **13** and also FIG. **15** should appropriately and effectively come up with the codewords of the different respective PU partitions. Therefore, the binary tree pictorially illustrated on the left-hand side of FIG. **13** need necessarily be used for the smallest CU, or SCU, in P slices only. That is to say, for other processing not specifically operating on P slices, the binary tree pictorially illustrated on the left-hand side of FIG. **13** may be efficiently reduced and modified thereby generating the modified binary tree as pictorially are illustrated in the right-hand portion of FIG. **13** and also FIG. **15**.

Also, as also mentioned above, in accordance with the currently developing HEVC standard, both B slices and P slices respectively used different respective codewords for the different PU modes. In accordance with operation on P slices, the CU prediction mode (whether performing inter-prediction or intra-prediction) and the PU partition mode are encoded separate syntax elements. However, in accordance with operation on B slices, the CU prediction mode and the PU partition mode are encoded jointly by using a single syntax element. In certain situations, this disparate processing, depending upon whether P slices or be slices are being operated on, can create an extra burden for syntax parsing. As such, is also described above, one possible embodiment may be implemented using the binary tree, as pictorially illustrated with respect to FIG. **14**, to be used for both B slices and P slices. That is to say, the very same binary tree is employed for both B slices and P slices in such an implementation. As may be understood with respect to such an embodiment, a very efficient implementation may be achieved by using the very same binary tree for both B slices and P slices, in that, only a singular codebook may be employed for both processing of the slices and P slices. For example, such an embodiment but operate by using the very same binary tree to unify processing of both B slices and P slices in this particular case and also to use the same approach with respect to signal CU prediction mode and PU partition mode. One possible embodiment may be implemented such that the CU prediction mode (whether performing inter-prediction or intra-prediction) and the PU partition mode are respectively encoded in separate syntax elements for both P slices and B slices. Another possible embodiment may be implemented such that the CU prediction mode (whether performing inter-prediction or intra-prediction) and the PU partition mode are jointly encoded in a single syntax element for both P slices and B slices.

As may be understood with respect to the various embodiments and/or diagrams herein, a variety of different implementations may be employed as may be desired within a particular application. In some instances, a unification of the processing for both P slices and B slices is effectuated using the common binary tree as pictorially illustrated with respect to FIG. **14**. In other instances, when two separately implemented and different respective binary trees and codebooks are employed respectively for B slices and P slices, and particularly when the smallest CU, or SCU, is not employed (e.g., the current CU is not the SCU), then a modified binary tree such as pictorially illustrated with respect to the right-hand side of FIG. **13** and in FIG. **15** may be employed for P slices while the binary tree as pictorially illustrated with respect to FIG. **14** may be employed for B slices.

FIG. **16**A, FIG. **16**B, FIG. **17**A, and FIG. **17**B, illustrate various embodiments of methods performed in accordance with video coding (e.g., within one or more communication devices).

Referring to method **1600** of FIG. **16**A, the method **1600** begins by operating a video encoder to encode an input video signal to generate an output bitstream, as shown in a block **1610**. The method **1600** continues by employing a single binary tree within the video encoder to process at least one P slice and at least one B slice in accordance with generating the output bitstream, as shown in a block **1620**.

Referring to method **1601** of FIG. **16**B, the method **1601** begins by operating a video encoder to encode an input video signal to generate an output bitstream, as shown in a block **1611**.

The method **1601** then operates by employing a single binary tree within the video encoder to encode jointly coding unit (CU) prediction and prediction unit (PU) partition mode in a single syntax element for both the at least one P slice and the at least one B slice in accordance with generating the output bitstream, as shown in a block **1621**.

Referring to method **1700** of FIG. **17**A, the method **1700** begins by operating a video encoder to encode an input video signal to generate an output bitstream, as shown in a block **1710**. The method **1700** continues by employing a single binary tree within the video encoder to encode coding unit (CU) prediction in a first syntax element for both the at least one P slice and the at least one B slice in accordance with generating the output bitstream, as shown in a block **1720**.

The method **1700** then operates by employing the single binary tree within the video encoder to encode prediction unit (PU) partition mode in a second syntax element for both the at least one P slice and the at least one B slice in accordance with generating the output bitstream, as shown in a block **1730**. As mentioned elsewhere herein with respect to other embodiments and/or diagrams, separate respective syntax elements may be employed for the CU prediction mode (whether performing inter-prediction or intra-prediction) and the PU partition mode. For example, a first syntax element may be employed for indicating intra-prediction processing or inter-prediction processing, and a second syntax element may be employed for indicating CU partition shape (e.g., indicating how to perform CU splitting and partitioning).

Referring to method **1701** of FIG. **17**A, the method **1701** begins by operating a video encoder to encode an input video signal to generate an output bitstream, as shown in a block **1711**.

The method **1701** continues by transmitting the output bitstream, or a signal generated from or corresponding to the output bitstream, to a video decoder via a communication channel, as shown in a block **1721**. For example, at the transmitter end of the communication channel, within some embodiments, any of a variety of operations (e.g., any desired analog and/or digital operations including filtering, scaling, frequency shifting, frequency conversion, rounding, digital to analog conversion, etc.) may be made with respect to the output bitstream to generate a signal that comports with and is suitable for transmission via a communication channel. That is to say, the output bitstream may undergo any of a number of transmitter front-end processing operations (e.g., such as within an analog front end (AFE)) including modulation, continuous-time signal generation, etc. to generate a signal that actually gets transmitted via the communication channel. In other embodiments, the output bitstream itself, without modification, may in fact be transmitted via the communication channel.

The method **1701** then operates by operating the video decoder to decode an input bitstream (e.g., output bitstream, or the signal generated from or corresponding to the output bitstream, after having traversed the communication channel) to generate an output video signal (e.g., re-generated version of the input video signal), as shown in a block **1731**. For example, analogously, at the receiving end of the communication channel, within some embodiments, it is noted that any additional processing may be performed on the received signal (e.g., any desired analog and/or digital operations including filtering, scaling, frequency shifting, frequency conversion, rounding, analog to digital conversion, etc.) to generate the signal (e.g., input bitstream) that undergoes video decoding. That is to say, the actual signal received from the communication channel may undergo any of a number of receiver front-end processing operations (e.g., such as within an analog front end (AFE)) including demodulation, etc. to generate the signal (e.g., input bitstream) that undergoes video decoding. Of course, in certain optional embodiments, when the output bitstream itself, without modification, is in fact be transmitted via the communication channel, no such receiver front-end processing operations may be required.

It is also noted that the various operations and functions as described with respect to various methods herein may be performed within a communication device, such as using a baseband processing module and/or a processing module implemented therein and/or other component(s) therein.

As may be used herein, the terms "substantially" and "approximately" provides an industry-accepted tolerance for its corresponding term and/or relativity between items. Such an industry-accepted tolerance ranges from less than one percent to fifty percent and corresponds to, but is not limited to, component values, integrated circuit process variations, temperature variations, rise and fall times, and/or thermal noise. Such relativity between items ranges from a difference of a few percent to magnitude differences. As may also be used herein, the term(s) "operably coupled to", "coupled to", and/or "coupling" includes direct coupling between items and/or indirect coupling between items via an intervening item (e.g., an item includes, but is not limited to, a component, an element, a circuit, and/or a module) where, for indirect coupling, the intervening item does not modify the information of a signal but may adjust its current level, voltage level, and/or power level. As may further be used herein, inferred coupling (i.e., where one element is coupled to another element by inference) includes direct and indirect coupling between two items in the same manner as "coupled to". As may even further be used herein, the term "operable to" or "operably coupled to" indicates that an item includes one or more of power connections, input(s), output(s), etc., to perform, when activated, one or more its corresponding functions and may further include inferred coupling to one or more other items. As may still further be used herein, the term "associated with", includes direct and/or indirect coupling of separate items and/or one item being embedded within another item. As may be used herein, the term "compares favorably", indicates that a comparison between two or more items, signals, etc., provides a desired relationship. For example, when the desired relationship is that signal **1** has a greater magnitude than signal **2**, a favorable comparison may be achieved when the magnitude of signal **1** is greater than that of signal **2** or when the magnitude of signal **2** is less than that of signal **1**.

As may also be used herein, the terms "processing module", "module", "processing circuit", and/or "processing unit" (e.g., including various modules and/or circuitries such as may be operative, implemented, and/or for encoding, for

decoding, for baseband processing, etc.) may be a single processing device or a plurality of processing devices. Such a processing device may be a microprocessor, micro-controller, digital signal processor, microcomputer, central processing unit, field programmable gate array, programmable logic device, state machine, logic circuitry, analog circuitry, digital circuitry, and/or any device that manipulates signals (analog and/or digital) based on hard coding of the circuitry and/or operational instructions. The processing module, module, processing circuit, and/or processing unit may have an associated memory and/or an integrated memory element, which may be a single memory device, a plurality of memory devices, and/or embedded circuitry of the processing module, module, processing circuit, and/or processing unit. Such a memory device may be a read-only memory (ROM), random access memory (RAM), volatile memory, non-volatile memory, static memory, dynamic memory, flash memory, cache memory, and/or any device that stores digital information. Note that if the processing module, module, processing circuit, and/or processing unit includes more than one processing device, the processing devices may be centrally located (e.g., directly coupled together via a wired and/or wireless bus structure) or may be distributedly located (e.g., cloud computing via indirect coupling via a local area network and/or a wide area network). Further note that if the processing module, module, processing circuit, and/or processing unit implements one or more of its functions via a state machine, analog circuitry, digital circuitry, and/or logic circuitry, the memory and/or memory element storing the corresponding operational instructions may be embedded within, or external to, the circuitry comprising the state machine, analog circuitry, digital circuitry, and/or logic circuitry. Still further note that, the memory element may store, and the processing module, module, processing circuit, and/or processing unit executes, hard coded and/or operational instructions corresponding to at least some of the steps and/or functions illustrated in one or more of the Figures. Such a memory device or memory element can be included in an article of manufacture.

The present invention has been described above with the aid of method steps illustrating the performance of specified functions and relationships thereof. The boundaries and sequence of these functional building blocks and method steps have been arbitrarily defined herein for convenience of description. Alternate boundaries and sequences can be defined so long as the specified functions and relationships are appropriately performed. Any such alternate boundaries or sequences are thus within the scope and spirit of the claimed invention. Further, the boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain significant functions are appropriately performed. Similarly, flow diagram blocks may also have been arbitrarily defined herein to illustrate certain significant functionality. To the extent used, the flow diagram block boundaries and sequence could have been defined otherwise and still perform the certain significant functionality. Such alternate definitions of both functional building blocks and flow diagram blocks and sequences are thus within the scope and spirit of the claimed invention. One of average skill in the art will also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, can be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

The present invention may have also been described, at least in part, in terms of one or more embodiments. An embodiment of the present invention is used herein to illustrate the present invention, an aspect thereof, a feature thereof, a concept thereof, and/or an example thereof. A physical embodiment of an apparatus, an article of manufacture, a machine, and/or of a process that embodies the present invention may include one or more of the aspects, features, concepts, examples, etc. described with reference to one or more of the embodiments discussed herein. Further, from figure to figure, the embodiments may incorporate the same or similarly named functions, steps, modules, etc. that may use the same or different reference numbers and, as such, the functions, steps, modules, etc. may be the same or similar functions, steps, modules, etc. or different ones.

Unless specifically stated to the contra, signals to, from, and/or between elements in a figure of any of the figures presented herein may be analog or digital, continuous time or discrete time, and single-ended or differential. For instance, if a signal path is shown as a single-ended path, it also represents a differential signal path. Similarly, if a signal path is shown as a differential path, it also represents a single-ended signal path. While one or more particular architectures are described herein, other architectures can likewise be implemented that use one or more data buses not expressly shown, direct connectivity between elements, and/or indirect coupling between other elements as recognized by one of average skill in the art.

The term "module" is used in the description of the various embodiments of the present invention. A module includes a functional block that is implemented via hardware to perform one or module functions such as the processing of one or more input signals to produce one or more output signals. The hardware that implements the module may itself operate in conjunction software, and/or firmware. As used herein, a module may contain one or more sub-modules that themselves are modules.

While particular combinations of various functions and features of the present invention have been expressly described herein, other combinations of these features and functions are likewise possible. The present invention is not limited by the particular examples disclosed herein and expressly incorporates these other combinations.

What is claimed is:

1. A video processing device comprising:

a video encoder configured to:

encode an input video signal to generate an output bitstream;

employ a single binary tree when processing at least one P slice and at least one B slice to generate the output bitstream, wherein the at least one P slice is used for unidirectional prediction forward or behind in at least one frame sequence, and wherein the at least one B slice is used for bidirectional prediction both forward and behind in the at least one frame sequence;

employ the single binary tree to encode coding unit (CU) prediction based on a selected CU that is selected from a plurality of CUs when generating a first syntax element for both the at least one P slice and the at least one B slice that undergo entropy encoding to generate the output bitstream, wherein the first syntax element specifies intra-prediction processing or inter-prediction processing for the selected CU; and

employ the single binary tree to encode prediction unit (PU) partition mode based on the selected CU when generating a second syntax element for both the at least one P slice and the at least one B slice that undergo the entropy encoding to generate to generate the output bitstream, wherein the second syntax ele-

ment specifies the PU partition mode for the selected CU, wherein the PU partition mode is based on a size N×N PU when the selected CU is a smallest CU (SCU) of the plurality of CUs and is based on a different size PU than the size N×N PU when the selected CU is another CU than the SCU of the plurality of CUs, wherein N is a positive integer.

2. The video processing device of claim **1**, wherein the PU partition mode is a size 2N×2N PU, a size N×2N PU, or a size 2N×N PU when the selected CU is the another CU than the SCU of the plurality of CUs.

3. The video processing device of claim **1**, wherein the video encoder is further configured to:

perform intra-prediction processing at or during a first time to generate the output bitstream; and

perform inter-prediction processing at or during a second time to generate the output bitstream.

4. The video processing device of claim **1**, wherein:

the video processing device is a first communication device; and further comprising:

a second communication device, in communication with the first communication device via at least one communication channel, including:

an input configured to receive the output bitstream; and

a video decoder configured to decode the output bitstream to generate an output video signal corresponding to the input video signal; and wherein:

the second communication device is at least one of computer, a laptop computer, a high definition (HD) television, a standard definition (SD) television, a handheld media unit, a set top box (STB), or a digital video disc (DVD) player.

5. The video processing device of claim **1** further comprising:

a communication device that is operative within at least one of a satellite communication system, a wireless communication system, a wired communication system, a fiber-optic communication system, or a mobile communication system.

6. A video processing device comprising:

a video encoder configured to:

encode an input video signal to generate an output bitstream; and

employ a single binary tree when processing at least one P slice and at least one B slice when generating a first syntax element and a second syntax element that undergo entropy encoding to generate the output bitstream, wherein the at least one P slice is used for unidirectional prediction forward or behind in at least one frame sequence, and wherein the at least one B slice is used for bidirectional prediction both forward and behind in the at least one frame sequence, wherein the first syntax element specifies intra-prediction processing or inter-prediction processing for a selected coding unit (CU) that is selected from a plurality of CUs, wherein the second syntax element specifies a prediction unit (PU) partition mode for the selected CU, wherein the PU partition mode is based on a size N×N PU when the selected CU is a smallest CU (SCU) of the plurality of CUs and is based on a different size PU than the size N×N PU when the selected CU is another CU than the SCU of the plurality of CUs, wherein N is a positive integer.

7. The video processing device of claim **6**, wherein the video encoder is further configured to:

employ the single binary tree to encode jointly CU prediction and PU partition mode in a single syntax element for both the at least one P slice and the at least one B slice to generate the output bitstream.

8. The video processing device of claim **6**, wherein the video encoder is further configured to:

employ the single binary tree to encode CU prediction based on the selected CU when generating the first syntax element for both the at least one P slice and the at least one B slice to generate the output bitstream; and

employ the single binary tree to encode PU partition mode based on the selected CU when generating the second syntax element for both the at least one P slice and the at least one B slice to generate the output bitstream.

9. The video processing device of claim **6**, herein the PU partition mode is a size 2N×2N PU, a size N×2N PU, or a size 2N×N PU when the selected CU is the another CU than the SCU of the plurality of CUs.

10. The video processing device of claim **6**, wherein the video encoder is further configured to:

perform intra-prediction processing at or during a first time to generate the output bitstream; and

perform inter-prediction processing at or during a second time to generate the output bitstream.

11. The video processing device of claim **6**, wherein:

the video processing device is a first communication device; and further comprising:

a second communication device, in communication with the first communication device via at least one communication channel, including:

an input to receive the output bitstream; and

a video decoder to decode the output bitstream to generate an output video signal corresponding to the input video signal.

12. The video processing device of claim **11**, wherein:

the second communication device is at least one of computer, a laptop computer, a high definition (HD) television, a standard definition (SD) television, a handheld media unit, a set top box (STB), or a digital video disc (DVD) player.

13. The video processing device of claim **6** further comprising:

a communication device that is operative within at least one of a satellite communication system, a wireless communication system, a wired communication system, a fiber-optic communication system, or a mobile communication system.

14. A method for operating a video processing device of a communication device, the method comprising:

operating a video encoder of the video processing device to encode an input video signal to generate an output bitstream; and

employing a single binary tree within the video encoder to process at least one P slice and at least one B slice when generating a first syntax element and a second syntax element that undergo entropy encoding to generate the output bitstream, wherein the at least one P slice is used for unidirectional prediction forward or behind in at least one frame sequence, and wherein the at least one B slice is used for bidirectional prediction both forward and behind in the at least one frame sequence, wherein the first syntax element specifies intra-prediction processing or inter-prediction processing for a selected coding unit (CU) that is selected from a plurality of CUs, wherein the second syntax element specifies a prediction unit (PU) partition mode for the selected CU, wherein the PU partition mode is based on a size N×N PU when the selected CU is a smallest CU (SCU) of the plurality of CUs and is based on a different size PU than the size

N×N PU when the selected CU is another CU than the SCU of the plurality of CUs, wherein N is a positive integer.

15. The method of claim **14** further comprising:

employing the single binary tree within the video encoder to encode jointly CU prediction and PU partition mode in a single syntax element for both the at least one P slice and the at least one B slice when generating the output bitstream.

16. The method of claim **14** further comprising:

employing the single binary tree to encode CU prediction based on the selected CU when generating the first syntax element for both the at least one P slice and the at least one B slice when generating the output bitstream; and

employing the single binary tree to encode PU partition mode based on the selected CU when generating the second syntax element for both the at least one P slice and the at least one B slice when generating the output bitstream.

17. The method of claim **14**, herein the PU partition mode is a size 2N×2N PU, a size N×2N PU, or a size 2N×N PU when the selected CU is the another CU than the SCU of the plurality of CUs.

18. The method of claim **14** further comprising:

performing intra-prediction processing at or during a first time when generating the output bitstream; and

performing inter-prediction processing at or during a second time when generating the output bitstream.

19. The method of claim **14** further comprising:

operating an additional communication device, in communication with the communication device via at least one communication channel, by:

receiving the output bitstream; and

operating a video decoder to decode the output bitstream to generate an output video signal corresponding to the input video signal, wherein the additional communication device is at least one of computer, a laptop computer, a high definition (HD) television, a standard definition (SD) television, a handheld media unit, a set top box (STB), or a digital video disc (DVD) player.

20. The method of claim **14**, wherein the communication device is operative within at least one of a satellite communication system, a wireless communication system, a wired communication system, a fiber-optic communication system, or a mobile communication system.

* * * * *

Exhibit 5

U.S. Patent No. 8,572,138

US008572138B2

(12) **United States Patent**
Sundar et al.

(10) **Patent No.: US 8,572,138 B2**
(45) **Date of Patent: Oct. 29, 2013**

(54) **DISTRIBUTED COMPUTING SYSTEM HAVING AUTONOMIC DEPLOYMENT OF VIRTUAL MACHINE DISK IMAGES**

(75) Inventors: **Jagane Sundar**, Saratoga, CA (US); **Sanjay Radia**, Fremont, CA (US); **David A. Henseler**, Maplewood, MN (US)

(73) Assignee: **CA, Inc.**, Islandia, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1339 days.

(21) Appl. No.: **11/694,483**

(22) Filed: **Mar. 30, 2007**

(65) **Prior Publication Data**

US 2007/0233698 A1 Oct. 4, 2007

Related U.S. Application Data

(60) Provisional application No. 60/787,280, filed on Mar. 30, 2006.

(51) **Int. Cl.**
G06F 12/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
USPC **707/828**; 707/831; 718/1

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,256,637	B1 *	7/2001	Venkatesh et al.	1/1
6,513,059	B1 *	1/2003	Gupta et al.	709/202
6,775,829	B1	8/2004	Kroening	
6,865,737	B1	3/2005	Lucas et al.	
7,093,239	B1 *	8/2006	van der Made	717/135
7,246,351	B2 *	7/2007	Bloch et al.	717/175
2002/0129129	A1 *	9/2002	Bloch et al.	709/220
2002/0156877	A1	10/2002	Lu et al.	
2003/0074360	A1 *	4/2003	Chen et al.	707/100
2003/0078958	A1 *	4/2003	Pace et al.	709/201
2003/0110173	A1 *	6/2003	Marsland	707/10
2003/0126265	A1	7/2003	Aziz et al.	

(Continued)

FOREIGN PATENT DOCUMENTS

WO	WO/03/085526	A1	10/2003
WO	WO/2006/081503	A1	8/2006

(Continued)

OTHER PUBLICATIONS

E.N. Herness et al., "WebSphere Application Server: A Foundation for on Demand Computing," IBM Systems Journal IBM, vol. 43, No. 2, pp. 213-237, XP-002383791, 2004.

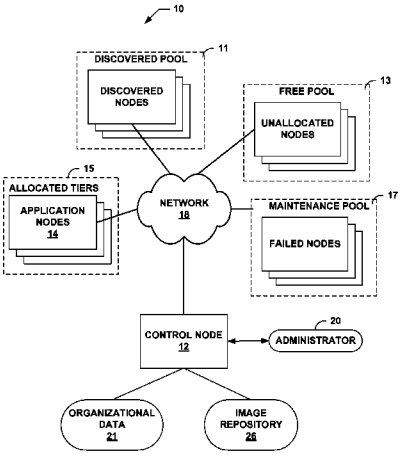
(Continued)

Primary Examiner — Angelica Ruiz
(74) *Attorney, Agent, or Firm* — Baker Botts, L.L.P.

(57) **ABSTRACT**

One or more control nodes provide for the efficient and automated allocation and management of computing functions and resources within the distributed computing system. The distributed computing system includes a software image repository storing: (i) one or more image instances of a virtual machine manager that is executable on the application nodes, wherein when executed on the applications nodes, the image instances of the virtual machine manager provide one or more virtual machines, and (ii) one or more image instances of one or more software applications that are executable on the virtual machines. The distributed computing system also includes a control node that comprises an automation infrastructure to provide autonomic deployment of the image instances of the virtual machine manager on the application nodes and to provide autonomic deployment of the image instances of the software applications on the virtual machines.

26 Claims, 29 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2003/0135509	A1*	7/2003	Davis et al.	707/100
2003/0135658	A1*	7/2003	Haggar et al.	709/312
2003/0140282	A1	7/2003	Kaler et al.	
2003/0177176	A1	9/2003	Hirschfeld et al.	
2003/0192035	A1*	10/2003	Duesterwald ald	717/138
2004/0088694	A1	5/2004	Ho	
2004/0162741	A1	8/2004	Flaxer et al.	
2004/0181794	A1	9/2004	Coleman et al.	
2004/0187104	A1	9/2004	Sardesai et al.	
2004/0230948	A1*	11/2004	Talwar et al.	717/114
2004/0260734	A1	12/2004	Ren et al.	
2005/0005200	A1	1/2005	Matena et al.	
2005/0039180	A1*	2/2005	Fultheim et al.	718/1
2005/0138370	A1*	6/2005	Goud et al.	713/164
2005/0193265	A1	9/2005	Lin et al.	
2006/0173856	A1	8/2006	Jackson et al.	
2006/0173857	A1	8/2006	Jackson	
2006/0173895	A1	8/2006	Engquist et al.	
2006/0173984	A1	8/2006	Emeis et al.	
2006/0173993	A1	8/2006	Henseler et al.	
2006/0173994	A1	8/2006	Emeis et al.	
2006/0174238	A1	8/2006	Henseler et al.	
2006/0259292	A1*	11/2006	Solomon et al.	703/27
2007/0168919	A1*	7/2007	Henseler et al.	717/101
2007/0169049	A1*	7/2007	Gingell et al.	717/151

FOREIGN PATENT DOCUMENTS

WO	WO/2006/083727	A1	8/2006
WO	WO/2006/083893	A1	8/2006
WO	WO/2006/083894	A1	8/2006
WO	WO/2006/083895	A1	8/2006
WO	WO/2006/083901	A1	8/2006

OTHER PUBLICATIONS

B. Urgaonkar et al., "Resource Overbooking and Application Profiling in Shared Hosting Platforms," Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 17 pgs., XP-002387427, 2002.

G. Lodi et al., "QoS-aware Clustering of Application Servers," Proceedings of the 1st IEEE Workshop on Quality of Service for Application Servers, In Conjunction With the 23rd International Symposium on Reliable Distributed Systems, 6 pages, XP-002383792, Oct. 17, 2004.

Preinstalling Microsoft Windows XP by Using the OEM Preinstallation Kit, Part 1, XP-002301441, Apr. 4, 2003, 24 pages.

R. Mark Koan et al., *It Takes a Village to Build an Image*, XP-002384269, 2003, pp. 200-207.

U.S. Appl. No. 11/607,819 entitled, "Automated Deployment and Configuration of Applications in an Autonomically Controlled Distributed Computing System", filed Dec. 1, 2006.

U.S. Appl. No. 11/607,820 entitled, "Automated Deployment and Configuration of Applications in an Autonomically Controlled Distributed Computing System", filed Dec. 1, 2006.

* cited by examiner

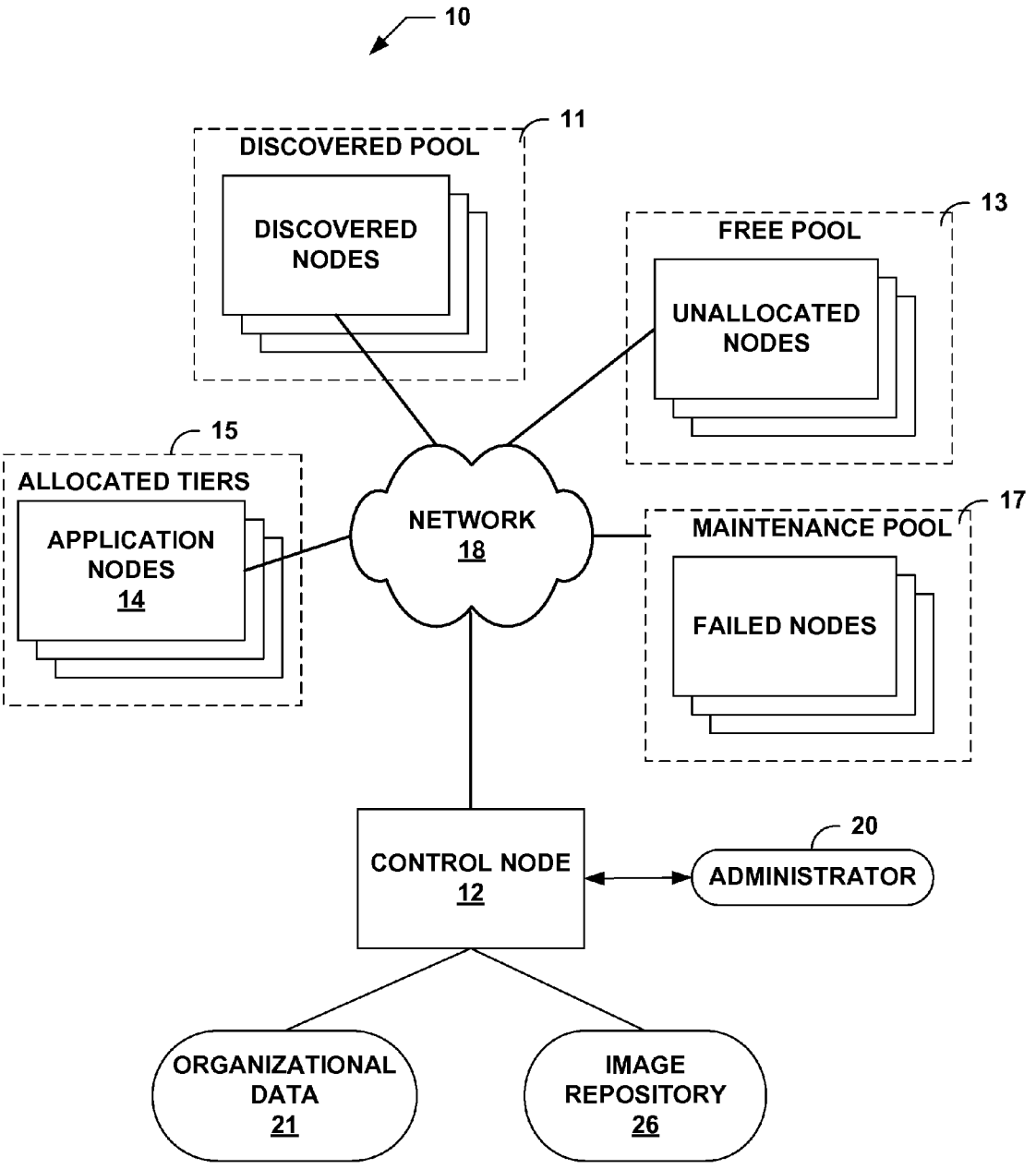

10
11
DISCOVERED POOL
DISCOVERED NODES
13
FREE POOL
UNALLOCATED NODES
15
ALLOCATED TIERS
APPLICATION NODES 14
NETWORK 18
17
MAINTENANCE POOL
FAILED NODES
CONTROL NODE 12
20
ADMINISTRATOR
ORGANIZATIONAL DATA 21
IMAGE REPOSITORY 26

FIG. 1

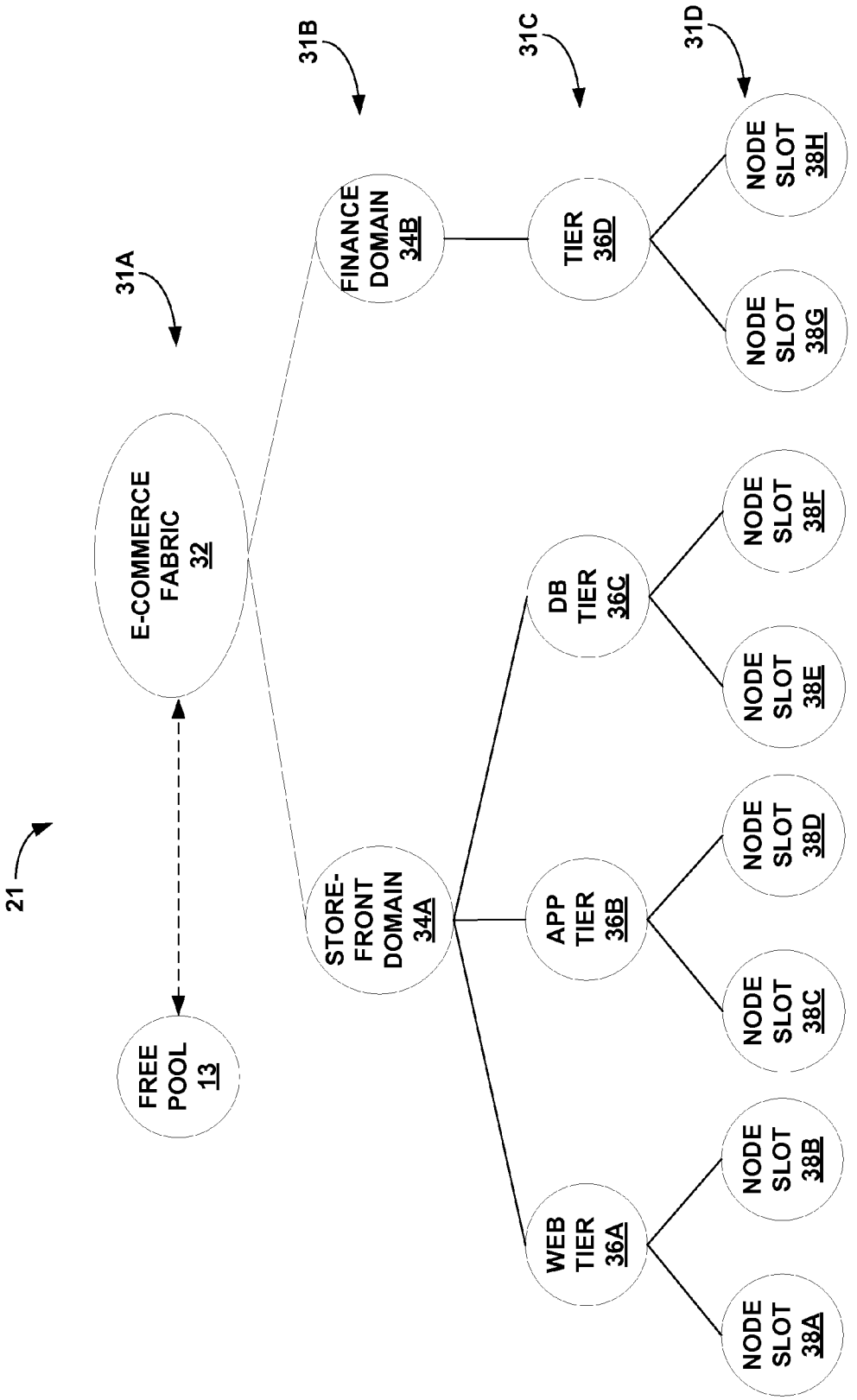

21
31A
31B
31C
31D
E-COMMERCE FABRIC 32
FREE POOL 13
FINANCE DOMAIN 34B
STORE-FRONT DOMAIN 34A
TIER 36D
DB TIER 36C
APP TIER 36B
WEB TIER 36A
NODE SLOT 38H
NODE SLOT 38G
NODE SLOT 38F
NODE SLOT 38E
NODE SLOT 38D
NODE SLOT 38C
NODE SLOT 38B
NODE SLOT 38A

FIG. 2

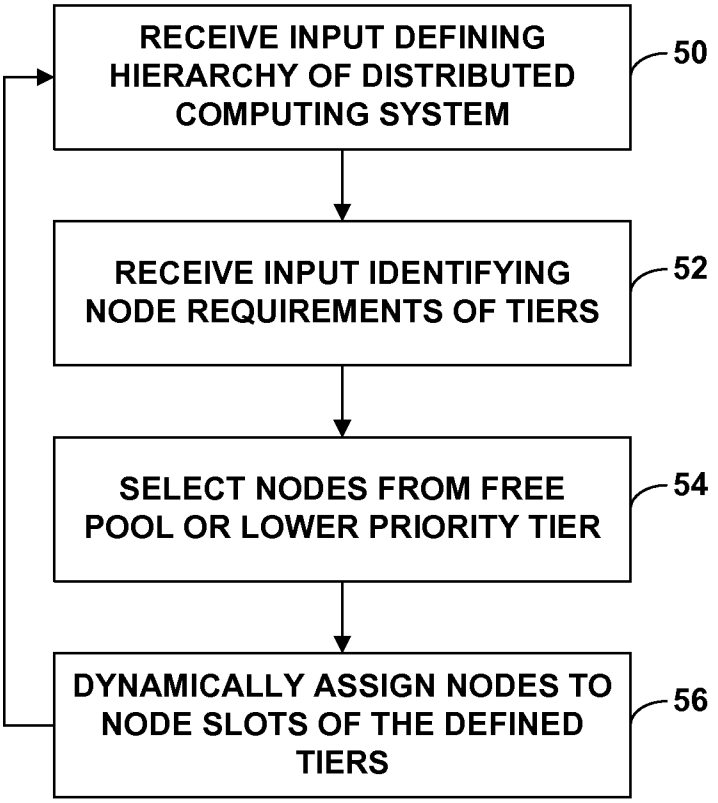

RECEIVE INPUT DEFINING HIERARCHY OF DISTRIBUTED COMPUTING SYSTEM
50
RECEIVE INPUT IDENTIFYING NODE REQUIREMENTS OF TIERS
52
SELECT NODES FROM FREE POOL OR LOWER PRIORITY TIER
54
DYNAMICALLY ASSIGN NODES TO NODE SLOTS OF THE DEFINED TIERS
56

FIG. 3

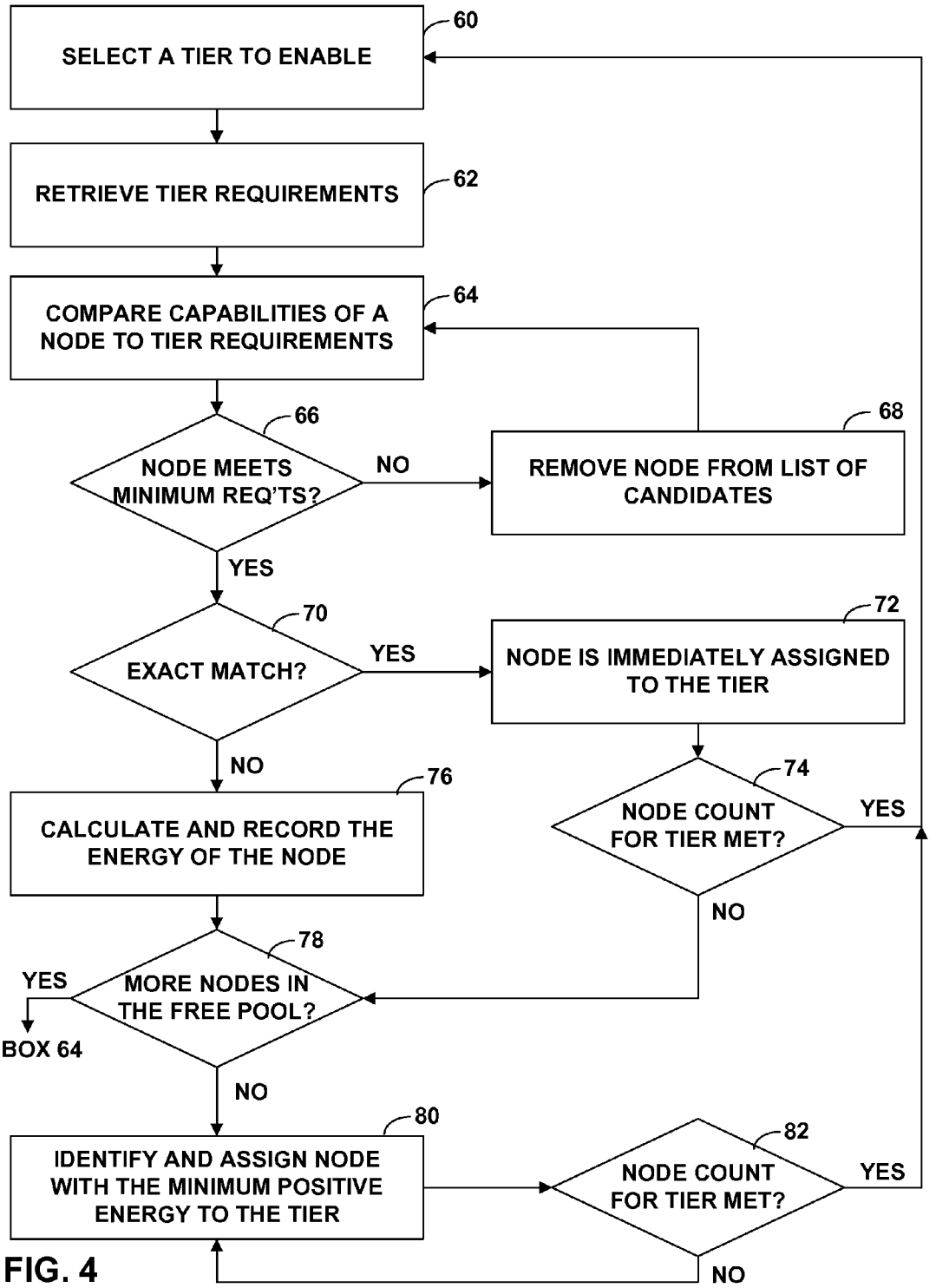

SELECT A TIER TO ENABLE
60
RETRIEVE TIER REQUIREMENTS
62
COMPARE CAPABILITIES OF A NODE TO TIER REQUIREMENTS
64
NODE MEETS MINIMUM REQ'TS?
66
NO
REMOVE NODE FROM LIST OF CANDIDATES
68
YES
EXACT MATCH?
70
YES
NODE IS IMMEDIATELY ASSIGNED TO THE TIER
72
NO
CALCULATE AND RECORD THE ENERGY OF THE NODE
76
NODE COUNT FOR TIER MET?
74
YES
NO
MORE NODES IN THE FREE POOL?
78
YES
BOX 64
NO
IDENTIFY AND ASSIGN NODE WITH THE MINIMUM POSITIVE ENERGY TO THE TIER
80
NODE COUNT FOR TIER MET?
82
YES
NO

FIG. 4

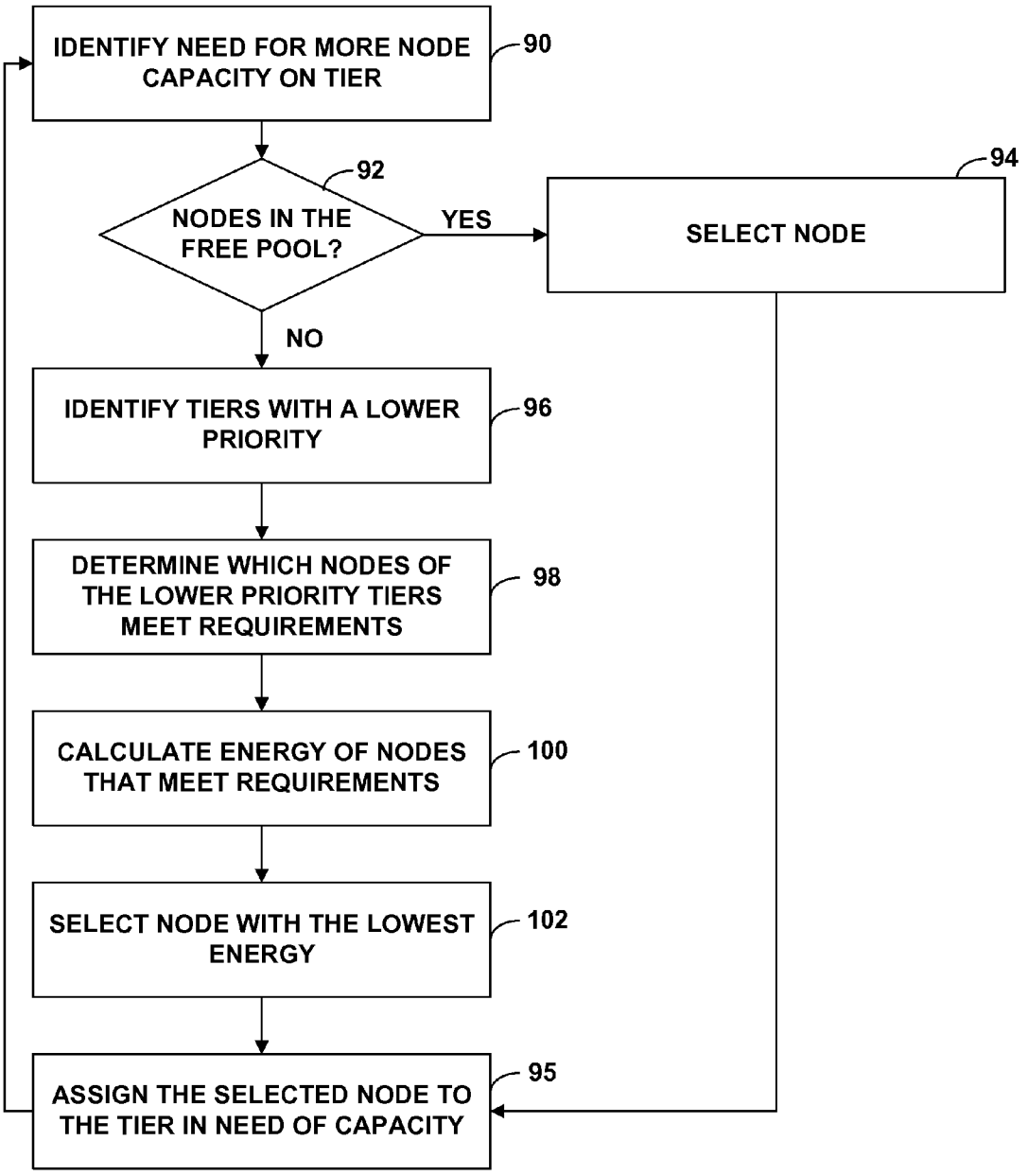

IDENTIFY NEED FOR MORE NODE CAPACITY ON TIER
90
NODES IN THE FREE POOL?
92
YES
SELECT NODE
94
NO
IDENTIFY TIERS WITH A LOWER PRIORITY
96
DETERMINE WHICH NODES OF THE LOWER PRIORITY TIERS MEET REQUIREMENTS
98
CALCULATE ENERGY OF NODES THAT MEET REQUIREMENTS
100
SELECT NODE WITH THE LOWEST ENERGY
102
ASSIGN THE SELECTED NODE TO THE TIER IN NEED OF CAPACITY
95

FIG. 5

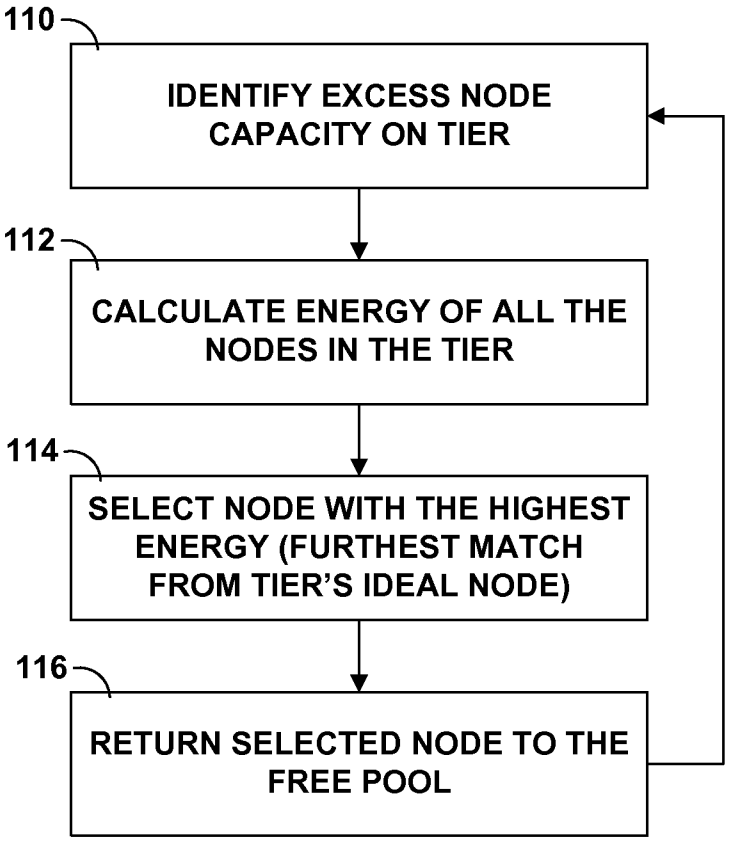

110
IDENTIFY EXCESS NODE CAPACITY ON TIER
112
CALCULATE ENERGY OF ALL THE NODES IN THE TIER
114
SELECT NODE WITH THE HIGHEST ENERGY (FURTHEST MATCH FROM TIER'S IDEAL NODE)
116
RETURN SELECTED NODE TO THE FREE POOL

FIG. 6

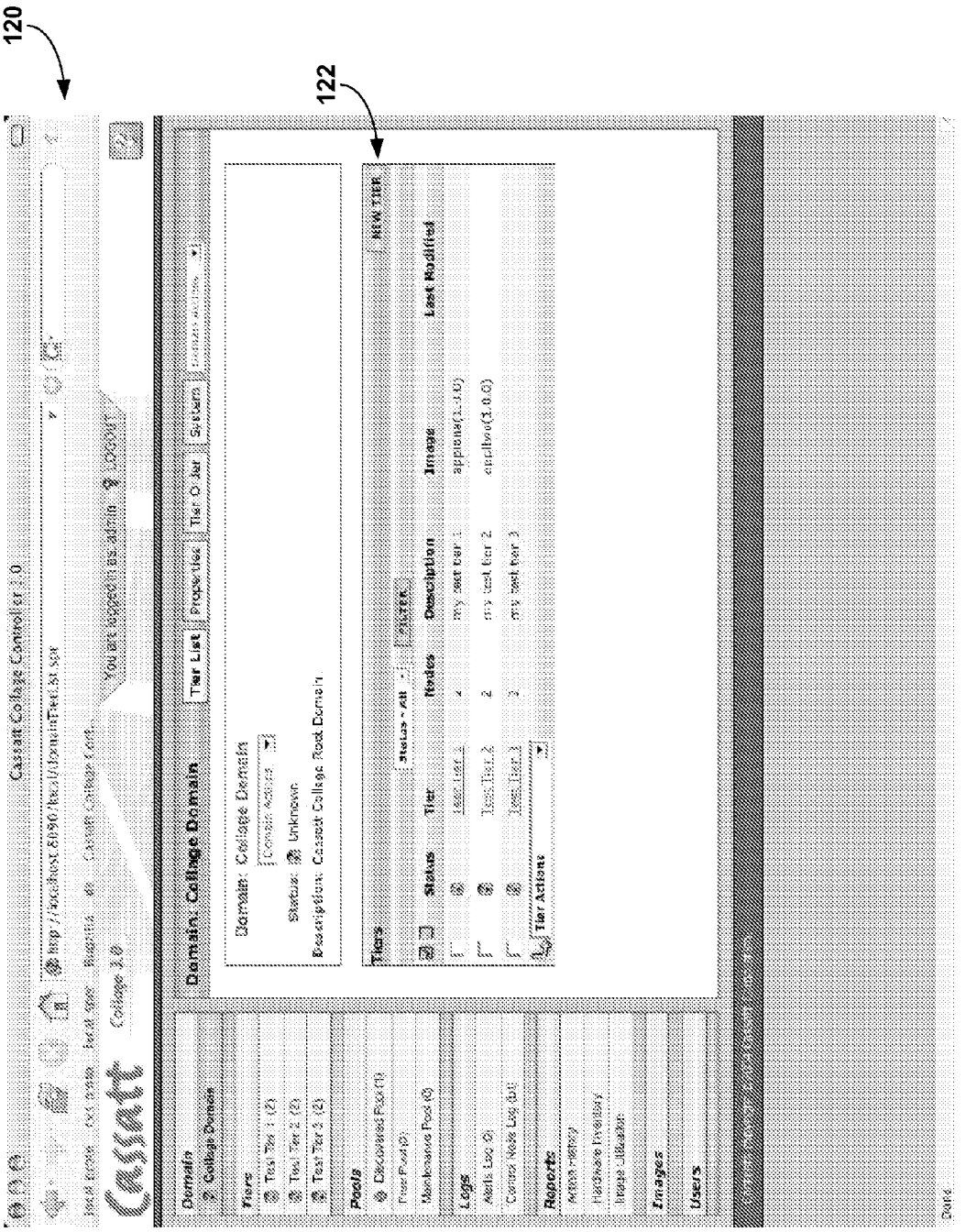

120
122
Cassat Collage Controller 1.0
Cassat
Collage 1.0
You are logged in as: admin
Logout
Domain
College Domain
Tiers
Test Tier 1 (0)
Test Tier 2 (0)
Test Tier 3 (0)
Pools
Discovered Pool (0)
Free Pool (0)
Maintenance Pool (0)
Logs
Alerts Log (0)
Control Node Log (0)
Reports
Activity History
Hardware Inventory
Image Utilization
Images
Users
Domain: College Domain
Tier List
Properties
Tier Or Jar
System
Domain: College Domain
Domain Actions
Status: Unknown
Description: Cassat College Pool Domain
Tiers
NEW TIER
Status - All
FILTER
Status
Tier
Nodes
Description
Image
Last Modified
Test Tier 1
my test tier 1
appliance(1.0.0)
Test Tier 2
my test tier 2
appliance(1.0.0)
Test Tier 3
my test tier 3
Tier Actions

FIG. 7

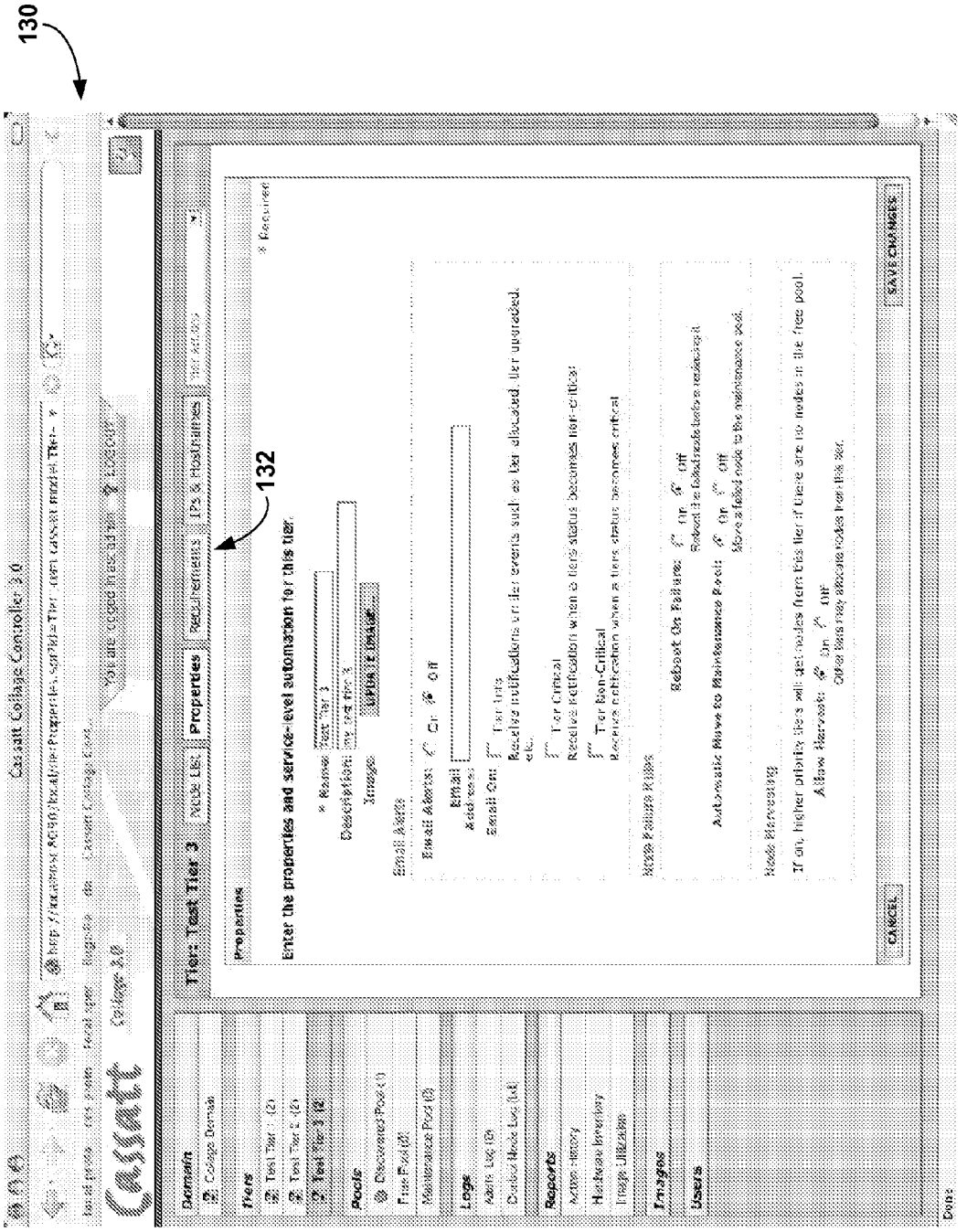

130
132
Cisco
Cisco Collage Controller 3.0
Tiers
Test Tier 1 (2)
Test Tier 2 (2)
Test Tier 3 (2)
Pools
Maintenance Pool (0)
Logs
Admin Log (0)
Device Mode Log (14)
Reports
Action History
Hardware Inventory
Usage Utilization
Images
Users
Tier: Test Tier 3
Node List
Properties
Requirements
IPs & Hostnames
Properties
Enter the properties and service-level automation for this tier.
* Required
* Name:
Test Tier 3
Description:
my test tier 3
Image:
Upload Image
Email Alerts
Email Alerts:
On
Off
Email:
Email Cisco:
Tier Info
Receive notifications when other events such as tier allocated, tier unrelated, etc.
Tier Critical
Receive notification when a tiers status becomes non-critical
Tier Non-Critical
Receive notification when a tiers status becomes critical
Node Failure Policy
Reboot On Failure:
On
Off
Reboot the failed node before redeploy
Automatic Move to Maintenance Pool:
On
Off
Move a failed node to the maintenance pool.
Node Harvesting
If any higher priority tiers will get no nodes from this tier if there are no nodes in the free pool.
Allow Harvesting:
On
Off
Other tiers may allocate nodes from this tier.
CANCEL
SAVE CHANGES
Done

FIG. 8

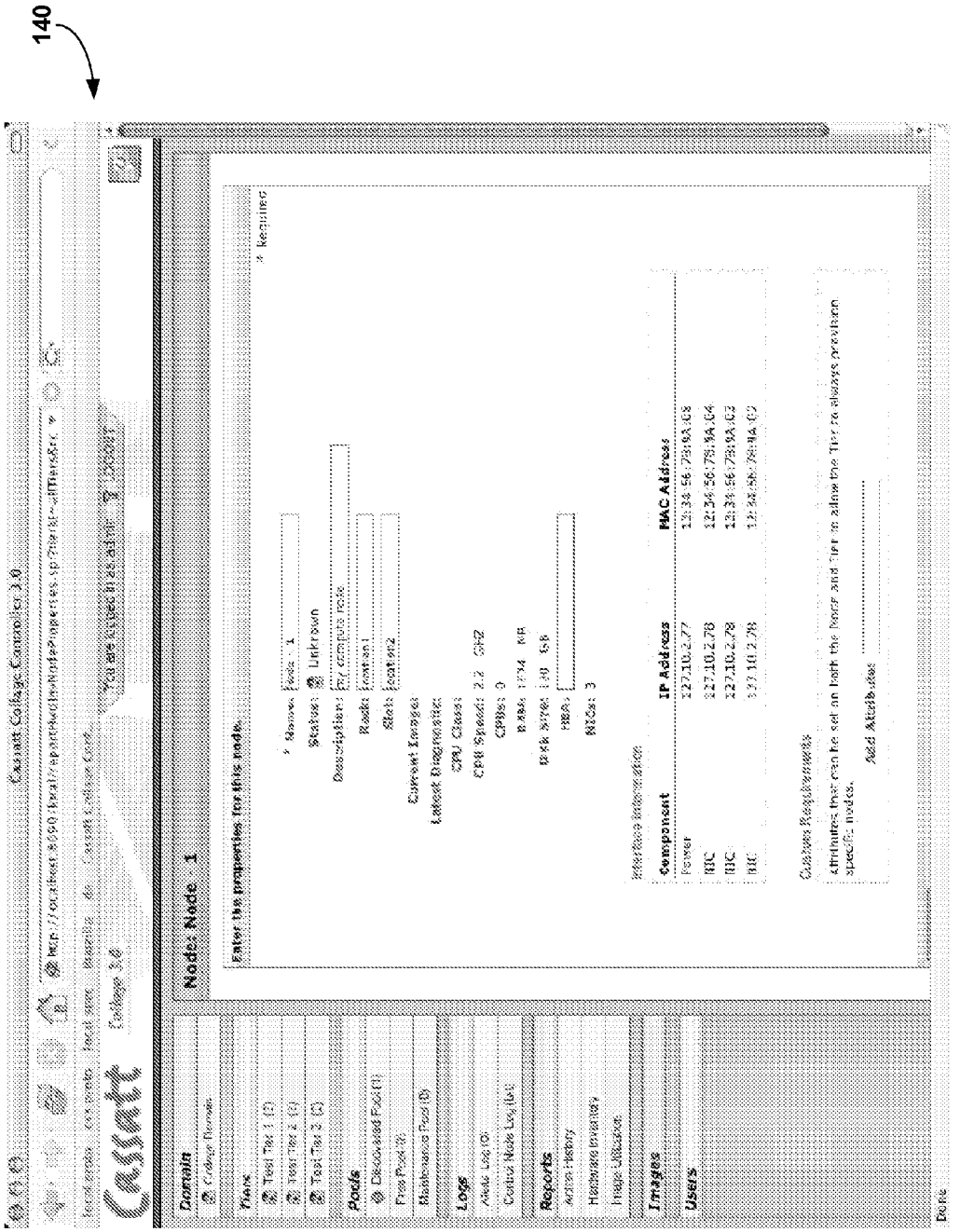

140
Cariat
Node: Node_1
Enter the properties for this node.
Domain
Runs
Pools
Logs
Reports
Images
Users
Component
IP Address
MAC Address

FIG. 9

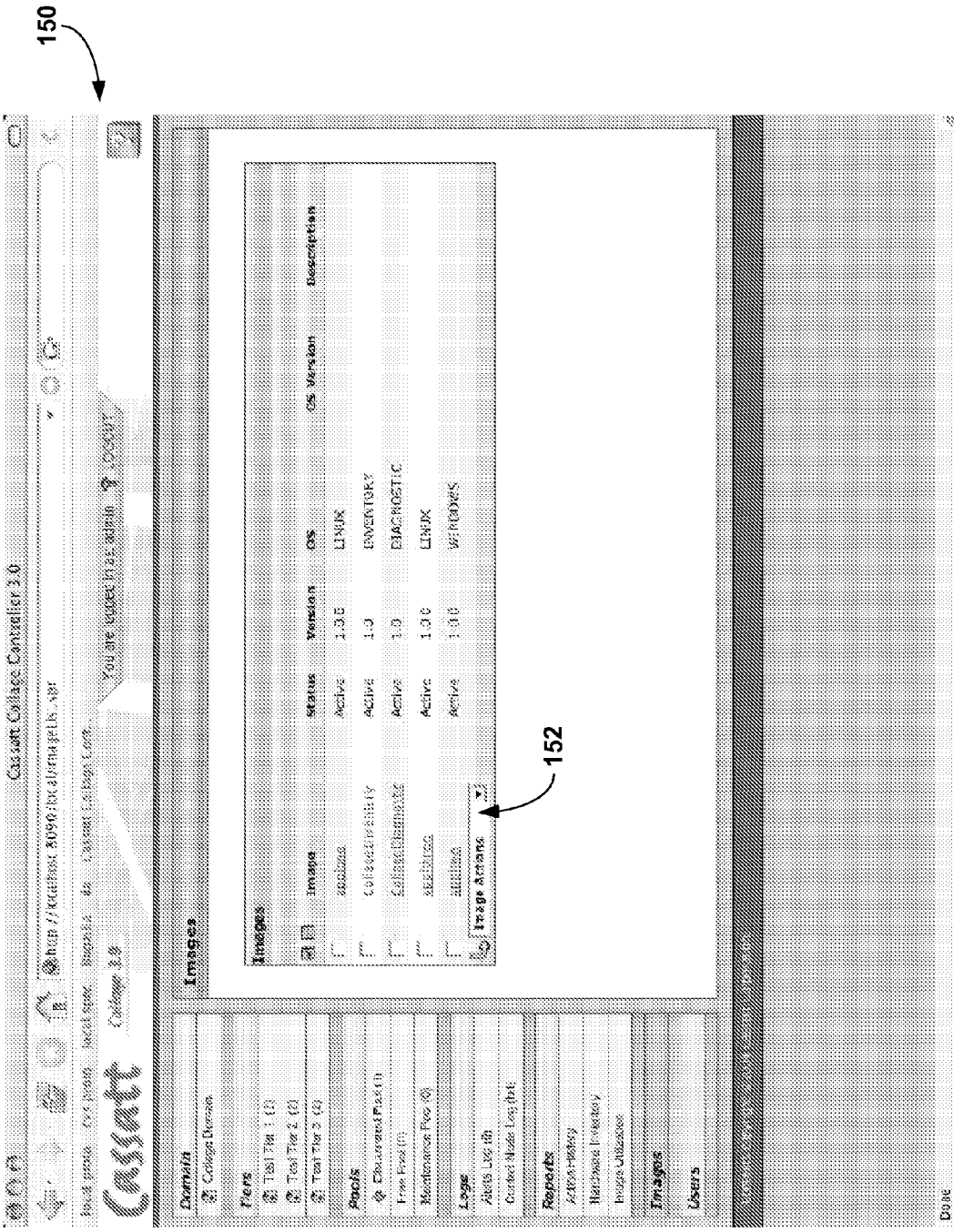

150
Cisco Unified Computing System Manager
Images
Image Actions
152

FIG. 10

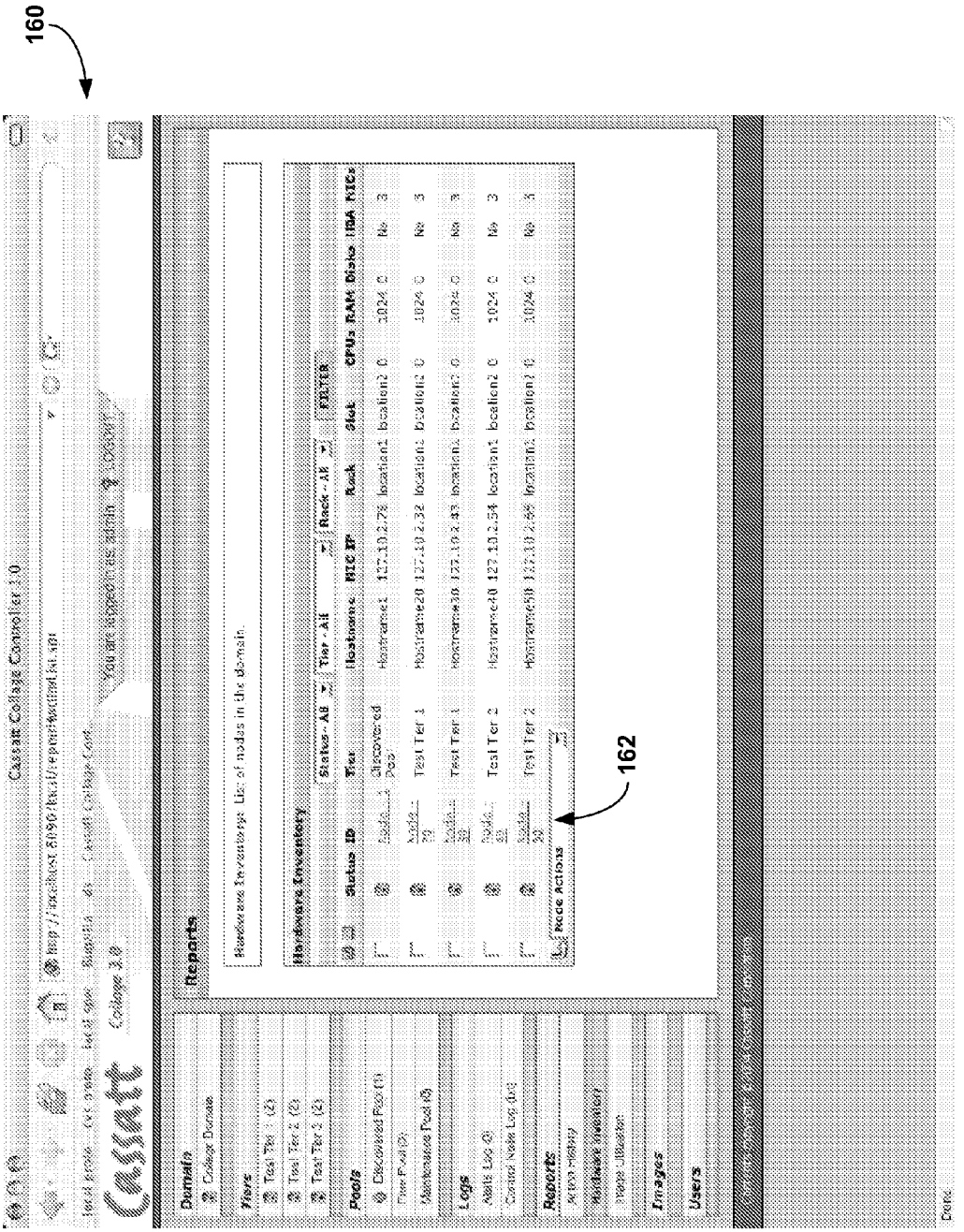

160
Cisco Collage Controller 3.0
Cisco Collage
Collage 3.0
You are logged in as admin
Domain
College Domain
Tiers
Test Tier 1 (2)
Test Tier 2 (2)
Test Tier 3 (2)
Pools
Discovered Pool (1)
Tier Pools (3)
Maintenance Pool (0)
Logs
Alarm Log (0)
Control Node Log (0)
Reports
Action History
Hardware Inventory
Image Utilization
Images
Users
Reports
List of nodes in the domain.
Hardware Inventory
Status - All
Tier - All
Rack - All
Edit Tab
Status ID
Tier
Hostname
NIC IP
Rack
Slot
CPUs
RAM
Disks
HBA NICs
Discovered Pool
Test Tier 1
Test Tier 2
Node Actions
162

FIG. 11

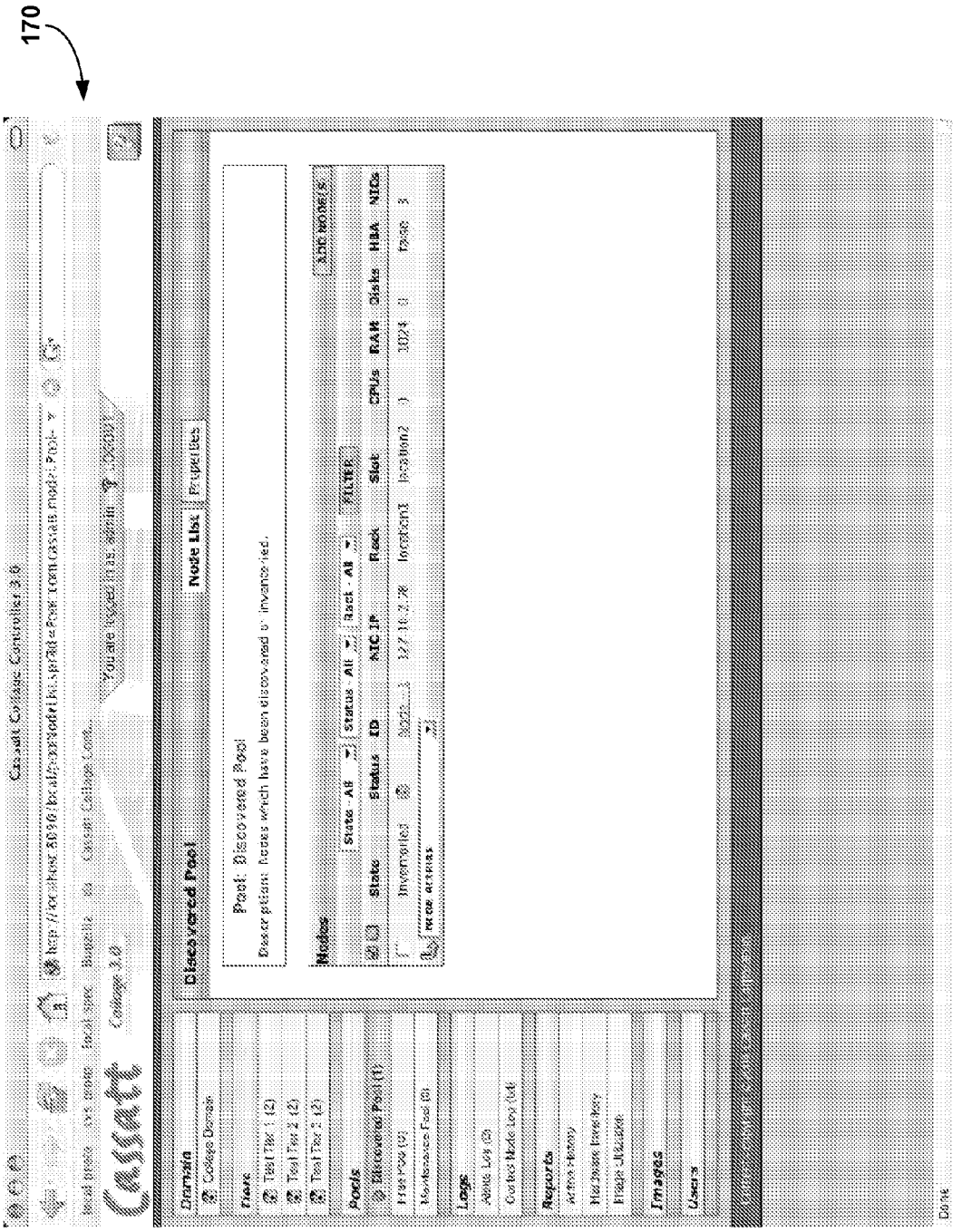

170
Cassatt Collage Controller 3.0
Cassatt
Collage 3.0
Domain
College Domain
Plan
Test Plan 1 (2)
Test Plan 2 (2)
Test Plan 3 (2)
Pools
Discovered Pool (1)
Free Pool (0)
Maintenance Pool (0)
Logs
Admin Log (0)
Control Node Log (14)
Reports
Action History
Hardware Inventory
Image Utilization
Images
Users
Discovered Pool
Node List
Properties
Pool: Discovered Pool
Description: Nodes which have been discovered or inventoried.
Nodes
State - All
Status - All
Rack - All
FILTER
State
Status
ID
NIC IP
Rack
Slot
CPUs
RAM
Disks
HBA
NICs
Inventoried
Location1
location2
1024
Done

FIG. 12

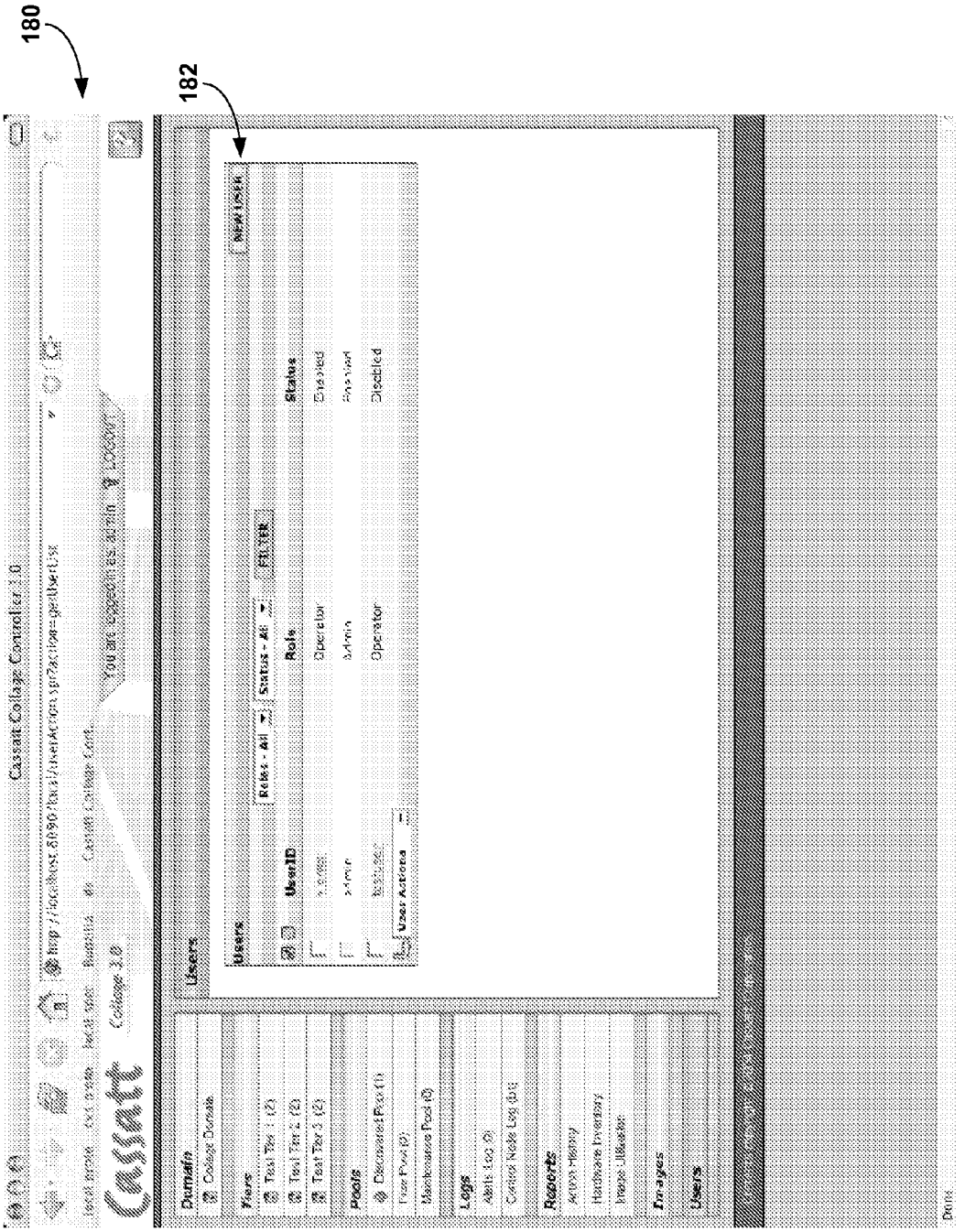

180
182
Cassat Collage Controller 3.0
Cassat
Collage 3.0
Domain
Users
Pools
Logs
Reports
Images
Users
UserID
Role
Status
Operator
Disabled

FIG. 13

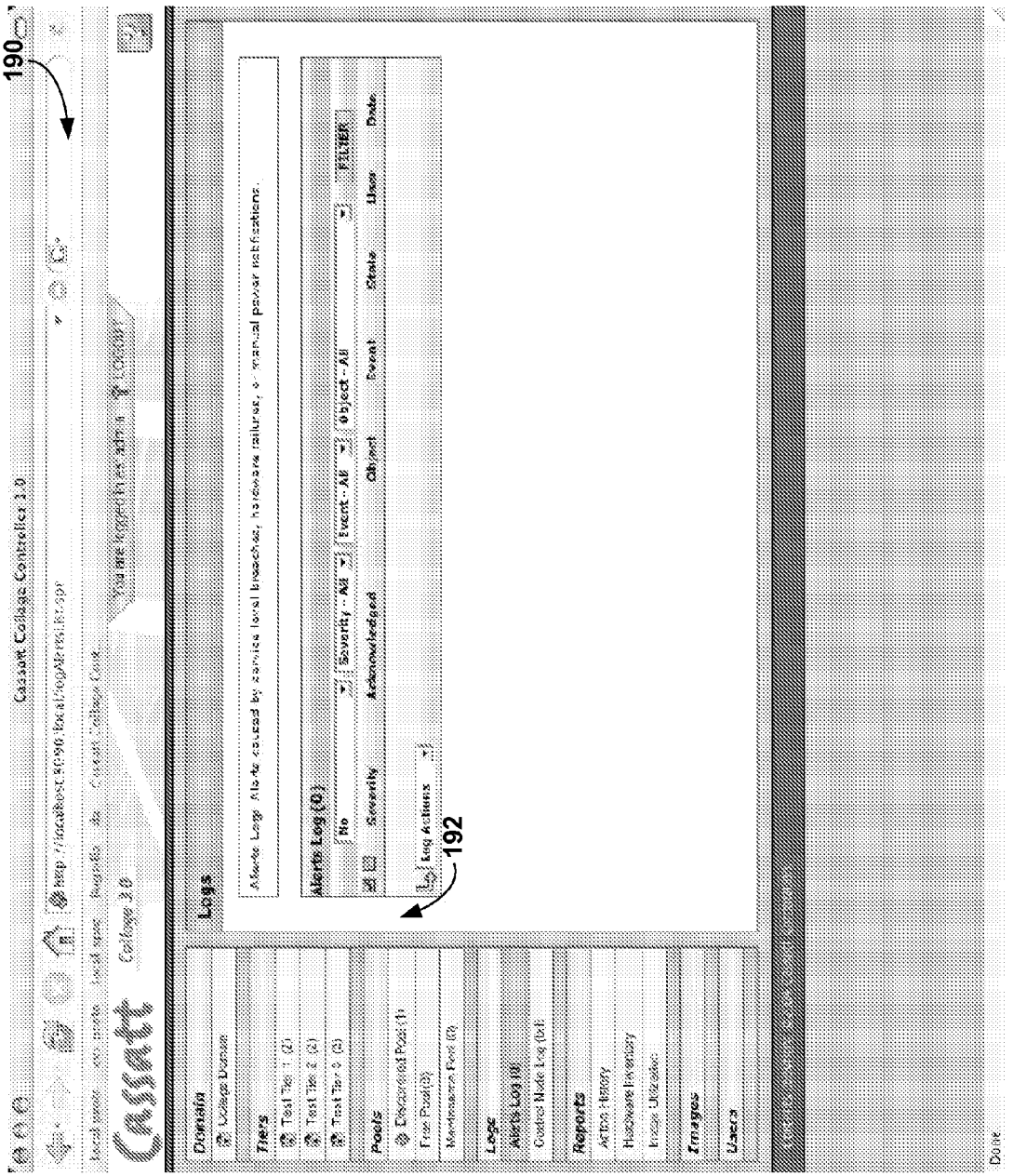

190
Cassat Collage Controller 3.0
Cassat
Collage 3.0
Logs
Domain
Tiers
Pools
Logs
Reports
Images
Users
Alerts Log (0)
Severity
Acknowledged
Object
Event
State
User
Date
FILTER
192
Done

FIG. 14

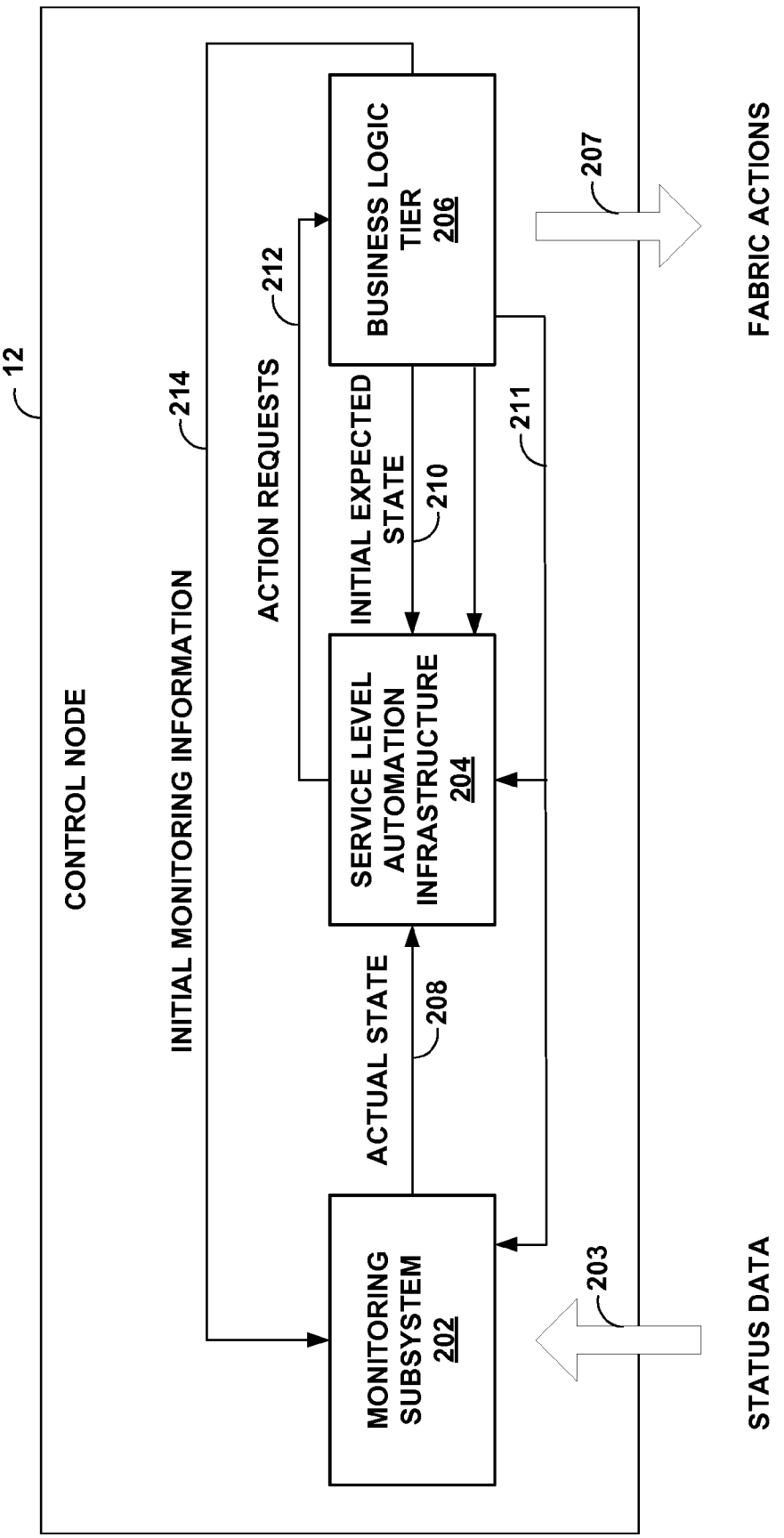

12
CONTROL NODE
214
INITIAL MONITORING INFORMATION
212
ACTION REQUESTS
BUSINESS LOGIC TIER 206
207
FABRIC ACTIONS
INITIAL EXPECTED STATE
210
211
SERVICE LEVEL AUTOMATION INFRASTRUCTURE 204
ACTUAL STATE
208
MONITORING SUBSYSTEM 202
203
STATUS DATA

FIG. 15

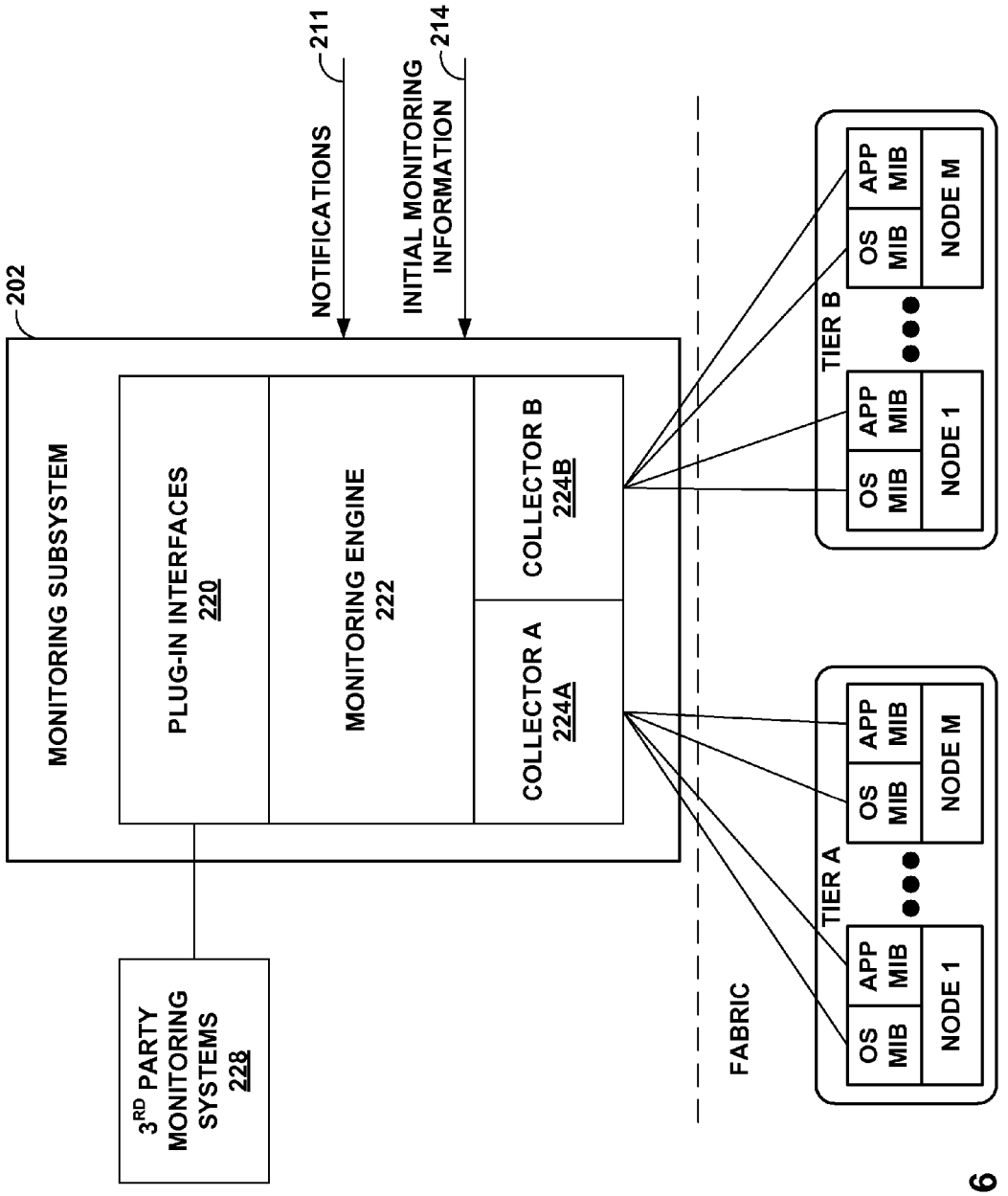

202
MONITORING SUBSYSTEM
PLUG-IN INTERFACES 220
MONITORING ENGINE 222
COLLECTOR B 224B
COLLECTOR A 224A
NOTIFICATIONS 211
INITIAL MONITORING INFORMATION 214
3RD PARTY MONITORING SYSTEMS 228
FABRIC
TIER A
TIER B
OS MIB
APP MIB
NODE 1
NODE M

FIG. 16

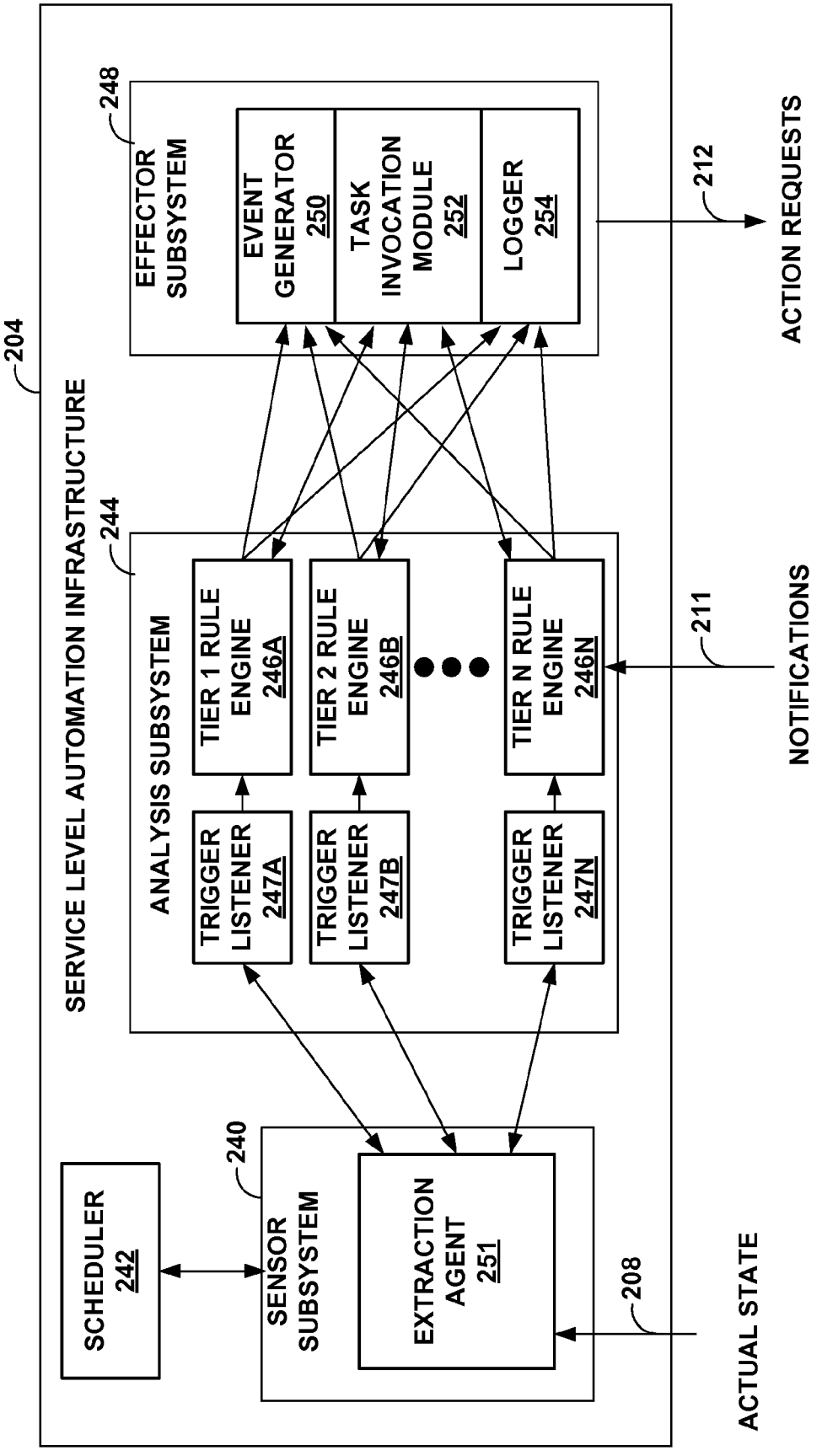

204
SERVICE LEVEL AUTOMATION INFRASTRUCTURE
248
EFFECTOR SUBSYSTEM
EVENT GENERATOR 250
TASK INVOCATION MODULE 252
LOGGER 254
212
ACTION REQUESTS
244
ANALYSIS SUBSYSTEM
TIER 1 RULE ENGINE 246A
TIER 2 RULE ENGINE 246B
TIER N RULE ENGINE 246N
211
NOTIFICATIONS
TRIGGER LISTENER 247A
TRIGGER LISTENER 247B
TRIGGER LISTENER 247N
240
SENSOR SUBSYSTEM
EXTRACTION AGENT 251
SCHEDULER 242
208
ACTUAL STATE

FIG. 17

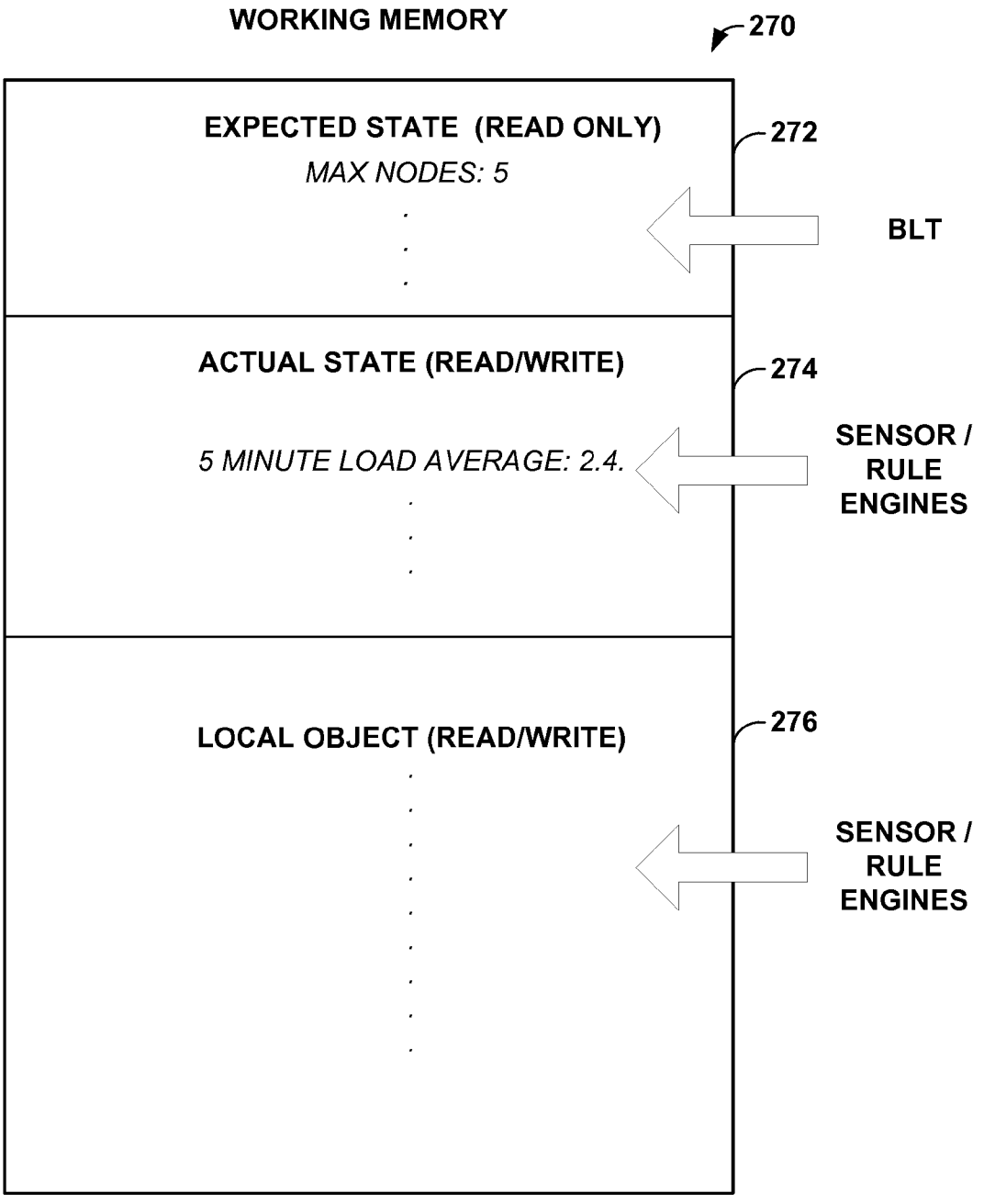

WORKING MEMORY
270
EXPECTED STATE (READ ONLY)
MAX NODES: 5
272
BLT
ACTUAL STATE (READ/WRITE)
274
5 MINUTE LOAD AVERAGE: 2.4.
SENSOR / RULE ENGINES
LOCAL OBJECT (READ/WRITE)
276
SENSOR / RULE ENGINES

FIG. 18

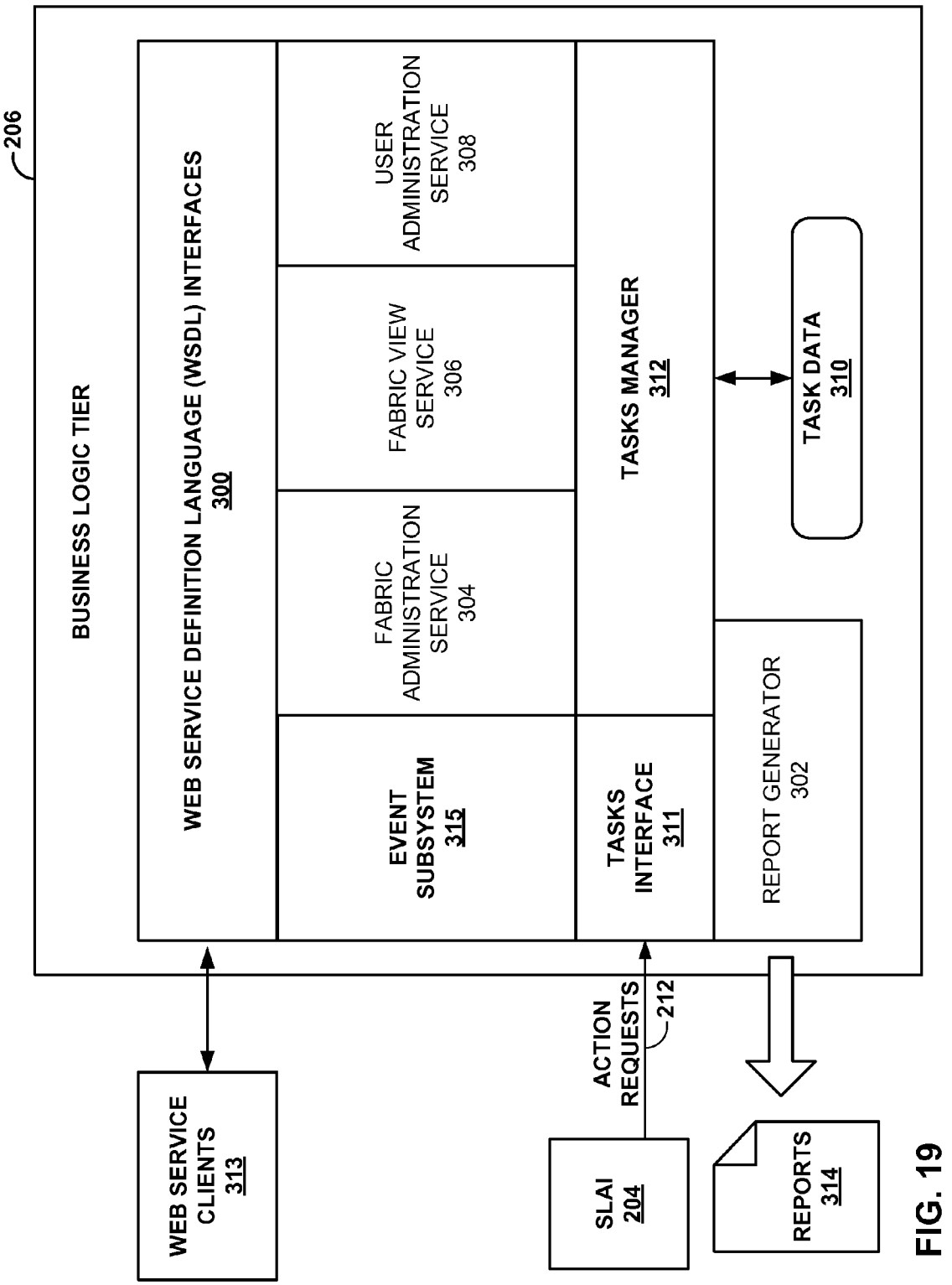

206
BUSINESS LOGIC TIER
WEB SERVICE DEFINITION LANGUAGE (WSDL) INTERFACES 300
USER ADMINISTRATION SERVICE 308
FABRIC VIEW SERVICE 306
FABRIC ADMINISTRATION SERVICE 304
EVENT SUBSYSTEM 315
TASKS MANAGER 312
TASKS INTERFACE 311
REPORT GENERATOR 302
TASK DATA 310
WEB SERVICE CLIENTS 313
ACTION REQUESTS 212
SLAI 204
REPORTS 314

FIG. 19

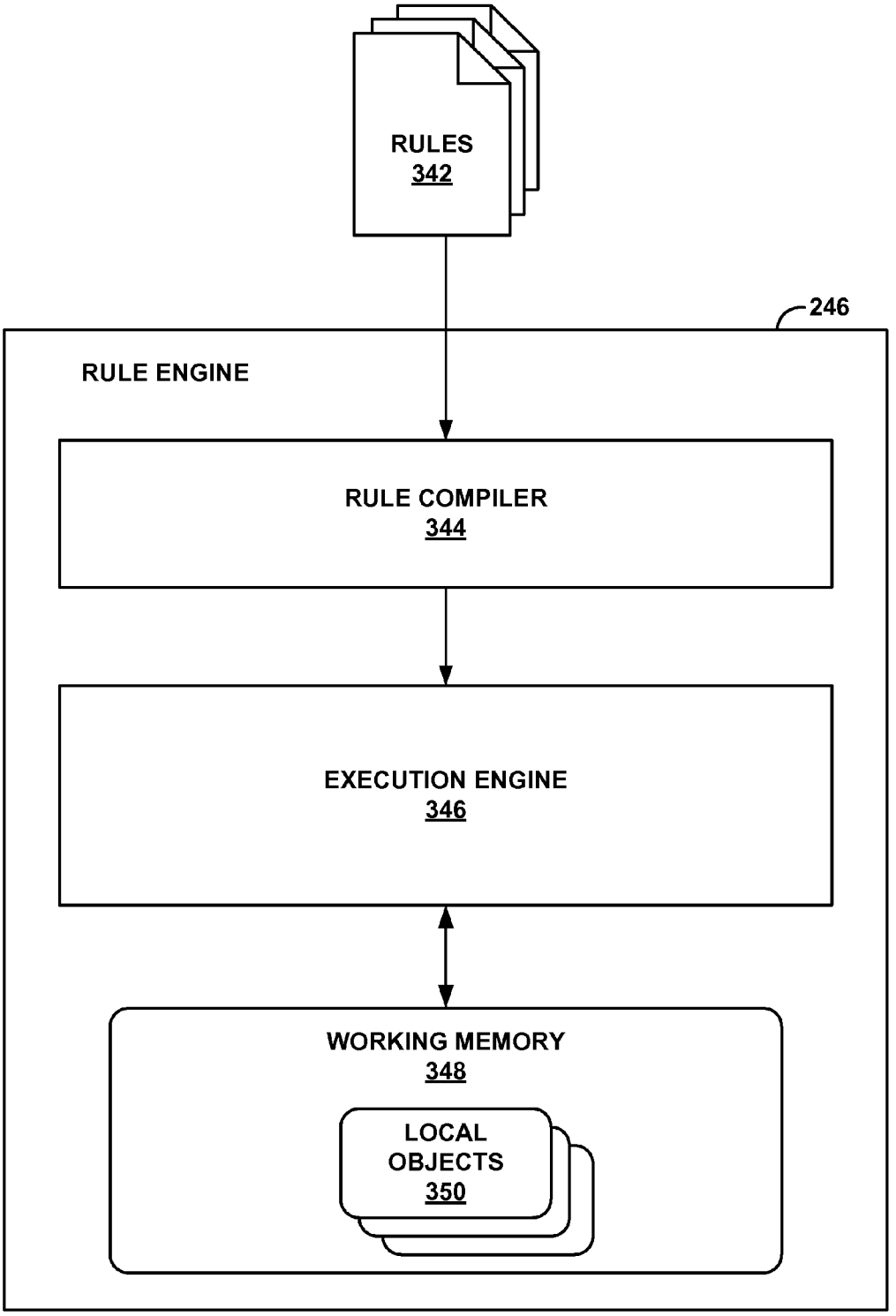

RULES
342
246
RULE ENGINE
RULE COMPILER
344
EXECUTION ENGINE
346
WORKING MEMORY
348
LOCAL
OBJECTS
350

FIG. 20

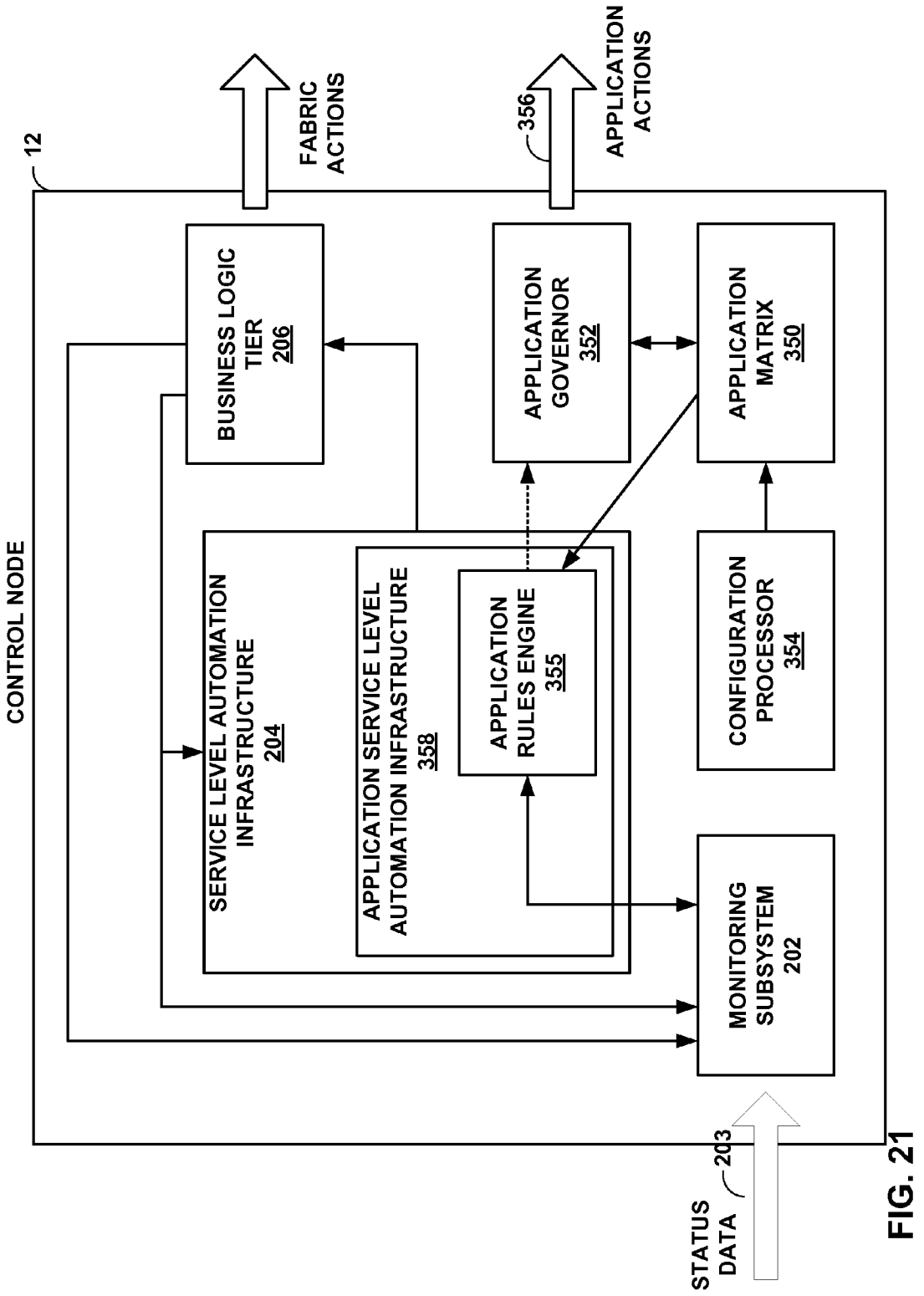

12
FABRIC ACTIONS
356
APPLICATION ACTIONS
CONTROL NODE
BUSINESS LOGIC TIER 206
APPLICATION GOVERNOR 352
APPLICATION MATRIX 350
SERVICE LEVEL AUTOMATION INFRASTRUCTURE 204
APPLICATION SERVICE LEVEL AUTOMATION INFRASTRUCTURE 358
APPLICATION RULES ENGINE 355
CONFIGURATION PROCESSOR 354
MONITORING SUBSYSTEM 202
203
STATUS DATA

FIG. 21

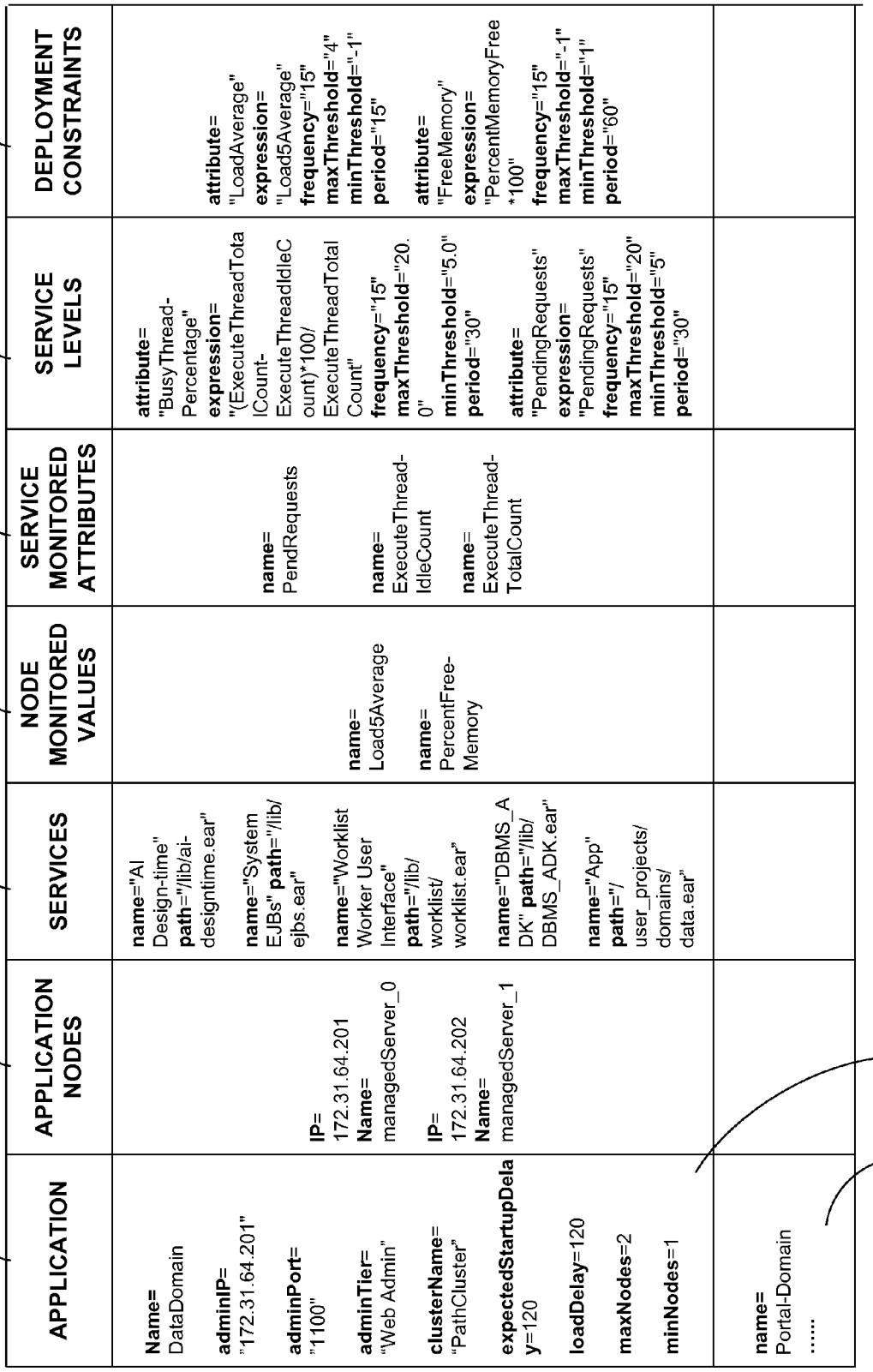

360
APPLICATION
362
APPLICATION NODES
364
SERVICES
366
NODE MONITORED VALUES
368
SERVICE MONITORED ATTRIBUTES
370
SERVICE LEVELS
372
DEPLOYMENT CONSTRAINTS
Name= DataDomain
adminIP= "172.31.64.201"
adminPort= "1100"
adminTier= "Web Admin"
clusterName= "PathCluster"
expectedStartupDelay=120
loadDelay=120
maxNodes=2
minNodes=1
IP= 172.31.64.201
Name= managedServer_0
IP= 172.31.64.202
Name= managedServer_1
name="Al Design-time" path="/lib/ai-designtime.ear"
name="System EJBs" path="/lib/ejbs.ear"
name="Worklist Worker User Interface" path="/lib/worklist/worklist.ear"
name="DBMS_ADK" path="/lib/DBMS_ADK.ear"
name="App" path="/user_projects/domains/data.ear"
name= Load5Average
name= PercentFree-Memory
name= PendRequests
name= ExecuteThread-IdleCount
name= ExecuteThread-TotalCount
attribute= "BusyThread-Percentage" expression= "(ExecuteThreadTotalCount-ExecuteThreadIdleCount)*100/ExecuteThreadTotalCount" frequency="15" maxThreshold="20.0" minThreshold="5.0" period="30"
attribute= "PendingRequests" expression= "PendingRequests" frequency="15" maxThreshold="20" minThreshold="5" period="30"
attribute= "LoadAverage" expression= "Load5Average" frequency="15" maxThreshold="4" minThreshold="-1" period="15"
attribute= "FreeMemory" expression= "PercentMemoryFree*100" frequency="15" maxThreshold="-1" minThreshold="1" period="60"
name= Portal-Domain
.....
374
376

FIG. 22

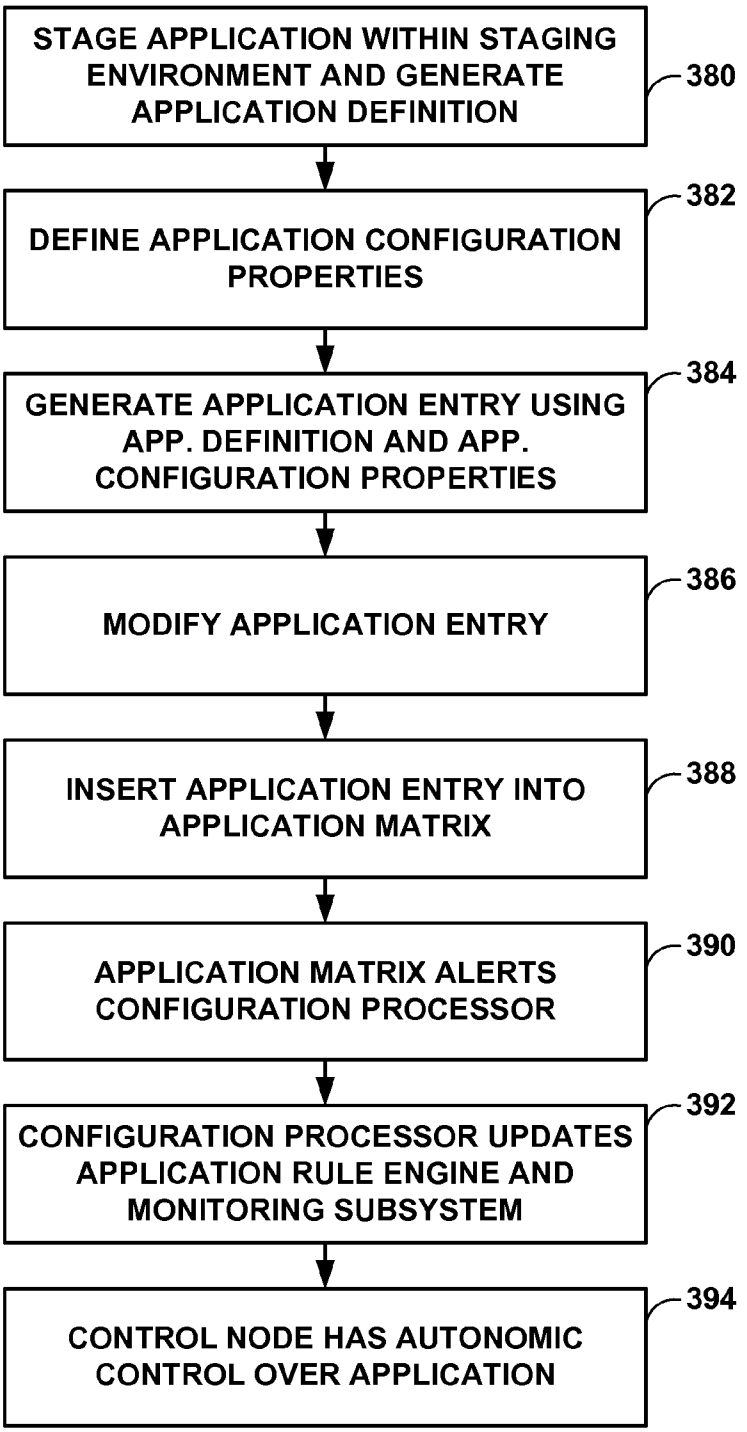

STAGE APPLICATION WITHIN STAGING ENVIRONMENT AND GENERATE APPLICATION DEFINITION
380
DEFINE APPLICATION CONFIGURATION PROPERTIES
382
GENERATE APPLICATION ENTRY USING APP. DEFINITION AND APP. CONFIGURATION PROPERTIES
384
MODIFY APPLICATION ENTRY
386
INSERT APPLICATION ENTRY INTO APPLICATION MATRIX
388
APPLICATION MATRIX ALERTS CONFIGURATION PROCESSOR
390
CONFIGURATION PROCESSOR UPDATES APPLICATION RULE ENGINE AND MONITORING SUBSYSTEM
392
CONTROL NODE HAS AUTONOMIC CONTROL OVER APPLICATION
394

FIG. 23

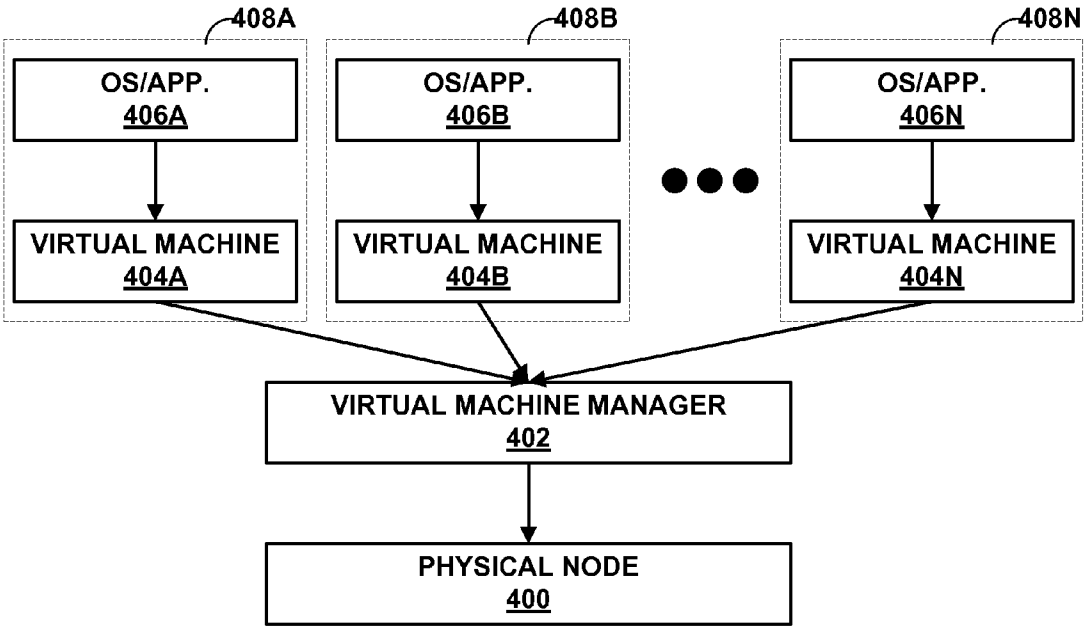

408A
408B
408N
OS/APP.
406A
OS/APP.
406B
OS/APP.
406N
VIRTUAL MACHINE
404A
VIRTUAL MACHINE
404B
VIRTUAL MACHINE
404N
VIRTUAL MACHINE MANAGER
402
PHYSICAL NODE
400

FIG. 24

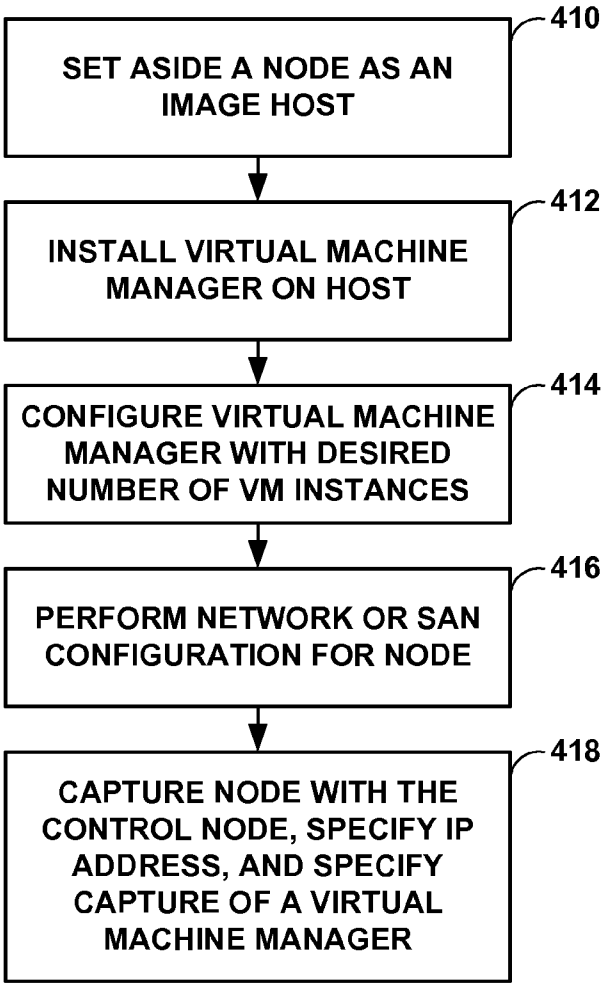

410
SET ASIDE A NODE AS AN IMAGE HOST
412
INSTALL VIRTUAL MACHINE MANAGER ON HOST
414
CONFIGURE VIRTUAL MACHINE MANAGER WITH DESIRED NUMBER OF VM INSTANCES
416
PERFORM NETWORK OR SAN CONFIGURATION FOR NODE
418
CAPTURE NODE WITH THE CONTROL NODE, SPECIFY IP ADDRESS, AND SPECIFY CAPTURE OF A VIRTUAL MACHINE MANAGER

FIG. 25

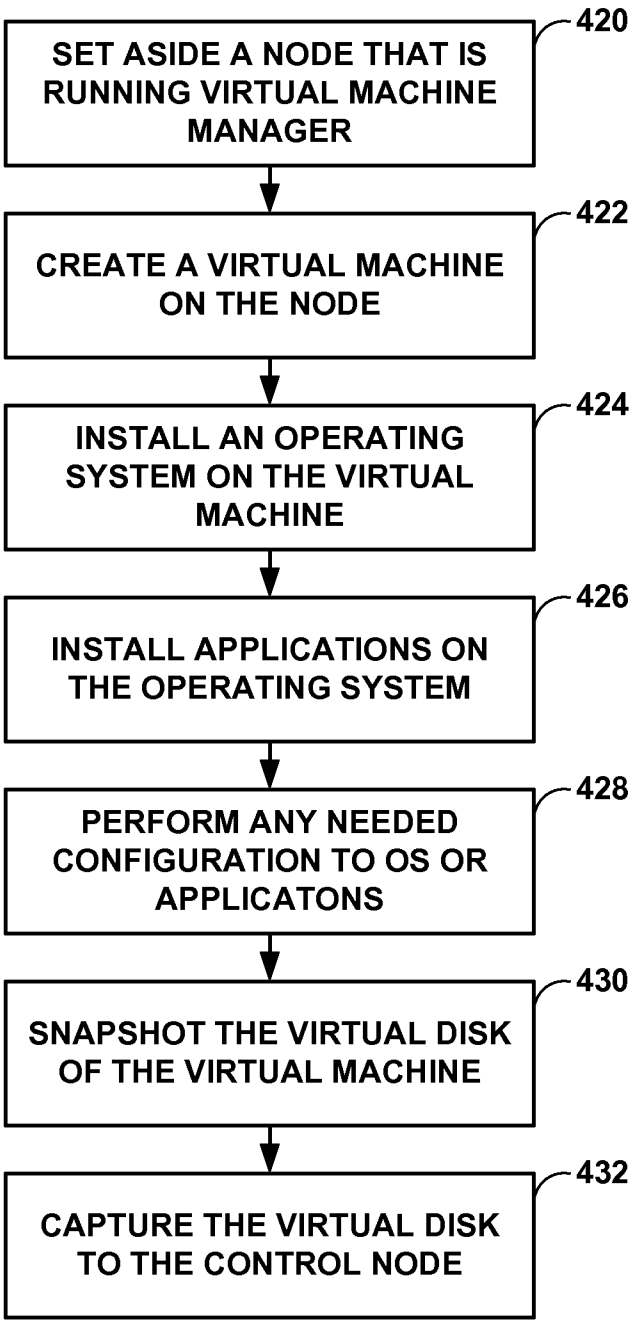

SET ASIDE A NODE THAT IS RUNNING VIRTUAL MACHINE MANAGER
420
CREATE A VIRTUAL MACHINE ON THE NODE
422
INSTALL AN OPERATING SYSTEM ON THE VIRTUAL MACHINE
424
INSTALL APPLICATIONS ON THE OPERATING SYSTEM
426
PERFORM ANY NEEDED CONFIGURATION TO OS OR APPLICATONS
428
SNAPSHOT THE VIRTUAL DISK OF THE VIRTUAL MACHINE
430
CAPTURE THE VIRTUAL DISK TO THE CONTROL NODE
432

FIG. 26

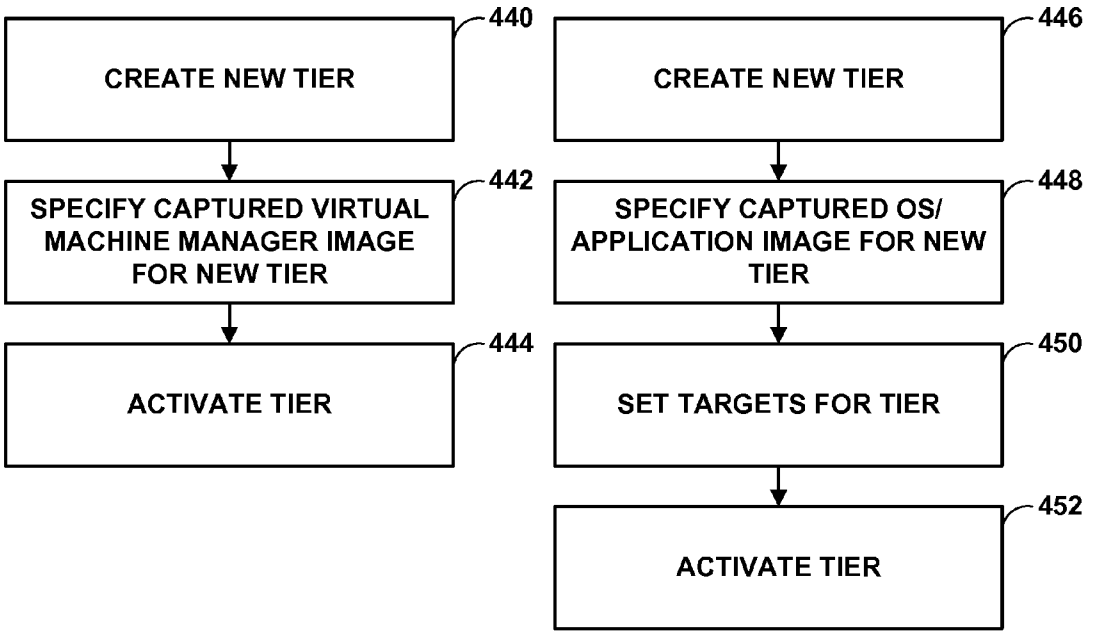


FIG. 27A

FIG. 27B

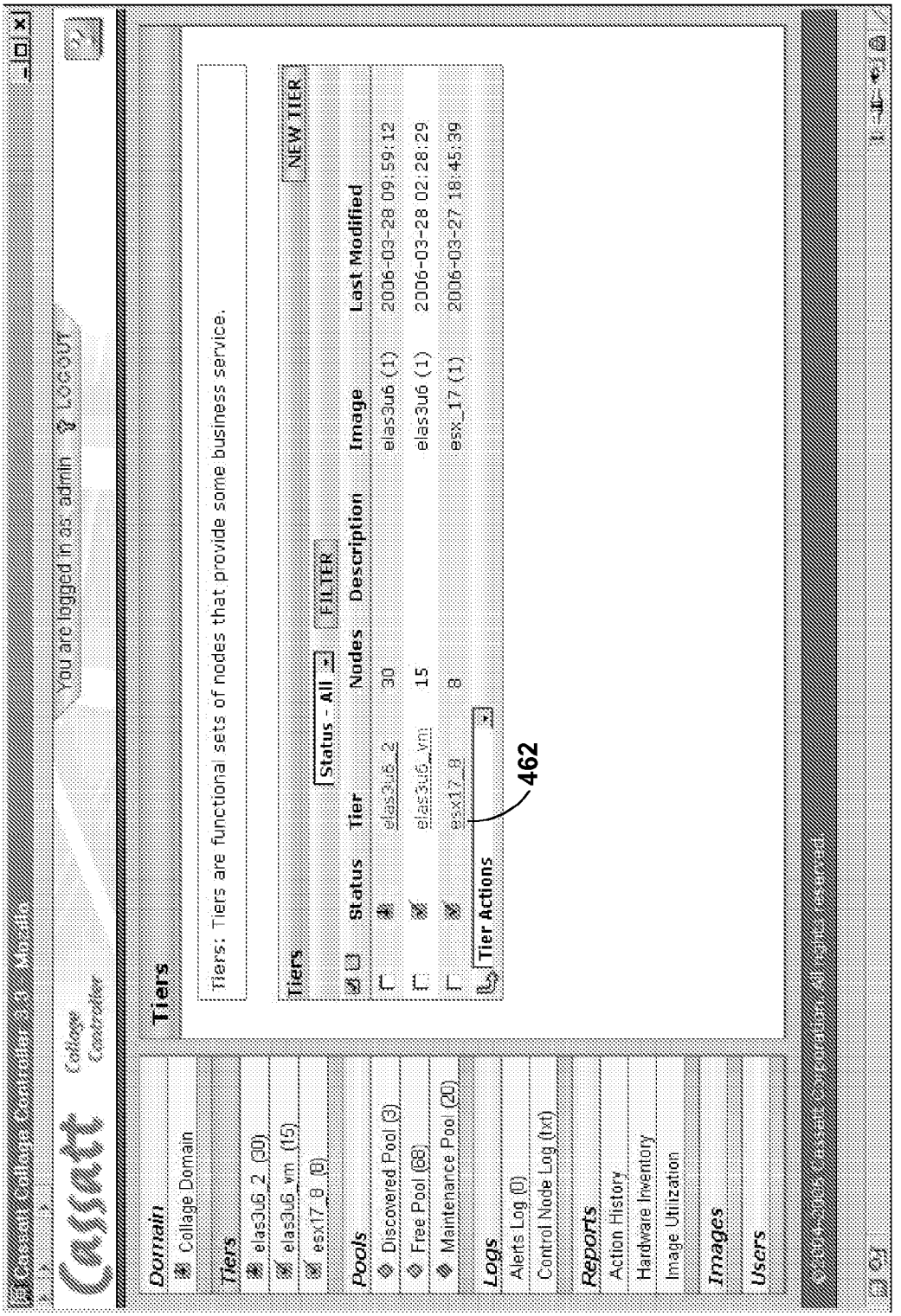

460
462
Tiers
Tiers: Tiers are functional sets of nodes that provide some business service.
NEW TIER
Status - All
FILTER
Tier Actions
Domain
Tiers
Pools
Logs
Reports
Images
Users

FIG. 28

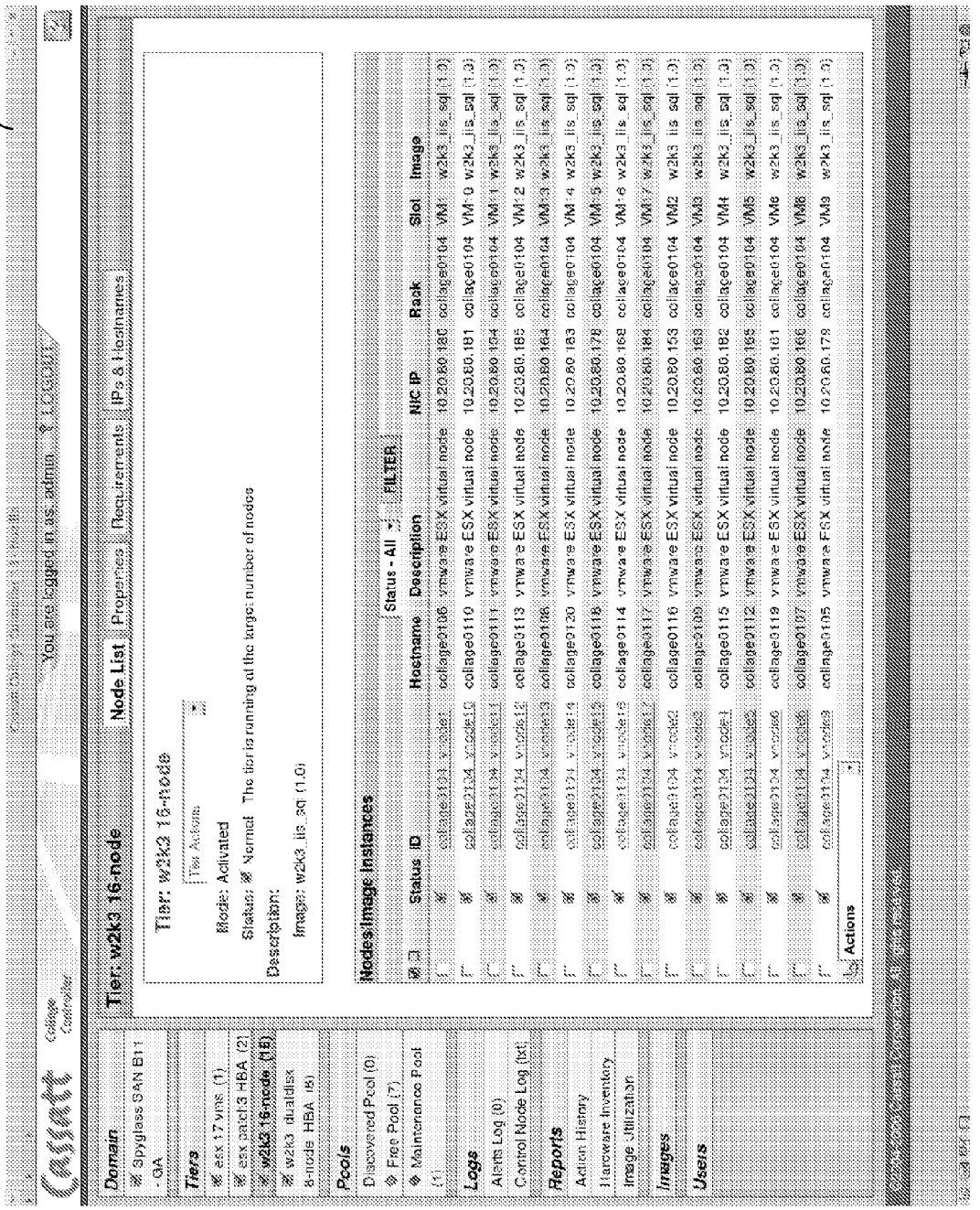

470
Tier: w2k3 16-node
Node List
Properties
Requirements
IPs & Hostnames
Nodes/image instances
FILTER
Actions

FIG. 29

DISTRIBUTED COMPUTING SYSTEM HAVING AUTONOMIC DEPLOYMENT OF VIRTUAL MACHINE DISK IMAGES

This application claims the benefit of U.S. provisional Application Ser. No. 60/787,280, filed Mar. 30, 2006, the entire content of which is incorporated herein by reference.

TECHNICAL FIELD

The invention relates to computing environments and, more specifically, to distributed computing systems.

BACKGROUND

Distributed computing systems are increasingly being utilized to support business as well as technical applications. Typically, distributed computing systems are constructed from a collection of computing nodes that combine to provide a set of processing services to implement the distributed computing applications. Each of the computing nodes in the distributed computing system is typically a separate, independent computing device interconnected with each of the other computing nodes via a communications medium, e.g., a network.

One challenge with distributed computing systems is the organization, deployment and administration of such a system within an enterprise environment. For example, it is often difficult to manage the allocation and deployment of enterprise computing functions within the distributed computing system. An enterprise, for example, often includes several business groups, and each group may have competing and variable computing requirements.

SUMMARY

In general, the invention is directed to a distributed computing system that conforms to a multi-level, hierarchical organizational model. One or more control nodes provide for the efficient and automated allocation and management of computing functions and resources within the distributed computing system in accordance with the organization model.

As described herein, the model includes four distinct levels: fabric, domains, tiers and nodes that provide for the logical abstraction and containment of the physical components as well as system and service application software of the enterprise. A user, such as a system administrator, interacts with the control nodes to logically define the hierarchical organization of the distributed computing system. The control nodes are responsible for all levels of management in accordance with the model, including fabric management, domain creation, tier creation and node allocation and deployment.

In one embodiment, a distributed computing system comprises a plurality of application nodes interconnected via a communications network and a control node. The control node comprises a set of one or more applications to be executed on the application nodes, an application matrix that includes parameters for controlling the deployment of the applications within the distributed computing system, and an automation unit having one or more rule engines that provide autonomic control of the application nodes and the applications in accordance with a set of one or more rules and the application matrix.

In another embodiment, a method comprises generating an application matrix that specifies data for controlling the deployment of a set of applications within a distributed computing system, and performing operations to provide autonomic control over the deployment of the applications within the distributed computing system in accordance with parameters of the application matrix.

In another embodiment, a computer-readable medium comprises instructions. The instructions cause the processor to generate an application matrix that specifies data for controlling the deployment of a set of applications within a distributed computing system, and perform operations to provide autonomic control over the deployment of the applications within the distributed computing system in accordance with parameters of the application matrix.

In another embodiment, a distributed computing system comprises a plurality of application nodes interconnected via a communications network. In addition, the distributed computing system includes a software image repository storing: (i) one or more image instances of a virtual machine manager that is executable on the application nodes, wherein when executed on the applications nodes, the image instances of the virtual machine manager provide one or more virtual machines, and (ii) one or more image instances of one or more software applications that are executable on the virtual machines. The distributed computing system also includes a control node that comprises an automation infrastructure to provide autonomic deployment of the image instances of the virtual machine manager on the application nodes and to provide autonomic deployment of the image instances of the software applications on the virtual machines.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

BRIEF DESCRIPTION OF DRAWINGS

FIG. **1** is a block diagram illustrating a distributed computing system constructed from a collection of computing nodes.

FIG. **2** is a schematic diagram illustrating an example of a model of an enterprise that logically defines an enterprise fabric.

FIG. **3** is a flow diagram that provides a high-level overview of the operation of a control node when configuring the distributed computing system.

FIG. **4** is a flow diagram illustrating exemplary operation of the control node when assigning computing nodes to node slots of tiers.

FIG. **5** is a flow diagram illustrating exemplary operation of a control node when adding an additional computing node to a tier to meet additional processing demands.

FIG. **6** is a flow diagram illustrating exemplary operation of a control node harvesting excess node capacity from one of the tiers and returning the harvested computing node to the free pool.

FIG. **7** is a screen illustration of an exemplary user interface for defining tiers in a particular domain.

FIG. **8** is a screen illustration of an exemplary user interface for defining properties of the tiers.

FIG. **9** is a screen illustration of an exemplary user interface for viewing and identify properties of a computing node.

FIG. **10** is a screen illustration of an exemplary user interface for viewing software images.

FIG. **11** is a screen illustration of an exemplary user interface for viewing a hardware inventory report.

FIG. **12** is a screen illustration of an exemplary user interface for viewing discovered nodes that are located in the free pool.

FIG. **13** is a screen illustration of an exemplary user interface for viewing users of a distributed computing system.

FIG. **14** is a screen illustration of an exemplary user interface for viewing alerts for the distributed computing system.

FIG. **15** is a block diagram illustrating one embodiment of control node that includes a monitoring subsystem, a service level automation infrastructure (SLAI), and a business logic tier (BLT).

FIG. **16** is a block diagram illustrating one embodiment of the monitoring subsystem.

FIG. **17** is a block diagram illustrating one embodiment of the SLAI in further detail.

FIG. **18** is a block diagram of an example working memory associated with rule engines of the SLAI.

FIG. **19** is a block diagram illustrating an example embodiment for the BLT of the control node.

FIG. **20** is a block diagram illustrating one embodiment of a rule engine in further detail.

FIG. **21** is a block diagram illustrating another example embodiment of the control node.

FIG. **22** is a flowchart illustrating an exemplary mode of initiating a control node utilizing an application matrix.

FIG. **23** is a block diagram illustrating an exemplary application matrix.

FIG. **24** is a block diagram illustrating an exemplary configuration of a physical node to which a virtual machine manager and one or more virtual machines are deployed.

FIG. **25** is a flowchart illustrating an exemplary procedure for creating and capturing an image of a virtual machine manager for autonomic deployment within the distributed computing system.

FIG. **26** is a flowchart illustrating an exemplary procedure for creating and capturing a virtual disk image file for autonomic deployment within the distributed computing system.

FIG. **27**A is a flowchart illustrating an exemplary procedure for deploying a virtual machine manager to a node within the distributed computing system.

FIG. **27**B is a flowchart illustrating an exemplary procedure for deploying an operating system and application image to a node that is executing a virtual machine manager within the distributed computing system.

FIG. **28** is a screen image illustrating an exemplary tier control interface for the embodiment described in FIGS. **24** through **27**.

FIG. **29** is a screen image illustrating an exemplary node control interface for the embodiment described in FIGS. **24** through **27**.

DETAILED DESCRIPTION

FIG. **1** is a block diagram illustrating a distributed computing system **10** constructed from a collection of computing nodes. Distributed computing system **10** may be viewed as a collection of computing nodes operating in cooperation with each other to provide distributed processing.

In the illustrated example, the collection of computing nodes forming distributed computing system **10** are logically grouped within a discovered pool **11**, a free pool **13**, an allocated tiers **15** and a maintenance pool **17**. In addition, distributed computing system **10** includes at least one control node **12**.

Within distributed computing system **10**, a computing node refers to the physical computing device. The number of computing nodes needed within distributed computing system **10** is dependent on the processing requirements. For example, distributed computing system **10** may include 8 to 512 computing nodes or more. Each computing node includes one or more programmable processors for executing software instructions stored on one or more computer-readable media.

Discovered pool **11** includes a set of discovered nodes that have been automatically "discovered" within distributed computing system **10** by control node **12**. For example, control node **12** may monitor dynamic host communication protocol (DHCP) leases to discover the connection of a node to network **18**. Once detected, control node **12** automatically inventories the attributes for the discovered node and reassigns the discovered node to free pool **13**. The node attributes identified during the inventory process may include a CPU count, a CPU speed, an amount of memory (e.g., RAM), local disk characteristics or other computing resources. Control node **12** may also receive input identifying node attributes not detectable via the automatic inventory, such as whether the node includes I/O, such as HBA. Further details with respect to the automated discovery and inventory processes are described in U.S. patent application Ser. No. 11/070,851, entitled "AUTOMATED DISCOVERY AND INVENTORY OF NODES WITHIN AN AUTONOMIC DISTRIBUTED COMPUTING SYSTEM," filed Mar. 2, 2005, the entire content of which is hereby incorporated by reference.

Free pool **13** includes a set of unallocated nodes that are available for use within distributed computing system **10**. Control node **12** may dynamically reallocate an unallocated node from free pool **13** to allocated tiers **15** as an application node **14**. For example, control node **12** may use unallocated nodes from free pool **13** to replace a failed application node **14** or to add an application node to allocated tiers **15** to increase processing capacity of distributed computing system **10**.

In general, allocated tiers **15** include one or more tiers of application nodes **14** that are currently providing a computing environment for execution of user software applications. In addition, although not illustrated separately, application nodes **14** may include one or more input/output (I/O) nodes. Application nodes **14** typically have more substantial I/O capabilities than control node **12**, and are typically configured with more computing resources (e.g., processors and memory). Maintenance pool **17** includes a set of nodes that either could not be inventoried or that failed and have been taken out of service from allocated tiers **15**.

Control node **12** provides the system support functions for managing distributed computing system **10**. More specifically, control node **12** manages the roles of each computing node within distributed computing system **10** and the execution of software applications within the distributed computing system. In general, distributed computing system **10** includes at least one control node **12**, but may utilize additional control nodes to assist with the management functions.

Other control nodes **12** (not shown in FIG. **1**) are optional and may be associated with a different subset of the computing nodes within distributed computing system **10**. Moreover, control node **12** may be replicated to provide primary and backup administration functions, thereby allowing for graceful handling a failover in the event control node **12** fails.

Network **18** provides a communications interconnect for control node **12** and application nodes **14**, as well as discovered nodes, unallocated nodes and failed nodes. Communications network **18** permits internode communications among the computing nodes as the nodes perform interrelated operations and functions. Communications network **18** may comprise, for example, direct connections between one or more of the computing nodes, one or more customer networks

maintained by an enterprise, local area networks (LANs), wide area networks (WANs) or a combination thereof. Communications network **18** may include a number of switches, routers, firewalls, load balancers, and the like.

In one embodiment, each of the computing nodes within distributed computing system **10** executes a common general-purpose operating system. One example of a general-purpose operating system is the Windows™ operating system provided by Microsoft Corporation. In some embodiments, the general-purpose operating system such as the Linux kernel may be used.

In the example of FIG. **1**, control node **12** is responsible for software image management. The term "software image" refers to a complete set of software loaded on an individual computing node to provide an execution environment for one or more applications. The software image including the operating system and all boot code, middleware files, and may include application files. As described below embodiments of the invention provide application-level autonomic control over the deployment, execution and monitoring of applications onto software images associated with application nodes **14**.

System administrator **20** may interact with control node **12** and identify the particular types of software images to be associated with application nodes **14**. Alternatively, administration software executing on control node **12** may automatically identify the appropriate software images to be deployed to application nodes **14** based on the input received from system administrator **20**. For example, control node **12** may determine the type of software image to load onto an application node **14** based on the functions assigned to the node by system administrator **20**. Application nodes **14** may be divided into a number of groups based on their assigned functionality. As one example, application nodes **14** may be divided into a first group to provide web server functions, a second group to provide business application functions and a third group to provide database functions. The application nodes **14** of each group may be associated with different software images.

Control node **12** provides for the efficient allocation and management of the various software images within distributed computing system **10**. In some embodiments, control node **12** generates a "golden image" for each type of software image that may be deployed on one or more of application nodes **14**. As described herein, the term "golden image" refers to a reference copy of a complete software stack for providing an execution environment for applications.

System administrator **20** may create a golden image by installing an operating system, middleware and software applications on a computing node and then making a complete copy of the installed software. In this manner, a golden image may be viewed as a "master copy" of the software image for a particular computing function. Control node **12** maintains a software image repository **26** that stores the golden images associated with distributed computing system **10**.

Control node **12** may create a copy of a golden image, referred to as an "image instance," for each possible image instance that may be deployed within distributed computing system **10** for a similar computing function. In other words, control node **12** pre-generates a set of K image instances for a golden image, where K represents the maximum number of image instances for which distributed computing system **10** is configured for the particular type of computing function. For a given computing function, control node **12** may create the complete set of image instance even if not all of the image instances will be initially deployed. Control node **12** creates different sets of image instances for different computing functions, and each set may have a different number of image instances depending on the maximum number of image instances that may be deployed for each set. Control node **12** stores the image instances within software image repository **26**. Each image instance represents a collection of bits that may be deployed on an application node.

Further details of software image management are described in co-pending U.S. patent application Ser. No. 11/046,133, entitled "MANAGEMENT OF SOFTWARE IMAGES FOR COMPUTING NODES OF A DISTRIBUTED COMPUTING SYSTEM," filed Jan. 28, 2005 and co-pending U.S. patent application Ser. No. 11/046,152, entitled "UPDATING SOFTWARE IMAGES ASSOCIATED WITH A DISTRIBUTED COMPUTING SYSTEM," filed Jan. 28, 2005, each of which is incorporated herein by reference in its entirety.

In general, distributed computing system **10** conforms to a multi-level, hierarchical organizational model that includes four distinct levels: fabric, domains, tiers and nodes. Control node **12** is responsible for all levels of management, including fabric management, domain creation, tier creation and node allocation and deployment.

As used herein, the "fabric" level generally refers to the logical constructs that allow for definition, deployment, partitioning and management of distinct enterprise applications. In other words, fabric refers to the integrated set of hardware, system software and application software that can be "knitted" together to form a complete enterprise system. In general, the fabric level consists of two elements: fabric components or fabric payload. Control node **12** provides fabric management and fabric services as described herein.

In contrast, a "domain" is a logical abstraction for containment and management within the fabric. The domain provides a logical unit of fabric allocation that enables the fabric to be partitioned amongst multiple uses, e.g. different business services.

Domains are comprised of tiers, such as a 4-tier application model (web server, application server, business logic, persistence layer) or a single tier monolithic application. Fabric domains contain the free pool of devices available for assignment to tiers.

A tier is a logically associated group of fabric components within a domain that share a set of attributes: usage, availability model or business service mission. Tiers are used to define structure within a domain e.g. N-tier application, and each tier represents a different computing function. A user, such as administrator **20**, typically defines the tier structure within a domain. The hierarchical architecture may provide a high degree of flexibility in mapping customer applications to logical models which run within the fabric environment. The tier is one construct in this modeling process and is the logical container of application resources.

The lowest level, the node level, includes the physical components of the fabric. This includes computing nodes that, as described above, provide operating environments for system applications and enterprise software applications. In addition, the node level may include network devices (e.g., Ethernet switches, load balancers and firewalls) used in creating the infrastructure of network **18**. The node level may further include network storage nodes that are network connected to the fabric.

System administrator **20** accesses administration software executing on control node **12** to logically define the hierarchical organization of distributed computing system **10**. For example, system administrator **20** may provide organizational data **21** to develop a model for the enterprise and

logically define the enterprise fabric. System administrator **20** may, for instance, develop a model for the enterprise that includes a number of domains, tiers, and node slots hierarchically arranged within a single enterprise fabric.

More specifically, system administrator **20** defines one or more domains that each correspond to a single enterprise application or service, such as a customer relation management (CRM) service. System administrator **20** further defines one or more tiers within each domain that represent the functional subcomponents of applications and services provided by the domain. As an example, system administrator **20** may define a storefront domain within the enterprise fabric that includes a web tier, an application tier and a database tier. In this manner, distributed computing system **10** may be configured to automatically provide web server functions, business application functions and database functions.

For each of the tiers, control node **12** creates a number of "node slots" equal to the maximum number of application nodes **14** that may be deployed. In general, each node slot represents a data set that describes specific information for a corresponding node, such as software resources for a physical node that is assigned to the node slot. The node slots may, for instance, identify a particular software image instance associated with an application node **14** as well as a network address associated with that particular image instance.

In this manner, each of the tiers include one or more node slots that reference particular software image instances to boot on the application nodes **14** to which each software image instance is assigned. The application nodes **14** to which control node **12** assigns the image instances temporarily inherit the network address assigned to the image instance for as long as the image instance is deployed on that particular application node. If for some reason the image instance is moved to a different application node **14**, control node **12**A moves the network address to that new application node.

System administrator **20** may further define specific node requirements for each tier of the fabric. For example, the node requirements specified by system administrator **20** may include a central processing unit (CPU) count, a CPU speed, an amount of memory (e.g., RAM), local disk characteristics and other hardware characteristics that may be detected on the individual computing nodes. System administrator **20** may also specify user-defined hardware attributes of the computing nodes, such as whether I/O (like HBA) is required. The user-defined hardware attributes are typically not capable of detection during an automatic inventory. In this manner, system administrator **20** creates a list of attributes that the tier requires of its candidate computing nodes. In addition, particular node requirements may be defined for software image instances.

In addition to the node requirements described above, system administrator **20** may further define policies that are used when re-provisioning computing nodes within the fabric. System administrator **20** may define policies regarding tier characteristics, such as a minimum number of nodes a tier requires, an indication of whether or not a failed node is dynamically replaced by a node from free pool **13**, a priority for each tier relative to other tiers, an indication of whether or not a tier allows nodes to be re-provisioned to other tiers to satisfy processing requirements by other tiers of a higher priority or other policies. Control node **12** uses the policy information input by system administrator **20** to re-provision computing nodes to meet tier processing capacity demands.

After receiving input from system administrator **20** defining the architecture and policy of the enterprise fabric, control node **12** identifies unallocated nodes within free pool **13** that satisfy required node attributes. Control node **12** automatically assigns unallocated nodes from free pool **13** to respective tier node slots of a tier. As will be described in detail herein, in one embodiment, control node **12** may assign computing nodes to the tiers in a "best fit" fashion. Particularly, control node **12** assigns computing nodes to the tier whose node attributes most closely match the node requirements of the tier as defined by administrator **20**. The assignment of the computing nodes may occur on a tier-by-tier basis beginning with a tier with the highest priority and ending with a tier with the lowest priority. Alternatively, or in addition, assignment of computing nodes may be based on dependencies defined between tiers.

As will be described in detail below, control node **12** may automatically add unallocated nodes from free pool **13** to a tier when more processing capacity is needed within the tier, remove nodes from a tier to the free pool when the tier has excess capacity, transfer nodes from tier to tier to meet processing demands, or replace failed nodes with nodes from the free pool. Thus, computing resources, i.e., computing nodes, may be automatically shared between tiers and domains within the fabric based on user-defined policies to dynamically address high-processing demands, failures and other events.

Distributed computing system **10** may provide one or more advantages. For example, distributed computing system **10** may deliver application services in a manner that is independent of the computational infrastructure which generates the application services. For example, the computation infrastructure that provides an application service to a user of distributed computing system **10** may be irrelevant to the user, so long as distributed computing system provides the application service. Distributed computing system **10** may deliver application services in a manner that is independent of the computational infrastructure by utilizing an infrastructure management facility (IMF) that may guarantee application service delivery using "service level automation."

This concept is itself based upon two concepts: The Service Delivery Model and Service Level Automation.

1. Service Delivery Model: The concept of computing as a service is based upon the ability of the consumer to be guaranteed availability of the quantity and quality of an application service required at the time and location necessary, independent of the type of service provided (e.g. audio, video, application programs, etc). This may be achieved if the application services are provided in a manner that is independent of the location and condition of the physical and logical infrastructure which generates the application service, limited only by total capability and capacity available. This disclosure may refer to this concept as the "Service Delivery Model."

Physical and logical infrastructure facilities that provide an application service may comprise a "computing infrastructure." A "computing infrastructure" may include computing, storage and communication hardware and related software infrastructure; the application software and application containers; and other enabling facilities and services that are part of the environment necessary to generate and deliver application services.

2. Service Level Automation: In order to implement the service delivery model, the IMF may provide service level automation. The IMF may implement the service delivery model for single and/or multiple application service flows. For example, the IMF may automatically and dynamically adjust application service levels of these application service flows based on a policy automation facility. The policy automation facility may automate delivery of application service based upon dynamically extensible policies. These dynami-

cally extensible policies may include but are not limited to: customer, location, priority, duration, date and time of day, service type, business demands and conditions, quantity and quality of the application service. The policy automation facility may also enable independent control of the application service by a service provider and by a consumer. Furthermore, the policy automation facility may allow the consumer provide the application service or a service based on the application service to another consumer. In addition, the IMF may measure both efficiency and cost as reflected in the behavior of the product. For example, the IMF may provide the ability to make decisions that account for the best uses of the resources on a dynamic basis in order to choose between functionally equivalent alternatives that make the best usage of the economics of the infrastructure at the time. This disclosure may refer to this concept as "Service Level Automation" (SLA).

Service-level automation may enable the separation of some or all aspects of the implementation of the computing infrastructure from the delivery of application services. Separating the implementation of the computing infrastructure from the delivery of application services may allow a consumer of the application services to decrease risk and cost while increasing business agility. Implementations of service-level automation may be based upon a new abstraction which operates independent of existing software categories (e.g. applications, middleware, data base, operating systems and system management) which for the purposes of this disclosure shall be referred to as "Metaware."

The IMF may comprise Metaware that regulates the delivery of application services through service-level automation. The IMF may use service-level automation to regulate the delivery of application services in a manner that is independent of the computing infrastructure that provides the application services. Because application services are the result of applications running within distributed computing system **10**, the IMF may automate application service levels and may make independent of all aspects of the computing infrastructure within the limits of the capability and capacity available for delivery to the consumer. Service-level automation may provide extensible, dynamic policy-driven control of service delivery for both service providers and service consumers. The IMF may include the following six functional design capabilities:

1. Application Service Independence: The basic concept which enables the IMF to deliver application services may be the ability of the IMF to dynamically manage service-level automation in a manner that is independent of the computing infrastructure. A key element of this is that application services themselves are independent and unmodified. The IMF may be able to accomplish this by abstracting the physical and logical computing infrastructure away from the quantity and quality of application services generated by the computing infrastructure.

2. Scale Independence: The incremental overhead of the IMF required by any server included in distributed computing system **10** may be independent of the number of nodes in distributed computing system **10**.

3. Container Capacity Metering: Distributed computing system **10** may achieve application service independence by dynamically metering higher-level software facilities of the computing infrastructure. The higher-level software facilities may generate the application services, service flow and flux. This disclosure may refer to these facilities as "container services." Examples of container services may include, but are not limited to, virtual machine managers, operating systems, and web application servers. Furthermore, some container services may contain other container services that the IMF manages. The system may conceptualize the application service as being independent of the type of computing infrastructure (e.g., middleware and hardware nodes) that generate these services.

4. Policy Automation: The IMF may deliver the required service-level automation through its extensible policy management facility. The policy management facility may dynamically adjust application service levels in order to implement user-defined policy. The ability to adjust application service levels may be limited only by the capacity and capability of the available computing infrastructure. The policy management facility may be capable of providing guaranteed service levels for single and/or multiple application service flows which the IMF automatically and dynamically adjust based on user-defined and extensible policies in response to changes in the environment. Examples of policies may include, but are not limited to: customer, location, service type, cost and/or pricing, quantity and quality of service, priority, duration, date and time of day, business demands and conditions, and other policies. The policy management facility may also allow the service provider to manage delivery and meter the capacity that the service provider delivers to individual customers. The policy management facility may allow the service provider to manage delivery and meter capacity independent of the ability of the customer to set and manage their own policies of how their capacity is utilized. This may be effectively done by leveraging application service independence.

5. Dynamic Provisioning: The IMF described herein may support the ability to dynamically provision at boot and run time all software in order to support installation, capacity management, service levels and other policies, and software upgrades. Offering a computing infrastructure architecture that is application service independent may create flexibility in deployment of a single instance of a service.

6. Data Capture Capability: The IMF described herein may acquire the data necessary to enable the service provider to implement metering and auditing facilities necessary to control, audit, bill and account for services provided in complete detail. The customer may also use the ability to capture this data as a means to educate the IMF about the performance characteristics of combinations of resources in order to allow the IMF to improve the quality of its scheduling decisions over time. Improvement of the quality of scheduling decisions may enhance the ability of the IMF to provide the requisite services with high efficiency.

The described system may allow customers to conveniently acquire the guaranteed quantity and quality of application services required at competitive market-driven rates with the ability to directly control the SLA capability of the IMF in order to maximize their operations efficiency and effectiveness without having to tradeoff operation risk, cost and business agility. Potential advantages may include, but are not limited to:

1. Efficiency and Effectiveness: The customer may not need to manage the technology, and can focus on the automation of the application service to improve their efficiency and effectiveness without having to tradeoff operation risk, cost and business agility which the IMF may enable through the service-level automation capability of the IMF.

2. Capabilities & Convenience: By leveraging the Internet as a generic, global delivery facility and separating the computing infrastructure from the application service delivery, the IMF may simplify the consumer's life in several ways:

i. Complexity: The IMF may eliminate the complexity of purchasing, installing, configuring, managing and administering the computing infrastructure.

ii. Access: The IMF may provide ubiquitous access to a full range of computing services wherever and whenever desired from any client device capable of accessing the Internet.

iii. Service Provider Independence: Consumers may be free to change providers by simply transferring their software images to a new service provider.

iv. Service Management: Service management may enable a consumer to provide services to customers of the consumer. For example, a corporation may be able to regulate and bill their individual business units for the capacity consumed.

3. Cost: By logically abstracting the application services from the computing infrastructure, the IMF may enable a competitive service provider industry that includes:

i. Capacity Utilization: A service provider using the IMF may be able to charge customers for capacity actually utilized by the customers. Presently, customers typically pay for the maximum capacity they would need at peak load.

ii. Choice of Suppliers: As the basic computing infrastructure facilities may be generic, the customer may have a choice of suppliers without the level of lock-in current vendors may enjoy. This may be enabled through the image and policy management implementation of the IMF. The image and policy management implementation of the IMF may allow portability across service providers through by enabling the customer to simply transfer the files to the new provider and initiating it with the IMF Metaware. The ability to easily migrate from a first service provider to a second service provider may drive the pricing competition. In this way, the ability of the IMF to provide application services independent of the computing infrastructure may enable customers to choose a service provider more effectively.

iii. Capital Expense Leverage: The customer may not need to own the capital equipment or software. Rather the service provider may be motivated to use the most cost effective components and amortize costs of these components across a customer base of the service provider. As a result, the cost to the consumer may be lower.

iv. Operations Expense Leverage: As the IMF automates most of the information technology operations functions, these costs may drop dramatically.

The IMF may enable the implementation of a service delivery model based on service-level automation by service providers:

1. A service provider may use the IMF to implement a service delivery model for application services independent of the computing infrastructure or of the service types provided. This capability may enable the service provider to deliver service-level automation on a global basis utilizing generic computing infrastructure for all types of digital services. Such digital service may include traditional applications and service flow processing, digital audio and digital video services independently to any consumer type (e.g. individual, organization or corporation) as long as the consumer has access to sufficient capacity of broadband communications for the services levels require, and other types of digital services. This capability of the IMF may also allow a consumer of an application service to, in turn, be a service provider as well.

2. The IMF described herein may enable a service provider to meter application services that distributed computing system **10** delivers. For example, the service provider may meter application services based on the capacity of computing infrastructure consumed by customer. The service provider may meter application services in order to control, audit, bill and account for the application services provided to individual consumers of the application services. This capability may represent a uniform model of service-level automation that is independent of the capacity of computing infrastructure for some or all types of application services. As a result, a pricing model of a service provider may reflect the goal of utility computing by enabling the service provider to bill based upon a combination of application service delivery parameters. These application service delivery parameters may include, but not limited to: customer, type of service, duration, date and/or time of day, location, point of delivery, quantity and quality of service and excess or reserved capacity, and other application service delivery parameters.

3. The IMF may also enable the service provider to leverage the economics of the service delivery model in several ways:

i. Utilization: By managing the computing infrastructure with the IMF, a service provider may be able to maintain much higher average utilization levels of the computing infrastructure. The maintenance of higher average utilization levels may dramatically reduce the amount of hardware purchased, software licensed and operations expenses.

ii. Global Service Demand Levels: Demand for an application service by an organization may vary by local time of day. By leveraging the global reach of the Internet, a service provider may be able to effectively manage the utilization of the computing infrastructure necessary to meet demand for the application service on a global "follow the sun" basis.

iii. Service Digitization: As more services become digital, service providers may deliver these services using a generic computing infrastructure architecture.

iv. Value-Added Services: A service provider may be free to provide value-added services beyond basic application services. A few examples of value-added services may include: a bundle of personal computer services that includes applications, storage, security, administration and related services; software as a service; compliance verification and certification, and other types of value-added services.

v. Hardware Commoditization: Hardware commoditization may enable a service provider to implement a computing infrastructure using generic low cost servers, storage facilities, and network switches. These hardware elements are undergoing dramatic commoditization which may allows for much lower capital expenses then in the past. This commoditization may continue for many years to come making capital expenses a much smaller part of the fixed costs for the service provider.

vi. Automation of Operations: One of the primary economic values of the IMF is that IMF may automate most of the runtime operations costs of the computing infrastructure. Operations expenses may constitute the majority of the costs of today. Because the IMF automates the runtime operations, the IMF may reduce runtime operations costs. Thus by leveraging the ability of the IMF to dramatically reduce the capital and operations expenses, the service provider may be able to implement an service delivery model that may justify the consumers switching cost.

vii. Software Commoditization: Software pricing is being commoditized based upon several factors which may continue to accelerate. These factors may include:

1. Service-Oriented Architecture: Software systems that utilize a service-oriented architecture may accelerate innovation, allow for incremental independent innovation, eliminate down stream lock in by software vendors, and may make it possible to use commodity servers to run applications that previously required large expensive servers. Service-oriented architecture may also reduce complex applications to their most fundamental independently divisible functional services which will be the basis for the complete commoditization of software further leveraging the open source movement.
2. Open Source Software: The open source software paradigm may serve as an accelerator for the commoditization of many types of software by both compromising the pricing power of software and serving as a reusable model of software services which may further compromise the value of horizontal software services.
3. Good Enough: Application services are becoming more and more generic, thus accelerating their commoditization further in combination with service-oriented architectures and Open Source.

FIG. **2** is a schematic diagram illustrating an example embodiment of organizational data **21** that defines a model logically representing an enterprise fabric in accordance with the invention. In the example illustrated in FIG. **2**, control node **12** (FIG. **1**) maintains organizational data **21** to define a simple e-commerce fabric **32**.

In this example, e-commerce fabric **32** includes a storefront domain **34**A and a financial planning domain **34**B. Storefront domain **34**A corresponds to the enterprise storefront domain and allows customers to find and purchase products over a network, such as the Internet. Financial planning domain **34**B allows one or more employees to perform financial planning tasks for the enterprise.

Tier level **31**C includes one or more tiers within each domain that represent the functional subcomponents of applications and services provided by the domain. For example, storefront domain **34**A includes a web server tier (labeled "web tier") **36**A, a business application tier (labeled "app tier") **36**B, and a database tier (labeled "DB tier") **36**C. Web server tier **36**A, business application tier **36**B and database tier **36**C interact with one another to present a customer with an online storefront application and services. For example, the customer may interact with web server tier **36**A via a web browser. When the customer searches for a product, web server tier **36**A may interacts with business application tier **36**B, which may in turn access a database tier **36**C. Similarly, financial planning domain **34**B includes a financial planning tier **36**D that provides subcomponents of applications and services of the financial planning domain **34**B. Thus, in this example, a domain may include a single tier.

Tier level **31**D includes one or more logical node slots **38**A-**38**H ("node slots **38**") within each of the tiers. Each of node slots **38** include node specific information, such as software resources for an application node **14** that is assigned to a respective one of the node slots **38**. Node slots **38** may, for instance, identify particular software image instances within image repository **26** and map the identified software image instances to respective application nodes **14**. As an example, node slots **38**A and **38**B belonging to web server tier **36**A may reference particular software image instances used to boot two application nodes **14** to provide web server functions. Similarly, the other node slots **38** may reference software image instances to provide business application functions, database functions, or financial application functions depending upon the tier to which the node slots are logically associated.

Although in the example of FIG. **2**, there are two node slots **38** corresponding to each tier, the tiers may include any number of node slots depending on the processing capacity needed on the tier. Furthermore, not all of node slots **38** may be currently assigned to an application node **14**. For example, node slot **28**B may be associated with an inactive software image instance and, when needed, may be assigned to an application node **14** for deployment of the software image instance.

In this example, organizational data **21** associates free node pool **13** with the highest-level of the model, i.e., e-commerce fabric **32**. As described above, control node **12** may automatically assign unallocated nodes from free node pool **13** to at least a portion of tier node slots **38** of tiers **36** as needed using the "best fit" algorithm described above or another algorithm. Additionally, control node **12** may also add nodes from free pool **13** to a tier when more processing capacity is needed within the tier, remove nodes from a tier to free pool **13** when a tier has excess capacity, transfer nodes from tier to tier to meet processing demands, and replace failed nodes with nodes from the free tier.

Although not illustrated, the model for the enterprise fabric may include multiple free node pools. For example, the model may associate free node pools with individual domains at the domain level or with individual tier levels. In this manner, administrator **20** may define policies for the model such that unallocated computing nodes of free node pools associated with domains or tiers may only be used within the domain or tier to which they are assigned. In this manner, a portion of the computing nodes may be shared between domains of the entire fabric while other computing nodes may be restricted to particular domains or tiers.

FIG. **3** is a flow diagram that provides a high-level overview of the operation of control node **12** when configuring distributed computing system **10**. Initially, control node **12** receives input from a system administrator defining the hierarchical organization of distributed computing system **10** (**50**). In one example, control node **12** receives input that defines a model that specifies a number of hierarchically arranged nodes as described in detail in FIG. **2**. Particularly, the defined architecture of distributed computing system **10** includes an overall fabric having a number of hierarchically arranged domains, tiers and node slots.

During this process, control node **12** may receive input specifying node requirements of each of the tiers of the hierarchical model (**52**). As described above, administrator **20** may specify a list of attributes, e.g., a central processing unit (CPU) count, a CPU speed, an amount of memory (e.g., RAM), or local disk characteristics, that the tiers require of their candidate computing nodes. In addition, control node **12** may further receive user-defined custom attributes, such as requiring the node to have I/O, such as HBA connectivity. The node requirements or attributes defined by system administrator **20** may each include a name used to identify the characteristic, a data type (e.g., integer, long, float or string), and a weight to define the importance of the requirement.

Control node **12** identifies the attributes for all candidate computing nodes within free pool **13** or a lower priority tier (**54**). As described above, control node **12** may have already discovered the computing nodes and inventoried the candidate computing nodes to identify hardware characteristics of all candidate computing nodes. Additionally, control node **12**

may receive input from system administrator **20** identifying specialized capabilities of one or more computing nodes that are not detectable by the inventory process.

Control node **12** dynamically assigns computing nodes to the node slots of each tier based on the node requirements specified for the tiers and the identified node attributes (**56**). Population of the node slots of the tier may be performed on a tier-by-tier basis beginning with the tier with the highest priority, i.e., the tier with the highest weight assigned to it. As will be described in detail, in one embodiment, control node **12** may populate the node slots of the tiers with the computing nodes that have attributes that most closely match the node requirements of the particular tiers. Thus, the computing nodes may be assigned using a "best fit" algorithm.

FIG. **4** is a flow diagram illustrating exemplary operation of control node **12** when assigning computing nodes to node slots of tiers. Initially, control node **12** selects a tier to enable (**60**). As described above, control node **12** may select the tier based on a weight or priority assigned to the tier by administrator **20**. Control node **12** may, for example, initially select the tier with the highest priority and successively enable the tiers based on priority.

Next, control node **12** retrieves the node requirements associated with the selected tier (**62**). Control node **12** may, for example, maintain a database having entries for each node slot, where the entries identify the node requirements for each of the tiers. Control node **12** retrieves the node requirements for the selected tier from the database.

In addition, control node **12** accesses the database and retrieves the computing node attributes of one of the unallocated computing nodes of free pool **13**. Control node **12** compares the node requirements of the tier to the node attributes of the selected computing node (**64**).

Based on the comparison, control node **12** determines whether the node attributes of the computing node meets the minimum node requirements of the tier (**66**). If the node attributes of the selected computing node do not meet the minimum node requirements of the tier, then the computing node is removed from the list of candidate nodes for this particular tier (**68**). Control node **12** repeats the process by retrieving the node attributes of another of the computing nodes of the free pool and compares the node requirements of the tier to the node attributes of the computing node.

If the node attributes of the selected computing node meet the minimum node requirements of the tier (YES of **66**), control node **12** determines whether the node attributes are an exact match to the node requirements of the tier (**70**). If the node attributes of the selected computing node and the node requirements of the tier are a perfect match (YES of **70**), the computing node is immediately assigned from the free pool to a node slot of the tier and the image instance for the slot is associated with the computing node for deployment (**72**).

Control node **12** then determines whether the node count for the tier is met (**74**). Control node **12** may, for example, determine whether the tier is assigned the minimum number of nodes necessary to provide adequate processing capabilities. In another example, control node **12** may determine whether the tier is assigned the ideal number of nodes defined by system administrator **20**. When the node count for the tier is met, control node **12** selects the next tier to enable, e.g., the tier with the next largest priority, and repeats the process until all defined tiers are enabled, i.e., populated with application nodes (**60**).

If the node attributes of the selected computing node and the node requirements of the tier are not a perfect match control node **12** calculates and records a "processing energy" of the node (**76**). As used herein, the term "processing energy" refers to a numerical representation of the difference between the node attributes of a selected node and the node requirements of the tier. A positive processing energy indicates the node attributes more than satisfy the node requirements of the tier. The magnitude of the processing energy represents the degree to which the node requirements exceed the tier requirements.

After computing and recording the processing energy of the nodes, control node **12** determines whether there are more candidate nodes in free pool **13** (**78**). If there are additional candidate nodes, control node **12** repeats the process by retrieving the computing node attributes of another one of the computing nodes of the free pool of computing nodes and comparing the node requirements of the tier to the node attributes of the computing node (**64**).

When all of the candidate computing nodes in the free pool have been examined, control node **12** selects the candidate computing node having the minimum positive processing energy and assigns the selected computing node to a node slot of the tier (**80**). Control node **12** determines whether the minimum node count for the tier is met (**82**). If the minimum node count for the tier has not been met, control node **12** assigns the computing node with the next lowest calculated processing energy to the tier (**80**). Control node **12** repeats this process until the node count is met. At this point, control node **12** selects the next tier to enable, e.g., the tier with the next largest priority (**60**).

In the event there are an insufficient number of computing nodes in free pool **13**, or an insufficient number of computing nodes that meet the tier requirements, control node **12** notifies system administrator **20**. System administrator **20** may add more nodes to free pool **13**, add more capable nodes to the free pool, reduce the node requirements of the tier so more of the unallocated nodes meet the requirements, or reduce the configured minimum node counts for the tiers.

FIG. **5** is a flow diagram illustrating exemplary operation of control node **12** when adding an additional computing node to a tier to meet increased processing demands. Initially, control node **12** or system administrator **20** identifies a need for additional processing capacity on one of the tiers (**90**). Control node **12** may, for example, identify a high processing load on the tier or receive input from a system administrator identifying the need for additional processing capacity on the tier.

Control node **12** then determines whether there are any computing nodes in the free pool of nodes that meet the minimum node requirements of the tier (**92**). When there are one or more nodes that meet the minimum node requirements of the tier, control node **12** selects the node from the free pool based the node requirements of the tier, as described above, (**94**) and assigns the node to the tier (**95**). As described in detail with respect to FIG. **4**, control node **12** may determine whether there are any nodes that have node attributes that are an exact match to the node requirements of the tier. If an exact match is found, the corresponding computing node is assigned to a node slot of the tier. If no exact match is found, control node **12** computes the processing energy for each node and assigns the computing node with the minimum processing energy to the tier. Control node **12** remotely powers on the assigned node and remotely boots the node with the image instance associated with the node slot. Additionally, the booted computing node inherits the network address associated with the node slot.

If there are no adequate computing nodes in the free pool, i.e., no nodes at all or no nodes that match the minimal node requirements of the tier, control node **12** identifies the tiers with a lower priority than the tier needing more processing capacity (**96**).

Control node **12** determines which of the nodes of the lower priority tiers meet the minimum requirements of the tier in need of processing capacity (**98**). Control node **12** may, for example, compare the attributes of each of the nodes assigned to node slots of the lower priority tiers to the node requirements of the tier in need of processing capacity. Lower priority tiers that have the minimum number of computing nodes may be removed from possible tiers from which to harvest an application node. If, however, all the lower priority tiers have the minimum number of computing nodes defined for the respective tier, the lowest priority tier is selected from which to harvest the one or more nodes.

Control node **12** calculates the processing energy of each of the nodes of the lower priority tiers that meet the minimum requirements (**100**). The energies of the nodes are calculated using the differences between the node attributes and the node requirements of the tier needing additional capacity. Control node **12** selects the computing node with the lowest processing energy that meets the minimum requirements, and assigns the selected computing node to the tier in need of processing capacity (**102**, **95**).

FIG. **6** is a flow diagram illustrating exemplary operation of control node **12** when harvesting excess node capacity from one of the tiers and returning the harvested computing node to free pool **13**. Initially, control node **12** identifies a tier having excess node capacity (**110**). Control node **12** may, for example, periodically check the node capacity of the tiers to identify any tiers having excess node capacity. Performing a periodic check and removal of excess nodes increases the likelihood that a capable computing node will be in free pool **13** in the event one of the tiers needs additional node capacity.

When harvesting a node, control node **12** calculates the processing energy of all the nodes in the tier as described above with reference to FIG. **4** (**112**). Control node **12** identifies the node within the tier with the highest processing energy and returns the identified node to the free pool of nodes (**114**, **116**). As described above, the node with the highest processing energy corresponds to the node whose node attributes are the most in excess of the node requirements of the tier.

Returning the node to the free pool may involve remotely powering off the computing node and updating the database to associate the harvested node with free pool **13**. In addition, control node **12** updates the database to disassociate the returned node with the node slot to which it was assigned. At this point, the node no longer uses the network address associated with the image instance mapped to the node slot. Control node **12** may, therefore, assign a temporary network address to the node while the node is assigned to free pool **13**.

FIG. **7** is a screen illustration of an exemplary user interface **120** presented by control node **12** with which administrator **20** interacts to define tiers for a particular domain. In the example illustrated in FIG. **7**, system administrator **20** has selected the "Collage Domain." User interface **120** presents the tiers that are currently in the selected domain. In the example illustrated, the Collage Domain includes three tiers, "test tier **1**," "test tier **2**," and "test tier **3**." As shown in FIG. **7**, in this example, each of the tiers includes two nodes. In addition, user interface **120** lists the type of software image currently deployed to application nodes for each of the tiers. In the example illustrated, image "applone (1.0.0)" is deployed to the nodes of test tier **1** and image "appltwo (1.0.0)" is deployed to the nodes of test tier **2**. System administrator **20** may add one or more tiers to the domain by clicking on new tier button **122**.

FIG. **8** is a screen illustration of an exemplary user interface **130** for defining properties of the tiers. In particular, user interface **130** allows system administrator **20** to input a name for the tier, a description of the tier, and an image associated with the tier. The image associated with the tier refers to a golden image from which image instances are generated and deployed to the nodes assigned to the tier.

When configuring a tier, system administrator **20** may elect to activate email alerts. For example, system administrator **20** may activate the email alerts feature in order to receive email alerts providing system administrator **20** with critical and/or non-critical tier information, such as a notification that a tier has been upgraded, a node of the tier has failed or the like. Furthermore, system administrator **20** may input various policies, such node failure rules. For example, system administrator **20** may identify whether control node **12** should reboot a node in case of failure or whether the failed node should automatically be moved to maintenance pool **17**. Similarly, system administrator **20** may identify whether nodes assigned to the tier may be harvested by other tiers.

User interface **130** may also allow system administrator **20** to input node requirements of a tier. In order to input node requirements of a tier, system administrator **20** may click on the "Requirements" tab **132**, causing user interface **130** to present an input area to particular node requirements of the tier.

FIG. **9** is a screen illustration of an exemplary user interface **140** for viewing and identifying properties of a computing node. User interface **140** allows system administrator **20** to define a name, description, and location (including a rack and slot) of a computing node. In addition user interface **140** may specify user-defined properties of a node, such as whether the computing node has I/O HBA capabilities.

User interface **140** also displays properties that control node **12** has identified during the computing node inventory process. In this example, user interface **140** presents system administrator **20** with the a CPU node count, a CPU speed, the amount of RAM, the disk size and other characteristics that are identifiable during the automated node inventory. User interface **140** additionally presents interface information to system administrator **20**. Specifically, user interface **140** provides system administrator **20** with a list of components and their associated IP and MAC addresses.

User interface **140** also allows system administrator **20** to define other custom requirements. For example, system administrator **20** may define one or more attributes and add those attributes to the list of node attributes presented to system administrator **20**.

FIG. **10** is a screen illustration of an exemplary user interface **150** for viewing software images. User interface **150** presents to a system administrator or another user a list of images maintained by control node **12** within image repository **26**. The image list further includes the status of each image (i.e., either active or inactive), the version of the image, the operating system on which the image should be run, the operating system version on which the image should be run and a brief description of the image.

System administrator **20** or another user may select an image by clicking on the box in front of the image identifier/name and perform one or more actions on the image. Actions that system administrator **20** may perform on an image include deleting the image, updating the image, and the like. System administrator **20** may select one of the image actions via dropdown menu **152**. In some embodiments, user interface **150** may further display other details about the images such as the node to which the images are assigned (if the node status is "active"), the network address associated with the images and the like.

FIG. **11** is a screen illustration of an exemplary user interface **160** for viewing a hardware inventory report. User interface **160** presents to system administrator **20** or another user a list of the nodes that are currently assigned to a domain. System administrator **20** may elect to view the nodes for the entire domain, for a single tier within the domain or for a single rack within a tier.

For each node, user interface **160** presents a node ID, a status of the node, the tier to which the node belongs, a hostname associated with the node, a NIC IP address, a rack location, a slot location, the number of CPU's of the node, the amount of RAM on the node, the number of disks on the node, whether the node has I/O HBA, and the number of NICs of the node.

System administrator **20** or other user may select a node by clicking on the box in front of the node identifier/name and perform one or more actions on the node. Actions that system administrator **20** may perform on the node include deleting the node, updating the node attributes or other properties of the node, and the like. System administrator **20** may select one of the node actions via dropdown menu **162**.

FIG. **12** is a screen illustration of an exemplary user interface **170** for viewing discovered nodes that are located in discovered pool **11**. For each node, user interface **170** presents a node ID, a state of the node, a NIC IP address, a rack location, a slot location, the number of CPU's of the node, the amount of RAM on the node, the number of disks on the node, whether the node has I/O HBA, and the number of NICs of the node.

FIG. **13** is a screen illustration of an exemplary user interface **180** for viewing users of distributed computing system **10**. User interface **180** presents a list of users as well as the role assigned to each of the users and the status of each of the users. Thus, system administrator **20** may define different roles to each of the users. For example, a user may be either an operator (i.e., general user) or an administrator. System administrator **20** may add a new user to the list of users by clicking on the "New User" button **182**.

FIG. **14** is a screen illustration of an exemplary user interface **190** for viewing alerts for distributed computing system **10**. For each of the alerts, user interface **190** identifies the severity of the alert, whether the alert has been acknowledged, an object associated with the alert, an event associated with the alert, a state of the alert, a user associated with the alert and a date associated with the alert.

System administrator **20** or other user may select an alert by clicking on the box in front of the logged alert and perform one or more actions on the logged alert. Actions that system administrator **20** may perform include deleting the alert, changing the status of the alert, or the like. System administrator **20** may specify the log actions via dropdown menu **192**.

FIG. **15** is a block diagram illustrating one embodiment of control node **12** in further detail. In the illustrated example, control node **12** includes a monitoring subsystem **202**, a service level automation infrastructure (SLAI) **204**, and a business logic tier (BLT) **206**.

Monitoring subsystem **202** provides real-time monitoring of the distributed computing system **10**. In particular, monitoring subsystem **202** dynamically collects status data **203** from the hardware and software operating within distributed computing system **10**, and feeds the status data in the form of monitor inputs **208** to SLAI **204**. Monitoring inputs **208** may be viewed as representing the actual state of the fabric defined for the organizational model implemented by distributed computing system **10**. Monitoring subsystem **202** may utilize well defined interfaces, e.g., the Simple Network Management Protocol (SNMP) and the Java Management Extensions (JMX), to collect and export real-time monitoring information to SLAI **204**.

SLAI **204** may be viewed as an automation subsystem that provides support for autonomic computing and acts as a central nervous system for the controlled fabric. In general, SLAI **204** receives monitoring inputs **208** from monitoring subsystem **202**, analyzes the inputs and outputs appropriate action requests **212** to BLT **206**. In one embodiment, SLAI **204** is a cybernetic system that controls the defined fabric via feedback loops. More specifically, administrator **20** may interact with BLT **206** to define an expected state **210** for the fabric. BLT **206** communicates expected state **210** to SLAI **204**. SLAI **204** receives the monitoring inputs from monitoring subsystem **202** and applies rules to determine the most effective way of reducing the differences between the expected and actual states for the fabric.

For example, SLAI **204** may apply a rule to determine that a node within a high priority tier has failed and that the node should be replaced by harvesting a node from a lower priority tier. In this example, SLAI **204** outputs an action request **212** to invoke BLT **206** to move a node from one tier to the other.

In general, BLT **206** implements high-level business operations on fabrics, domains and tiers. SLAI **204** invokes BLT **206** to bring the actual state of the fabric into accordance with the expected state. In particular, BLT **206** outputs fabric actions **207** to perform the physical fabric changes. In addition, BLT **206** outputs an initial expected state **210** to SLAI **204** and initial monitoring information **214** to SLAI **204** and monitoring subsystem **202**, respectively. In addition, BLT **206** outputs notifications **211** to SLAI **204** and monitoring subsystem **202** to indicate the state and monitoring changes to distributed computing system **10**. As one example, BLT **206** may provide control operations that can be used to replace failed nodes. For example, BLT **206** may output an action request indicating that a node having address 10.10.10.10 has been removed from tier ABC and a node having address 10.10.10.11 has been added to tier XYZ. In response, monitoring subsystem **202** stops attempting to collect status data **203** from node 10.10.10.10 and starts monitoring for status data from node 10.10.10.11. In addition, SLAI **204** updates an internal model to automatically associate monitoring inputs from node 10.10.10.11 with tier XYZ.

FIG. **16** is a block diagram illustrating one embodiment of monitoring subsystem **202**. In general, monitoring subsystem **202** dynamically detects and monitors a variety of hardware and software components within the fabric. For example, monitoring subsystem **202** identifies, in a timely and efficient manner, any computing nodes that have failed, i.e., any node that does not respond to a request to a known service. More generally, monitoring subsystem **202** provides a concise, consistent and constantly updating view of the components of the fabric.

As described further below, monitoring subsystem **202** employs a modular architecture that allows new detection and monitoring collectors **224** to be "plugged-in" for existing and new protocols and for existing and new hardware and software. As illustrated in FIG. **16**, monitoring subsystem **202** provides a plug-in architecture that allows different information collectors **224** to be installed. In general, collectors **224** are responsible for protocol-specific collection of monitoring information. The plug-in architecture allows for new protocols to be added by simply adhering to a collector plug-in signature. In this example, monitoring subsystem **202** includes collectors **224**A and **224**B for collecting information from operating systems and applications executing on nodes within tier A and tier B, respectively.

In one embodiment, collectors **224** are loaded at startup of control node **12** and are configured with information retrieved from BLT **206**. Monitoring engine **222** receives collection requests from SLAI **204**, sorts and prioritizes the requests, and invokes the appropriate one of collectors **224** based on the protocol specified in the collection requests. The invoked collector is responsible for collecting the required status data and returning the status data to monitoring engine **222**. If the collector is unable to collect the requested status data, the collector returns an error code.

In one embodiment, collectors **224** are Java code compiled into a jar file and loaded with a class loader at run time. Each of collectors **224** has an associated configuration file written in a data description language, such as the extensible markup language (XML). In addition, a user may interact with BLT **206** to add run-time configuration to dynamically configure collectors **224** for specific computing environments. Each of collectors **224** expose an application programming interface (API) to monitoring engine **222** for communication and data exchange.

A user, such as a system administrator, specifies the protocol or protocols to be used for monitoring a software image when the image is created. In addition, the users may specify the protocols to be used for monitoring the nodes and each service executing on the nodes. Example protocols supported by the collectors **224** include Secure Shell (SSH), Simple Network Management Protocol (SNMP), Internet Control Message Protocol (ICMP) ping, Java Management Extensions (JMX) and the Hypertext Transfer Protocol (HTTP).

Some protocols require special privileges, e.g., root privileges, to perform the required data collection. In this case, the corresponding collectors **224** communicate with a separate process that executes as the root. Moreover, some protocols may require deployment and/or configuration of data providers within the fabric. Software agents may, for example, be installed and configured on nodes and configured on other hardware. If needed, custom in-fabric components may be deployed.

In this example, the modular architecture of monitoring subsystem **202** also supports one or more plug-in interfaces **220** for data collection from a wide range of third-party monitoring systems **228**. Third-party monitoring systems **228** monitor portions of the fabric and may be vendor-specific.

FIG. **17** is a block diagram illustrating one embodiment of SLAI **204** in further detail. In the illustrated embodiment, SLAI **204** is composed of three subsystems: a sensor subsystem **240**, an analysis subsystem **244** and an effector subsystem **248**.

In general, sensor subsystem **240** receives actual state data from monitoring subsystem **202** in the form of monitoring inputs **208** and supplies ongoing, dynamic input data to analysis subsystem **244**. For example, sensor subsystem **240** is notified of physical changes to distributed computing system **10** by monitoring subsystem **202**. Sensor subsystem **240** uses the state data received from monitoring subsystem **202** to maintain ongoing, calculated values that can be sent to analysis subsystem **244** in accordance with scheduler **242**.

In one embodiment, sensor subsystem **240** performs time-based hierarchical data aggregation of the actual state data in accordance with the defined organization model. Sensor subsystem **240** maintains organizational data in a tree-like structure that reflects the current configuration of the hierarchical organization model. Sensor subsystem **240** uses the organizational data to perform the real-time data aggregation and map tiers and domains to specific nodes. Sensor subsystem **240** maintains the organizational data based on notifications **211** received from BLT **206**.

Sensor subsystem **240** sends inputs to analysis subsystem **244** to communicate the aggregated data on a periodic or event-driven basis. Analysis subsystem **244** may register an interest in a particular aggregated data value with sensor subsystem **240** and request updates at a specified frequency. In response, sensor subsystem **240** interacts with monitoring subsystem **202** and scheduler **242** to generate the aggregated data required by analysis subsystem **244**.

Sensor subsystem **240** performs arbitrary data aggregations via instances of plug-in classes (referred to as "triggers") that define the aggregations. Each trigger is registered under a compound name based on the entity being monitored and the type of data being gathered. For example, a trigger may be defined to aggregate and compute an average computing load for a tier every five minutes. Analysis subsystem **244** requests the aggregated data based on the registered names. In some embodiments, analysis subsystem **244** may define calculations directly and pass them to sensor subsystem **240** dynamically.

Analysis subsystem **244** is composed of a plurality of forward chaining rule engines **246**A-**246**N. In general, rule engines **246** match patterns in a combination of configuration data and monitoring data, which is presented by extraction agent **251** in the form of events. Events contain the aggregated data values that are sent to rule engines **246** in accordance with scheduler **242**.

Sensor subsystem **240** may interact with analysis subsystem **244** via trigger listeners **247** that receives updates from a trigger within sensor subsystem **240** when specified events occur. An event may be based on system state (e.g., a node transitioning to an up or down state) or may be time based.

Analysis subsystem **244** allows rule sets to be loaded in source form and compiled at load time into discrimination networks. Each rule set specifies trigger-delivered attributes. Upon loading the rule sets, analysis subsystem **244** establishes trigger listeners **247** to receive sensor notifications and update respective working memories of rule engines **246**. As illustrated in FIG. **17**, each of rule engines **246** may serve a different tier defined within the fabric. Alternatively, multiple rule engines **246** may serve a single tier or a single rule engine may serve multiple tiers.

Rule engines **246** process the events and invoke action requests via calls to effector subsystem **248**. In addition, rule engines **246** provide a call-back interface so that effector subsystem **248** can inform a rule engine when an action has completed. Rule engines **246** prevent a particular rule from re-firing as long as any action invoked by the rule has not finished. In general, rules contain notification calls and service invocations though either may be disabled by configuration of effector subsystem **248**. BLT **206** supplies initial system configuration descriptions to seed each of rule engines **246**.

In general, rule engines **246** analyze the events and discover discrepancies between an expected state of the fabric and an actual state. Each of rule engines **246** may be viewed as software that performs logical reasoning using knowledge encoded in high-level condition-action rules. Each of rule engines **246** applies automated reasoning that works forward from preconditions to goals defined by system administrator **20**. For example, rule engines **246** may apply modus ponens inferences rules.

Rule engines **246** output requests to effector subsystem **248** which produce actions requests **212** for BLT **206** to resolve the discrepancies. Effector subsystem **248** performs all operations on behalf of analysis subsystem **244**. For example, event generator **250**, task invocation module **252**

and logger **254** of effector subsystem **248** perform event generation, BLT action invocation and rule logging, respectively. More specifically, task invocation module **252** invokes asynchronous operations within BLT **206**. In response, BLT **206** creates a new thread of control for each task which is tracked by a unique task identifier (task id). Rules engine **246** uses the task id to determine when a task completes and, if needed, to re-fire any rules that were pended until completion of the task. These tasks may take arbitrary amounts of time, and rules engine **246** tracks the progress of individual task via change notifications **211** produced by BLT **206**.

Event generator **250** creates persistent event records of the state of processing of SLAI **204** and stores the event records within a database. Clients uses these event records to track progress and determine the current state of the SLAI **204**.

Logger **254** generates detailed trace information about system activities for use in rule development and debugging. The logging level can be raised or lowered as needed without changing operation of SLAI **204**.

FIG. **18** is a block diagram of an example working memory **270** associated with rule engines **246**. In this example, working memory **270** includes a read-only first data region **272** that stores the expected state received from BLT **206**. Data region **272** is read-only in the sense that it cannot be modified in response to a trigger from sensor subsystem **240** or by rule engines **246** without notification from BLT **206**.

In addition, working memory **270** includes a second data region **274** that is modifiable (i.e., read/write) and may be updated by monitoring subsystem **202** or used internally by rule engines **246**. In general, data region **274** stores aggregated data representing the actual state of the fabric and can be updated by sensor subsystem **240** or by rule engines **246**. The actual state may consist of a set of property annotations that can be attached to objects received from BLT **206** or to objects locally defined within a rule engine, such as local object **276**.

FIG. **19** is a block diagram illustrating an example embodiment for BLT **206**. In this example, BLT **206** includes a set of one or more web service definition language (WSDL) interfaces **300**, a report generator **302**, a fabric administration interface service **304**, a fabric view service **306**, a user administration service **308**, a task interface **311**, a task manager **312** and an event subsystem **315**.

As described, BLT **206** provides the facilities necessary to create and administer the organizational model (e.g., fabric, domains, tiers and nodes) implemented by distributed computing system **10**. In general, BLT **206** abstracts access to the persisted configuration state of the fabric, and controls the interactions with interfaces to fabric hardware services. As such, BLT **206** provides fabric management capabilities, such as the ability to create a tier and replace a failed node. WSDL interfaces **300** provide web service interfaces to the functionality of BLT **206** that may be invoked by web service clients **313**. Many of WSDL interfaces **300** offered by BLT **206** allow administrator **20** to define goals, such as specifying a goal of the expected state of the fabric. As further described below, rule engines **246** within SLAI **204**, in turn, invoke task manger **312** to initiate one or more BLT tasks to achieve the specified goal. In general, web service clients **313** may be presentation layer applications, command line applications, or other clients.

BLT **206** abstracts all interaction with physical hardware for web service clients **313**. BLT **206** is an enabling component for autonomic management behavior, but does not respond to real-time events that either prevent a goal from being achieved or produce a set of deviations between the expected state and the actual state of the system. In contrast, BLT **206** originates goals for autonomic reactions to changing configuration and state. SLAI **204** analyzes and acts upon these goals along with real-time state changes. BLT **206** sets the goals to which SLAI **204** strives to achieve, and provides functionality used by the SLAI in order to achieve the goals.

In general, BLT **206** does not dictate the steps taken in pursuit of a goal since these are likely to change based on the current state of distributed computing system **10** and changes to configurable policy. SLAI **204** makes these decisions based on the configured rule sets for the fabric and by evaluating monitoring data received from monitoring subsystem **202**.

Fabric administration service **304** implements a set of methods for managing all aspects of the fabric. Example methods include methods for adding, viewing, updating and removing domains, tiers, nodes, notifications, assets, applications, software images, connectors, and monitors. Other example methods include controlling power at a node, and cloning, capturing, importing, exporting or upgrading software images. Rule engines **246** of SLAI **204** may, for example, invoke these methods by issuing action requests **212**.

Task manager **312** receives action requests **212** via task interface **311**. In general, task interface **311** provides an interface for receiving action requests **212** from SLAI **204** or other internal subsystem. In response, task manager **312** manages asynchronous and long running actions that are invoked by SLAI **204** to satisfy a goal or perform an action requested by a client.

Task manager **312** generates task data **310** that represents identification and status for each task. Task manager **312** returns a task identifier to the calling web service clients **313** or the internal subsystem, e.g., SLAI **204**, that initiated the task. Rule engines **246** and web service clients **313** use the task identifiers to track progress and retrieve output, results, and errors associated with achieving the goal.

In one embodiment, there are no WSDL interfaces **300** for initiating specific tasks. Rather, administrator **20** interacts with BLT **206** though goal interfaces presented by WSDL interfaces **300** to define the goals for the fabric. In contrast, the term task is used to refer to internal system constructs that require no user interaction. Tasks are distinct, low-level units of work that affect the state of the fabric. SLAI **204** may combine tasks to achieve or maintain a goal state.

For example, administrator **20** can request configuration changes by either adding new goals to an object or by modifying the attributes on existing goals. Scheduled goals apply a configuration at a designated time. For example, the goals for a particular tier may specify the minimum, maximum, and target node counts for that tier. As a result, the tier can increase or decrease current node capacity by scheduling goals with different configuration values.

This may be useful, for example, in scheduling a software image upgrade. As another example, entire domains may transition online and offline per a defined grid schedule. Administrator **20** may mix and match goals on a component to achieve configurations specific to the application and environment. For example, a tier that does not support autonomic node replacement would not be configured with a harvesting goal.

In some embodiments, goals are either "in force" or "out of force." SLAI **204** only works to achieve and maintain those goals that are currently in force. SLAI **204** may applies a concept of "gravity" as the goals transition from in force to out of force. For example, SLAI **204** may transition a tier offline when an online goal is marked out of force. Some goal types may have prerequisite goals. For example, an image upgrade goal may require as a prerequisite that a tier be

transitioned to offline before the image upgrade can be performed. In other embodiments, goals are always in force until modified.

SLAI **204** may automatically formulate dependencies between goals or may allow a user to specify the dependencies. For example, a user may request that a newly created tier come online. As a result of this goal, SLAI **204** may automatically direct task manager **312** to generate a task of harvesting a target number of nodes to enable the tier. Generally, all goals remain in-force by SLAI **204** until modified by BLT **206**. In one embodiment, each goal remains in-force in one of three states: Satisfied, Warning, or Critical depending on how successful SLAI **204** was in achieving the goal at the time the event record was generated and stored.

In this manner, SLAI **204** controls the life cycle of a goal (i.e., the creation, scheduling, update, deletion of the goal), and provides a common implementation of these and other services such as timeout, event writing, goal conflicts, management of intra-goal dependencies, and tracking tasks to achieving the goals.

Progress toward a goal is tracked though event subsystem **315**. In particular, event subsystem **315** tracks the progress of each in force goal based on the goal identifiers. Tasks executed to achieve a particular goal produce events to communicate result or errors. The events provide a convenient time-based view of all actions and behaviors.

Examples of goal types that may be defined by administrator **20** include software image management goals, node allocation goals, harvest goals, tier capacity goals, asset requirement goals, tier online/offline goals, and data gathering goals.

In one embodiment, BLT **206** presents a task interface to SLAI **204** for the creation and management of specific tasks in order to achieve the currently in force goals. In particular, rule engines **246** invoke the task interface based on evaluation of the defined rule sets in view of the expected state and actual state for the fabric. Example task interfaces include interfaces to: reserve node resources; query resources for a node slot; associate or disassociate an image with a node in a tier node slot; allocate, de-allocate, startup or shutdown a node; move a node to a tier; apply, remove or cycle power of a node; create a golden image; create or delete an image instance; and delete an activity, node or tier.

Report generator **302** provides an extensible mechanism for generating reports **314**. Typical reports include image utilization reports that contain information with respect to the number of nodes running each software image, inventory reports detailing both the logical and physical aspects of the fabric, and system event reports showing all events that have occurred within the fabric. Report generator **302** gathers, localizes, formats and displays data into report form for presentation to the user. Report generator **302** may include one or more data gathering modules (not shown) that gather events in accordance with a schedule and update an events table to record the events. The data gathering modules may write the events in XML format.

FIG. **20** is a block diagram illustrating one embodiment of a rule engine **246** (FIG. **17**). In the illustrated embodiment, rule engine **246** includes a rule compiler **344** and an execution engine **346**. Each of rules **342** represents a unit of code that conforms to a rule language and expresses a set of triggering conditions and a set of implied actions. When the conditions are met, the actions are eligible to occur. The following is one example of a configuration rule:

```
rule checkTierLoad {
    Tier t where status != "overloaded";
    LoadParameter p where app == t.app && maxload < t.load;
} -> {
    modify t {
        status: "overloaded";
    };
}
```

When translated, this example rule marks a tier as overloaded if an application is implemented by the tier and the maximum specified load for the application has been exceeded. Another example rule for outputting a notification that a tier is overloaded and automatically invoking a task within BLT **206** to add a node is:

```
rule tierOverloadNotify {
    Tier t where status == "overloaded";
} -> {
    notify "Tier: " + t + "is overloaded.";
    BLT.addNode(f);
}
```

Rule compiler **344** compiles each of rules **344** and translates match conditions of the rules into a discrimination network that avoids redundant tests during rule execution. Execution engine **346** handles rule administration, object insertion and retrieval, rule invocation and execution of rule actions. In general, execution engine **346** first matches a current set of rules **342** against a current state of working memory **348** and local objects **350**. Execution engine **346** then collects all rules that match as well as the matched objects and selects a particular rule instantiation to fire. Next, execution engine **346** fires (executes) the instantiated rule and propagates any changes to working memory **348**. Execution engine **346** repeats the process until no more matching rule instantiations can be found.

Firing of a rule typically produces a very small number of changes to working memory **348**. This allows sophisticated rule engines to scale by retaining match state between cycles. Only the rules and rule instantiations affected by changes are updated, thereby avoiding the bulk of the matching process. One exemplary algorithm that may be used by execution engine **346** to handle the matching process includes the RETE algorithm that creates a decision tree that combines the patterns in all the rules and is intended to improve the speed of forward-chained rule system by limiting the effort required to re-compute a conflict set after a rule is fired. One example of a RETE algorithm is described in Forgy, C. L.: 1982, 'RETE: a fast algorithm for the many pattern/many object pattern match problem', Artificial Intelligence 19, 1737, hereby incorporated by reference. Other alternatives include the TREAT algorithms, and LEAPS algorithm, as described by Miranker, D. P.: 'TREAT: A New and Efficient Match Algorithm for AI Production Systems'. ISBN 0934613710 Daniel P. Miranker, David A. Brant, Bernie Lofaso, David Gadbois: On the Performance of Lazy Matching in Production Systems. AAAI 1990: 685692, each of which is hereby incorporated by reference.

FIG. **21** is a block diagram illustrating an alternative embodiment of control unit **12** (FIG. **15**). In this embodiment, control unit **12** operates substantially as described above, but includes an application matrix **350**, an application governor **352**, a configuration processor **354**, and an application service level automation infrastructure ("application SLAI") **358**. As

described below, application matrix **350**, application governor **352**, configuration processor **354**, and application SLAI **358** provide a framework that allows control unit **12** to autonomically control the deployment, execution and monitoring of applications across application nodes **14** of distributed computing system **10**.

Application matrix **350** contains all the information needed by control unit **12** to interact with one or more applications or application servers and provide autonomic control over a set of applications. Specifically, application matrix **350** provides a logical definition for deploying and controlling the set of applications to one or more tiers within distributed computing system **10**. In one embodiment, application matrix **350** is an electronic document that conforms to a data description language, e.g., the extensible markup language (XML). Application SLAI **358** includes an application rules engine **355** dedicated to processing application-level rules, i.e., forward-chaining rules to provide autonomic control over the applications defined within application matrix **350**. Like rules engine **246**, application rules engine **355** contains a rule compiler, an execution engine, and a working memory. In order to give effect to changes in application matrix **350**, application SLAI **358** automatically updates application rules engine **355** and monitoring subsystem **202**. In particular, application matrix **350** sends an alert whenever application matrix **350** changes. In response to this alert, application SLAI **358** captures application-specific attributes from application matrix **350**. Specifically, application SLAI **358** captures configuration attributes and rule attributes contained in application matrix **350**. Application SLAI **358** transfers any new rule attributes to the working memory of application rules engine **355** to provide autonomic control over the deployment, monitoring and the execution of the applications defined within application matrix **350**. In addition, application SLAI **358** updates monitoring subsystem **202** to collect information required to control the applications In this manner, administrator **20** may continue to add new application definitions and configurations to application matrix **350** after distributed control node **12** has started.

As described in further detail below, configuration processor **354** is a software module that generates an application matrix entry based on an application definition and application configuration properties of a "staged" application. A staged application is an application that has been deployed in a staging environment and customized for subsequent deployment within distributed computing system **10**. After creating the application matrix entry, administrator **20** may insert the application matrix entry into application matrix **350**.

Configuration processor **354** is "pluggable." That is, administrator **20** can "plug in" different implementations of configuration processor **354** as needed. For example, administrator **20** may need to "plug in" a different implementation of configuration processor **354** to handle applications that do not use an application server.

Application governor **352** is a software engine that performs application-level actions based on requests received from application rules engine **355**. In this manner, BLT **206** effects fabric-level actions (e.g., deployment and monitoring of nodes and images) based on request from fabric-level rules engines **246** (FIG. **17**), while matrix governor **352** performs application-level actions (e.g., deployment and monitoring of applications) based on requests from application rules engine **355**.

Application governor **352** uses application matrix **350** as a source of parameters when carrying out the application-level operations requested by application rules engine **355**. For example, application rules engine **355** may detect that a node to which the application is not deployed is ready for use by the application. As a result, application rules engine **355** directs application governor **352** to handle the details of deploying the application to the node. In turn, application governor **352** accesses application matrix **350** to retrieve application-specific parameters necessary to deploy the application. Storing application-specific parameters in application matrix **350** allows the application-specific parameters to change without having to recompile the rules within working memory of application rules engine **355**.

Application governor **352** performs a similar procedure to undeploy an application. That is, application rules engine **355** may detect that a second application needs to use a node more than a first application that is currently deployed to the node. In this situation, application rules engine **355** sends an instruction to application governor **352** to undeploy the first application and deploy the second application. To carry out this instruction, application governor **352** accesses application matrix **350** to discover configuration parameters of both applications. Application governor **352** then uses the discovered configuration parameters to communicate with the applications.

Like configuration processor **354**, application governor **352** is also "pluggable." That is, administrator **20** can easily install or remove implementations of application governor **352** depending on the circumstances. Because application governor **352**, and configuration processor **354** represent interchangeable, plug-in modules, the other parts of control unit **12** and system **10**, including application rules engine **355**, can remain generic and application neutral while providing autonomic control over distributed computing system **10**.

FIG. **22** provides a conceptual view of an exemplary application matrix **350**. Although application matrix **350** is typically represented in an encoded, electronic document (e.g., an XML document), FIG. **22** provides a conceptual view for ease of illustration.

In this example, application matrix **350** contains seven columns and two rows. Each row represents a different application entry for deployment within distributed computing system **10**. In FIG. **22**, only the first entry is shown in detail.

Each of the columns represents a different category of elements that make up an application's logical definition. In this example, the columns include:

(1) an application column **360** that includes elements generally related to the deployment of the application,

(2) an application nodes column **362** that contains elements related to the tier node slots to which the application may be assigned,

(3) a services column **364** that contains elements related to the executable services launched when the application is deployed,

(4) a node monitor values column **366** that contains elements related to attributes of the nodes that are to be monitored after the application is deployed to that node,

(5) a service monitored attributes column **368** that contains elements related to attributes of the services that are to be monitored after the application is deployed,

(6) a service levels column **370** that contains elements related to attributes for use when constructing rules to monitor execution of the services, and

(7) a deployment constraints column **372** that contains elements related to attributes for use when constructing rules to control deployment of the application.

Different types of applications may have different elements in each column, and different numbers of columns.

In the example of FIG. **22**, application matrix **350** contains two applications **360** "DataDomain" **374** and "PortalDomain" **376**. DataDomain application **374** has eleven attributes that define how control node **12** accesses and launches the application. For instance, the "adminIP" and "adminPort" attributes instruct governor **352** as to which the server and port hosts the administrative part of the logically defined application. Other attributes like "maxNodes" and "minNodes" instruct application rules engine **355** to run the application no less than minNodes and no more than maxNodes. Applications other than application **374** may have different numbers or types of attributes. In XML format, the attributes of application **374** may appear as follows:

```
<WebServerDomain
    name="DataDomain"
    adminIP="172.31.64.201"
    adminPort="1100"
    adminTier="Web Admin"
    clusterName="PathCluster"
    expectedStartupDelay="120"
    loadDelay="120"
    maxNodes="2"
    minNodes="1"
    nodeManagerPort="5811"
    nodeTier="Web App"
>
```

In addition to these attributes, application **374** contains a series of elements (columns **362**-**372**). In general, application nodes **362** contain a list of all of the available tier-node slots to which control node **12** may deploy application **374**. In this example, two tier-node slots are specified. In XML, managed servers **362** appears as:

```
<ManagedServers IP="172.31.64.201" name="managedServer_0"
state="STOPPED" />
<ManagedServers IP="172.31.64.201" name="managedServer_1"
state="STOPPED" />
```

Each services **364** element identifies a service that control node **12** launches when deploying application **374** on a given application node. A service element comprises a service name and a path to a file containing the executable service. Governor **352** uses the path to locate and launch the service. For example, the following XML code indicates that governor **352** must access the file at "/lib/worklistApp/worklistApp.ear".

```
<services name="Worklist User Interface"
path="/lib/worklistApp/worklistApp.ear" />
```

Node Monitored Values **366** elements represent characteristics for use in constructing rules for monitoring nodes to which the applications are deployed. Similarly, Service Monitored Values **368** elements represent characteristics for use in constructing rules for monitoring services that are launched once the application is deployed. In this example, a "nodeMonitoredValues" element defines characteristics of a particular node that are to be monitored. For instance, the amount of free memory in a node is one example of a characteristic listed as a "nodeMonitoredValues" element. On the other hand, a "serviceMonitoredValues" element is specific attribute of a service that is to be monitored. For example, the number of pending operating system-level requests for the service, the number of idle threads, etc., could be service monitored values. In a XML rendition of application matrix **350**, node monitored values and service monitored values could appear as follows:

```
<nodeMonitoredValues name="Load5Average" />
<nodeMonitoredValues name="PercentMemoryFree" />
<serviceMonitoredValues name="PendingRequests" />
<serviceMonitoredValues name="ExecuteThreadIdleCount" />
<serviceMonitoredValues name="ExecuteThreadTotalCount" />
```

Deployment constraint elements **372** specify characteristics of a node under which application rules engine **355** should (or should not) deploy a service to the node. In this example, a deployment constraint element has five attributes: "attribute", "expression", "frequency", "maxThreshold", "minThreshold", and "period." The "attribute" attribute names the deployment constraint. The "expression" attribute specifies an arithmetic expression manipulating a monitored value. For example, the expression could be "PercentMemoryFree*100", meaning monitor the value of the "PercentMemoryFree" node monitored value multiplied by 100. The "expression" attribute may specify one or more node monitored values. The "frequency" attribute informs application rules engine **355** how frequently to check the monitored value. The "maxThreshold" attribute tells application rules engine **355** to invoke a rule when the value of the expression exceeds the value specified by the "maxThreshold" attribute. Similarly, the "minThreshold" attribute tells application rules engine **355** to invoke the rule when the value of the expression drops below the value specified by the "minThreshold" attribute. Finally, the "period" attribute informs application rules engine **355** of the period over which to collect monitored value. For example, a deployment constraint element may specify that application rules engine **355** should monitor the PercentMemoryFree attribute of a node every 15 seconds for 60 seconds. If the value of PercentMemoryFree*100 should drop below 1.0 (i.e. 1% of memory free) for 60 seconds, then application rules engine **355** should not deploy the application to that node. In XML, this rule would be represented as:

```
<deploymentConstraints attribute="FreeMemory"
expression="PercentMemoryFree*100" frequency="15" maxThreshold="-
1.0" minThreshold="1.0" period="60" />
```

Service level elements **370** have the same five attributes as deployment constraints elements: "attribute", "expression", "frequency", "maxThreshold", "minThreshold", and "period". However, the "expression" attribute deals with service monitored attributes rather than node monitored attributes. For example, a service level element may specify that application rules engine **355** check the "pendingRequest" service-monitored attribute every 15 seconds for 30 seconds. Then, if there are more than 20 pending requests for more than 30 seconds, application rules engine **355** should take the action of starting a new application. On the other hand, if there are fewer than 5 pending requests for 30 seconds, application rules engine **355** enables the action to remove the application to free up space on a node. Such a service level element could be represented as:

```
<serviceLevels attribute="PendingRequests"
expression="PendingRequests" frequency="15" maxThreshold="20.0"
minThreshold="5.0" period="30" />
```

Put together, an XML representation of an application matrix logically defining a single application for deployment within autonomically controlled distributed computing system **10** may appear as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<appMatrixRegister>
<WebServerDomain adminIP="172.31.64.201" adminPort="1100"
adminTier="Web
Admin" clusterName="PathCluster" expectedStartupDelay="120"
loadDelay="120" maxNodes="2" minNodes="1" name="DataDomain"
nodeManagerPort="5811" nodeTier="Web App">
    <ManagedServers IP="172.31.64.201" name="managedServer_0"
    state="STOPPED" />
    <ManagedServers IP="172.31.64.201" name="managedServer_1"
    state="STOPPED" />
    <applications name="AI Design-time"
    path="/lib/ai-designtime.ear" />
    <applications name="System EJBs" path="/lib/ejbs.ear" />
    <applications name="Worklist Worker User Interface"
    path="/lib/worklist/worklist.ear" />
    <applications name="DBMS_ADK" path="/lib/DBMS_ADK.ear"
    />
    <applications name="UserApp"
    path="/user_projects/domains/userApp.ear"
    />
    <nodeMonitoredValues name="Load5Average" />
    <nodeMonitoredValues name="PercentMemoryFree" />
    <serviceMonitoredValues name="PendingRequests" />
    <serviceMonitoredValues name="ExecuteThreadIdleCount" />
    <serviceMonitoredValues name="ExecuteThreadTotalCount" />
    <deploymentConstraints attribute="LoadAverage"
    expression="Load5Average" frequency="15"
    maxThreshold="4.0"
    minThreshold="-1.0" period="15" />
    <deploymentConstraints attribute="FreeMemory"
    expression="PercentMemoryFree*100" frequency="15"
    maxThreshold="-1.0"
    minThreshold="1.0" period="60" />
    <serviceLevels attribute="BusyThreadPercentage"
    expression="(ExecuteThreadTotalCount-
    ExecuteThreadIdleCount) *100/ExecuteThreadTotalCount"
    frequency="15"
    maxThreshold="20.0" minThreshold="5.0" period="30" />
    <serviceLevels attribute="PendingRequests"
    expression="PendingRequests" frequency="15"
    maxThreshold="20.0"
    minThreshold="5.0" period="30" />
</WebserverDomain>
</appMatrixRegister>
```

FIG. **23** is a flowchart illustrating an exemplary series of general steps that are performed to add an application to application matrix **350** in accordance with the principles of this invention. Administrator **20** first stages the application within a staging environment including deploying the application on a desired image, creating a domain, specifying any services and external system connectivity (e.g., connections to databases, queues) (**380**).

During this process, administrator **20** directs the application to generate an application definition file that represents an application-specific configuration based on the environment specified by the administrator. The application definition file contains information specific to an installation instance of the application. For example, the application definition file typically specifies the locations (paths) for all software components of the application as staged within the staging environment, a list of files to execute, connectivity information for the application, and other information that may be recorded by the configured application.

After staging the application, administrator **20** defines a set of application configuration properties that specify desired behavior of the application within distributed computing system **10** (**382**). Examples of such application configuration properties include a minimum number of nodes, minimum resource requirements for a node, deployment timing information, network addresses of tier node slots that are able to execute the application and other properties.

Next, administrator **20** directs configuration processor **354** to generate an application entry for the application using the application definition and the application configuration properties (**384**). The application entry contains configuration attributes used by application governor **352** to interact with the application. In addition, the application entry contains rule attributes used by application rules engine **355** that define how control node **12** monitors the deployment and execution of the application within distributed computing environment **10**.

After configuration processor **354** creates the application entry, administrator **20** may modify the application entry (**386**). In particular, administrator **20** may update start scripts or shell scripts for the application due to path changes between the staging environment and distributed computing system **10**.

Once administrator **20** has finished modifying the application entry, administrator **20** inserts the application entry into application matrix **350** (**388**). Because application matrix **350** detects that it has changed, application matrix **350** sends an alert to application SLAI **358** (**390**).

In response to the alert, application SLAI **358** may update application rules engine **355** and monitoring subsystem **202** (**392**). In particular, application SLAI **358** automatically scans application matrix **350**. If application SLAI **358** detects new rule attributes, application SLAI **358** creates local objects reflecting the new rule attributes in the working memory of application rules engine **355**. In addition, if application SLAI **358** detects new monitored values, application SLAI **358** updates monitoring subsystem **202** to add new monitoring collectors **224** (FIG. **16**).

After application SLAI **358** updates application rules engine **355** and monitoring subsystem **202**, control node **12** has autonomic control over the deployment of applications to tiers based on the configuration of application matrix **350** (**394**). For example, control node **12** deploys images, deploys applications, monitors the state of nodes and the execution of the applications, and applies tier-level rules as well as application-specific rules. Application SLAI **358** continues to listen for alerts from application matrix **350**.

FIGS. **24** through **29** illustrate exemplary embodiments of distributed computing system **10** that provides autonomic control and deployment of virtual machines and virtual disk image files. For example, FIG. **24** is a block diagram illustrating an exemplary configuration of a physical node **400**. Node **400** may be any node within distributed computing system **10** (FIG. **1**). For instance, node **400** may be any one of application nodes **14**.

As illustrated in FIG. **24**, a virtual machine manager **402** is executing on node **400**. As used herein, a virtual machine manager is a software program that is capable of managing one or more virtual machines. For example, virtual machine manager **402** may be an instance of VMWare ESX software produced by VMWare, Inc. of Palo Alto, Calif.

When deployed on physical node **200**, virtual machine manager **402** hosts virtual machines **404**A through **404**N (collectively, virtual machines **404**). The term virtual

machine is used herein to refer to software that creates an environment that emulates a computer platform. For example, virtual machines **40** provide environments on which guest operating systems (OS) execute as through the operating systems were operating directly on a physical computing platform. In this manner, multiple operating systems and software applications **406**A through **406**N (collectively, operating systems and applications **406**) may execute on virtual machines **404**A through **404**N, respectively, and the virtual machines **404** may execute on a single physical node **400** or multiple physical nodes. For example, if virtual machine **404**A emulates an x86 machine, an instance of the Microsoft Windows operating system (e.g., OS **406**A) may execute on virtual machine **404**A as through the instance of Windows were running on a physical x86 machine. Other software applications may be installed and configured on the guest operating system running on virtual machine **404**. For example, an instance of Microsoft Internet Information Server may execute on an instance of Microsoft Windows that is executing on a virtual machine that emulates an x86 machine. In this manner, virtual machine manager **402** and virtual machines **404** provide a level of independence from the particular hardware environment provided by physical node **400**.

Virtual machine manager **402** utilizes operating system data and application data for execution of each of virtual machines **404**. The operating system data and application data, as well as instance-specific configuration data for the underlying virtual machines **404**, is stored as respective virtual disk image files **408**. From the perspective of the guest operating system, each of virtual disk image files **408** appears as a bootable hard disk of the host computer. In other words, the virtual disk image file provides a bootable software image of the operating system and applications, and includes all necessary configuration data and file structures. As illustrated in FIG. **24**, virtual machine disk image file **408**A provides a bootable software image for operating system/application information **406**A that is capable of being executed on virtual machine **404**A. For instance, if virtual machine manager **402** is an instance of VMWare ESX, virtual machine disk image files **408** may be specially captured .vmdk files.

A two-phase capture process may be used to provide autonomic control and deployment of the virtual machine disk image files **408** as well as virtual machine manager **402** to physical nodes within distributed computing system **10**. In particular, control node **12** may first capture an image of virtual machine manager **402** installed on physical node **400** prior to installation and configuration of any guest operating system and applications **406**. Control node **12** may utilize this captured image of virtual machine manager **402** as a "golden image" from which instance images can be created for this type virtual machine. The instance images may be stored within application matrix **350** for autonomic deployment as other software images (FIG. **21**).

After the first capture is complete, an administrator may install one or more virtual machines **404** to virtual machine manager **402**. After the user installs and configures guest operating systems and any necessary applications **406** on the virtual machines **404**, control node **12** may capture software images for storage as virtual disk image files **408**. Specifically, control node **12** may utilize the captured images as a golden images for the particular operating systems, applications and specific configuration of the virtual machine. The golden image may then be used to produce image instances, i.e., virtual disk images in this case, that are bootable on the virtual machines. The image instances may likewise be stored in application matrix **350** for autonomic deployment.

After generation of image instances for virtual machine manager **402** and virtual disk image files **408**, control node **12** may deploy the image instance of virtual machine manager **402** to one or more nodes in free pool **13**. Once control node **12** deploys the image instance of virtual machine manager **402** to a node, the node appears to be one or more new, unallocated nodes. For instance, if the image instance of virtual machine manager **402** is configured to support five virtual machines, control node **12** represents the newly deployed image instance as though five new, unallocated nodes have been added to free pool **13**. Subsequently, control node **12** may deploy an image instance of virtual disk image files **408** to one of the new unallocated nodes.

In one embodiment, during deployment, administrator **20** may specify whether data in virtual disk image files is to be treated as persistent or non-persistent. If administrator **20** specifies that the virtual disk image files are persistent, changes made to a virtual disk image file persist when node is restarted. To ensure the persistence of the data, administrator **20** may configure virtual machine manager **402** to store virtual disk image files in a storage area network (SAN). Otherwise, if administrator **20** specifies that virtual disk image files are non-persistent, the original virtual disk image file is reloaded from control node **12**. Non-persistence may be valuable to minimize risks (e.g., potential corruption due to viruses) associated with prolonged use of an operating system. Further, corrupt or invalid data in the virtual disk image file may be corrected by use of non-persistent image instances for the virtual disk image files.

In this way, distributed computing system **10** may autonomically utilize application nodes, e.g, node **400**, to flexibly support a wide variety of operating systems and applications, and provide virtualized execution environment for those operating systems and applications. For instance, node **400** may autonomically configured to support a Macintosh Operating System designed for a PowerPC processor, and a Microsoft Windows Operating System designed for an x86 processor based on the particular goals of the autonomic system. Thus, control node **12** may not need to distinguish between hardware processor architectures when deploying operating systems and applications. However, this system may also allow administrator **20** to specify that control node **12** operate an image instance directly on a node without the intervening virtual machine and virtual machine manager. Furthermore, by executing multiple operating systems on a single node, distributed computing system **10** may be able to use and deploy the resources of each node more efficiently.

FIG. **25** is a flowchart illustrating an exemplary procedure for creating and capturing a golden image for a particular virtual machine manager **402** (FIG. **24**). Initially, administrator **20** sets aside a node (e.g., node **400**) as an image host (**410**). Administrator **20** may then install a particular type of virtual machine manager **402** on node **400** (**412**). After administrator **20** installs virtual machine manager **402** on node **400**, administrator **20** may configure the virtual machine manager to support a desired number of virtual machine instances (**414**). For example, administrator **20** could configure virtual machine manager **402** to support one or more virtual machines. Administrator **20** may then perform other network or Storage Area Network (SAN) configuration on virtual machine manager **402** (**416**). After administrator **20** performs all of the necessary configuration, administrator **20** instructs control node **12** to capture the image of virtual machine manager **402** with the configurations (**418**). Control node **12** may store the captured image as a golden image in application matrix **350** for use in subsequent autonomic deployment of image instances.

FIG. **26** is a flowchart illustrating an exemplary procedure for creating and capturing a virtual disk image file, i.e., a golden image of a guest operating system and applications for execution on a virtual machine. Initially, administrator **20** sets aside a node (e.g., node **400**) that is running a virtual machine manager (e.g., virtual machine manager **402**) (**420**). Administrator **20** may then configure the virtual machine manager to create a virtual machine (e.g., virtual machine **404**) on node **400** (**422**). After creating the virtual machine, administrator **20** installs the particular operating system on virtual machine **404** (**424**). Administrator **20** may then install one or more applications on the operating system (**426**). After installation, administrator **20** may configure the operating system and applications as needed (**428**). In some embodiments, virtual machine manager **402** stores definition data for the particular virtual machine, operating system, and applications in a virtual disk image file. Once configuration is complete, administrator **20** takes a "snapshot" of the data in the virtual disk image file (**430**). Administrator **20** may then capture this "snapshot" to control node **12** as a golden image (**432**). Control node **12** may store the "snapshot" in application matrix **350** as a golden image for use in autonomic deployment of one or more image instances to virtual machines hosting the particular operating system and applications.

FIG. **27**A is a flowchart illustrating an exemplary procedure for autonomic deploying a virtual machine manager to a node. Initially, administrator **20** may instruct control node **12** to create a new tier (**440**). Subsequently, administrator **20** specifies one or more captured virtual machine manager image instances for assignment to the slots of the new tier (**442**). Administrator **20** may also specify whether the image instances of the tier run in persistent or non-persistent mode. Administrator **20** may then activate the new tier (**444**). After these physical nodes boot with the virtual machine manager image instance, each of the nodes creates a set of virtual machines as specified by the image instances. Control node **12** may then discover each of these virtual machines and place them in free pool **13** as if they were independent nodes. In this manner, each of the virtual machines appear to be physical nodes available for use as application nodes **14** when needed. However, control node **12** maintains tags or other metadata that distinguish between virtual machines and actual physical nodes. Finally, based on the current state and goals of distributed computing system **10**, control node **12** may allocate one or more of the virtual machines to the slots within the tier, similar to the process by which nodes are deployed from the free pool **13** to a tier as needed.

FIG. **27**B is a flowchart illustrating an exemplary procedure for deploying an operating system and application image instance to a node that is executing a virtual machine manager. To start the procedure, administrator **20** may instruct control node **12** to create a new tier, including defining the number of slots for the tier, or select an existing tier (**446**). Administrator **20** may then specify a captured operating system and application image for the tier (**448**). In addition, administrator **20** may set minimum, maximum, and target values for the image instances to be deployed to the slots new tier (**450**). Administrator **20** may then activate the tier (**452**). After administrator **20** activates the tier, based on the current state and goals of distributed computing system **10**, control node **12** autonomically selects a virtual machine from free pool **13**, associates the virtual machine with a slot in the tier, identifies the virtual disk image file (image instance) associated with the particular slot, copies the virtual disk image file to the local hard disk of the physical node on which the virtual node resides or the an associated SAN, and finally boots the virtual machine from the virtual disk image file.

FIG. **28** is a screen image illustrating an exemplary tier control interface **460** for the embodiment described in FIGS. **24** through **27** above. In this example, control interface **460** displays a set of three tiers "elas3u6_2", "elas3u6_vm", and "esx17_8". As illustrated in FIG. **28**, tier "esx17_8", labeled **462**, contains eight physical nodes. The image "esx_17" is operating on each node in tier **462**. If the "esx_17" image is a virtual machine manager image that operates seventeen virtual machines, the eight nodes of tier **462** operate as 136 virtual machines.

FIG. **29** is a screen image illustrating an exemplary node control interface **470** for the embodiment described in FIGS. **24** through **27** above. As illustrated in FIG. **29**, a single physical node "collage0104" hosts seventeen virtual machines "collage0104_vnode1" through "collage0104_vnode17". Each of these virtual machines is loaded with the "w2k3_iis_sql (1.0)" image. For instance, the "w2k3_iis_sql (1.0)" image may be an image containing instances of Microsoft Windows 2003 Internet Information Server and Microsoft SQL Server.

Various embodiments of the invention have been described. These and other embodiments are within the scope of the following claims.

The invention claimed is:

1. A distributed computing system comprising:

a software image repository comprising non-transitory, computer-readable media operable to store: (i) a plurality of image instances of a virtual machine manager that is executable on a plurality of application nodes, wherein when executed on the applications nodes, the image instances of the virtual machine manager provide a plurality of virtual machines, each of the plurality of virtual machine operable to provide an environment that emulates a computer platform, and (ii) a plurality of image instances of a plurality of software applications that are executable on the plurality of virtual machines; and

a control node that comprises an automation infrastructure to provide autonomic deployment of the plurality of image instances of the virtual machine manager on the application nodes by causing the plurality of image instances of the virtual machine manager to be copied from the software image repository to the application nodes and to provide autonomic deployment of the plurality of image instances of the software applications on the virtual machines by causing the plurality of image instances of the software applications to be copied from the software image repository to the application nodes.

2. The distributed computing system of claim **1**, wherein the image instances of the plurality of software applications comprise virtual disk image files.

3. The distributed computing system of claim **1**, wherein a first image instance of the plurality of image instances of the plurality of software applications comprises an operating system.

4. The distributed computing system of claim **1**, wherein a first image instance of the plurality of image instances of the plurality of software applications comprises a server application.

5. The distributed computing system of claim **1**, wherein a first one of the virtual machines provides an environment that emulates an x86 platform.

6. The distributed computing system of claim **1**, wherein the automation infrastructure captures a first image instance of the image instances of the virtual machine manager from a first application node of the plurality of application nodes.

7. The distributed computing system of claim 6, wherein the first application node executes the virtual machine manager.

8. The distributed computing system of claim 1, wherein the control node maintains the plurality of image instances of the virtual machine manager and the plurality of image instances of the plurality of software applications as persistent or non-persistent based on a configuration.

9. The distributed computing system of claim 1, wherein the control node further comprises one or more rule engines that provide autonomic deployment of the software applications to the virtual machines in accordance with a set of one or more rules.

10. The distributed computing system of claim 9, wherein the one or more rule engines are forward-chaining rule engines.

11. The distributed computing system of claim 9, wherein the automation infrastructure automatically updates the one or more rules engines to autonomically control the deployment of the software applications to the application nodes in accordance with a current state of an application matrix.

12. The distributed computing system of claim 1, wherein the control node further comprises an application matrix that includes parameters for controlling the deployment, undeployment, and execution of the software applications to the virtual machines within the distributed computing system.

13. The distributed computing system of claim 12, further comprising an application governor that accesses the application matrix to communicate instructions from the automation infrastructure to the software applications.

14. The distributed computing system of claim 9, wherein the automation infrastructure automatically updates the one or more rules engines to monitor the execution of the software applications when deployed to the application nodes in accordance with a current state of the application matrix.

15. The distributed computing system of claim 1, wherein the automation subsystem includes a rule compiler that compiles rules into one or more discrimination networks.

16. The distributed computing system of claim 1, wherein the control node further comprises:

a monitoring subsystem that collects status data from the application nodes and communicates the status data to the automation subsystem, wherein the status data represents an actual state for the application nodes; and

a business logic tier that provides expected state data to the automation subsystem, wherein the expected state data represents an expected state for the application nodes,

wherein one or more rule engines analyze the status data from the monitoring subsystem and apply a set of rules to produce action requests to the business logic tier to control the application nodes to reduce any difference between the actual state and the expected state.

17. The distributed computing system of claim 1, wherein the application nodes are organized into tiers, and wherein status data is collected based on the tiers.

18. The distributed computing system of claim 1, further comprising:

a database storing data that defines a model for a hierarchical organization of the distributed computing system, wherein the model specifies a fabric having one or more domains, and wherein each domain has at least one tier that includes at least one of the application nodes,

wherein one or more rule engines automatically produce action requests to control the deployment of the software applications in accordance with the hierarchical organization of the distributed computing system defined by the model.

19. A method comprising:

storing a plurality of image instances of a virtual machine manager that is executable on a plurality of application nodes in a distributed computing system, wherein the image instances of the virtual machine manager provide a plurality of virtual machines, each virtual machine operable to provide an environment that emulates a computer platform;

storing a plurality of image instances of a plurality of software applications that are executable on the plurality of virtual machines;

executing a rules engine to perform operations to autonomically deploy the image instances of the virtual machine manager on the application nodes by causing the image instances of the virtual machine manager to be copied to the application nodes; and

executing the rules engine to perform operations to autonomically deploy the image instances of the software applications on the virtual machines by causing the image instances of the software applications to be copied to the application nodes.

20. The method of claim 19, wherein storing the image instances of the plurality of software applications comprises storing virtual disk image files.

21. The method of claim 19, wherein a first image instance of the plurality of image instances of the plurality of software applications comprises an operating system.

22. The method of claim 19,

further comprising generating an application matrix that specifies data for controlling the deployment of the software applications within the distributed computing system; and

wherein performing operations to provide autonomic control comprises providing autonomic control over the deployment, undeployment, and execution of the software applications on the virtual machines within the distributed computing system in accordance with parameters of the application matrix.

23. The method of claim 19, wherein performing operations to autonomically deploy the image instances of the software applications on the virtual machines comprises:

receiving status data for the distributed computing system, wherein the status data represents an actual state of the application nodes;

processing the status data with rules in a set of rule engines to determine operations for reducing any difference between an expected state and the actual state of the application nodes; and

performing the operations to provide autonomic control over the deployment of the software applications on the virtual machines within the distributed computing system in accordance with the rules.

24. The method of claim 19, further comprising accessing parameters of an application matrix and determining one or more rules for use in autonomically controlling the deployment of the software applications on the virtual machines.

25. The method of claim 24, further comprising processing the one or more rules with a plurality of forward-chaining rule engines that compile the one or more rules into one or more discrimination networks.

26. A non-transitory, computer-readable medium comprising instructions stored thereon, that when executed by a programmable processor:

store a plurality of image instances of a virtual machine manager that is executable on a plurality of application nodes in a distributed computing system, wherein the image instances of the virtual machine manager provide

a plurality of virtual machines, each virtual machine operable to provide an environment that emulates a computer platform;

store a plurality of image instances of a plurality of software applications that are executable on the plurality of virtual machines;

perform operations to autonomically deploy the image instances of the virtual machine manager on the application nodes by causing the image instances of the virtual machine manager to be copied to the application nodes; and

perform operations to autonomically deploy the image instances of the software applications on the virtual machines by causing the image instances of the software applications to be copied to the application nodes.

* * * * *