EXHIBIT A

Part 2 of 3

FIG. **23** is a diagrammatic illustration showing an embodiment of a system according to the invention.

DETAILED DESCRIPTION

Among other aspects and innovations, the invention provides structure, system, method, method of operation, computer program product, and business model and method for providing distributed on-demand application processing.

There is a missing category in the available internet infrastructure based on-demand services. On-demand services fail to provide on-demand application processing, delivered as an on demand infrastructure service.

In one embodiment, the present invention provides for on-demand application processing, delivered as an on demand internet (or other networked) infrastructure service. Application processing may for example include one or more of, but is not limited to, deploying, instantiating, running and operating an application. One of the major benefits of providing this type of on-demand service is improvement in operational and other economics. The novel on-demand application processing method and system of the present invention improves: operational economics such as the elimination of costly server infrastructure expansion, simplifying and reducing capacity planning and an economic cost based on use; and user satisfaction by providing a maximum and assured application, such as an internet site, responsiveness under substantially any user load and for users located substantially anywhere. The present inventive on-demand application processing method and system changes the economic focus of server infrastructure.

The novel on-demand application processing method and apparatus of the present invention solves an application provider's capacity planning problem. For example, an application provider is an entity that provides a service via a computer network such as, Charles Schwab, WebVan-like entities, Walmart.com, and Intuit, which provide various types of applications accessed by individuals or entities over the internet. One of the problems that such companies face is that it is very difficult for them to predict how much demand they will have for their services and applications. Therefore it is extremely difficult for them to determine how large a server farm to deploy to allow greater user access to their services.

The present on-demand application processing method and apparatus solves this problem by providing on-demand processing capacity. Thus, the on-demand system provides the application provider with additional access to further processing capabilities without the need or expense of the application provider trying to predict how much processing capability will be needed. Further, one of the advantages of the present on-demand application processing method and system is that the application provider's cost is based on the amount of processing capability actually used. Thus, instead of having a huge up front capital investment to provide the expected processing capabilities and thus take all the risk to create these services, the present on-demand application processing method and system provides the application processing capacity based on demand, avoiding the need to predict processing needs, and eliminating the up-front capital investment.

Another major benefit provided by the novel on-demand application processing method and system is application user or customer satisfaction. An application user's satisfaction is achieved and maintained because the on-demand application processing substantially improves the response time of applications by increasing the processing capacity as the demand increases, is capable of spreading the load across a plurality servers, and enhancing consistency. The on-demand application processing system is further able to put a cap or limit on how much response time is built into the server side of application processing.

The present on-demand application processing method and system solves the growth spiral and the exponential cost per user increase in providing applications and services over the internet by supplying resource capacity based on the demand for the applications. The present on-demand method and system will increase resource capacity to an application provider as user access grows, and will also reduce resource capacity as user access decreases. Thus, the application provider simply pays for the amount of resources needed and used.

The present invention provides an ideal solution to the fixed capacity problem shown in FIG. **3**, through a flexible on-demand variable capacity providing substantially unlimited server infrastructure capacity which responds within seconds because demand patterns change within seconds. FIG. **4** shows a graphical representation of the on-demand response of the present on-demand system. The present on-demand application processing system solves the unpredictable capacity problem by providing on-demand server or application processor capabilities with a response time of seconds or less. If the capacity demand increases, the on-demand capacity of the present invention adjusts to supply further capacity **90***a* and **90***b*. If the capacity demand decreases, the on-demand capacity of the present invention adjusted to supply less capacity **92***a* and **92***b*, thus reducing the overall cost.

The present invention provides an ideal solution, by providing substantially instant variable capacity. As an example, the present invention provides an infrastructure or virtual infrastructure, which comes on-line or is activated for those peak times (i.e., those 10 minutes) when a site gets a rush of Web traffic, and then the virtual infrastructure reduces or go away when the Web traffic is reduced. Further the present invention provides substantially unlimited processing resources, thus providing as much processing as is needed. The present invention further provides unlimited processing resources with a global reach, because application providers now have users all over the world. The present invention further provides this substantially unlimited capacity to application providers at a pricing scheme which charges the application providers for the amount of capacity utilized, obviating the need for capital expenditures. The present on-demand application processing method and system is flexible and capable of running substantially any application, thus the application providers are not limited or locked into a particular application. The present invention provides the application providers with the ability to have the freedom to choose their own application sets. Further, the present invention allows the application sets to be completely under the application provider's control. As an example, once an application provider deploys an application set, the application provider maintains control over that application set, the data in and related to the application set, and other such control features. Thus preventing an application provider from being at the mercy of someone else owning their application set. Instead, the application provider maintains complete control over the services provided through the distributed application set.

FIG. **5** depicts a simplified block diagram of a business operating over the Internet, sometimes referred to as an e-business. Generally, an e-business has a set of servers **110** that run several or all of their different applications. The servers **110** have back end ERPs **112**, back end transaction systems or services **114**, and front end systems **116** including, but not limited to, personalization service **118**, an e-selling

system **120**, and a one-to-one marketing system **122**, which is found in a central site. Users **124** gain access through the internet **126** to the central server or central site **110**. As the number of users **124** accessing the server **110** increases, a tornado effect **130** results, and a bottleneck **132** is created, adversely affecting the response time and reliability of the site.

FIG. **6** depicts a simplified schematic block diagram of one embodiment of the novel distributed on-demand application processing system **140** of the present invention which substantially eliminates the bottleneck and tornado effects seen in the prior art. In one embodiment, the present on-demand system **140** pushes or distributes application processes, such as the front end systems, including, but not limited to, personalization **118**, eSales **120**, and one-to-one marketing **122** (see FIG. **5**), out into the Internet **126**, and out into the infrastructure of the Internet. In one embodiment, distributed compute resources, such as processors, computers and/or servers **148**, of the on-demand system **140** are geographically distributed, and in one embodiment located and installed globally all around the world. By geographically distributing the servers **148**, the application processing can also be distributed, allowing traffic from users **124** to be distributed and routed to the servers **148** distributed across the Internet **126**. In one embodiment, final applications or transactions, such as final purchases, and any other conventional transaction, are routed back across the internet **126** to the transactions system **1 14** and the ERP system **112**. In one embodiment, the transaction system **114** and the ERP system **112** are not moved out or distributed across the distributed servers **148**. As an example, a user **124** does his/her shopping and configuring, and the like as well as determining what he/she would like to buy, through a distributed server **148** which is geographically closer to the user in the on-demand system **140**. In one embodiment, once the user **124** selects or hits the "buy" button of the interactive website to complete the sales transaction, that transaction is forwarded to the backend systems **112** and **114** maintained on the application provider's central servers to complete the transaction. Thus, significantly reducing the amount of traffic into the application provider's central servers, eliminating the bottle neck effect, improving performance, enhancing response time, and thus improving and maintaining user satisfaction.

In one embodiment, the entire central site including the back end ERP **112** and transactions service **114** are distributed out onto the distributed on-demand system **140**. Thus, even the final transactions, such as the final purchase, are preformed on the distributed servers **148**.

FIG. **7** illustrates in high level block diagram form one implementation of one embodiment of the overall structure of the present invention as used in connection with a computer network **150** such as the Internet. In one embodiment, computer network **150** is a direct link between one or more remote entities, such as the users **152-1** and **152-2**, a separate application, a server, a process, computational device and substantially any other entity capable of issuing requests for application processing. In one embodiment, computer network **150** is a network providing an indirect link, such as an intranet or global network (i.e., the Internet). Remote users **152-1** and **152-2** utilize the computer network **150** to gain access to a plurality of computers or servers **158-1**, **158-2**, through **158-n**. In one embodiment, the computers **158** are protected by a firewall **154**. In one embodiment, computers **158** are edgepoints (described more fully below), groups of edgepoints, global dispatchers or other components of a private network **156**. In one embodiment, computers **158** are used to run various applications, such as hosting web sites for access by remote users **152**. In one embodiment, the present invention is implemented on computer network **156** in the form of virtual environments **160-1** and **160-2**. While only two virtual environments are illustrated, it is to be understood that any number of virtual environments may be utilized in connection with the present invention

In one embodiment, the method and system of the present invention is implemented in a computer readable medium, such as a computer program **164** and executed on a computer **166** as illustrated in the high level block diagram of FIG. **8**. As shown, computer **166** incorporates a processor **168** utilizing, in one embodiment, a central processing unit (CPU) and supporting integrated circuitry. A memory **170** which is any type or combination of memory including fast semiconductor memory (for example, RAM, NVRAM or ROM), slower magnetic memory (for example, hard disk storage), optical memory and substantially any conventional memory known in the art, to facilitate storage of the computer program **164** and the operating system software. In one embodiment, also included in computer **166** are interface devices including, but not limited to, keyboard **172**, pointing device **174**, and monitor **176**, which allow a user to interact with computer **166**. Mass storage devices such as disk drive **178** and CD ROM **180** may also be included in computer **166** to provide storage of information. Computer **166** may communicate with other computers and/or networks via modem **182** and telephone line **184** to allow for remote operation, or to utilize files stored at different locations. Other media may also be used in place of modem **182** and telephone line **184**, such as a direct connection, high speed data line or a wireless connection, and the like. In one embodiment, the components described above may be operatively connected by a communications bus **186**. In one embodiment, the components may be operatively connected by wireless communication.

FIG. **9** shows a simplified block diagram of one embodiment of an overall system architecture **200** for the distributed on-demand application processing service and system **140**. The on-demand system **140** includes an application switching architecture or technology **202** configured to provide application switching, an edge processing network **204**, which, in one embodiment, is hundreds of machines, edgepoints or servers in hundreds of data centers distributed throughout the world and/or the internet, automated deployment **206**, remote control **210**, security architecture **212**, and performance monitoring **214**, all coupled to cooperate and provide application processing, and deployment.

Some of the advantages provided by the on-demand method and system **140** include: protection during peak loads, in one embodiment, with guaranteed application response time SLA; global reach with application provider control of distributed web presence; freedom to grow aggressively including elastic web-processing infrastructure on demand; no capital investment with costs based on the amount of capacity used; supporting substantially any application on substantially any platform to preserve application provider's current application investment; and higher reliability because the system provides superior response time and automatically routes around failures.

FIG. **10** shows a simplified block diagram of one embodiment of the application switching architecture **202**. In one embodiment, the application switching architecture **202** includes an application snapshot or appshot **220**. An appshot **220** is a set of all data and/or state necessary to halt (and then restore and restart) at least one application at a given point in time, such that, the application may be restored at a later time on substantially any machine. For example, an appshot **220** can be an already running application halted at a point in time

without the application knowing it was halted. In one embodiment, an appshot **220** is the encapsulation of an application stack of at least one running application including the different processes, states, and interprocess communication. For example, a set of interdependent and/or interacting applications halted together may be included in an appshot **220**. In one embodiment, the appshot **220** includes, data **222**, and a set of interactive applications, **224***a*-**224***n*.

One example of an appshot **220** is a configuration engine, which allows users to shop online and decide exactly what they want to purchase. A snapshotted application and/or process, and the method for performing a snapshot is more fully described in co-pending U.S. patent application Ser. No. 09/680,847, filed on Oct. 5, 2000, incorporated in its entirety herein by reference.

In one embodiment, an appshot **220** encapsulates a multi-tier applications stack, including data **222**. The present on-demand application processing method and system **140** performs this appshot encapsulation or snapshotting which saves the states of a running set of processes. The encapsulation of an appshot **220** allows the on-demand system **140** to replicate an application and provide a plurality of instances of the same application to be operated at substantially the same time utilizing a plurality of subsets of the on-demand computational resources. The replication allows the on-demand system **140**, among other things, to move the appshot **220** to another set of compute resources such as another server, computer or machine, to duplicate the appshot **220** to other servers, and to replace or upgrade an appshot **220**. Further, the encapsulated appshot **220** allows the on-demand system **140** to put an application when operating as an instance of an application into a form which allows the system to remove the instance of the application from an idle server when the application instance associated with an appshot **220** is not being used, and to store the appshot **220** in a memory with accompanying application states. As an example, an appshot **220** is an already running application halted at a point in time. Thus the on-demand system is capable of freeing up resources to allow other applications to utilize the idle resources.

In one embodiment, the on-demand application system **140** is capable of relocating or replicating an appshot **220** to other or alternate sets of computational resources such as other compute modules and/or other edgepoints **350** (see FIG. **14**A) distributed throughout the worldwide on-demand system **140** providing at least a portion of the distributed on demand computational resources. In one embodiment, an edgepoint **350** is a computing facility with intelligent routing and load balancing architecture or capabilities. The edgepoint is capable of operating as a server. An edgepoint includes at least one and usually a plurality of servers or processors, such as a Sun Server **420** available from Sun Microsystems, Windows NT server from Microsoft, and a Linux server available from Linux, and substantially any other conventional server known in the art. The edgepoints are deployed, in one embodiment, throughout the world making up at least a portion of the on-demand system **140**. Application providers will generate an appshot **220** of the application, applications or site which they want to distribute throughout the on-demand system **140**. The appshot **220** can then be distributed to specific edgepoints, or distributed globally to every edgepoint **350** of the on-demand system **140**. Thus, when an entity **124**, such as a user, a separate application, a server, a process, computational device and substantially any other entity capable of issuing requests for application processing, wants to access the application or site, the edgepoint **350** activates or restores the appshot **220** to an activate instance of the application or applications encapsulated within the appshot **220**. The configuration and structure of the appshot **220** also allows the edgepoint **350** to re-encapsulate or snapshot the application or applications back into an appshot **220** and store the appshot **220** in a memory when the application or applications are not in use. As discussed above, the appshot **220** is capable of being restored or reactivated when needed. In one embodiment, the application can be restored from an appshot **220**, usually in less than **5** seconds, and more usually less than **3** seconds, depending on the available edgepoint resources. Thus, in one embodiment, the on-demand system **140** provides capacity on demand by restoring an appshot **220** when needed to provide one or more instances of an application. The system monitors the resources utilized to provide processing for the active application instances. The application provider is then charged according to the amount of computational resources utilized in providing users with access to their distributed applications.

FIG. **11** depicts a simplified flow diagram of one implementation of a sequence of steps executed by the present invention to perform a snapshot of a process or application instance. In step **250**, a snapshot of an active application is requested. The processes that are snapshotted together in the form of an application chain share the same application ID (AID). As such, the AID is looked up (decision step **252**) in memory containing a list of the AID's present. If the AID is not found control returns at step **254**. However, if the AID is found, control continues to decision step **256** where a search is performed for a process belonging to the application having the matched AID. If a process is found, control continues to step **258**, where the process is suspended. If the state is consistent and the threads are quiesced (decision step **260**), control loops to step **256** and the remaining processes belonging to the application are located and suspended. However, if a process is located that does not have a consistent state or a thread is not quiesced, suspended processes are resumed and the snapd module **262** returns a notice that the snapshot was not completed.

In one embodiment, a snapd module (snapshot daemon module) comprises a daemon listening on a port that does the snap-shotting of all processes that have the same snapshot id (snapid). The state of the applications after a snapshot is taken is stored in one or more files. The state that is typically saved includes process state information (for example, in a snapshot file per process), and memory information (for example, in a file per anonymous and shared memory segments). In one embodiment, the snapshot file stores all process state information as apseudo ELF file. A different ELF_NOTE section is created for each process state record (such as file descriptor). Another file called snaplist.snapid identifies all the processes in that snapid along with any parent/child relationship. In one embodiment, the process state information is collected during execution in preload libraries or when the snapshotting is done from the kernel.

Once the related processes are suspended, the states of the suspended processes are checked to see if they are virtualized (step **268**). A virtualized state is any process state that reflects a virtualized resource. If the state is virtualized, it is retrieved at step **270** otherwise the non-virtualized state is retrieved at step **272**. If the state has changed since the last snapshot (step **274**), the new state is recorded. Control then loops to step **266** and executes through the above sequence of steps until the states of the processes are checked. Once completed, control proceeds to step **282**, registered global states, such as semaphores, are removed. A registered global state is a state that is not specifically associated with any one process (i.e., private state). A global state is usually exported (accessible) to all

processes and its state can be modified (shared) by all processes. Control proceeds to step **284**, where the process is terminated. If there are remaining processes (step **286**), these are also terminated. This sequence of steps is concluded to create a snapshot of an application instance which is stored, in one embodiment, as a file and made available for reactivation or transmission to another compute modules, and/or other edgepoints.

FIG. **12** illustrates a simplified flow diagram of one implementation of the sequence of steps executed to restore a snapshotted application. The snapshotted application is accessed via a shared storage mechanism through a restore call at step **300**. The AID for the snapshotted application is looked up and (decision step **302**) if not found a notification is issued that the snapshotted application has not been restored. However, if the AID is found, control continues to decision step **304** where, if the snapshotted application matching the AID is located, the global/shared state for each process associated with the snapshot are found. Control then continues to step **308**, where remaining global or shared state for the processes are recreated. Then a process is created that inherits the global/shared state restored in step **308**, and the created processes are isolated to prevent inter-process state changes. At step **314**, for each type of state within the processes, the process-private resources are recreated to their state at the time the application was snapshotted. If the state is virtualized (decision step **316**), the system state is bound to a virtual definition. In one embodiment, as part of the restore a step is performed to create a virtual mapping. This is done by taking the system resource that was created in step **314** and binding it to the virtual definition that was saved during the snapshot in step **266**. This allows the application to see a consistent view of resources, since it may not be guaranteed that at restore time the same system resource will be available. If the state is shared with another process, such as via a pipe (decision state **320**), the shared state is reconnected with the other process at step **322**. If there are more states (decision step **324**) steps **314** through **322** are repeated. Once steps **314** through **322** have been executed for all states, control continues to step **326**, where the process is placed in synchronized wait. If there are remaining images in the snapshotted application (decision step **328**), steps **310** through **326** are repeated. Once all images have been processed, control continues to step **330**, where traces and states induced during restore of the process are removed, and a synchronized operation of the processes occurs at step **332**. Once steps **300** through **332** have executed without error, the restored application can continue to run without interruption. Thus, the present invention avoids the overhead and delay of shutting down an application, storing data to a separate file, moving both the application and data file elsewhere, and restarting the program or application.

FIGS. **13**A-C depict one implementation of one embodiment of the system and method or process for providing on-demand compute resources provided by the present invention. FIG. **13**A shows a simplified block diagram of one embodiment of an edgepoint **350** of the present invention. The edgepoint **350** includes a memory **352**, which is any type or combination of memory including fast semiconductor memory (e.g., RAM or ROM), slower magnetic memory (e.g., hard disk storage), optical memory substantially and any conventional memory known in the art. Within memory **352** is stored appshots **220***a-f*. Edgepoint **350** further includes compute resources **354**. Compute resources include but are limited to at least one of a microprocessor, memory, control logic, or combination thereof. In operation, the edgepoint **350** is accessed by at least one entity, such as users **124***a-b*, over a network, such as the internet **126**. When a user **124***a* is routed to the edgepoint **350** and requests one or more applications, the edgepoint **350** determines which appshot **220***a-f* provides the desired application. The edgepoint pulls or duplicates the appshot **220***b* from a halted or snapshotted state, and unencapsulates or restores the appshot **220***b* onto a first set of compute resources, such as a first server **354***a*. The server **354***a* activates an instance of the application **356***a* to allow the user **124***a* to utilize the application **356***a*. In one embodiment, the edgepoint **350** restores an application from an appshot **220** immediately upon request.

Referring to FIG. **13**B, once a server **354***a* is running the application instance **356***a*, the application **356***a* is fully active and operating, so that additional users **124***b* can be routed to and gain access to the active application **356***a*. Because the application **356***a* is already active, additional users **124***b* get an immediate response with substantially no delay. However, as more users request access to the application **356***a*, the response time begins to suffer, and the effectiveness of this application begins to deteriorate because the application **356***a* becomes overloaded. As additional users **124***c* attempt to access the instance of the application **356***a* response time degrades. In one embodiment, a predetermined response time threshold or limit is set, or a predefined number of users is set which limits the number of users allowed to access one instance of an application. Thus, when a new user **124***c* attempts to access the application **356***a*, and this new user **124***c* exceeds the predefined threshold, the edgepoint **350** activates the appshot **220***b* to initiate a second instance of the application **356***b*. Thus, this demonstrates the ability of the present invention to provide capacity on the run or on-demand, and provide an optimal response time for the applications **356***a-f*.

Referring to FIG. **13**C, as the numbers of users accessing the second instance of the application **356***b* continues to increase, the threshold will once again be reached. Once additional users **124***e* attempting to access the second instance of the application **356***b* exceeds the limit, the edgepoint **350** will again activate the appshot **220***b* to activate a third instance of the application **356***c*. This will continue until the servers **354***a-f* are occupied to capacity running at least one of the applications **356***a-f*. At which point, the edgepoint **350** will signal the system **140** and the on-demand application system **140** will then direct or route additional users to other edgepoints **350** throughout the distributed on-demand system **140**. Thus, the system **140** ensures an effective response time and reliability, and thus improves user satisfaction.

The present invention also provides for the freeing up of system resources to be utilized by alternative applications. As the number of users **124** decrease below a threshold, one of the application instances, such as the third instance **356***c*, can be terminated or snapshotted to free up a set of resources. The freed resources allows the edgepoint **350** to activate and run an alternative appshot **220***a-f*. Thus, the on-demand system **140** not only provides resources but reduces resources when not needed, resulting in a reduced cost to the application provider. Further, the present inventive on-demand method and system **140** provides for the ability to share resources among application providers because applications can be initiated as well as removed from compute resources allowing a substantially unlimited number of applications to utilize the same resources.

In one embodiment the edgepoint **350** is not limited to activating a single application **356** from a single appshot **220**. A single edgepoint **350** can activate a plurality of different applications **356** on a variety of different sets of compute resources, such as servers **354**, based on the applications requested by the users **124**.

In prior art systems, application providers wishing to provide applications had to buy a server, then they must buy or develop the applications that are going to be loaded and run on that server, load the server, and activate the server to provide access to that application. The server is a fully dedicated resource, so that 100% of the time an application is dedicated to a specific server. The present on-demand application system **140** reverses or switches this paradigm and instead of applications being dedicated to a server, the on-demand system **140** provides computing resources on-demand, when demand for an application is received, and additionally frees up resources when demand falls off for the restoring of completely different applications. Further, application providers no longer need to purchase the servers. Application providers simply take advantage of the on-demand application processing system **140** already deployed by loading their applications onto the distributed on-demand system **140**. The on-demand system **140** allows an application provider to allow substantially an unlimited number of users to access substantially the same application at substantially the same time without over loading the system **140** or the application, all without the need to incur the extremely expensive capital expense of providing their own system. Instead, the application provider pays for the amount of resources utilized to provide their users access to the applications. As demand increase, the on-demand system **140** increases the number of applications running, increasing the amount of compute resources and capacity, thus the application provider is charged more; as demand falls, the on-demand system **140** decreases the number of application instances, reducing the amount of computational capacity, thus the application provider is charged less. Thus, the application provider is charged for the amount of resources used.

In one embodiment, the present invention provides for a distributed on-demand system **140** such that potentially thousands of servers **354** in hundreds of edgepoints **350** are globally deployed and linked to create a virtual single server view throughout the world. This virtual single server view provides an application provider with access and control over their own applications in the system **140**.

FIG. **14**A-C show one implementation of one embodiment of the novel method and system **140** which allows the application providers to dictate the distribution of their applications, the potential compute resources to be available, upgrade or alter their applications, replace applications, monitor applications, and monitor the amount of resources utilized by entities accessing their distributed applications.

Prior art application processing systems require an application provider to route a user to a single central site to allow access to the applications. Every user attempting to access the application is directed to the single central site. Thus, resulting in the bottle neck as discussed above. In the prior art single server or single central site, the application provider, however, does maintain access to and control over the application. In some systems where the application provider outsources their server capacity, the application provider must select from a preselected, limited number of applications. Further, the application provider no longer has direct control over the application. Any changes desired by the application provider are submitted by request to the server provider. Then the server provider must schedule a time at low demands to take the server down to make the changes. This process results in large lag times between the decision to make changes and the implementation of those changes.

FIG. **14**A shows a simplified block diagram of one embodiment of the present on-demand application processing system **140** in cooperation with the preexisting internet infrastructure **126**. The present distributed on-demand application processing method and system **140** provides for distributed processing capabilities, with on-demand capacity, as well as providing the application provider with direct access to the on-demand system **140** and thus direct access and control over their applications. The application provider has control to distribute new applications or change already distributed applications throughout the on-demand system **140**. In one embodiment, the on-demand system **140** is configured to provide an application provider with a virtual single server view of the on-demand system **140**. This virtual single server view allows an application provider complete access and control over the applications which the application provider decides to distribute over the system **140**. In one embodiment, the system **140** provides an application provider with the ability to deploy applications throughout the distributed on-demand system **140**, change and update deployed applications, and monitor any one or all of the applications deployed throughout the internet **126** from a single terminal or site. In one embodiment, the application provider can deploy or access their applications through any one of a plurality of terminals or locations, including computers located at their own facilities. The present on-demand system **140** provides the application provider with the ability to deploy applications and access those applications once distributed through the system **140**. As referred to herein, conduit **360** is a staging facility that enables the application provider to deploy applications across a computer network. Through the conduit **360** application provider deploys applications, retains control over their deployed applications, monitors the operation of their applications on the system **140**, checks billing information, checks metering information, checks performance information, and so forth.

By structuring the on-demand system **140** as a single distributed system, and allowing the application provider with access to the on-demand system **140** through a single point, the on-demand system **140** appears to the application provider as a single server. Thus, when the application provider wishes to load and implement a new application onto the system **140**, the application provider simply accesses the on-demand system **140** through conduit **360**. Still referring to FIG. **14**A, in one embodiment, application provider is capable of loading an application as an appshot **220** onto the on-demand system **140** from a single origin site **362** through the conduit **360**. The application provider loads the appshot **220** onto the system **140**. The application provider is then capable of designating specific locations (for example, areas of the world wide system **140**, such as, edgepoints **350***a* and **350***b* representing Asia, Western Europe, the United States, Germany, a north-eastern portion of the United States, or London) or the entire system to allow users to access the deployed applications. Once the areas of distribution are designated, the on-demand system **140** distributes the appshot **220** through a hub **370** to the edgepoints **350** in the areas designated by the application provider. FIG. **14**B shows a simplified block diagram of one embodiment of the on-demand system **140** with the appshot **220** distributed to edgepoints **350***a* and **350***b*. Once the appshot**222** is distributed, auser **124** can then be routed **378** to the most optimal edgepoint **350***a* having the specified application, instead of being routed to the central site **362**. Because the on-demand system **140** is configured to appear as a single virtual server, the application provider loads the appshot **222** once on to the system **140**. The application provider does not need to load the application onto each edgepoint to distribute the application to specific areas of the on-demand system **140** or throughout the system **140**.

Further, the virtual single server mechanism also allows the application provider access to the appshot **220** through conduit **360** from a single point in the on-demand system **140**. Referring to FIG. **14**C, the application provider is capable of making changes or up-grading **374** the appshot **220** through conduit **360** and/or replacing the appshot. In one embodiment, the change or up-grade **374** is made once. This change or up-grade **374** is then distributed throughout the system to those edgepoints **350***a-b* providing the desired application without additional input from the application provider. Thus, the application providers maintain control over their applications and how the applications are distributed. The application providers are further capable of directly implementing changes and updates and replacements to their applications.

FIGS. **15**A-C show a simplified block diagram of one implementation of one embodiment of the optimal user and entity routing provided by the present invention. FIG. **15**A shows a block diagram of one embodiment of the on-demand application processing system **140**. One of the advantages of the on-demand system **140** and virtual single server mechanism is the ability to route a user **124** to an optimal edgepoint **350** which will provide the user **124** with the least amount of latency delays while avoiding overloading a single edgepoint **350**. For example, referring to FIG. **15**A, to reduce the amount of latency delay, it would be preferable to route the user **124** to the geographically closest edgepoint **350***a*, as designated by dashed line **1**. However, because of an overriding condition, for example edgepoint **350** is overloaded by other users, the user **124** is routed to a second edgepoint **350***b*, as designated by arrow **2**, adding a minimal amount of latency delay but providing enhanced response time because the second edgepoint **350***b* is not as heavily loaded, thus providing an overall superior interaction.

Referring to FIG. **15**B, another advantage of the on-demand system **140** is that the applications **356** can be interactive. Thus allowing the user **124** to make changes or additions **380** to the information provided through the application **356**. Further, once changes are made to the application **356**, the on-demand system **140** will continue to route the user **124** to that same edgepoint **350***b*, as designated by arrow **3**, for future connections of that user **124** to the on-demand system **140** for that application **356**. This affinity for that user **124** ensure that the user **124** continues to interact with the most current version of the application **356** according to the user's information and changes. The on-demand system **140** also provides the capability to synchronize or update the system **140** to reflect the changes **380** made by the user **124**. The updates are made at anytime as dictated by the on-demand system **140**, such as periodically, randomly or by schedule. As shown in FIG. **15**B, the changes **380** made by the user **124** are forwarded to the origin site **362** through conduit **360**, as shown with reference to arrows **4** and **5** updating the origin site. In one embodiment, the on-demand system **140** is further capable of distributing the changes **380** to the other edgepoints **350***a* which provide access to the application **356**, shown by arrow **6**. Thus, the on-demand system **140** synchronizes data and user's changes **380** throughout the system, maintaining an active and current system, without further interaction by the application provider. Examples of user changes include changes to a user profile, items added to a shopping cart, and other such changes. One example of data changes from the application provider would be an online catalog update from the application provider to the various edgepoints. Referring to FIG. **15**C, once the on-demand system **140** is updated and the user's changes **380** are distributed throughout the on-demand system **140**, the on-demand system **140** is again free to re-route or route the user **124** for future connections to any optimal edgepoint **350** in the system **140** providing the needed application **356**, as noted by arrow **7**. Thus, the present on-demand method and system **140** provides affinity in the routing scheme along with the ability to synchronize the on-demand system **140**.

Some of the additional features and benefits provided by the novel on-demand application processing method and system **140** include edge staging and load testing of applications and sites. The on-demand system **140** allows application providers to directly install new versions of a website or application onto the system **140**. The system **140** allows the application provider to limit the access to the new application or website. Thus, application providers are able to access the new website or application and functionally test the distributed site or application. Further, the on-demand system **140** allows the application provider to load test the application or site prior to allowing public access and use. For example, utilizing the on-demand system resources, numerous synthetic simultaneous sessions are able to be activated at a single time to load test the application or site. The application provider is able to perform these load tests without the capital expenditure of having to purchase additional equipment to perform load testing. Further, because of the pricing scheme of the present on-demand method, the application provider then pays for the amount of capacity utilized during this testing. Which is significantly less expensive than purchasing additional equipment.

FIG. **16** shows a simplified flow diagram of one implementation of one embodiment of the process or method **600** and system providing on-demand computing resources. In step **604**, an application request is received from an entity. Once a request is received, the process **600** enters step **606** where it is determined if the request from the entity is bound to a specific compute resource, such as a specific edgepoint. If the request is not bound, step **610** is entered where the optimal edgepoint is determine for providing the compute resources for application processing. In one embodiment, the optimal edgepoint is determined based on a plurality of criteria, including minimal latency delay, capacity of the edgepoint, the distribution of the desired application across the network, and other such parameters. Step **612** is entered if, in step **606**, the request is bound or once the optimal edgepoint is determined in step **610**. In step **612** it is determined if the bound or optimal edgepoint is operating at capacity. If the edgepoint is operating at capacity, step **614** is entered where it is determined whether the edgepoint can free up sufficient resources by snapshotting one or more application instances. If resources cannot be freed up, the process **600** returns to step **610** to reevaluate and determine the optimal edgepoint. Step **616** is entered, if in step **612**, it is determined that the edgepoint is not at capacity, or in step **614** it is determined that capacity can be freed up on the edgepoint. In step **616**, the user is routed to the edgepoint where the edgepoint determines optimal routing of the user to an instance of the desired application.

FIG. **17** shows a simplified block diagram of one implementation of one embodiment of the on-demand apparatus **140** including a plurality of edgepoints **350***a-b*. The network **140** can include substantially any number of edgepoints. FIG. **17** depicts two edgepoints for simplicity, however, it will be apparent to one skilled in the art that the network **140** can include substantially an unlimited number of edgepoints. Network **140** is configured to at least provide application processing for remote users **124** over the internet **126**. Network **140** can include any number of edgepoints allowing the computational capacity of network **140** to be scaled. Network **140** provides user **124** with differential computational capacity through either of the edgepoints **380***a-b*. In one embodi-

ment, network **140** includes a global dispatcher (GD) **430** which at least provides routing of application requests from user **124** to an edgepoint of the network. Based on network parameters including, but not limited to, edgepoint load and which applications are currently provisioned on each edgepoint **380***a-b*, GD **430** routes the user to the optimal edgepoint, for example first edgepoint **380***a*. Once the first edgepoint **380***a* receives the user request, the request is accepted and dispatched by a local dispatcher **434***a*. A resource manager **432***a* determines which of a plurality of compute resources or modules **436**a_1-**436**a_3 would be the optimal compute module in which to route the application request. In determining the optimal compute module, the resource manager **432***a* determines if a compute module is currently running the desired application or whether the application needs to be restored from a snapshotted state. The resource manager further determines if compute resources need to be freed up. If resources need to be freed, the resource manager signals a snapd module **440** of the optimal compute module, where the snapd module snapshots one or more application instances, thus freeing up the resources which were associated with the snapshotted applications. Once the optimal compute module has the available resources, the resource manager **432***a* signals the optimal compute module, for example first compute module **436**a_1, where the restored module **442** restores the desired application from memory **352***a* if necessary, and initializes the application to allow the user to interact with or operate the desired application.

In one embodiment, restored is a daemon that restores the state of all processes that belong to a single snapid. Snaplist identifies the list of processes that need to be restored. In one embodiment, the restore program "morphs" into the target process. The restore program creates processes in the order determined by, for example, a parent/child relationship. Each restore program is designated a process-id (pid) that it morphs to and each will do so by reading the appropriate snapshot file.

In one embodiment, the resource manager **432** further monitors the compute resources utilized by each application instance and records the compute resources utilized. The on-demand system **140** uses the recorded resource utilization for determining the amount an application provider is charged for the use of the compute resources. In one embodiment, the system includes a monitoring module **464**, which monitors the compute resources and provides the monitored results to the resource manager and other components of the system. In one embodiment, the resource manager (RM) utilizes the perfd module to collect at least some of the data to determine the amount of resources utilized per application instance. The resource manager **432** monitors such things as CPU utilization, network bandwidth, disk utilization, license usage, response time latencies, and other such parameters for determining resource utilization.

In one embodiment, a perfd module comprises an agent or other entity running on the computer node (CN) that computes or otherwise determines the performance of each of the CNs. A call to this perfd agent is initiated by the resource manager (RM). This perfd agent makes system calls to collect the statistics and sends it to resource manager. Viewed somewhat differently, perfd is or includes a daemon or other mechanism that collects performance information, hence the abbreviation perfd for performance daemon. In a more general way, a perfd module is any performance determining module.

In one embodiment, a compute node is a component within an embodiment of an edgepoint that runs the components of an Application Instance. Typically, these are application tiers such as a webserver connected to a middle-tier and a database.

In one embodiment, an edgepoint is a machine or collection of machines that run the customers site. An administrative node (AN) is the component within an edgepoint that runs the administrative components of an edgepoint. For example, a configuration database, deployer components, data synchronization components, and monitoring components are run in the administrative or admin node.

In one embodiment, the network **140** further includes at least one deployment center **444**, deployment database (DDB) **446**, conduit **360**, and dashboard (i.e., a network operating center (NOC) dashboard **454** and/or a customer dashboard **456**). In one embodiment, the edgepoints further include a global dispatch module **460**, a data synchronization module **462** and the metering module **464**. The plurality of edgepoints communicate such that snapshotted instances of applications can be transferred and users rerouted.

The global dispatch module **460** provides communication with and access to the network **140**, and aids in network routing to allow entities, such as users **124**, access to the applications and compute resources. The global dispatch module **460** further receives information from the network and other edgepoints regarding the amount of compute resources available on other edgepoints **350** and throughout the network **140** to aid in optimizing the resources of the network. The data synchronization module **462** communicates with the network to receive data and information from the network to update the edgepoint **350** with the data. The data synchronization module **462** allows new or changed data distributed across the network to be forwarded to the compute modules **436** and/or the memory **352** of the edgepoint **350**. In one embodiment, the data synchronization module **462** allows data added or changed by a user **124** to be distributed across the network **140**. The metering module **464** monitors the edgepoint and the compute resources utilized by an application, user, group of applications, and any combination thereof. The metering module **464** acts in cooperation with the resource manager **432** to monitor the compute modules **436** and collects data from the compute modules regarding usage. In one embodiment, the metering module is further configured to determine an amount to charge the application providers based on the amount of compute resources utilized by each application provider for application processing.

In one embodiment, the deployment center **444** acts as the hub that collects data, policies and applications and distributes them to the edgepoints **350***a-b*. Deployment center **444** maintains application and data versions and distributes updates, revisions and replacements which are forwarded to the deploy modules **433***a-b* of the edgepoints **350***a-b*. In one embodiment, the deployment through the deployment center **444** includes capturing application states (initial and updates), policies and testing methods as released to the network **140** from application providers and network administrators and moving it to the edgepoints **350***a-b*. The policies include deployment and execution policies. Application states include the actual application data/binaries and the method to create the snapshots.

In one embodiment, the DDB **446** is the repository for the NOC **452** and serves also as the repository for deployment by the deployment center **444**. Conduit **360** provides an application provider with access to the network **140** to distribute, update and monitor their applications distributed throughout the edgepoints **350***a-b*. In one embodiment, the conduit **360** abstracts or virtualizes the distributed nature of the network **140** and allows the application provider to update, manage and view their data and applications without being burdened by the location and load of the actual edgepoints **350***a-b*.

In one embodiment, the dashboards provide at least two forms of data viewing, including immediate and aggregated. Immediate viewing allows a current, up-to-date view of an entity of the network **140**, such as edgepoints **350***a-b*, global dispatcher **430**, deployment center **444** and conduit **360**. In one embodiment, immediate viewing is updated on a predefined schedule, periodically when a predefined change occurs or upon request. Aggregated viewing provides a cumulative temporal view of the entities of the network **140**, such as application instance usage, user patterns, edgepoint usage, etc. In one embodiment, immediate views are obtained by polling the edgepoints **350***a-b* and conduits **360**. The aggregate view is accumulated at the deployment center **444**. In one embodiment, dashboards receive resource utilization information and determine an amount to charge each application provider based on the amount of resources utilized.

The NOC dashboard **454** allows network operators and controllers of network **140** to gain access to and view information and data relating to components on the network **140**. In one embodiment, NOC dashboard **454** allows access to components on the network at machine levels and application instance levels.

Customer dashboards **456** allow application providers to view the state of their outsourced applications, such as response time, data arrival rates, comp-utilities used, amount of compute resources utilized, cost per application, and other such information. In one embodiment, the network **140** prevents customer dashboards to gain access to the actual edgepoints **350***a-b* and the applications stored and operated by the edgepoints **350***a-b*.

In one embodiment, network **140** includes additional dashboards which allow other operators and users of the network access to information on and about the network. One example of an additional dashboard is an independent software vendor (ISV) dashboard which allows ISV's to view the usage patterns of applications on a per application provider or per application basis. This is an audit for the ISV's to understand how their applications are behaving in a real environment.

FIG. **18** depicts a simplified flow diagram of one implementation of one embodiment of a process **702** for an application provider to access and distributing applications onto the distributed system **140**. In step **704**, the application provider gains access to the system **140**. In one embodiment, the application provider gains access through conduit **360**. In step **706** the application provider dictates to the system the distribution of the application being deployed. As discussed above, the application provider is capable of limiting the distribution to specific geographic locations, to specific markets, to populated areas (such as only to resources located near metropolitan areas) and other such deployment distribution. The application provider can also allow an application to be distributed across the entire system **140**. In step **710**, the application provider specifies limits on the amount of compute resources to be utilized. The limit may be based on a monitory limit or other limits. In step **712**, the application provider dictates a maximum response time, such that when demand is such that the response time is exceeded, the system **140** will activate additional instances of a desired application. In step **714**, the application is distributed. In step **716**, entities, such as users, servers and computers are allowed to access the distributed applications.

FIG. **19** depicts a simplified flow diagram of one embodiment of a process **730** for an application provider to monitor and update applications distributed onto the system **140**. In step **732**, the application provider monitors the applications distributed. In one embodiment, the application provider is capable of monitoring the system through the dashboards **456** and conduit **360**. In step **734**, it is determined whether the application provider wishes to update or change the distributed application or applications. If updates are to be made, step **736** is entered where the application provider submits the updates and the updates are distributed. If, in step **734**, updates are not to be made, or following updates, it is determined in step **740** whether the application provider wishes to replace a distributed application. If yes, then step **742** is entered where the application provider submits the replacement application and the replacement application is distributed. If, in step **740**, no replacement is needed, step **744** is entered where it is determined whether the application provider wishes to adjust the distribution of one or more application. If the adjustments are received and the adjustments to distribution are made in step **746**. If adjustments are not needed, step **750** is entered where it is determined if the application provider wishes to adjust resource limits, such as response time limits, monetary limits, capacity limits and other such limits. If yes the adjustments are received and implemented in step **752**.

FIG. **20** depicts a flow diagram of one embodiment of a process **770** for monitoring demand and determining an amount to bill an application provider. In step **772**, the on-demand system **140** monitors the demand for each application distributed on the system. In step **774**, it is determined whether the response time for an application exceeds limits. In one embodiment, the limits are default limits. In one embodiment the limits are previously specified by the application provider. If the response time limits are exceeded, the system determines if the system is at capacity, where capacity is dictated by several factors including, but not limited to compute resource availability, resource limits specified by the application provider, and other such factors. If the system is not at capacity, step **780** is entered where a new instance is restored where an instance of the application is restored and activated to allow entities to access the application. In step **782**, it is determined if demand exceeds a first threshold, where the threshold is defined as a number of users per instance or other such parameters. The threshold is a default threshold or defined by the application provider. If the demand exceed the first threshold, then step **784** is entered where it is determined if the system is at capacity (as described above). If the system is not at capacity then step **786** is entered where an instance of a desired application is activated to allow entities to continue to access the application without exceeding the threshold. In step **790** it is determined whether the demand is below a second threshold. If the demand is below the second threshold, at least one instance of an application will be snapshotted in step **792** to free up resources and reduce the compute resource cost to the application provider.

FIG. **21** depicts a simplified flow diagram of one implementation of one embodiment of a process **808** for determining an amount of resources utilized for an application and the amount to be charged to the application provider based on the amount of resources utilized. In step **810**, the compute resources providing application processing is monitored. In step **812**, the amount of resources utilized by each distributed application is determined. In step **814**, a total amount of resources utilized by each individual application provider all applications distributed by each application provider is determined. In step **816**, an amount to charge each application provider is determined based on the amount of compute resources utilized in application processing of all application distributed by each application provider. In step **818**, the

metered data is presented via a portal to application providers and partners (such as resellers and independent software vendors).

The capability to meter resource usage creates the further ability to develop pricing schemes which reflect the economic value or cost of providing service, including, for example, the opportunity cost of using compute resources for a given application. As an example, demand for computing resources during the business day may be higher than demand for processing during the night. The system provides a mechanism for pricing for peak usage versus off-peak usage.

The system facilitates a transfer pricing mechanism for computing resources. By metering resources used on an application level, the system enables a compute resource provider to determine how much resource usage is related to each specific application or customer. In one embodiment, this enables a corporation to allocate the cost of a centralized computing resource between different departments according to usage.

In one embodiment, more than one party may own the compute resources within the distributed network. In this case, the present invention enables the trading of compute resources between any number of compute resource suppliers and users. For example, a party may be an owner of compute resources that in the normal course of events provide sufficient capacity for its processing needs. This party can, by deploying the present invention and interconnecting its computing network with those of others, sell underutilized computing resources, and buy compute resources from other parties at times of peak demand. This enables parties to significantly improve the utilization of their computing infrastructure.

In the more general case, the present invention enables the development of an efficient market for trading computing resources, facilitating economic pricing. By way of example, a spot market could develop, better reflecting supply and demand of computing resources. Instruments for managing the financial risk of fluctuations in price on the spot market could then develop, for example forward contracts, options and derivatives.

In one embodiment, the novel system **140** is configured to provide at least six data paths including: a data path that connects an entity **124** to an application instance at an edgepoint **380**; a data path that sets up application instances for application providers; a data path which implements the snapshot/restore framework; a data path which provides a centralized view of edgepoints to application providers, the network provider and other such entities for monitoring the edgepoints and the compute resources utilized; a data path which provides database and file updates; and a path which prepares an application or plurality of applications for deployment and data synchronization.

As discussed above, the edgepoint **350** is capable of performing a plurality of actions or functions based on parameter and compute resource utilization information collected, such as performing snapshots of active application instances; restoring applications based on demand, response time and other parameters; effecting a move of an application instance; identifying optimal compute resources, such as an optimal compute module for routing a request; monitoring the performance and available capacity of the edgepoint to optimize performance and to signal the global dispatcher **430** when the edgepoint is operating at or near capacity; and monitoring the compute resources utilized per application instance such that the application provider is charged for the resources utilized in operating the application provider's distributed applications.

In one embodiment, the edgepoint **350** effects moves of application instances based on the results of an overload of a compute module, under-utilization of another compute module, and/or prioritization of one application instance over another (based for example, on distribution and prioritization specified by the system provider and the application providers).

The edgepoint **350** determines if the edgepoint is overloaded and hence notifies the global dispatcher **430** such that bind requests are re-routed back to the global dispatcher or initially routed by the global dispatcher to alternative edgepoints. In one embodiment, the resource manager **432** sends periodic load or resource utilization messages to the global dispatcher, such that the global dispatcher can accommodate the server weighting in the databases and memory.

The edgepoint further monitors, meters, and/or collects resource consumption information from application instances. This information is used by the dashboards **454**, **456** and for billing. In one embodiment, the information that is collected is logged into the databases or memory. The information collected by the edgepoint includes, but is not limited to, CPU usage, memory usage, disk usage, network bandwidth usage on a per application instance basis. In the case of CPU usage, information is collected at the software component level, providing a greater level of granulating than prior art systems. This information may be used to identify and allocate resources, manage partnerships and for usage based billing purposes.

The edgepoint additionally collects performance information such as application response time. In one embodiment, for each application instance, the resource manager **432** performs a periodic response time check. This information is also used to initiate the snapshot or move actions.

In one embodiment, the conduit **360** allows an application provider to create and distribute one or more application onto the network **140** producing the outsourced applications. The conduit performs a plurality of function including, but not limited to: determining cleave points; capturing distributed applications and associated data; capturing distribution policy, response time policy and other such policies as designated by the application provider; and test captures.

Cleaving includes a process of dividing one or more applications into applications that are distributed across the on-demand network **140**, and applications that are not to be distributed, and instead, for example, maintained by the application provider. One example is a website, where the site is cleaved to allow some of the application processing of the site to be handled by the distributed on-demand network, and some of the application processing of the site to be handled by the central site of the application provider. Thus, cleaving separates the applications that are outsourced by the application provider to be distributed over the present invention to take advantage of the on-demand compute resources.

The novel on-demand network **140** acquires and captures the applications and data associated with those applications through a capture process. The capture process analyzes how to bring up the necessary applications associated with a request, such as all the applications in a website operated from the on-demand system. The capture process further collects files and database data files for the application instances to satisfy a request. The capture process maps applications to application instances and documents the process of capturing the process of creating an application instance, and maps data to an application instance for data synchronization and captures.

In one embodiment, application and data capture is a process for determining how an outsourced application is constructed or produced. Some of the steps for application and data capture include:

a) analyzing how to bring up applications in the network, such as bringing up applications configured to produce a web site;

b) collecting the files and database data files for the operation of application instances;

c) mapping applications to application instances and documenting the process of creating an application instance. In one embodiment, this is predominantly the data used by a packager (not shown) for initial deployment, and the instructions to start and stop application instances; and

d) mapping data to an application instance, data synchronization and capturing the data synchronization components.

In one embodiment, the application provider dictates the distribution of the applications onto the distributed, on-demand network **140**. The application provider is capable of designating specific geographic areas for distribution, high traffic areas, such as specific metropolitan areas, and other such distribution. The application provider is also capable of designating the quantity of distribution, which will allow the application provider to limit the cost by limiting the compute resources utilized based on the distribution designation.

Policy capturing includes collecting deployment and execution policies. Deployment policies determine coverage information and are used by the deployment center **444** to aid in selecting the edgepoints **350** that will hold the outsourced applications. Execution policies relate to user-level SLAs and priorities for execution. Policy capture allows the application provider to limit and determine the potential cost spent by the application provider in utilizing the on-demand network.

In one embodiment, the on-demand network **140** is further capable of providing capacity testing of applications to aid the application provider to determine the accurate and operational capacity of application. One example is the testing of the capacity of a web site before allowing users to access the web site. Test capture includes a method, data and frequency to run tests on edgepoints before enabling the applications.

In one embodiment, the conduit **360** includes a studio module (not shown) which is a module used to perform application/data, policy and test captures in the conduit. The studio module includes at least six functional modules, including: a catalog module (not shown) which creates an inventory of deployed application; a camera module (not shown) which is the portion of the studio used to capture the process of bringing up and/or restoring an application, and establishing an initial snapshot; a packager module (not shown) configured to assemble an installed application and snapshots into a format (package) suitable for deployment; a publisher module (not shown) capable of transferring the packaged application to a deployment center; a cleaver module (not shown) which identifies the portions that are handled by the out-sourced applications, and will initiate data synchronization; and a policy editor module (not shown) configured to specify deployment policies. Application strength, deployment coverage and level of service are specified through the policy editor module. In one embodiment, the coverage is coarsely defined as a number of application instances, a number of edgepoints and/or which geographic location or locations. A level of service is also coarsely defined as a response time of an application.

The on-demand method and system **140** further provides remote control capabilities. The remote control feature provides: policy-based server management with the alignment to business objectives; deployment policies with application provider's selection of coverage, along with deployment and initiation timing; resource use policy to aid in obtaining the desired response time within application provider's budget constraints; and web-based policy editor.

Another advantage of the novel on-demand method and system **140** is providing application providers with direct access to the system **140** and allowing the application provider to make immediate changes or version updates to existing applications or site. Further, application providers are able to immediately load completely new applications or sites onto the system **140** without the need to bring the application or site down.

Another advantage of the present method and system **140** is the ability to provide fast rollback. Because of the design of the system **140** and ability to maintain applications in an inactive or snapshotted state as an appshot **220**, prior versions of applications can be maintained on the edgepoints **350** while new versions are loaded onto the edgepoints **350**. If there is a glitch or error in the new version, the system **140** is able to quickly redirect the system **140** to reinstate the old version. Thus, avoiding catastrophic errors and glitches.

Another advantage provided by the novel on-demand method and system **140** is the ability to distribute users **124** to applications **356** throughout the system **140** thus spreading the load of the users. This provides the application provider with additional benefits which were not available through the prior art without enormous capital expenditures. One benefit of the on-demand system **140** is the ability to handle extremely large numbers of entities **124** at a single time because entities can be directed to application instances distributed throughout the system **140**. As an example, application and web providers have the ability to announce and broadcast an event which may attract abnormally large numbers of users without overloading the system. Because the users can be routed to edgepoints all over the system **140**, the maximum load on any given edgepoint will not exceed the capacity of the resources of the edgepoints. Thus allowing abnormally large numbers of entities to utilize or view the application or website. This is all achieved without the need for the application provider to purchase large numbers of servers and accompanying hardware, as well as the need to configure, load and maintain these large numbers of machines for such a limited surge in user load.

An additional benefit is that application providers are able to interact with abnormally large numbers of users instead of just broadcast to those users. Because the applications are distributed, the capacity to operate those applications is also distributed. Thus, allowing each instance of an application to utilize a larger amount of resources without exhausting the resources of the system. Thus, more interactive applications and sites are capable without the need for additional capital expenditures by the application provider. These are significant advantages provided by embodiments of the invention that are not available in conventional content delivery systems, networks, or methods.

As an example, big media companies have the capability through the present on-demand method and system **140** to now start getting a one to one opportunity with users accessing their applications and sites As a comparison, television allows the distribution of a single program without any interaction. However, with the present method and system **140**, a plurality of different programs can be distributed while allowing direct interaction with the user without overloading a system and without cost prohibitive capital expenditures. As a further example, if a prior art application or site were to get