

The chart below illustrates infringement of the Sundar '138 Patent by Defendant Netflix, Inc.'s ("Netflix") cloud-based distributed computing system, which relies on the Titus architecture, and which Netflix made and uses to power various aspects of Netflix's business, such as video streaming, recommendations, machine learning, big data, content encoding, studio technology, internal engineering tools, and other Netflix workloads (the "Accused System").

This chart is preliminary and based on information presently available to Plaintiff Avago Technologies International Sales Pte. Ltd. ("Avago"). Netflix has yet to produce any documents or materials in this matter, or to provide any discovery relating to the accused products, systems, and methods. Notably, Avago does not currently have access to the internal technical documentation and source code associated with the Accused System. This infringement chart was therefore made without the benefit of these materials. Avago expressly reserves the right to amend this infringement chart as additional documents, materials, and information becomes available to Avago, including but not limited to the internal technical documentation and source code described above.

These contentions are not an admission of any kind, including with regard to claim construction, as claim construction proceedings have not yet occurred. Avago expressly reserves the right to amend these contentions based on the outcome of those proceedings. This claim chart is exemplary in nature, and merely designed to put Netflix on notice of Avago's current infringement contentions.

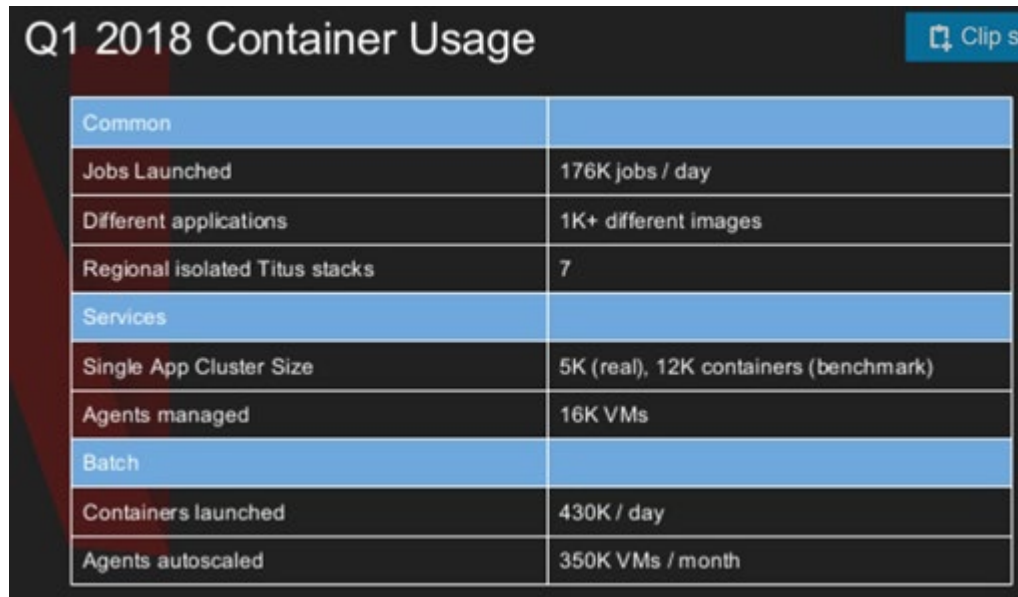
Claim Language	Contentions
<b>1.p.</b> A distributed computing system comprising:	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System includes a distributed computing system.</p> <p>For example, Netflix developed and uses the Titus Container Management System ("Titus") to "provide[] scalable and reliable container execution and cloud-native integration with Amazon AWS."<sup>1</sup> "Titus was built internally at Netflix and is used in production" by Netflix.<sup>2</sup></p> <p>As Netflix explains, "Titus powers critical aspects of the Netflix business, from video streaming, recommendations, and machine learning, big data, content encoding, studio technology, internal engineering tools, and other Netflix workloads."<sup>3</sup></p>

<sup>1</sup> See <https://netflix.github.io/titus/>.

<sup>2</sup> *Id.*

<sup>3</sup> <https://netflixtechblog.com/titus-the-netflix-container-management-platform-is-now-open-source-f868c9fb5436>.

In Netflix's words, "Titus offers a convenient model for managing compute resources."<sup>4</sup> Netflix publications indicate that Netflix uses the Titus platform to launch and manage millions of containers per week (if not more), and to host thousands of applications globally across tens of thousands of virtual machines (if not more).<sup>5</sup>



Q1 2018 Container Usage	
Common	
Jobs Launched	176K jobs / day
Different applications	1K+ different images
Regional isolated Titus stacks	7
Services	
Single App Cluster Size	5K (real), 12K containers (benchmark)
Agents managed	16K VMs
Batch	
Containers launched	430K / day
Agents autoscaled	350K VMs / month

Source: <https://www.slideshare.net/aspyker/container-world-2018>.

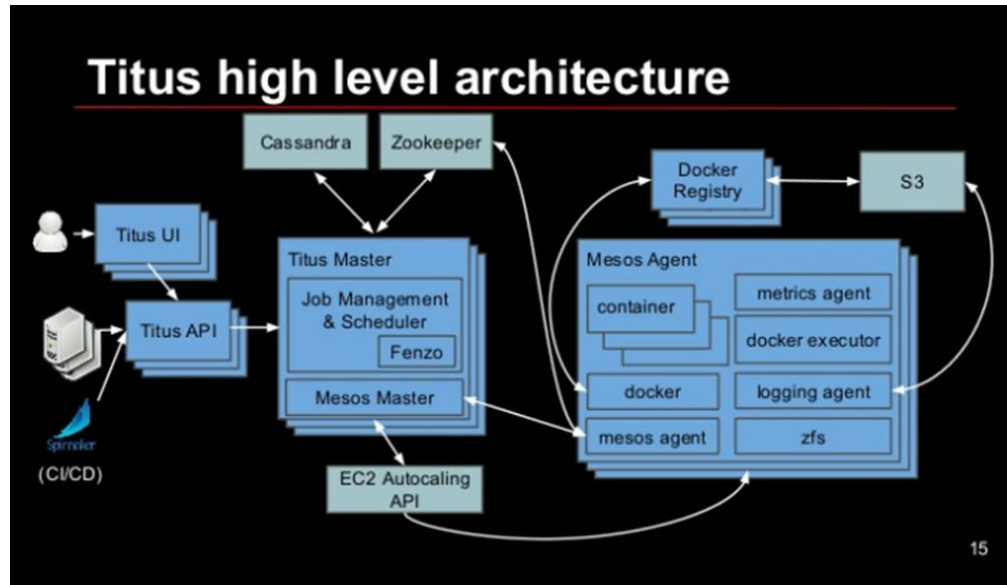
"Titus is a framework [that runs] on top of Apache Mesos, a cluster-management system that brokers available resources across a fleet of machines. . . . Titus consists of a replicated leader-elected scheduler called Titus Master."<sup>6</sup> According to Netflix, the Titus Master is responsible for

<sup>4</sup> See <https://medium.com/netflix-techblog/titus-the-netflix-container-management-platform-is-now-open-source-f868c9fb5436>.

<sup>5</sup> See *id.* ("...container use at Netflix has grown from thousands of containers launched per week to as many as three million containers launched per week in April 2018. Titus hosts thousands of applications globally over seven regionally isolated stacks across tens of thousands of EC2 virtual machines.").

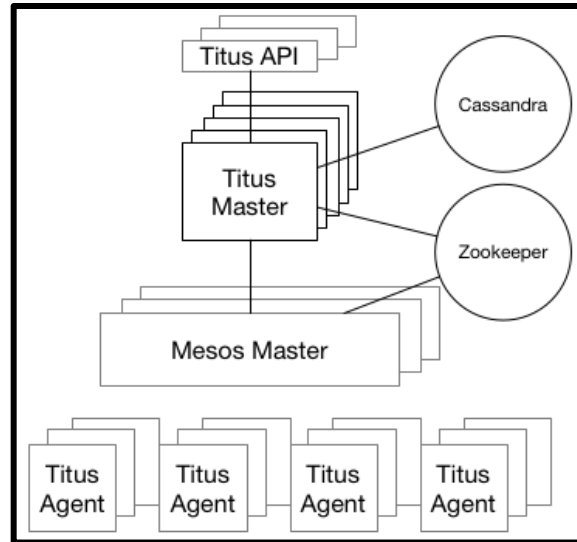
<sup>6</sup> <https://netflix.github.io/titus/overview/>.

persisting job and task information, scheduling tasks, and managing the pool of EC2 virtual machine instances on which the Titus Agents execute in order to launch and run containerized applications.<sup>7</sup>



Source: <https://www.slideshare.net/aspyker/netflix-and-containers-titus>.

<sup>7</sup> *Id.*



Source: <https://netflix.github.io/titus/overview/>.

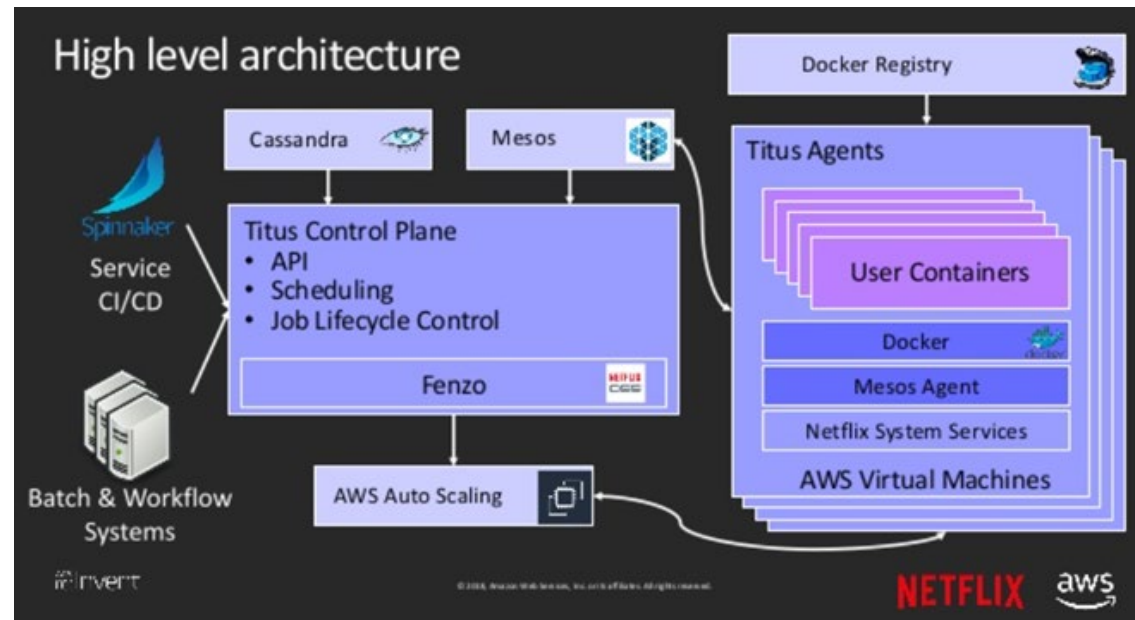
For instance, Netflix developers explain that “Titus consists of a replicated, leader-elected scheduler called Titus Master, which handles the placement of containers onto a large pool of EC2 virtual machines called Titus Agents, which manage each container’s life cycle.”<sup>8</sup>

To the extent the Accused System relies on third-party cloud computing resources, such as Elastic Computing virtual machine services (“EC2”) and Simple Storage Solutions data storage services (“S3”) supplied by Amazon Web Services (“AWS”), Netflix still directs, controls, and directly benefits from the infringing functionality described in this chart. For example, while Netflix utilizes generic cloud computing resources purchased from AWS—which can be used for virtually any computing task—Netflix developed, uses, configures, and controls the Titus software and other software that runs on and controls the relevant operations of these generic computing resources.

<sup>8</sup> <https://queue.acm.org/detail.cfm?id=3158370>.

**1.a.** 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 application nodes, the image instances of the virtual machine manager provide a plurality of virtual machines, each of the plurality of virtual machines 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

The Accused System includes a software image repository comprising non-transitory, computer-readable media operable to store a plurality of image instances of a virtual machine manager that is executable on a plurality of application nodes.



Source: <https://www.slideshare.net/aspyker/cmp376-another-week-another-million-containers-on-amazon-ec2>.

For example, Netflix uses one or more S3 “cloud” servers, git repositories, and/or Docker registries to store a plurality of Titus Agent image instances. The S3 “cloud” servers, git repositories, and Docker registries operate on physical server hardware. Like all, or nearly all, modern computer systems, the server hardware uses memory (non-transitory, computer-readable media) to store the files and data its holds (in this case the repository and its contents).

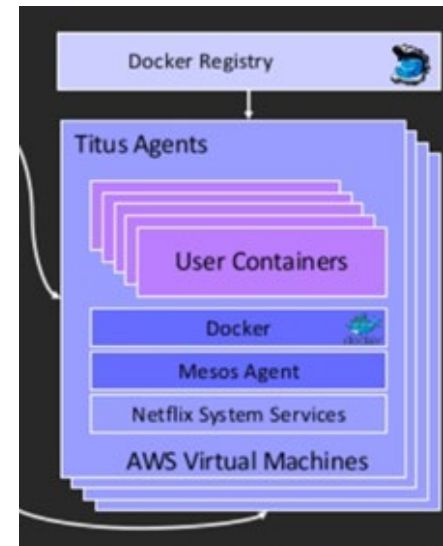
As Netflix explains, the Titus Agent images may be configured for different CPU, GPU, and memory configurations, such as “r3.8xl,” “p2.8xl,” and “m4.4xl” configurations.

	<p>We run a peak of 500 r3.8xl instances in support of our batch users. That represents 16,000 cores of compute with 120 TB of memory. We also added support for GPUs as a resource type using p2.8xl instances to power deep learning with neural nets and mini-batch. In the early part of 2017, our stream-processing-as-a-service team decided to leverage Titus to enable simpler and faster cluster management for their Flickr based system. [. . .] These and other services use thousands of m4.4xl instances.<sup>9</sup></p> <p>Netflix further explains that the Titus Master causes the Titus Agent image instances to be copied from storage to a plurality of EC2 instances during a scaling operation.<sup>10</sup></p> <p>The Titus Master is responsible for persisting job and task information, scheduling tasks, and managing the pool of EC2 Agents. [. . .] The Master schedules tasks onto Agents with available resources and scales the pool of Titus Agents up or down in response to demand.<sup>11</sup></p>
--	---

<sup>9</sup> <https://netflixtechblog.com/the-evolution-of-container-usage-at-netflix-3abfc096781b>.

<sup>10</sup> <https://netflix.github.io/titus/overview/>; see also, <https://queue.acm.org/detail.cfm?id=3158370> (explaining that “Titus uses Fenzo, an extensible scheduler library for Mesos frameworks . . . to assign[] tasks to resource offers presented by Mesos and support[] a variety of scheduling objectives that can be configured and extended.”).

<sup>11</sup> <https://netflix.github.io/titus/overview/>.



Source: <https://www.slideshare.net/aspyker/cmp376-another-week-another-million-containers-on-amazon-ec2> (depicting Titus Agents executing on multiple “AWS Virtual Machines,” (EC2 servers)).

The Netflix system further includes the previously discussed plurality of image instances of a virtual machine manager, wherein when executed on the application nodes, the image instances of the virtual machine manager provide a plurality of virtual machines, each of the plurality of virtual machines operable to provide an environment that emulates a computer platform.

For example, each Titus Agent image instance, when executed on an application node (e.g., an EC2 server) provides a Titus Agent virtual machine; hence, multiple Titus Agent image instances provide a plurality of virtual machines.

Further, each Titus Agent operates to provide an environment that emulates a computer platform. For example, each Titus Agent comprises virtual memory and processors, an operating system

	<p>(“OS”), and various other dependencies required to launch and manage software applications.<sup>12</sup> For example, an “r3.8xl” Titus Agent instance provides a virtual machine comprising 32 virtual CPUs and 244 GB of memory<sup>13</sup> and an “m4.4xl” Titus Agent instance provides a virtual machine comprising 16 virtual CPUs and 64 GB of memory.<sup>14</sup></p> <p>The Accused System includes the software image repository previously discussed, further operable to store a plurality of image instances of a plurality of software applications that are executable on the plurality of virtual machines.</p> <p>For example, Netflix uses S3 “cloud” servers and/or git repositories to host Docker registries that store a curated set of container images that may be executed on the Titus Agents.<sup>15</sup> As Docker explains, a Docker registry is a “stateless, highly scalable server side application that stores and lets you distribute Docker [container] images.”<sup>16</sup> As Netflix explains:</p> <p style="padding-left: 40px;">We also added things to Docker. Some of the things we’ve done is we’ve curated a set of base images . . . for people that are looking for a base Java image to start with or a base Node image to start with, we have a curated set of images for that.<sup>17</sup></p>
--	---

<sup>12</sup> *Id.* (“Titus Agents are responsible for setting up and running task containers and managing their lifecycle. The Agent sets up on host resources, such as storage and networking resources, and launches the container using Docker.”).

<sup>13</sup> See, e.g., <https://aws.amazon.com/about-aws/whats-new/2014/04/10/r3-announcing-the-next-generation-of-amazon-ec2-memory-optimized-instances/>.

<sup>14</sup> See, e.g., <https://aws.amazon.com/ec2/instance-types/>.

<sup>15</sup> <https://www.infoq.com/presentations/netflix-titus-2018/>; see also, <https://netflix.github.io/titus/install/prereqs/> (explaining that Netflix “operate[s] a docker registry based on Docker Distribution”); <https://docs.docker.com/registry/> (explaining that companies, such as Netflix, may create a private registry to store these container image instances, or they may use the public repository (“Docker Hub”) available through Docker’s website).

<sup>16</sup> <https://docs.docker.com/registry/>.

<sup>17</sup> <https://www.infoq.com/presentations/netflix-titus-2018/>.



	<div data-bbox="987 230 1522 685" data-label="Diagram"> </div> <p>Source: <a href="https://www.slideshare.net/aspyker/netflix-and-containers-titus">https://www.slideshare.net/aspyker/netflix-and-containers-titus</a> (depicting integration of Docker image instances with S3 “cloud” server; red box added).</p> <p>As Docker explains each container provides “everything needed to run an application: code, runtime, system tools, system libraries and settings.”<sup>18</sup> Netflix further explains that “[u]sing containers, Titus runs any batch application letting the user specify exactly what application code and dependencies are needed.”<sup>19</sup></p>
<p><b>1.b.</b> 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</p>	<p>The Netflix system includes 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.</p>

<sup>18</sup> <https://www.docker.com/resources/what-container>.

<sup>19</sup> <https://netflixtechblog.com/the-evolution-of-container-usage-at-netflix-3abfc096781b>.

<p>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.</p>	<p>For example, the Titus framework employs an automation infrastructure known as a “Titus Master.”<sup>20</sup> As Netflix explains, the Titus Master is responsible for persisting job and task information, scheduling tasks, and automatically scaling the pool of Titus Agents up and down in response to demand.<sup>21</sup> For example, the Titus Master “configures both the AWS Auto Scaling policies and CloudWatch alarms for the service” and monitors performance metrics for both the application and the container-level system metrics (e.g., CPU and memory usage) to determine whether the policy thresholds are being breached.<sup>22</sup> As Netflix explains:</p> <p style="padding-left: 40px;">As service apps running on Titus emit metrics, AWS analyzes the metrics to determine whether CloudWatch alarm thresholds are being breached. If an alarm threshold has been breached, AWS triggers the alarm’s associated scaling actions. These actions result in calls to the configured API Gateway endpoints to adjust instance counts. Titus responds to these calls by scaling up or down the Job accordingly.<sup>23</sup></p> <p>In such a scaling operation, the Titus Master autonomically causes a plurality of Titus Agent images to be deployed from Netflix’s S3 “cloud” servers, git repositories and/or Docker registries onto the newly provisioned EC2 virtual machine instances.<sup>24</sup></p> <p>The Titus Master of the Netflix system additionally provides 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.</p>
---	---

<sup>20</sup> The Titus Master is also known as the “Titus control plane.” See <https://www.slideshare.net/InfoQ/disenchantment-netflix-titus-its-feisty-team-and-daemons>, at slide 12 (identifying the Titus control plane); see also, <https://www.slideshare.net/aspyker/netflix-and-containers-titus>, at slide 15 (identifying the Titus Master).

<sup>21</sup> <https://netflix.github.io/titus/overview/> (As discussed *supra* at fn.10 and *infra* at claims 9, 11 and 12, the Netflix Titus Master employs an internally developed scheduler known as “Fenzo” to carry out task scheduling and resource autoscaling operations).

<sup>22</sup> <https://netflixtechblog.com/auto-scaling-production-services-on-titus-1f3cd49f5cd7>.

<sup>23</sup> *Id.*

<sup>24</sup> <https://queue.acm.org/detail.cfm?id=3158370> (The Titus Master “handles the placement of containers onto a large pool of EC2 virtual machines called Titus Agents, which manage each container’s life cycle.”).

	<p>For example, as previously discussed, the Titus Master causes Titus Agents to be autonomically scaled up or down in response to demand.<sup>25</sup> The Titus Master then provides for the autonomic deployment of the container image instances onto the Titus Agents virtual machines that are executing on one or more EC2 instances. In Netflix’s words, the Titus Master “handles the placement of containers onto a large pool of EC2 virtual machines called Titus Agents, which manage each container’s life cycle.”<sup>26</sup></p>
<p><b>4.</b> 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.</p>	<p>The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System also includes 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.</p> <p>For example, as previously discussed with regard to claim 1 (which is incorporated by reference herein), the container image files execute on one or more Titus Agents. In so doing, the containers enable an application capable of receiving and responding to requests received from the Titus Master, as well as from other Titus containers and Netflix applications.</p> <p>For instance, Netflix states, “Titus supports both service jobs that run ‘forever’ and batch jobs that run ‘until done’.”<sup>27</sup> As Netflix further explains, the Titus container platform enables “service to service communication.”<sup>28</sup> In Netflix’s words, “[w]e avoided reimplementing service discovery and service load balancing. Instead we provided a full IP stack enabling containers to work with existing Netflix service discovery and DNS (Route 53) based load balancing.”<sup>29</sup></p>

<sup>25</sup> <https://netflix.github.io/titus/overview/>.

<sup>26</sup> *Id.*

<sup>27</sup> <https://netflixtechblog.com/the-evolution-of-container-usage-at-netflix-3abfc096781b>.

<sup>28</sup> *Id.*

<sup>29</sup> *Id.*

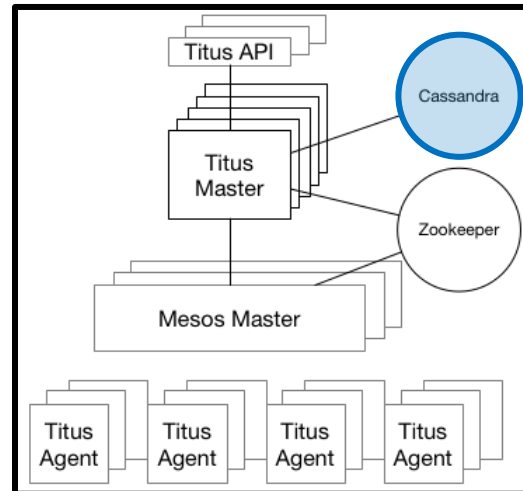
	<div data-bbox="798 230 1680 719" data-label="Image"> </div> <p>Source: <a href="https://www.infoq.com/presentations/netflix-titus-2018/">https://www.infoq.com/presentations/netflix-titus-2018/</a> (explaining that Titus enables service to service communication).</p>
<p><b>8.</b> The distributed computing 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.</p>	<p>The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System includes 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.</p> <p>For example, as previously discussed with regard to claim 1 (which is incorporated herein by reference), the Titus Master maintains a plurality of Titus Agent image instances and container application image instances in one or more S3 “cloud” servers, git repositories, and/or Docker registries. As the AWS documentation explains, S3 provides “persistent virtual disk storage similar to a physical disk drive for files or other data that must persist longer than the lifetime of a single EC2 instance.”<sup>30</sup></p>

<sup>30</sup> <https://d0.awsstatic.com/whitepapers/AWS%20Storage%20Services%20Whitepaper-v9.pdf>.

As Netflix developers explain, the Titus system works in conjunction with the Apache Cassandra software in order to store Titus Agent images and container images in persistent or non-persistent memory.<sup>31</sup> In Netflix’s words,

Titus consists of a replicated, leader-elected scheduler called Titus Master, which handles the placement of containers onto a large pool of EC2 virtual machines called Titus Agents, which manage each container’s life cycle. Zookeeper manages leader election, and *Cassandra persists the master’s data*.<sup>32</sup>

Hence, the Titus system—specifically the Titus Master—utilizes Cassandra to persist the plurality of Titus Agents image instances and container image instances (i.e., the “master’s data”).



Source: <https://netflix.github.io/titus/overview/> (blue circle added).

<sup>31</sup> <https://queue.acm.org/detail.cfm?id=3158370>; see also, [https://en.wikipedia.org/wiki/Apache\\_Cassandra](https://en.wikipedia.org/wiki/Apache_Cassandra) (Apache Cassandra is a “NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency for all clients.”).

<sup>32</sup> <https://netflix.github.io/titus/overview/> (emphasis added); see also, <https://cassandra.apache.org/> (explaining that Netflix uses Cassandra to persist data across its network and processes over 1 trillion requests per day).

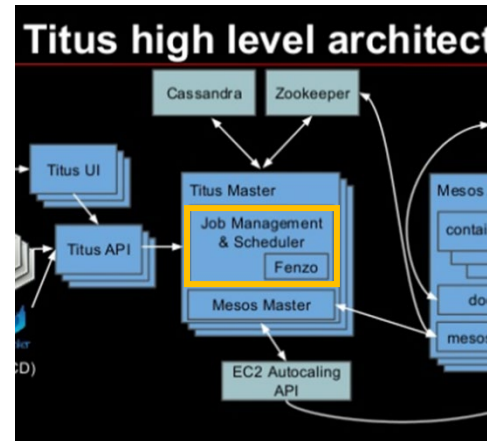
9. The distributed computing system of claim 1, wherein the control node further comprises one or more rule engines that provide autonomic deployment of the software applications to the virtual machines in accordance with a set of one or more rules.

The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.

The Accused System also includes 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.

For example, as previously discussed with regard to claim 1 (which is incorporated by reference herein), the Titus Master utilizes Fenzo, an extensible scheduler library for Mesos frameworks developed by Netflix, to schedule job tasks (user requests to run a particular software application or process) onto resources.<sup>33</sup> As Netflix developers explain,

Titus uses Fenzo's bin packing in conjunction with its agent autoscaling capabilities to grow and shrink the agent pool dynamically as workload demands. . . . Fenzo supports the concept of a *fitness calculator*, which allows the quality of the scheduling decision to be tuned.<sup>34</sup>

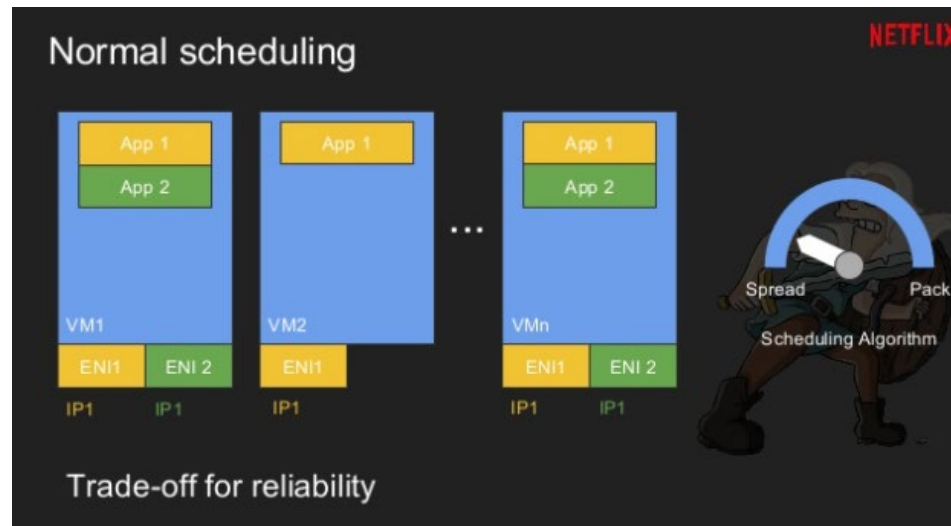


Source: <https://www.slideshare.net/aspyker/netflix-and-containers-titus> (yellow box added).

<sup>33</sup> <https://queue.acm.org/detail.cfm?id=3158370>.

<sup>34</sup> *Id* (emphasis in original).

As Netflix explains, the Fenzo scheduler implements various user-defined “constraints” in order to select a resource on which to run a task.<sup>35</sup> For instance, “[y]ou can tell the Fenzo scheduler the circumstances under which it should perform autoscaling and what criteria it should use to make autoscaling decisions.”<sup>36</sup>



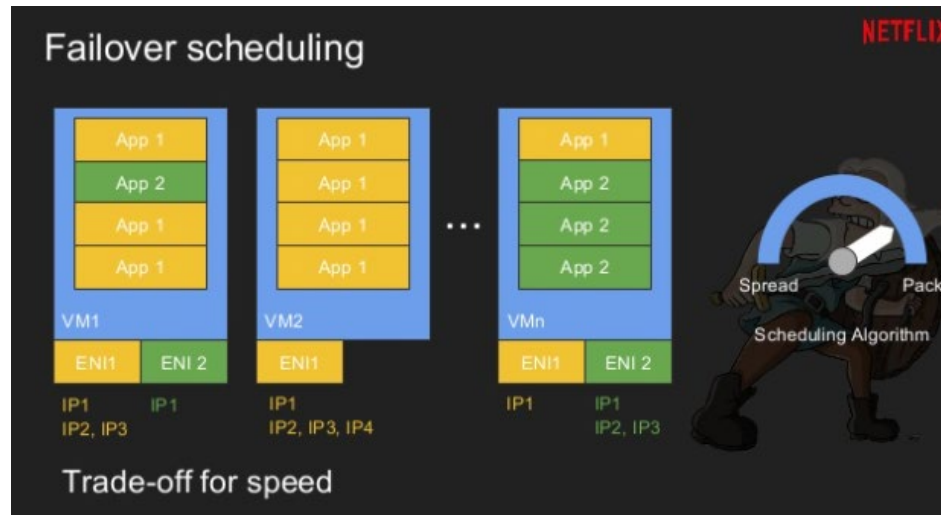
Source: <https://www.slideshare.net/InfoQ/disenchantment-netflix-titus-its-feisty-team-and-daemons>.

For example, as depicted above, the Titus Master (via Fenzo) operating under normal conditions may schedule containers onto Titus Agents using a “spread” algorithm in order to maximize distribution of tasks across the entire network of Titus Agents.<sup>37</sup>

<sup>35</sup> <https://github.com/Netflix/Fenzo/wiki/Introduction>.

<sup>36</sup> *Id.*

<sup>37</sup> <https://www.slideshare.net/InfoQ/disenchantment-netflix-titus-its-feisty-team-and-daemons>.



Source: <https://www.slideshare.net/InfoQ/disenchantment-netflix-titus-its-feisty-team-and-daemons>.

As another example, during a failover event, the Titus Master (via Fenzo) may optionally implement a “pack” algorithm to distribute tasks across Titus Agents more efficiently in exchange for certain quality of service tradeoffs.<sup>38</sup> As Netflix explains,

Fenzo adopts a strategy of scoring hosts for a task placement, with higher scores indicating better ‘fitness.’ This scoring is performed by a *fitness calculator* plugin that rates the fitness of a task on a given host. Fenzo includes some built-in plugins that you can use for some common bin packing use cases. . . . In this way, Fenzo can evaluate fitness on a subset of the hosts, until one is found with a good enough fitness value. You can dynamically control the speed of task assignment and its trade off with fitness optimization as needed by changing this ‘good enough’ function.<sup>39</sup>

<sup>38</sup> *Id.*

<sup>39</sup> <https://github.com/Netflix/Fenzo/wiki/Introduction> (emphasis in original).



	Thus, the Titus Master “fine-tune[s] how the Fenzo scheduler assigns tasks [i.e., containers] to compatible hosts [Titus Agents] by using fitness calculators and constraints.” <sup>40</sup>
<b>10.</b> The distributed computing system of claim 9, wherein the one or more rule engines are forward-chaining rule engines.	<p>The Accused System provides the system of claim 9, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System also includes the distributed computing system of claim 9, wherein the one or more rules engines are forward-chaining rules engines.</p> <p>For example, as previously discussed with regard to claim 9 (which is incorporated by reference herein), the Fenzo software comprises a forward-chaining rules engine. As previously explained, Fenzo implements various user-selected, customizable constraints to select a resource on which to run a task.<sup>41</sup> As Netflix explains:</p> <p style="padding-left: 40px;">[Y]ou provide a function to Fenzo that determines if the fitness of a host is “good enough.” If that function says that the fitness of the host is good enough, Fenzo will stop its search for a more optimal host. . . . You may also apply certain <i>constraints</i> to tasks, such that a host must meet these constraints before Fenzo will consider assigning the task to that host. . . . If that host does meet all hard constraints, Fenzo will then evaluate the soft constraints along with the fitness calculators. . . .<sup>42</sup></p> <p>As Netflix further explains, “frequently there are many hosts that meet these baseline qualifications.”<sup>43</sup> The Fenzo scheduler allows a framework (such as the Titus container system) to fine-tune how tasks are assigned to compatible hosts using fitness calculators and constraints. In Netflix’s words, “[i]t is also possible to combine multiple fitness calculators and constraints into a single, weighted, task scheduling optimization strategy.”<sup>44</sup></p>

<sup>40</sup> *Id.*

<sup>41</sup> *Id.*

<sup>42</sup> *Id.*

<sup>43</sup> *Id.*

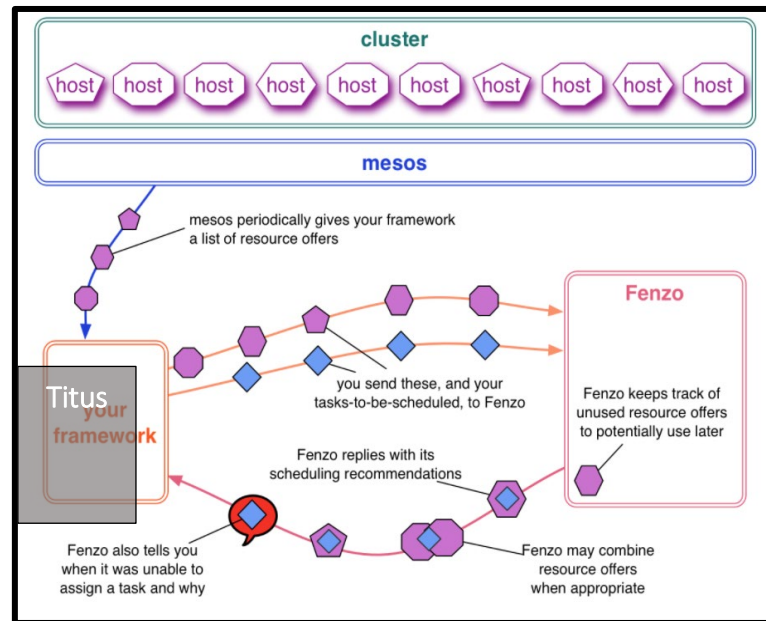
<sup>44</sup> *Id.*

**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.

The Accused System provides the system of claim 9, as described above, which allegations are incorporated by reference herein.

The Netflix system also includes 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.

For example, as previously discussed with regard to claim 9 (which is incorporated by reference herein), the Fenzo scheduler autonomically controls the deployment of containers to Titus Agents executing on a plurality of EC2 instances in accordance with a scheduling loop.



Source: <https://github.com/Netflix/Fenzo/wiki> (depicting a high-level overview of the scheduling loop utilized by the Titus/Fenzo architecture in order to deploy containers to Titus Agents; grey box labeled “Titus” added).

As Netflix explains, the Fenzo scheduler applies a sequence of steps to create a scheduling loop.<sup>45</sup> Netflix describes the “typical task scheduling loop” for Fenzo as comprising the following steps:

- [Step 1]** You get the latest resource offers from Mesos.
- [Step 2]** You create the list of tasks you want to assign.
- [Step 3]** You call the Fenzo task scheduler to assign tasks to hosts.
- [Step 3.1]** The task scheduler rejects any old, unused offers that have expired.
- [Step 3.2]** The task scheduler combines new and old offers to determine total resource availability.
- [Step 3.3]** The task scheduler assigns tasks to hosts.
- [Step 3.4]** The task scheduler rejects any offers that remain unused and that have expired.
- [Step 3.5]** The task scheduler (optionally) launches the autoscaling evaluator, asynchronously.
- [Step 3.6]** The task scheduler returns a report on the task assignments.
- [Step 4]** You check this result for tasks that Fenzo was unable to assign and may need to be reattempted.
- [Step 5]** You tell Mesos to launch any tasks that Fenzo has successfully assigned to hosts.
- [Step 6]** You call Fenzo’s *taskAssigner* for each task that you launch.
- [Step 7]** If you choose to not launch the tasks that Fenzo assigned resources to from a host, you will have to either reject the resource offers in the assignment result for that host, or re-offer those resource offers to the task scheduler again. Otherwise, those resource offers would be "leaked out".
- [Step 8]** You return to the beginning and start this process again.<sup>46</sup>

<sup>45</sup> <https://github.com/Netflix/Fenzo/wiki/Scheduling-Tasks>.

<sup>46</sup> *Id.*

	<p>Additionally, the Titus Master automatically updates the Fenzo scheduler by, for example, collecting and providing to Fenzo utilization metrics for the existing host resources.<sup>47</sup> For example, the Fenzo scheduler may receive information regarding the number of CPU cores (CPUs), the amount of disk space, memory, and/or network bandwidth for each resource that is known to it.<sup>48</sup></p> <p>Using this information, Fenzo can be configured to employ a different set of constraints when determining where to run a particular software application.<sup>49</sup> For instance, upon determining that a certain resource allocation threshold has been breached (<i>see, e.g.</i>, claim element 1.b., which is incorporated by reference herein), Fenzo may employ different policies to achieve a desired scheduling objective.<sup>50</sup></p> <p>As another example, the utilization metrics received from the Titus Master concerning the actual state of the application nodes causes Fenzo to automatically scale the number of resources up or down in accordance with an autoscale rule.<sup>51</sup></p>
<b>12.</b> 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	<p>The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.</p> <p>The Netflix system also includes 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.</p>

<sup>47</sup> <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-the-amount-of-resources-currently-available-on-particular-hosts>.

<sup>48</sup> *Id.*

<sup>49</sup> *See, e.g.*, <https://github.com/Netflix/Fenzo/wiki/Resource-Allocation-Limits> (“You can restrict the amount of resources (CPU cores, disk space, memory, and bandwidth) available to a group of tasks, such that a new task in that group will not be assigned even to an available and fit host if that task would cause the group of tasks to exceed those resource limits.”).

<sup>50</sup> *Id.*

<sup>51</sup> *See, e.g.*, <https://github.com/Netflix/Fenzo/wiki/Autoscaling> (explaining how to configure an autoscale rule that “[r]eturn[s] true if that host has sufficient resources to be considered an idle but potentially-useful host” and “return false if that host will not be sufficient for the tasks assigned to this autoscale group”).

<p>of the software applications to the virtual machines within the distributed computing system.</p>	<p>For example, as previously described with regard to claims 1, 9 and 11 (which are incorporated by reference herein), the Titus Master, via the Fenzo scheduler, comprises a scheduling loop that controls the execution of the containers on Titus Agents. As previously described, this scheduling loop further includes autoscaling rules that control the deployment and undeployment of containers to Titus Agents.<sup>52</sup> As Netflix explains:</p> <p style="padding-left: 40px;">Your <i>AutoScaler</i> callback method takes as its parameter an object that implements the <i>AutoScaleAction</i> interface. This object indicates whether your method is to scale the cluster up or down.<sup>53</sup></p>
<p><b>14.</b> 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.</p>	<p>The Accused System provides the system of claim 9, as described above, which allegations are incorporated by reference herein.</p> <p>The Netflix system includes 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.</p> <p>For example, as previously described with regard to claims 9, 11 and 12 (which are incorporated by reference herein), the Fenzo scheduler applies a sequence of steps to create a scheduling loop.<sup>54</sup> This scheduling loop includes, among others, the step of assigning tasks to hosts based on a task scheduling optimization strategy that includes multiple fitness calculators and/or constraints.<sup>55</sup></p> <p>As previously discussed with regard to claim 11, as part of the scheduling loop, the Titus Master automatically updates the Fenzo scheduler by, for example, collecting and providing to Fenzo</p>

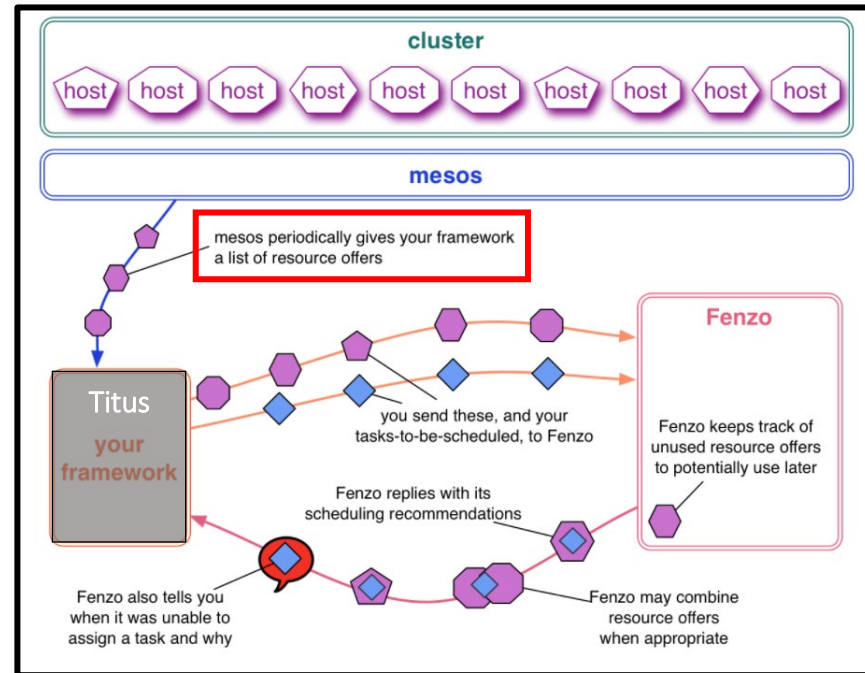
<sup>52</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.

<sup>53</sup> *Id.*; see also, <https://queue.acm.org/detail.cfm?id=3158370> (“Titus uses Fenzo’s bin packing in conjunction with its agent autoscaling capabilities to grow and shrink the agent pool dynamically as workload demands.”).

<sup>54</sup> <https://github.com/Netflix/Fenzo/wiki/Scheduling-Tasks>.

<sup>55</sup> *Id.*; see also, <https://github.com/Netflix/Fenzo/wiki/Introduction>.

utilization metrics for the existing host resources.<sup>56</sup> For instance, the Fenzo scheduler may receive information regarding the number of CPU cores (CPUs), the amount of disk space, memory, and/or network bandwidth for each resource that is known to it.<sup>57</sup> This information is then used to update the list of resource offers provided to the scheduler so that Fenzo may assign tasks in a manner that best achieves the desired scheduling objective based on the updated state information.<sup>58</sup>



Source: <https://github.com/Netflix/Fenzo/wiki> (depicting how Mesos periodically provides an updated list of resource offers that the Titus Master then provides to Fenzo; red box, grey box labeled “Titus” added).

<sup>56</sup> <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-the-amount-of-resources-currently-available-on-particular-hosts>.

<sup>57</sup> *Id.*

<sup>58</sup> <https://github.com/Netflix/Fenzo/wiki>.

<p><b>15.</b> The distributed computing system of claim 1, wherein the automation subsystem includes a rule compiler that compiles rules into one or more discrimination networks.</p>	<p>The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System includes the distributed computing system of claim 1, wherein the automation subsystem includes a rule compiler that compiles rules into one or more discrimination networks.</p> <p>For example, as previously described with regard to claim 9 (which is incorporated by reference herein), the Fenzo scheduler compiles various fitness calculators and hard and/or soft constraints in order to select a resource on which to run a task.<sup>59</sup> In Netflix’s words, “[i]t is also possible to combine multiple fitness calculators and constraints into a single, weighted, task scheduling optimization strategy.”<sup>60</sup></p> <p>Further, Netflix explains that Fenzo’s scheduling strategy is compiled to avoid redundant tests during rule execution. For example, Netflix states the following:</p> <p style="padding-left: 40px;">In this way, Fenzo can evaluate fitness on a subset of the hosts, until one is found with a good enough fitness value. You can dynamically control the speed of task assignment and its trade off with fitness optimization as needed by changing this “good enough” function.</p> <p style="padding-left: 40px;">You may also apply certain <i>constraints</i> to tasks, such that a host must meet these constraints before Fenzo will consider assigning the task to that host. Constraints may be “hard” or “soft.” Fenzo evaluates the hard constraints first. If a host fails to meet a hard constraint, Fenzo will not consider that host for the task. If that host does meet all hard constraints, Fenzo will then evaluate the soft constraints along with the fitness calculators: The fitness score for a host is an aggregation of the results of the fitness calculators and the soft constraints.<sup>61</sup></p>
--	---

<sup>59</sup> <https://github.com/Netflix/Fenzo/wiki/Introduction>.

<sup>60</sup> *Id.*

<sup>61</sup> *Id.* (italics in original).

Regarding how Fenzo avoids redundant tests, Netflix provides the following summary of the Fenzo scheduling strategy as well as a fitness evaluator used in Titus:

### Fenzo scheduling strategy

```
For each (ordered) task
  On each available host
    Validate hard constraints
    Score fitness and soft constraints
  Until score good enough, and
    A minimum #hosts evaluated
  Pick host with highest score
```

### Current fitness evaluator in Titus

Combines resource request bin packing with task type bin packing

```
resBinpack = (cpuFit + memFit + networkFit) / 3.0
taskTypePack = numSameType / totTasks

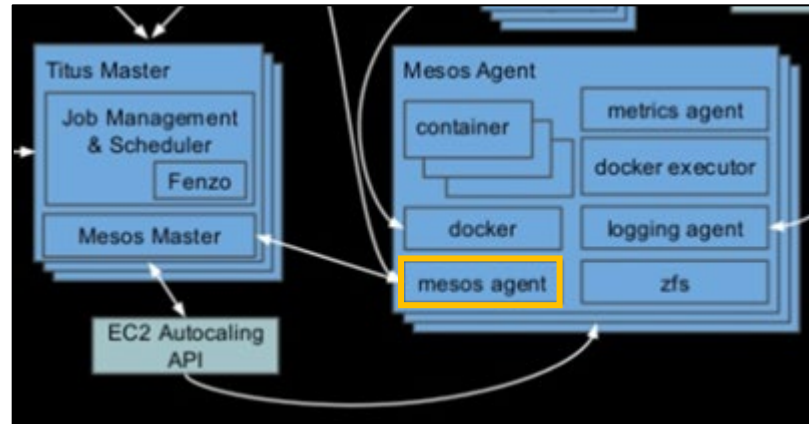
fitness = resBinpack * 0.4 + taskTypePack * 0.6
```

Source: [http://platformlab.stanford.edu/Seminar%20Talks/Netflix\\_seminar.pdf](http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf).



<p><b>16.p.</b> The distributed computing system of claim 1, wherein the control node further comprises:</p>	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System includes a distributed computing system.</p> <p><i>See</i> claim 1, as described above, which allegations are incorporated by reference herein.</p>
<p><b>16.a.</b> 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</p>	<p>The Accused System includes 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.</p> <p>For example, as previously described with regard to claims 1 and 9 (which are incorporated by reference herein), the Fenzo software comprises an automation subsystem wherein the Fenzo scheduler autonomically assigns tasks to containers using a series of fitness calculators and/or constraints. As discussed with regard to claim 11 (which is incorporated by reference herein), Fenzo may dynamically respond to status information, such as resource utilization information, collected from Titus Agents running on one or more EC2 instances.<sup>62</sup></p> <p>To do so, the Titus framework provides a “mesos agent” that collects and emits performance metrics from the EC2 Titus Agent instances to the Titus Master.</p>

<sup>62</sup> *See, e.g.*, <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.



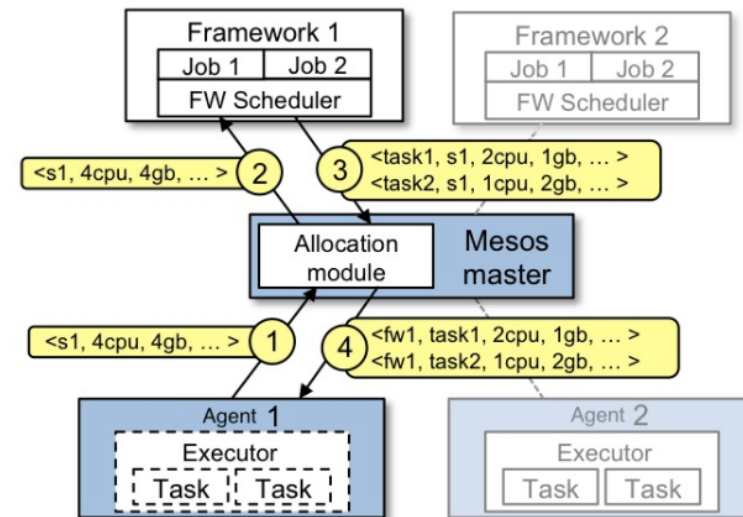
Source: <https://www.slideshare.net/aspyker/netflix-and-containers-titus> (yellow box added)

The Apache Mesos software documentation describes the interaction between the mesos agent and the framework (i.e., the Titus Master) using the following figure and explanation.<sup>63</sup>

<sup>63</sup> <http://mesos.apache.org/documentation/latest/architecture/>.

## Example of resource offer

The figure below shows an example of how a framework gets scheduled to run a task.



Let's walk through the events in the figure.

1. Agent 1 reports to the master that it has 4 CPUs and 4 GB of memory free. The master then invokes the allocation policy module, which tells it that framework 1 should be offered all available resources.
2. The master sends a resource offer describing what is available on agent 1 to framework 1.
3. The framework's scheduler replies to the master with information about two tasks to run on the agent, using <2 CPUs, 1 GB RAM> for the first task, and <1 CPUs, 2 GB RAM> for the second task.
4. Finally, the master sends the tasks to the agent, which allocates appropriate resources to the framework's executor, which in turn launches the two tasks (depicted with dotted-line borders in the figure). Because 1 CPU and 1 GB of RAM are still unallocated, the allocation module may now offer them to framework 2.

Source: <http://mesos.apache.org/documentation/latest/architecture/>.

Consistent with this implementation, Netflix explains that the Titus Master receives the current status information from the mesos agent and then communicates that information to the Fenzo

scheduler using, for example, the resource allocation functions that are built into the Fenzo scheduler.<sup>64</sup> In Netflix’s words:

You can call your task scheduler’s *getVmCurrentStates()* method to get a list of *VirtualMachineCurrentState* objects, one for each host that is known to the task scheduler. Each of these objects has a *getCurrAvailableResources()* method that returns an object that implements the *VirtualMachineLease* interface.<sup>65</sup>

method	description
<code>cpuCores()</code>	Get the number of cores (CPUs) available to be assigned on this host.
<code>diskMB()</code>	Get the amount of disk space, in megabytes, available to be assigned on this host.
<code>hostname()</code>	Get the name of this host.
<code>memoryMB()</code>	Get the amount of memory, in megabytes, available to be assigned on this host.
<code>networkMbps()</code>	Get the amount of network bandwidth, in megabits per second, available to be assigned on this host.
<code>portRanges()</code>	Get the list of port ranges that are available for assigning on this host.

Source: <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-which-resources-are-assigned-to-a-task> (depicting functions or “methods” that the Fenzo scheduler may use to retrieve metrics regarding the actual state information for the EC2 instances).

**16.b.** 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,

The Accused System includes the distributed computing system of claim 1, wherein the control node further comprises 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.

For example, the Fenzo scheduler includes a “shortfall evaluator” that works in conjunction with autoscaler methods—previously discussed with regard to claims 11 and 12 (which are incorporated

<sup>64</sup> <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-which-resources-are-assigned-to-a-task>.

<sup>65</sup> *Id.*

	<p>by reference herein)—to “ensure there will be enough hosts [e.g., EC2 instances or application nodes] up and running to be able to accept all of the incoming tasks.”<sup>66</sup></p> <p>As Netflix explains:</p> <p style="padding-left: 40px;">The resource shortfall analysis attempts to estimate the number of hosts required to satisfy the pending workload. This complements the rules based scale up during demand surges. Fenzo’s autoscaling also complements predictive autoscaling systems.<sup>67</sup></p> <p>The shortfall analysis uses information about EC2 instance utilization metrics, task assignment failures, and pending tasks to make a prediction regarding the resource demands for EC2 instances. As Netflix explains, Fenzo “evaluates task assignment failures and assesses pending tasks to make an estimate of whether, and which, additional resources will be needed.”<sup>68</sup></p> <p>As another example, Netflix utilizes the concept of “tiers” and “capacity groups” to enable Fenzo to optimize scheduling with respect to expected states. As Netflix developers explain, “applications running in EC2 VMs have become accustomed to AWS’s concept of Reserved Instances that guarantee VM capacity if purchased in advance.”<sup>69</sup> These concepts of tiers and capacity groups “enable a more consistent concept of capacity between VMs and containers” and “allow[] applications to make tradeoffs between cost (setting aside possibly unused agent resources for an application) and reliable task execution (ensuring an application cannot be starved by another).”<sup>70</sup></p> <p>For instance, Titus utilizes a “critical” and “flexible” tier. The critical tier “enables Titus to launch containers immediately, without having to wait for EC2 to provision a VM. This tier optimizes around launch latency at the expense of running more VMs than the application may require at the</p>
--	--

<sup>66</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.

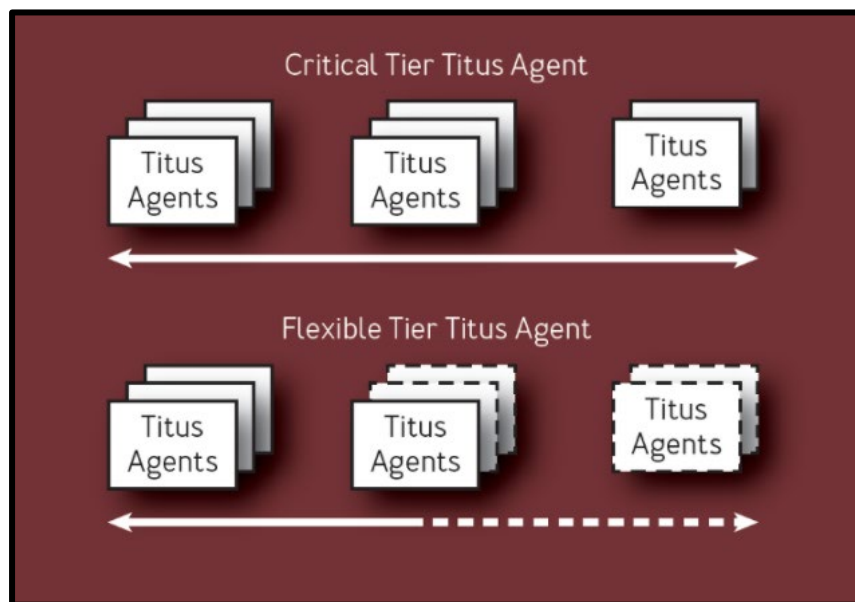
<sup>67</sup> <https://netflixtechblog.com/fenzo-oss-scheduler-for-apache-mesos-frameworks-5c340e77e543>.

<sup>68</sup> <https://github.com/Netflix/Fenzo/wiki/Building-Your-Scheduler>.

<sup>69</sup> <https://queue.acm.org/detail.cfm?id=3158370>.

<sup>70</sup> *Id.*

moment.”<sup>71</sup> By contrast, the flexible tier “provisions only enough agent VMs to handle the current workload (though it does keep a small headroom of idle instances to avoid overly aggressive scale-up and down).”<sup>72</sup>



Source: <https://queue.acm.org/detail.cfm?id=3158370>.

In addition, Netflix utilizes a concept it refers to as capacity groups. As Netflix explains, “[c]apacity groups are a logical concept on top of each tier that guarantee an application or set of applications some amount of dedicated capacity.”<sup>73</sup> As depicted below, these capacity groups may include, for example, a minimum cluster size (guaranteed capacity), an expected or desired cluster size, and a maximum cluster size.<sup>74</sup> The Fenzo scheduler in turn assigns tasks to hosts and

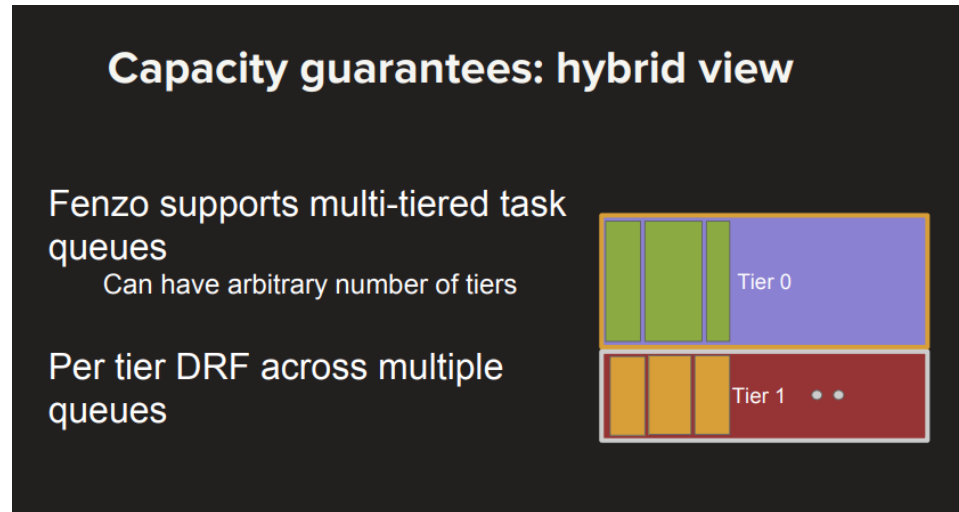
<sup>71</sup> *Id.*

<sup>72</sup> *Id.*

<sup>73</sup> *Id.*

<sup>74</sup> See [http://platformlab.stanford.edu/Seminar%20Talks/Netflix\\_seminar.pdf](http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf).

optionally conducts scale-up or down operations based on the specified capacity groups in order to achieve a desired scheduling objective such as optimizing throughput for batch jobs and start latency for service jobs.



Source: [http://platformlab.stanford.edu/Seminar%20Talks/Netflix\\_seminar.pdf](http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf) (depicting Netflix's concept of "tiers" and "capacity groups" as implemented in the Titus/Fenzo architecture).

	<div data-bbox="764 228 1722 789" data-label="Diagram"> <h3>Sizing clusters for capacity guarantees</h3> <p>Tier 0: Used capacity, Idle capacity, Autoscaled. Cluster min size (guaranteed capacity), Cluster max Size.</p> <p>Tier 1: Used capacity, Autoscaled. Cluster desired size, Cluster max Size. (Idle size kept near zero).</p> </div> <p>Source: <a href="http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf">http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf</a> (depicting Netflix’s concept of “tiers” and “capacity groups” as implemented in the Titus/Fenzo architecture).</p>
<p><b>16.c.</b> 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</p>	<p>Finally, the Accused System includes the distributed computing system of claim 1, 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.</p> <p>As described above, the Fenzo scheduler analyzes EC2 instance status data that is generated by the mesos agent to determine the actual state of the EC2 instances.<sup>75</sup> The Fenzo scheduler then executes a shortfall analysis based on, for example, current utilization of the EC2 instances and pending, unassigned job requests, to determine the anticipated resource demands.<sup>76</sup> After completing these steps, the Fenzo scheduler applies autoscaling rules to scale the cluster up or down</p>

<sup>75</sup> See claim element 16.a., *supra*.

<sup>76</sup> See claim element 16.b., *supra*.



<p>between the actual state and the expected state.</p>	<p>in order to minimize differences between the actual utilization of the EC2 instances and the output of the shortfall evaluation. As Netflix explains:</p> <p style="padding-left: 40px;">By default, Fenzo evaluates task assignment failures and assesses pending tasks to make an estimate of whether, and which, additional resources will be needed. This is useful for determining how to scale up resources.<sup>77</sup></p> <p>Said differently, “Fenzo regulates autoscaling . . . by also applying Fenzo’s resource shortfall evaluation to estimate resource needs based on the current task workload, and trigger a scale-up action when Fenzo believes resources are wanting.”<sup>78</sup></p> <p>In addition, as described <i>infra</i> at claim 17 (which is incorporated by reference herein), Fenzo uses threshold based autoscaling, which reduces any difference between an actual state and an expected state. As Netflix explains:</p> <p style="padding-left: 40px;">Thresholds based autoscaling lets users specify rules per host group (e.g., EC2 Auto Scaling Group, ASG) being used in the cluster. For example, there may be one ASG created for compute intensive workloads using one EC2 instance type, and another for network intensive workloads. Each rule helps maintain a configured number of idle hosts available for launching new jobs quickly.<sup>79</sup></p>
---	--

<sup>77</sup> <https://github.com/Netflix/Fenzo/wiki/Building-Your-Scheduler>.

<sup>78</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.

<sup>79</sup> <https://netflixtechblog.com/fenzo-oss-scheduler-for-apache-mesos-frameworks-5c340e77e543>.

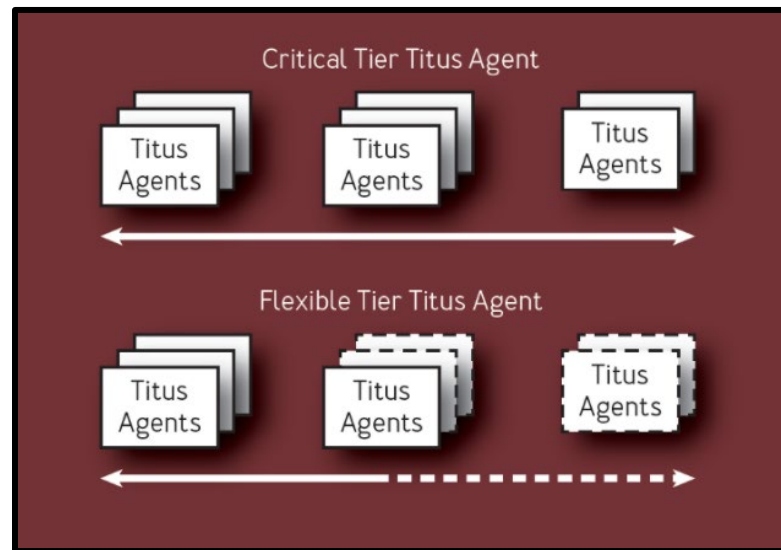
	<div><h3>Threshold based autoscaling</h3><ul style="list-style-type: none"><li>• Set up rules per agent attribute value</li><li>• Sample:</li></ul><table><thead><tr><th>Cluster Name</th><th>Min Idle</th><th>Max Idle</th><th>Cooldown Secs</th></tr></thead><tbody><tr><td>MemosyClstr</td><td>2</td><td>5</td><td>360</td></tr><tr><td>ComputeClstr</td><td>5</td><td>10</td><td>300</td></tr></tbody></table><div><div>#Idle hosts</div><div><div>max</div><div>min</div></div><div><div>Trigger down scale</div><div>Trigger up scale</div></div></div></div> <p>Source: <a href="http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf">http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf</a>.</p> <p>As another example, the Fenzo scheduler compares predefined resource capacity groups—such as the desired cluster size—with EC2 instance data received from the mesos agent in order to determine the optimal scheduling strategy.<sup>80</sup> Based on this evaluation, the Fenzo scheduler assigns tasks to resources based on a series of fitness calculators and constraints in order to minimize any difference between the actual resource capacity and the desired resource capacity.</p>	Cluster Name	Min Idle	Max Idle	Cooldown Secs	MemosyClstr	2	5	360	ComputeClstr	5	10	300
Cluster Name	Min Idle	Max Idle	Cooldown Secs										
MemosyClstr	2	5	360										
ComputeClstr	5	10	300										
<p>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.</p>	<p>The Accused System provides the system of claim 1, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System also includes 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.</p>												

<sup>80</sup> [http://platformlab.stanford.edu/Seminar%20Talks/Netflix\\_seminar.pdf](http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf).

For example, in addition to autoscaling methods based on a resource shortfall analysis, the Titus/Fenzo architecture utilizes “thresholds based” autoscaling, which collects status information and applies constraints according to the designated tier of each EC2 instance. As Netflix explains:

Thresholds based autoscaling lets users specify rules per host group (e.g., EC2 Auto Scaling Group, ASG) being used in the cluster. For example, there may be one ASG created for compute intensive workloads using one EC2 instance type, and another for network intensive workloads. Each rule helps maintain a configured number of idle hosts available for launching new jobs quickly.<sup>81</sup>

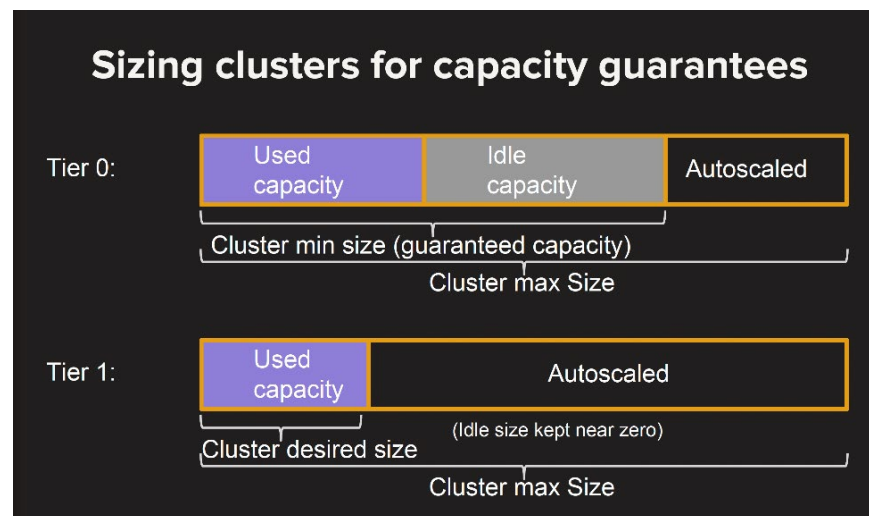
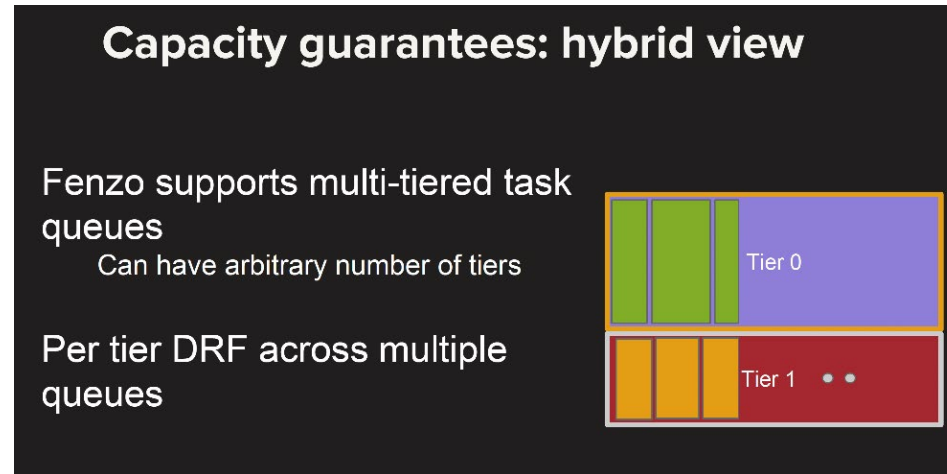
As previously discussed with regard to claim element 16.b. (which is incorporated by reference herein), the Titus system utilizes the concepts of “tiers” and “capacity groups” to organize EC2 instances into separate business logic tiers.



Source: <https://queue.acm.org/detail.cfm?id=3158370>.

<sup>81</sup> <https://netflixtechblog.com/fenzo-oss-scheduler-for-apache-mesos-frameworks-5c340e77e543>.

By dividing hosts into groups, Netflix explains that Fenzo is able to “apply a different autoscale rule to each group.”<sup>82</sup>



<sup>82</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.

	<div data-bbox="774 232 1740 761" data-label="Diagram"> <h3>Capacity guarantees: hybrid view</h3> <p>Tier Capacity = SUM (App1-cap + App2-cap + ... + AppN-cap) + BUFFER</p> <p>BUFFER:</p> <ul style="list-style-type: none"> <li>• Accommodate some new or ad hoc jobs with no guarantees</li> <li>• Red-black pushes of apps temporarily double app capacity</li> </ul> </div> <p>Source: <a href="http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf">http://platformlab.stanford.edu/Seminar%20Talks/Netflix_seminar.pdf</a>.</p> <p>Thus, using the shortfall analysis method described above with regard to claim 16 (which is incorporated by reference herein), Fenzo separately collects actual and expected status data for the EC2 instances in the critical and flexible tiers as well as for EC2 instances in each capacity group.</p>
<p><b>19.p.</b> A method comprising:</p>	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System practices a method.</p> <p><i>See</i> claim 1, as described above, which allegations are incorporated by reference herein.</p>
<p><b>19.a.</b> 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</p>	<p><i>See</i> claim element 1.a., as described above, which allegations are incorporated by reference herein.</p>

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;	
<b>19.b.</b> storing a plurality of image instances of a plurality of software applications that are executable on the plurality of virtual machines;	See claim element 1.a., as described above, which allegations are incorporated by reference herein.
<b>19.c.</b> 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	<p>The Accused System also practices the step of 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.</p> <p>For example, as previously discussed with regard to claim 1 (which is incorporated by reference herein), the Titus Master causes image instances of the Titus Agent to be deployed from the S3 “cloud” servers, git repositories, and/or Docker registries to the EC2 instances. To achieve this, the Titus Master utilizes Fenzo, as previously discussed with regard to claim 12 (which is incorporated by reference herein), to scale up resources thereby causing image instances of a Titus Agent to be copied to one or more EC2 instances. As Netflix explains:</p> <p style="padding-left: 40px;">You can tell Fenzo how, and under what conditions, it should scale the number of available hosts up or down. . . . Your <i>AutoScaler</i> callback method takes as its parameter an object that implements the <i>AutoScaleAction</i> interface. This object indicates whether your method is to scale the cluster up or down.<sup>83</sup></p>

<sup>83</sup> *Id.*; see also, <https://queue.acm.org/detail.cfm?id=3158370> (“Titus uses Fenzo’s bin packing in conjunction with its agent autoscaling capabilities to grow and shrink the agent pool dynamically as workload demands.”).

	<p><i>See also</i> claim 11 and claim element 16.c., as described above, which allegations are incorporated by reference herein (further describing the Fenzo autoscaling features).</p>
<p><b>19.d.</b> 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.</p>	<p>The Accused System also practices the step of 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.</p> <p>For example, as previously discussed with regard to claim 1 (which is incorporated by reference herein), the Titus Master causes containers to be autonomically deployed from the S3 “cloud” servers, git repositories, and/or Docker registries to Titus Agents running on one or more EC2 instances. To achieve this result, the Titus Master utilizes Fenzo, as previously discussed with regard to claim 9 (which is incorporated by reference herein), to schedule job tasks (user requests to run a particular software application or process) onto resources.<sup>84</sup></p> <p>As Netflix explains, the Fenzo scheduler implements various user-defined “constraints” in order to select a resource on which to run a task.<sup>85</sup> For instance, “[y]ou can tell the Fenzo scheduler the circumstances under which it should perform autoscaling and what criteria it should use to make autoscaling decisions.”<sup>86</sup> Fenzo uses these constraints to evaluate the job criteria and autonomically deploy tasks (i.e., a software application to run) to containers.<sup>87</sup> Thus, the Titus Master, through Fenzo, causes the autonomic deployment of containers by copying container image instances to the Titus Agents that are executing on the EC2 instances.</p>
<p><b>22.p.</b> The method of claim 19,</p>	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System practices a method.</p> <p><i>See</i> claim 19, as described above, which allegations are incorporated by reference herein.</p>

<sup>84</sup> <https://queue.acm.org/detail.cfm?id=3158370>.

<sup>85</sup> <https://github.com/Netflix/Fenzo/wiki/Introduction>.

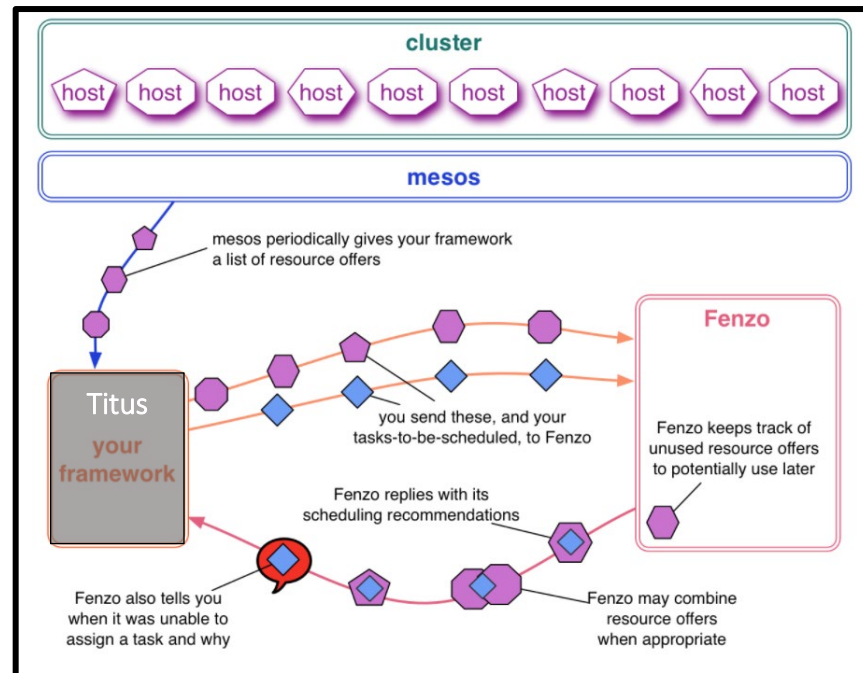
<sup>86</sup> *Id.*

<sup>87</sup> <https://github.com/Netflix/Fenzo/wiki> (“Fenzo schedules resources to tasks by using a variety of possible scheduling objectives such as bin packing, balancing across resource abstractions (such as AWS availability zones or data center racks), and resource affinity.”).

**22.a.** further comprising generating an application matrix that specifies data for controlling the deployment of the software applications within the distributed computing system; and

The Accused System practices the method of claim 19, further comprising the step of generating an application matrix that specifies data for controlling the deployment of the software applications within the distributed computing system.

For example, as previously discussed with regard to claim 11 (which is incorporated by reference herein), the Fenzo software generates a scheduling loop that specifies data for controlling the deployment of containers to Titus Agents executing on EC2 instances.



Source: <https://github.com/Netflix/Fenzo/wiki> (depicting a high-level overview of the scheduling loop utilized by the Titus/Fenzo architecture in order to deploy containers to Titus Agents; grey box labeled “Titus” added).



	<p>As Netflix explains, the Fenzo scheduler applies a sequence of steps to create a scheduling loop.<sup>88</sup> Netflix describes the “typical task scheduling loop” for Fenzo as comprising the following steps:</p> <ul style="list-style-type: none"> <li><b>[Step 1]</b> You get the latest resource offers from Mesos.</li> <li><b>[Step 2]</b> You create the list of tasks you want to assign.</li> <li><b>[Step 3]</b> You call the Fenzo task scheduler to assign tasks to hosts.</li> <li><b>[Step 3.1]</b> The task scheduler rejects any old, unused offers that have expired.</li> <li><b>[Step 3.2]</b> The task scheduler combines new and old offers to determine total resource availability.</li> <li><b>[Step 3.3]</b> The task scheduler assigns tasks to hosts.</li> <li><b>[Step 3.4]</b> The task scheduler rejects any offers that remain unused and that have expired.</li> <li><b>[Step 3.5]</b> The task scheduler (optionally) launches the autoscaling evaluator, asynchronously.</li> <li><b>[Step 3.6]</b> The task scheduler returns a report on the task assignments.</li> <li><b>[Step 4]</b> You check this result for tasks that Fenzo was unable to assign and may need to be reattempted.</li> <li><b>[Step 5]</b> You tell Mesos to launch any tasks that Fenzo has successfully assigned to hosts.</li> <li><b>[Step 6]</b> You call Fenzo’s <i>taskAssigner</i> for each task that you launch.</li> <li><b>[Step 7]</b> If you choose to not launch the tasks that Fenzo assigned resources to from a host, you will have to either reject the resource offers in the assignment result for that host, or re-offer those resource offers to the task scheduler again. Otherwise, those resource offers would be "leaked out".</li> <li><b>[Step 8]</b> You return to the beginning and start this process again.<sup>89</sup></li> </ul> <p>As seen above, this scheduling loop specifies various data—for example, “total resource availability” (Step 3.2), “autoscaling evaluator” (Step 3.5), and task assignment “report[s]” (Step</p>
--	--

<sup>88</sup> <https://github.com/Netflix/Fenzo/wiki/Scheduling-Tasks>.

<sup>89</sup> *Id.*

	<p>3.6)—that are used to control the autonomic deployment (Step 5) of containers to Titus Agents executing on EC2 instances.</p> <p><i>See also</i> claim 11, as described above, which allegations are incorporated by reference herein.</p>
<p><b>22.b.</b> 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.</p>	<p>The Accused System practices the method of claim 19, further comprising the step 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.</p> <p>For example, as previously described with regard to claims 1, 9, and 11 (which are incorporated by reference herein), the Titus Master, via the Fenzo scheduler comprises a scheduling loop that controls the execution of the containers onto Titus Agents. As previously described, this scheduling loop further includes autoscaling rules that control the deployment and undeployment of container images to containers.<sup>90</sup> As Netflix explains:</p> <p style="padding-left: 40px;">Your <i>AutoScaler</i> callback method takes as its parameter an object that implements the <i>AutoScaleAction</i> interface. This object indicates whether your method is to scale the cluster up or down.<sup>91</sup></p> <p><i>See also</i> claim 12, as described above, which allegations are incorporated by reference herein.</p>
<p><b>23.p.</b> The method of claim 19, wherein performing operations to autonomically deploy the image instances of the software applications</p>	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System practices a method.</p> <p><i>See</i> claim 19, as described above, which allegations are incorporated by reference herein.</p>

<sup>90</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>; *see also*, <https://github.com/Netflix/Fenzo/wiki/Scheduling-Tasks> (“[Step 3.5] The task scheduler (optionally) launches the autoscaling evaluator, asynchronously”).

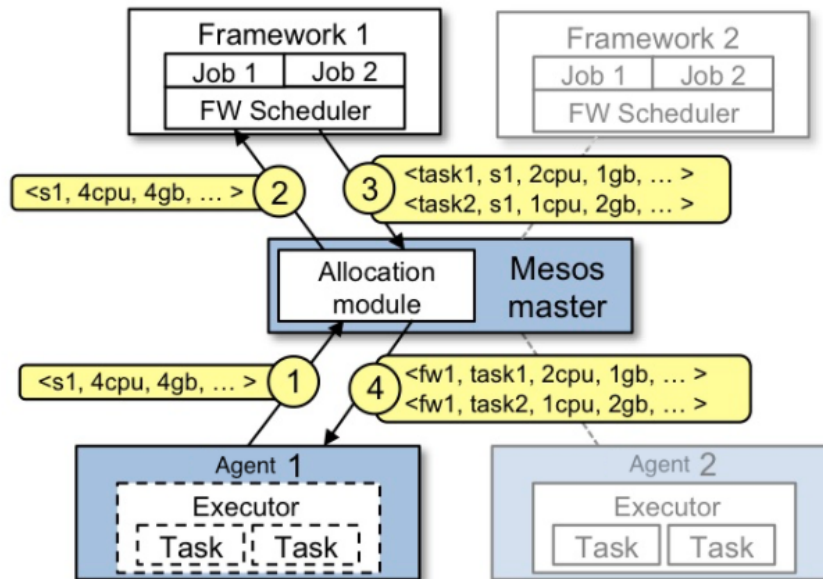
<sup>91</sup> *Id.*; *see also*, <https://queue.acm.org/detail.cfm?id=3158370> (“Titus uses Fenzo’s bin packing in conjunction with its agent autoscaling capabilities to grow and shrink the agent pool dynamically as workload demands.”).

<p>on the virtual machines comprises:</p>	
<p><b>23.a.</b> receiving status data for the distributed computing system, wherein the status data represents an actual state of the application nodes;</p>	<p>The Accused System practices the method of claim 19, further including the step of receiving status data for the distributed computing system, wherein the status data represents an actual state of the application nodes.</p> <p>For example, as previously described with regard to claim element 16.a. (which is incorporated by reference herein), the Titus framework provides a “mesos agent” that collects and emits performance metrics from the EC2 Titus Agent instances to the Titus Master.</p> <div data-bbox="823 596 1623 1016" data-label="Diagram"> <p>The diagram illustrates the architecture of the Titus framework. On the left, the 'Titus Master' is shown as a stack of components: 'Job Management &amp; Scheduler' (containing 'Fenzo') and 'Mesos Master'. Below this is the 'EC2 Autocaling API'. On the right, the 'Mesos Agent' is shown as a stack of components: 'container', 'metrics agent', 'docker executor', 'docker', 'logging agent', 'mesos agent' (highlighted with a yellow box), and 'zfs'. Arrows indicate the flow of data: from the 'mesos agent' to the 'Mesos Master' and from the 'mesos agent' to the 'metrics agent'.</p> </div> <p>Source: <a href="https://www.slideshare.net/aspyker/netflix-and-containers-titus">https://www.slideshare.net/aspyker/netflix-and-containers-titus</a> (yellow box added)</p> <p>The Apache Mesos documentation describes the interaction between the mesos agent and the framework (i.e., the Titus Master) using the following figure and explanation.<sup>92</sup></p>

<sup>92</sup> <http://mesos.apache.org/documentation/latest/architecture/>.

## Example of resource offer

The figure below shows an example of how a framework gets scheduled to run a task.



Let's walk through the events in the figure.

1. Agent 1 reports to the master that it has 4 CPUs and 4 GB of memory free. The master then invokes the allocation policy module, which tells it that framework 1 should be offered all available resources.
2. The master sends a resource offer describing what is available on agent 1 to framework 1.
3. The framework's scheduler replies to the master with information about two tasks to run on the agent, using <2 CPUs, 1 GB RAM> for the first task, and <1 CPU, 2 GB RAM> for the second task.
4. Finally, the master sends the tasks to the agent, which allocates appropriate resources to the framework's executor, which in turn launches the two tasks (depicted with dotted-line borders in the figure). Because 1 CPU and 1 GB of RAM are still unallocated, the allocation module may now offer them to framework 2.

Source: <http://mesos.apache.org/documentation/latest/architecture/>.

Consistent with this implementation, Netflix explains that the Titus Master receives the current status information from the mesos agent and then communicates that information to the Fenzo scheduler using, for example, the resource allocation functions that are built into the Fenzo scheduler.<sup>93</sup> In Netflix’s words:

You can call your task scheduler’s *getVmCurrentStates()* method to get a list of *VirtualMachineCurrentState* objects, one for each host that is known to the task scheduler. Each of these objects has a *getCurrAvailableResources()* method that returns an object that implements the *VirtualMachineLease* interface.<sup>94</sup>

method	description
<code>cpuCores()</code>	Get the number of cores (CPUs) available to be assigned on this host.
<code>diskMB()</code>	Get the amount of disk space, in megabytes, available to be assigned on this host.
<code>hostname()</code>	Get the name of this host.
<code>memoryMB()</code>	Get the amount of memory, in megabytes, available to be assigned on this host.
<code>networkMbps()</code>	Get the amount of network bandwidth, in megabits per second, available to be assigned on this host.
<code>portRanges()</code>	Get the list of port ranges that are available for assigning on this host.

Source: <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-which-resources-are-assigned-to-a-task> (depicting functions or “methods” that the Fenzo scheduler may use to retrieve metrics regarding the actual state information for the EC2 instances).

See also claim element 16.a., as described above, which allegations are incorporated by reference herein.

<sup>93</sup> <https://github.com/Netflix/Fenzo/wiki/Insights#how-to-learn-which-resources-are-assigned-to-a-task>.

<sup>94</sup> *Id.*

<p><b>23.b.</b> 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</p>	<p>The Accused System practices the method of claim 19, further including the step of 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.</p> <p>For example, as previously described with regard to claim elements 16.c. and 23.a. (which are incorporated by reference herein), the Fenzo scheduler analyzes EC2 instance status data that is generated by the metrics agent to determine the actual state of the EC2 instances.<sup>95</sup> The Fenzo scheduler then executes a shortfall analysis based on, for example, current utilization of the EC2 instances and pending, unassigned job requests, to determine the anticipated resource demands.<sup>96</sup> After completing these steps, the Fenzo scheduler applies autoscaling rules to scale the cluster up or down in order to minimize differences between the actual utilization of the EC2 instances and the output of the shortfall evaluation. As Netflix explains:</p> <p style="padding-left: 40px;">By default, Fenzo evaluates task assignment failures and assesses pending tasks to make an estimate of whether, and which, additional resources will be needed. This is useful for determining how to scale up resources.<sup>97</sup></p> <p>Said differently, “Fenzo regulates autoscaling . . . by also applying Fenzo’s resource shortfall evaluation to estimate resource needs based on the current task workload, and trigger a scale-up action when Fenzo believes resources are wanting.”<sup>98</sup></p> <p>See claims 19 and claim element 16.c., as described above, which allegations are incorporated by reference herein.</p>
<p><b>23.c.</b> performing the operations to provide autonomic control over the</p>	<p>The Accused System practices the method of claim 19, further including the step of 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.</p>

<sup>95</sup> See, e.g., claim element 23.a. *supra*.

<sup>96</sup> See claim element 16.b., *supra*.

<sup>97</sup> <https://github.com/Netflix/Fenzo/wiki/Building-Your-Scheduler>.

<sup>98</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>.

<p>deployment of the software applications on the virtual machines within the distributed computing system in accordance with the rules.</p>	<p>For example, as previously discussed with regard to claim element 19.d. (which allegations are incorporated by reference herein), the Titus Master—through the Fenzo software—performs operations that cause containers to be autonomically deployed from Netflix’s S3 “cloud” servers, git repositories, and/or Docker registries to Titus Agents in accordance with constraints applied by the Fenzo scheduler.</p> <p>As Netflix explains, the Fenzo scheduler implements various user-defined “constraints” in order to select a resource on which to run a task.<sup>99</sup> For instance, “[y]ou can tell the Fenzo scheduler the circumstances under which it should perform autoscaling and what criteria it should use to make autoscaling decisions.”<sup>100</sup> Fenzo uses these constraints to evaluate the job criteria and autonomically deploy tasks (i.e., a software application to run) to Titus Agents.<sup>101</sup> Thus, the Titus Master, through Fenzo, performs operations to cause the autonomic deployment of containers to Titus Agents executing on one or more EC2 instances.</p>
<p><b>24.</b> 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.</p>	<p>The Accused System performs the method of claim 19, as described above, which allegations are incorporated by reference herein.</p> <p>The Accused System practices 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.</p> <p>For example, as previously described with regard to claim 12 (which allegations are incorporated by reference herein), the Titus Master, via the Fenzo scheduler comprises a scheduling loop that controls the deployment of containers to Titus Agents. As previously described, this scheduling</p>

<sup>99</sup> <https://github.com/Netflix/Fenzo/wiki/Introduction>.

<sup>100</sup> *Id.*

<sup>101</sup> <https://github.com/Netflix/Fenzo/wiki> (“Fenzo schedules resources to tasks by using a variety of possible scheduling objectives such as bin packing, balancing across resource abstractions (such as AWS availability zones or data center racks), and resource affinity.”).

	<p>loop further includes autoscaling rules that Fenzo accesses to autonomically control the deployment and undeployment of containers to Titus Agents.<sup>102</sup> As Netflix explains:</p> <p style="padding-left: 40px;">Your <i>AutoScaler</i> callback method takes as its parameter an object that implements the <i>AutoScaleAction</i> interface. This object indicates whether your method is to scale the cluster up or down.<sup>103</sup></p>
<p><b>25.</b> 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.</p>	<p>The Accused System performs the method of claim 24, as described above, which allegations are incorporated by reference herein.</p> <p><i>See</i> claims 9, 10, and claim element 16.a., as described above, which allegations are incorporated by reference herein.</p>
<p><b>26.p.</b> A non-transitory, computer-readable medium comprising instructions stored thereon, that when executed by a programmable processor:</p>	<p>To the extent the Court determines this preamble is a limitation of the claim, the Accused System includes a non-transitory computer-readable medium comprising instructions stored thereon and executable by a programmable processor.</p> <p>For example, the Titus/Fenzo software executes on “cloud” servers Netflix purchases from AWS. <i>See, e.g.</i>, claim element 1.p., <i>supra</i> (which allegations are incorporated by reference herein). Like all, or nearly all, modern computer systems, these servers execute the Titus/Fenzo software, and any other software associated with the Accused System (instructions), by storing it in computer memory (a non-transitory computer-readable medium) and executing it using one or more programmable processors.</p>

<sup>102</sup> <https://github.com/Netflix/Fenzo/wiki/Autoscaling>; *see also*, <https://github.com/Netflix/Fenzo/wiki/Scheduling-Tasks> (“[Step 3.5] The task scheduler (optionally) launches the autoscaling evaluator, asynchronously”).

<sup>103</sup> *Id.*; *see also*, <https://queue.acm.org/detail.cfm?id=3158370> (“Titus uses Fenzo’s bin packing in conjunction with its agent autoscaling capabilities to grow and shrink the agent pool dynamically as workload demands.”).



	<p>For instance, when running the Titus/Fenzo system, Netflix specifies the number of virtual processors (vCPUs) that will run on each virtual machine instance.<sup>104</sup> Whenever a new instance is created—such as during a scale up operation—the Titus/Fenzo framework allocates additional vCPUs for that resource.<sup>105</sup></p> <p>Further, the virtual computing resources utilized by Netflix within the Accused System ultimately run on physical servers. These physical servers, like all or nearly all modern computer systems, utilize physical processors to execute the software running on those servers, including Netflix’s Titus and Fenzo software.</p> <p><i>See</i> claim 1, as described above, which allegations are incorporated by reference herein.</p>
<b>26.a.</b> 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;	<i>See</i> claim element 19.a., as described above, which allegations are incorporated by reference herein.
<b>26.b.</b> store a plurality of image instances of a plurality	<i>See</i> claim element 19.b., as described above, which allegations are incorporated by reference herein.

<sup>104</sup> *See* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html> (explaining that AWS users must specify desired CPU and vCPU options whenever launching a new instance).

<sup>105</sup> *Id.*

of software applications that are executable on the plurality of virtual machines;	
<b>26.c.</b> 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	<i>See claim element 19.c., as described above, which allegations are incorporated by reference herein.</i>
<b>26.d.</b> 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.	<i>See claim element 19.d., as described above, which allegations are incorporated by reference herein.</i>