Service-Centric Globally Distributed Computing

An automated service demand–supply control system can improve a large-scale grid infrastructure comprising a federation of distributed utility data centers.

Sven Graupner
Hewlett-Packard Laboratories

Vadim Kotov
Carnegie Mellon University

Artur Andrzejak
Zuse Institute Berlin

Holger Trinks
Santa Clara University

The growing complexity of large-scale systems raises concerns about their effectiveness and manageability. New approaches to system design, use, and management address these concerns through the aggregation and consolidation of system and application components into larger building blocks; systematic and standard ways of integrating such systems and communicating between them; sharing of distributed resources; and automated system management and operation control. Due to their scale and dynamism, large-scale systems cannot provide global information about resource availability and service demand. Decision-making algorithms thus need to deal with partial information, yet provide good approximations of localized assignment solutions. Moreover, they need to make decisions within the duration of the overall control cycle of an automated resource demand–supply control system. Various control cycles occur simultaneously, ranging from short-term reactive response (range of seconds to minutes) to mid-term system rebalancing (hours to days) to longer-term adjustments and system evolvement (months). Usually, decision algorithms implement trade-offs between the time spent identifying a solution and the quality of that solution.

Two of the most prominent and practical approaches to solving these problems are utility computing and grid computing. Adopting the utility computing approach, Hewlett-Packard developed the Utility Data Center (UDC; see www.hp.com/go/hpudc/) to consolidate computing resources. Virtual data centers[1] consolidate the resources of a federation of distributed UDCs into virtual resources that users, user applications, and user services can share using grid-type mechanisms, thus significantly reducing deployment and operation costs.

We propose an architecture for such an

 Published by the IEEE Computer Society 1089-7801/03/$17.00

automated demand–supply control system based on a formalization of service demands and supplies in an overlay metasystem.[2] The architecture features a loosely coupled, federative structure of nodes, each representing server capacities and service utilizations. Nodes can freely join or disappear. Distributed decision-making algorithms in the metasystem face trade-offs between solution quality and reactiveness, but this architecture is general enough for a variety of large-scale distributed systems such as a federation of distributed UDCs.

The Utility Data Center

Besides automating fail-over techniques in high-availability systems, management systems typically automate monitoring and information collection. Major service-capacity adjustments imply manual involvement in hardware as well as in software to adjust, reinstall, and reconfigure systems.

Deployment and operational costs for executing these manual processes dominate enterprise IT customers' balance sheets. HP's UDC is a new type of data center infrastructure that provides automated support for these tasks, thus providing an automated service capacity demand–control system for a large service grid. This control system is based on a federation of geographically distributed data centers whose UDC capabilities provide immediate support for service demand–supply control.

HP's UDC supports the automated deployment of services (installation and configuration of software and data) as well as the virtual wiring of machines into application environments (here referred to as virtual server environments) independently of the physical wiring in a data center. UDCs allow programmatic rearrangements of service applications among machines, dynamic sizing of service capacities, and isolation of different environments hosted in a given data center. In addition, UDCs enable automated control systems.

Virtualizing Resources

Resource virtualization is a vital aspect of a UDC. The storage virtualization fabric attaches storage elements (disks) to processing elements (machines) across the storage area network. The virtualization fabric is a high-speed interconnect layer between storage arrays and machines (storage interconnect fabric) and between machines (network interconnect fabric). The network fabric links processing elements together in a private virtual LAN.

The UDC virtualizes storage resources by providing virtual disks with images from storage arrays. Persistent states of application environments — file systems, bootable operating system images, application software, and so on — do not reside in server machines anymore; rather, they are programmatically attached from separate storage arrays to machines. By making disks programmatically attachable to machines, operators can vary operating systems and applications on machines without having to manually reconfigure them. This capability exploits the storage fabric's programmability, making storage images appear on the SCSI interfaces of server machines from which other machines obtain boot images and further data.

The UDC virtualizes network resources by permitting the programmable rewiring of server machines and devices to create a virtual LAN. It achieves virtual wiring by programming the network switches that connect machines and by programmatically connecting machines to or removing them from virtual networks.

Automating Functions

A UDC platform enables two new functions: automated service deployment provisioning and dynamic capacity sizing of services. It automates service deployment provisioning by maintaining persistent services' states entirely in the separate storage system and conducting programmatic control over attaching storage to machines.

The UDC size services capacity dynamically through its ability to automatically launch additional service instances, thus absorbing additional load. The UDC launches service instances by first allocating spare machines from a pool the data center maintains, then virtually wiring them into the specific service environment by attaching the appropriate storage to those machines and launching the operating systems and applications the attached storage provides.

The Global Grid

Although scientific supercomputing's need for computing power seems infinite, resource sharing means more than just distributing computational tasks transparently among heterogeneous nodes. From a commercial point of view, even more important is transparent and secure data sharing between the software and services needed to access, maintain, and process data across organizations. In this context, a grid is a collaborative domain that spans multiple networks and organizations, facilitating secure and efficient sharing of the data and services needed for collaboration.[1,3] Such grid domains comprise resources, documents, and all other data, including accompanying appli-

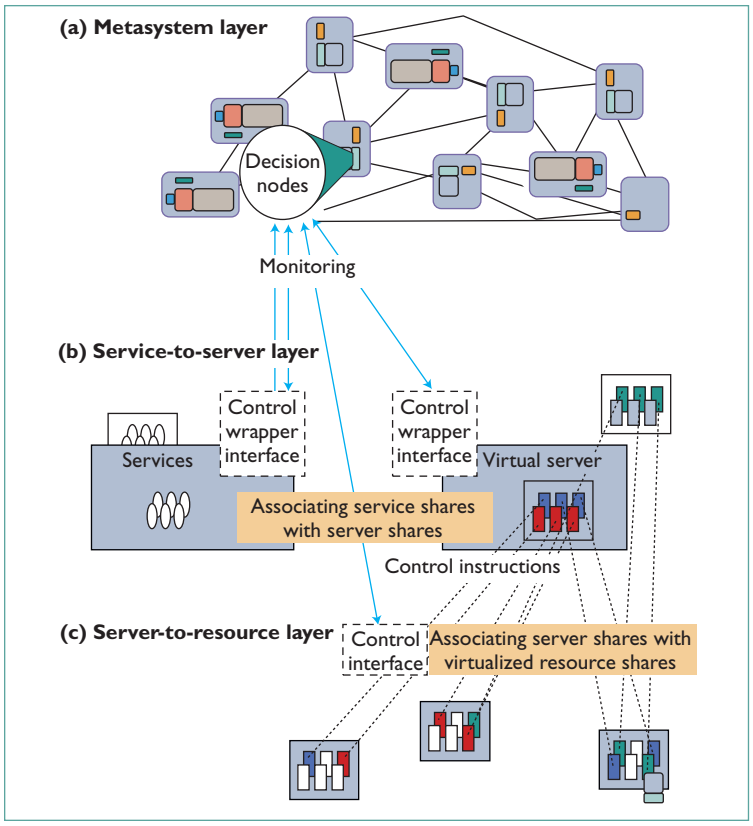

Figure 1. Utility Data Center's control system architecture. (a) The metasystem layer is an overlay system of distributed nodes. (b) The service-to-server layer associates service shares, which relate demand for services to server utilization, with server shares. (c) The server-to-resource layer associates server shares, which represent how much of a service load the server configuration can handle, with virtualized resource shares, which relate demand for virtual resources to server configuration.

cations, all consolidated under the uniform view of services.[4] Uniformly treating the diversity of resources and applications as services significantly simplifies use and management.

Some people use terms like collaborative virtual environments or virtual organizations to describe such domains as well. Whatever you call them, their goal is to provide access to resources and services seamlessly, transparently, and securely across organizational boundaries in a controlled manner.

An international group of researchers founded the Global Grid Forum (GGF; www.ggf.org) to coordinate activities and establish standards for emerging service grids. The Globus project (www.globus.org) provides a public-domain reference implementation of the standards defined by GGF. The Open Grid Services Architecture (OGSA)[5] provides the foundation for service grids. One essential part is the Open Grid Services Interface (OGSI) definition, which GGF has defined. For more information, see the "Related Work in Autonomous System Management" sidebar on p. 43.

Supply Control

One major goal of service grids is to balance service supply with demand for services. Grids are resource supply infrastructures, in which supply and demand are quantitative metrics. Service supply induces demand for resources in the underlying infrastructure that provides resources.

Infrastructure such as the UDC can control the balance on both sides. The UDC can achieve demand–control instruments in various ways:

- refusing further demands for services,
- redirecting demands to where capacity is still available, or
- calculating and imposing price adjustments as an indirect, longer-term control instrument on the demand side.

To achieve supply control, the UDC can adjust service capacity at existing locations, move service capacity toward locations or time frames where demands occur, or utilize capacity elsewhere in the system.[6]

Service Control

Because the UDC separates the storage system from the machine resources, it can maintain multiple images of a service, each representing different capacity configurations. During low demand, the control system activates the service's low-capacity configuration and during high demand, the high-capacity configuration. UDC operators can adjust "virtual server" capacity by programming the resource allocations in the utility data center. Control instructions use the special XML-based UDC Control Language (UCL). A document in that language describes hosting environments with all hardware resources and their wiring relationships.

Human operators or management systems can send documents to the utility controller software, instantiating a farm in the UDC that causes the controller to allocate and configure the described resources properly. A human operator or a management system can provision several capacity configurations of the same service in the UDC as preconfigured farms and instantiate (and switch between them) according to demands. Switching configurations implies shutting down all services' applications and writing out all persistent states to the storage system.

Demand-Capacity Control System

The UDC's control system consists of three building blocks. First, it contains a monitoring and information dissemination infrastructure for collecting utilization data and traces of workloads. One important aspect of this monitoring system is that it aggregates monitored data and transforms the data into a set of abstracted metrics that the system can use to correlate demands with capacities. The second block is the built-in decision-making capability, a new component that is an essential part of the control system. The third block is the actuator that executes decisions by imposing control actions on the demand or capacity side.

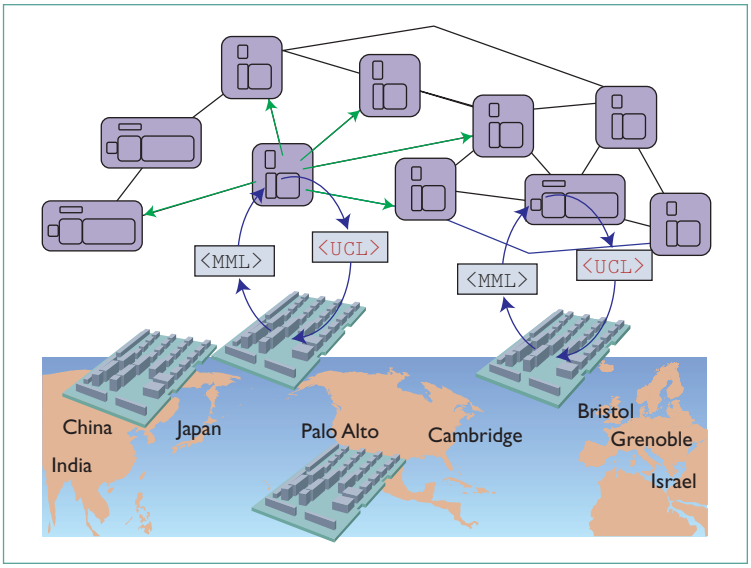

Figure 2. Federation of Utility Data Center control systems. The metasystem at the top of the figure performs monitoring, decision-making, and actuation.

Virtual Servers

Figure 1 introduces the general architecture of the control system, which consists of three layers:

- an infrastructure layer, made up of the resources offered by data centers with utility capabilities that perform (virtual) server-to-resource mapping,
- a layer that maps services to servers, and
- a metasystem — an overlay structure of nodes representing server capacities and service demands — that makes the instructions (decisions) upon which the service-to-server mappings are based.

This architecture is based on the notion of virtual servers — environments that can host services as encapsulated units. The notion of a virtual server generalizes from a machine to a whole operational environment for hosting and performing one or a multitude of services. OGSA uses the term "hosting environment" to describe the generalized virtual server.[5] In this sense, a virtual server is a set of virtual resources allocated in a data center or spread across resources from different data centers.

Figure 1a is the decision-making layer — a metasystem that manages descriptive data about the two lower layers. An overlay-structured network of nodes maintains information about available virtual server environments and the services they need to host. Deployment of nodes automatically establishes this network, which forms an inherently decentralized, distributed structure that adapts to the envisioned global scale of service grids. The UDC uses this overlay structure to perform distributed algorithms that determine how to allocate resources to server environments within nodes, as well as service-to-server environments among nodes. Distributed algorithms constantly observe whether capacity–demand conditions are balanced throughout the overlay topology and eventually trigger control actions directed to entities in the two underlying layers to adjust the balance as needed.

The control architecture's second layer (Figure 1b) allocates services to virtual server environments. This service-to-server mapping function entails deploying services' software and data as well as management and control components belonging to services' applications.

To deploy a virtual server, the server-to-resource layer (Figure 1c) must allocate the necessary resources and configure them to form an operational environment that can host services. This bottom layer of the control architecture allocates machine resources and sets up the overall operational environment. The UDC provides direct hardware support to this layer of the control system by representing a virtual server as a so-called UDC farm, a programmable set of resources and binary images configured in the UDC's storage system.

Federation of Nodes

Figure 2 shows a distributed federation of UDCs, each represented by a node in the metasystem. These nodes are interconnected, and each represents either a virtual server environment (server descriptors) or a service (service descriptors).

Each node contains static descriptive data about the entity it represents, as well as dynamic para-

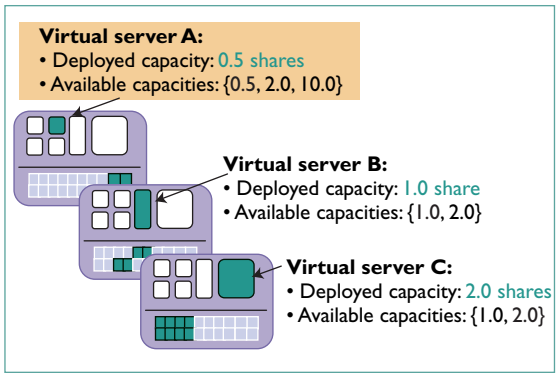

Figure 3. Virtual server environments with different deployed and available capacities. Three data centers provide various capacities (server shares A, B, and C) for a service at three geographic locations.

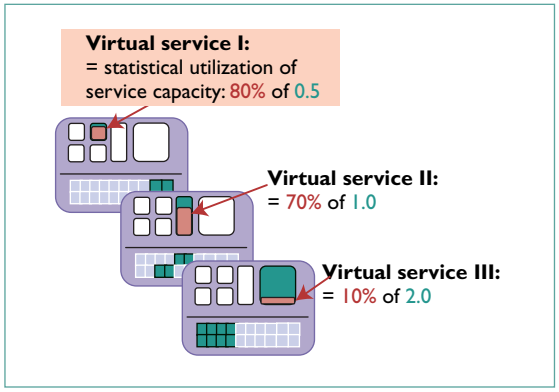

Figure 4. Deployed services on virtual server environments, expressed with service shares. The services, deployed on the three data centers from Figure 3, utilize various amounts of server capacity.

meters about the current condition. The monitoring subsystem updates dynamic parameters using an event or time-triggered push method. The entity and the node transmit data to each other in Monitoring Markup Language (MML) format.

The node then observes dynamic parameters and initiates action (decision and actuation) when conditions require it. Because the nodes are interconnected, they can communicate based on neighborhood relationships. Distributed control algorithms in the metasystem ensure constant communication.

The federated structure of the overall service demand–supply control system takes the form of communicating nodes in the metasystem. Each node represents one member of the control system that is itself a control system for the entity it represents (as the two control loops in the middle of Figure 2 illustrate).

After a node has made a decision (in conjunction with distributed control algorithms), the node translates that decision into an action represented in the UDC's control language, which the node sends to the control interfaces in the underlying system, closing the entity's control loop.

Demand-Capacity Metrics

Our approach characterizes virtual server capacity for a class of services in terms of server shares. Server shares represent a server configuration's ability to handle a certain maximum load of a service when deployed in that server environment. We apply an approach similar to expressing processing capacity in terms of benchmark metrics: machine configuration X can process load Y of application or service Z. For example, say a specific machine configuration can handle 100 transactions per second (TA/sec) of a business application or an industry-standard benchmark. Another machine environment equipped with more resources might be capable of handling 250 TA/sec of that application. A server share represents a normalized metric that expresses a virtual server's ability to handle a maximum amount of load related to a particular benchmark application that is suitable for characterizing the service.

Server shares are normalized to a chosen base unit — to the first example with 100 TA/sec, for instance. This server configuration represents a server share of 1.0. The second configuration would then have 2.5 server shares, expressing its relationship to the considered benchmark's reference configuration.

Formalizing server capacities relative to benchmarks provides an outside-the-box perspective rather than aggregating internal server parameters such as numbers of CPUs, cache sizes, disk and memory configurations, and so on. Our approach lets us consolidate the aggregated behavior of inner parameters into one number.

Figure 3 shows three data centers offering various server environments for hosting services. The top one has a deployed capacity of 0.5 server shares among three possible server configurations with capacities {0.5, 2.0, or 10.0}. *Deployed capacity* means that the data center has already allocated that amount of resources to a particular server environment. The other configurations have what we call *available capacities* because they currently do not have resources assigned, although a control command issued to the data center can deploy and activate that amount of resources later.

Akin to server shares, *service shares* express service demand relative to the utilization of server

capacities among the same class of services. Figure 4 shows three services (or three instances of one service) being allocated to the three deployed server environments. Our metrics formulate service demands in terms of service shares that represent the current utilization of the hosting server's capacity. In the figure, the upper service has a utilization of 0.8 (80 percent) of the server capacity of 0.5. The middle service utilizes 70 percent of the capacity 1.0, and the bottom service utilizes only 10 percent of the capacity 2.0. Expressing service shares in terms of utilization of normalized hosting server environments' capacities indirectly normalizes service shares.

Figure 5 shows how the global metasystem can extract server capacities (expressed in server shares) and service utilizations of those capacities (service shares) from the real system (Figure 1a). Decision-making algorithms then operate in this metasystem of distributed nodes, each of which represents a server capacity, service allocation, and utilization. Nodes publish information about themselves (including capacities, utilization, and so forth) in XML documents called *descriptors*, which communicate information to other nodes in the metasystem. Two main types of descriptors exist: a server descriptor and a service descriptor.[4]

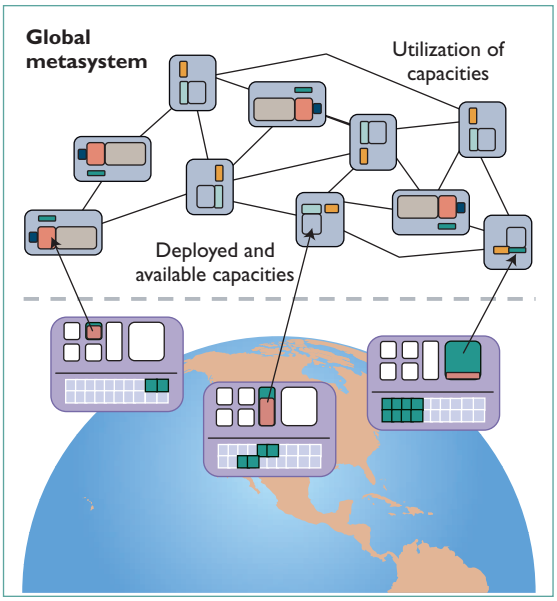

Figure 5. Extracting information to the metasystem. The global metasystem makes available to other nodes the local information about data centers' available capacities and current utilization.

Distributed Control Algorithms

In making decisions about managing resource demands and supplied capacities, the biggest challenge is finding reactive algorithms that deliver high-quality solutions for the control scale we are dealing with. In practice, we must trade the algorithm's reactiveness against the solution's quality. Reactiveness is the time between detecting an abnormality, such as a sudden peak in demand, and the final computation of a decision on how to deal with the situation. Thus, reactiveness constitutes one parameter of the design space. Another is the control system's degree of distribution.

We investigated five distributed algorithms in regard to solution quality versus reactiveness[7] and related them to integer programming:

- agents in overlay networks,
- ant colony algorithms,[7,8]
- broadcast of local eligibility (BLE),
- local random/local round robin, and
- Local greedy distribution.

We considered three time scales:

- the design stage of an initial service placement, sometimes reiterated as long-term adjustment processes in the system;
- a mid-term period for periodic operational adjustments; and
- a short-term period for discharging sudden hot spots.

The local random, local round robin, and local geedy distribution algorithms are characterized by simplicity and statelessness. Much like random or round-robin scheduling, the algorithms push load from an overloaded node to a neighbor, chosen randomly or in a round-robin fashion, that will absorb the load if it has the capacity or push the load further on to another node chosen in the same fashion. Once the algorithm finds a place that can absorb the load, the underlying system initiates the actual load migration. Figure 6 (next page) illustrates the distribution cycle.

Testing Our Approach

In general, goals for optimal placement of applications or services on resources or servers might vary. Therefore, we designed the algorithms to be generic enough to support new objectives without fundamental changes. However, we decided on a few aspects that control decisions must achieve:

- balancing the server load such that each serv-

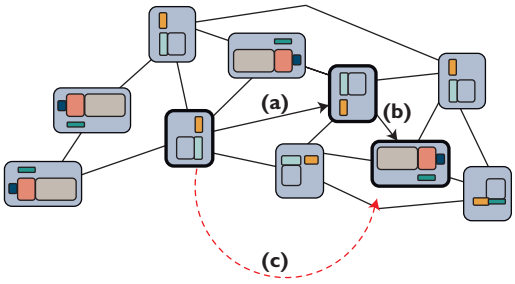

Figure 6. Load distribution algorithms. (a) The algorithm pushes a load from an overloaded node to a neighboring node. (b) The neighboring node either accepts the load or passes it on. (c) Once a node accepts the load, the algorithm migrates the actual load.

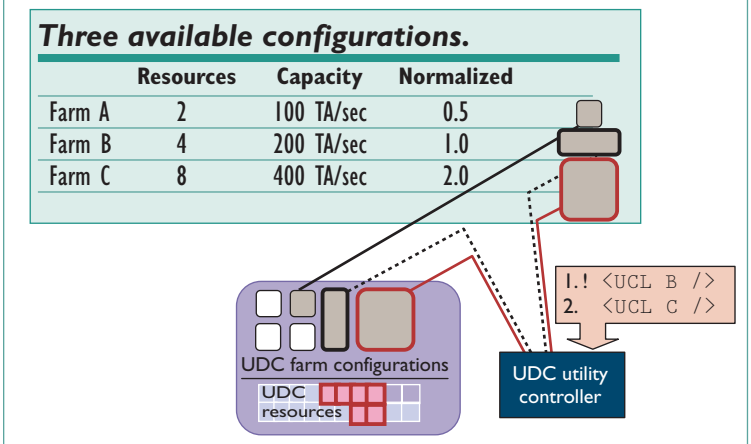

Figure 7. Switching service capacity from 1.0 (farm B) to service capacity 2.0 (farm C). The Utility Data Center (UDC) utility controller switches a service from capacity configuration B (providing 200 TA/sec) to capacity configuration C (providing 400 TA/sec).

er's utilization level is within a desired range,
- placing services in such a way that communication demand among them does not exceed the capacity of the links between the hosting server environments, and
- minimizing the overall network traffic by placing services with heavy traffic close to one another.

We tested our UDC deployment with three available server configurations realized as three different UDC farms, each representing a different capacity of the same service type:

- farm A: 100 TA/sec, normalized to 0.5
- farm B: 200 TA/sec, normalized to 1.0 (base unit), and
- farm C: 400 TA/sec, normalized to 2.0

Figure 7 shows how our system switches service capacity from 1.0 (farm B) to 2.0 (farm C). The metasystem sends the control document to the UDC utility controller. The figure represents the exchange of control documents with `! <UCL B />` and `<UCL C />`. The "!" means that the UDC utility controller deactivates farm B's control document and terminates the corresponding configuration B; the controller then activates the new configuration C when it receives it from the metasystem.

The example shows how a system can automatically adjust service capacity based on information summarized in the global metasystem. The underlying mechanisms come from data center control systems such as the UDC. The UDC demonstrates that a global service supply system can fully automate supply control based on distributed decision-making algorithms.

Conclusions

Capacity management for services contributes to further automation in data center operations. The UDC's programmable capabilities enable our approach. Exposing programmable interfaces to data center resources lets us establish a metalayer that can bring together relevant information about deployments and utilization of services, as well as induced load in underlying resources; decentralized decision algorithms can use that information to optimize demand–supply control in the overall system.

We developed the presented approach as a research project under the mission of establishing globally distributed enterprise grids (grids that connect data centers within one or across many corporations). The idea is to automate management tasks with the ultimate goal of achieving adaptive behavior, a vision HP calls adaptive enterprise. The UDC is one step toward that goal.

References

1. V. Kotov, *On Virtual Data Centers and Their Operating Environments*, tech. report HPL-2001-44, HP Labs, Mar. 2001, www.hpl.hp.com/techreports/2001/HPL-2001-44.html.
2. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. SIGCOMM 2001*, ACM Press, 2001, pp. 161–172.
3. S. Graupner, V. Kotov, and H. Trinks, "Resource-Sharing and Service Deployment in Virtual Data Centers," *Proc. 22nd Int'l Conf. Distributed Computing Systems Workshops (ICDCSW'02)*, IEEE CS Press, 2002, pp. 666–674.
4. V. Kotov, *Towards Service-Centric System Organization*, tech. report HPL-2001-54, HP Labs, Mar. 2001, www.hpl.hp.com/techreports/2001/HPL-2001-54.html.

Related Work in Autonomous System Management

IBM's Autonomic Computing vision[1] aims to provide self-managing systems to create systems that respond to capacity demands and system failures without human intervention. These systems will be self-configuring, self-healing, self-protecting, and self-optimizing. We share this vision, but extend it beyond data center boundaries into global-scale service grids.

IBM's Project eLiza (www-1.ibm.com/servers/autonomic/) is an ongoing effort under the Autonomic Computing aegis to create servers that automatically respond to unexpected capacity demands and system glitches. eLiza's goals are to increase reliability, availability, and serviceability while decreasing downtime and cost of ownership. Project eLiza has made self-management possible throughout some IBM product lines. The xSeries, iSeries, zSeries, and pSeries servers share the following traits:

- support for dynamic clustering,
- support for dynamic partitioning,
- EZSetUp wizards that allow self-installation,
- user authentication, directory integration, and other tools to protect access to network resources, and
- heterogeneous enterprise-wide workload management.

The Océano project (www.research.ibm.com/oceanoproject/) is designing and developing a prototype of a scalable, manageable infrastructure for a large-scale "computing utility powerplant" that enables multicustomer hosting on a virtualized collection of hardware resources. A computing utility infrastructure consists of a "farm" of massively parallel, densely packaged servers connected by high-speed LANs.

Both Océano and eLiza are solutions for inside the data center. They do not encompass virtual environments and distributed services grids.

Traditional grid approaches[2] such as Globus (www.globus.org) focus on distributed supercomputing, in which schedulers make decisions about where to perform computational tasks. Typically, schedulers are based on simple policies such as round-robin due to the lack of a feedback infrastructure reporting load conditions back to schedulers. Globus researchers are planning more sophisticated approaches for grids (see www.globus.org/research/papers/ogsa.pdf). However, we cannot foresee whether Globus will evolve into a service demand–supply control system such as the one presented in this article.

References

1. P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," white paper, IBM Research, 2001, www.research.ibm.com/autonomic/manifesto/.
2. K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems," *Software—Practice and Experience*, vol. 32, no. 2, 2002, pp. 135–164.
5. I. Foster et al., "The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration," draft, Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf.
6. A. Andrzejak et al., "Self-Organizing Control in Planetary-Scale Computing," *Proc. 2nd IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid'02)*, IEEE CS Press, 2002, pp. 359–367.
7. A. Andrzejak et al., *Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems*, tech. report HPL-2002-259, HP Labs, Sept. 2002, www.hpl.hp.com/techreports/2002/HPL-2002-259.html.
8. R. Schoonderwoerd et al., "Ant-Based Load Balancing in Telecommunications Networks," *Adaptive Behavior*, vol. 5, no. 2, 1996, pp. 169–207.

Sven Graupner is a research scientist at Hewlett-Packard Laboratories, Palo Alto, California. His research background is in architecture and management of large distributed systems. He is specifically interested in the intersection with grid technology for operating and managing data centers across large enterprises. He received a PhD in computer science from Chemnitz University of Technology, Germany. He has more than 50 publications, many patents, and is a member of the German Gesellschaft fuer Informatik and the ACM. Contact him at sven.graupner@hp.com.

Vadim Kotov is director of engineering at Carnegie-Mellon University, West Campus, Moffett Field, California. His main research interest is in the architecture of large-scale concurrent and distributed systems. He received a PhD from the Russian Academy of Sciences. From 1991 to 2002, he worked at Hewlett-Packard Laboratories in Palo Alto; before that, he was the director of the Institute of Informatics Systems and the head of the Russian fifth-generation computing consortium START. Contact him at vadim.kotov@cs.cmu.edu.

Artur Andrzejak is a researcher at the Konrad Zuse-Institute, Berlin. His research interests includes utility computing, systems management, and P2P computing. He received a PhD in computer science from ETH Zurich. From 1991 to 1992, he worked at Hewlett-Packard Laboratories on distributed algorithms for an automated demand–supply control system. Contact him at andrzejak@zib.de.

Holger Trinks is currently working toward an MS in engineering management and leadership at Santa Clara University, California. His research interests include distributed software systems, distributed computing, and systems management. He received an MS in computer science from the Chemnitz University of Technology, Germany. Contact him at htrinks@gmx.net.