

# Adaptive Control Overlay for Service Management

Sven Graupner

Hewlett-Packard Laboratories  
Palo Alto, CA 94304  
sven\_graupner@hp.com

Artur Andrzejak

Zuse Institute Berin (ZIB)  
Takustr. 7, D-14195 Berlin  
andrzejak@zib.de

Vadim Kotov

Carnegie Mellon University  
Moffett Field, CA 94035  
vadim.kotov@cs.cmu.edu

Holger Trinks

Santa Clara University  
Santa Clara, CA 95053  
htri@gmx.de

## Abstract

With application systems increasingly built around the services-centric model, management of these eventually large distributed systems of interacting services is a challenge. In contrast to traditional application environments and their related management systems, management of distributed services systems will not have central control and is inherently dynamic.

Furthermore, with dependability of services from hosting locations decreasing, services gain flexibility for identifying “best locations” and “appropriate capacity” for satisfying demands. With demands and other conditions changing over time, services may migrate among locations or adapt their capacity in order to keep the system in a desired balance.

We propose an architecture and algorithms for a service-oriented control system that constantly re-evaluates system conditions and re-adjusts service placements and capacities. The control system is organized as an overlay topology with monitoring and actuation interfaces to underlying services. Evaluation of system conditions is performed by distributed control algorithms in the overlay which possess self-organizing capabilities.

## 1 Introduction

The service-centric model provides abstractions for classical applications not only in terms of unified access methods (common identification by URI's, common protocols XML/SOAP and interface definitions WSDL). An entire suite of standards is created around common application properties and functions in W3C and OASIS. Integration and collaboration of applications adopting these standards, and by thus becoming services, is significantly easier making them more attractive for customers and application providers. Services of such kind are also often called web-services. We consider web

services as an precursory step and state of the current technology towards the service-centric vision where applications can feely interact and collaborate as services and adapt to changing needs and conditions. For us, service-centric implies self-organizing and –managing, capabilities that have not yet developed in web services.

We make several assumptions, not all of them are fully achievable with current technology, but are foreseeable in not too far future as trends indicate [1]. Assumptions are:

- applications are wrapped into an abstraction called *service* allowing them freely to interact and collaborate;
- service are deployed at “hosting locations” – we abstract from platform diversity and introduce a notion of (*virtual*) *servers* that possess the capability to host services<sup>1</sup>;
- as services form interaction topologies (graphs) among each other, so servers are embedded into topologies (graphs) determined by an underlying connection infrastructure giving servers a location within that topology;
- for our control system, we consider induced demands (initially estimated and later measured) into services that are occurring in a certain region requiring a certain capacity of that service in that region; service capacity translates into demands on an underlying hosting server;
- we assume that capacity of services can be altered during operation, for instance by adapting the number of service instances; we assume fixed server capacity at a particular location;

---

<sup>1</sup> Virtualization techniques are an example how a server abstraction can be realized. For instance, a virtual machine environment can host a service and by thus be its server.

- we assume the capability of services to migrate from one server to another in order to keep service capacity close to where demands occur (also helping reducing cross-network traffic).

The focus of this paper is a control system that can determine initial service placements among choices of distributed hosting locations and initial service capacity based on demand estimates. During operation, estimates are constantly re-evaluated, and capacity or locations of services are adjusted.

The distributed service-to-server allocation problem is hard by itself. We assume that no global information about resource availability and service demand can be provided due to scale and dynamism. Decision-making algorithms thus need to deal with partial information, yet provide good approximations of localized allocation solutions, and yet need to be reactive that decisions are made in time for an automated service demand-supply control system.

In the first part of the paper we propose an architecture for such an automated demand-supply control system. It is based on a formalization of service demands and supplies in an overlay system. In the second part, we discuss distributed decision-making algorithms performed in that overlay system with their tradeoffs between quality of solutions and reactivity.

## 2 Architecture for the Self-Managing Service Demand-Capacity Control System

The control system consists of conventional and new building blocks. First, it contains a monitoring and information dissemination infrastructure for collecting utilization data and/or workload traces. One important aspect here is the aggregation of monitored data and transforming data into a set of abstracted metrics that can be used for correlating demands with capacities. The built-in decision-making capability is a new component and the essential part of the control system. The third part performs the actuation of decisions by imposing control actions on the demand or on the capacity side.

The following figure gives an overview of the general architecture of the control system. The architecture consists of three layers:

1. an infrastructure layer consisting of resources offered from data centers with utility capabilities [2] performing a “server-to-resource” mapping,
2. a layer above performs a mapping of “services-to-servers” based on instructions (decisions) made in

3. a meta-system, an overlay structure of nodes representing server capacities and service demands.

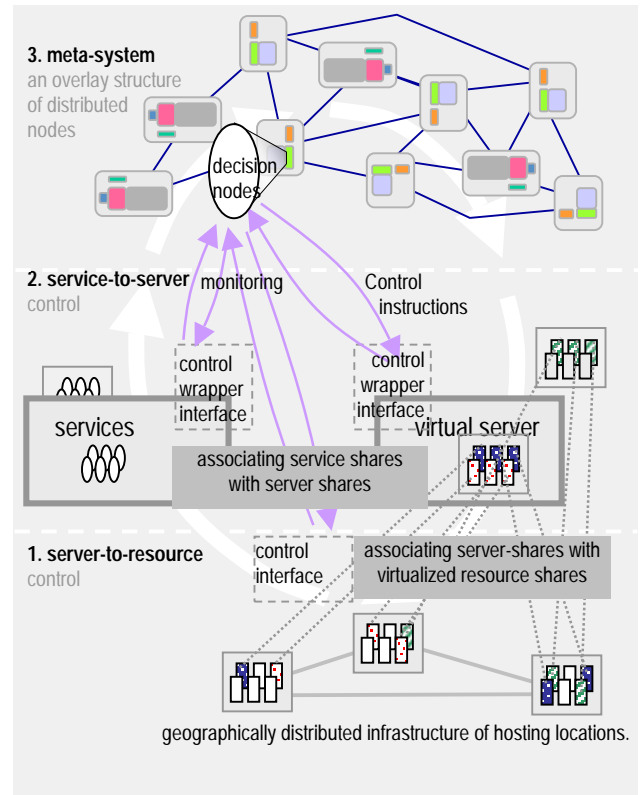


Figure 1: Three-layer control architecture.

The architecture is based on a notion of “virtual servers” as introduced before, environments that can host services as encapsulated units. The notion of a “virtual server” generalizes from a machine to a whole operational environment needed for hosting and performing one or a multitude of services. A virtual server in this sense consists of a set of resources allocated in a data center or even spread across resources from different data centers. A virtual server needs to be materialized or deployed by allocating needed resources and configuring them to form the operational environment that can host services. Resources provide the smallest allocatable entities including machine resources, storage and networks resources as well as the software for configuring and managing them. It also comprises software entities in a service’s environment such as DNS. This bottom layer of the control architecture performs the mapping of “virtual servers-to-resources” including the allocation of machine resources and setting up the overall operational environment. This layer of the control system benefits from Utility Data Center (UDC) capabilities [2] providing means to represent a virtual server by a so-called farm, a programmable set of resources plus binary

images for service's software and data in the storage system.

The second layer of the control architecture builds upon a variety of virtual server environments. This layer performs the allocation of services to virtual server environments. This "service-to-server" mapping function includes the deployment of services' software and data as well as management and control components belonging to the service's applications.

The third layer is the decision-making layer. It is formed as a meta-system managing descriptive data about the two layers underneath. Information about available virtual server environments and services to be hosted are maintained in an overlay-structured network of nodes. It is automatically established during deployment [9] and forms an inherently decentralized, distributed structure adapting it to the envisioned planetary scales of service grids. This overlay structure is used to perform distributed algorithms for decision-making about allocations of resources to server environments within nodes as well as allocating services to server environments among nodes. Distributed algorithms constantly observe whether capacity-demand conditions are in balances throughout the overlay topology and eventually trigger control actions directed to entities in the two underlying layers causing adjustments there.

An infrastructure exists that allows to collect and process monitoring data from sensors updating nodes in the meta-system and disseminates control decisions to control points of servers and services (actuators).

### 2.1 Control Instruments

Control of the demand-supply balance can be exercised on the demand side as well as on the supply side.

*Demand control* instruments are, for instance, admission control (refusing further demands coming into services), redirecting demands in the system to where capacity is still available, or even calculating and imposing price adjustments as indirect, longer-term control instrument on the demand side.

*Supply control* can be achieved by adjusting service capacity at existing locations, by moving service capacity towards locations or time frames where demands occur or by utilizing available service capacity elsewhere [10].

Since the storage system is separated from the machine resources in the UDC, multiple images of a service each representing different capacity configurations can be maintained. During low demand, the control system will activate the low-capacity configuration of the service and during high-demand the high-capacity configuration. In the UDC, virtual server capacity can be adjusted by

programming the resource allocations in the utility data center. Control instructions are described in a special language and are sent to the utility controller software. Switching configurations implies that all service's applications are shut down, and all persistent states are written out to the storage system. Next, the higher (or lower) capacity configuration of the service is launched by allocating needed machine resources and connecting the service's storage images to them. After machines have booted, the service is available again with the adjusted capacity configuration.

In order to make decisions, supplies and demands need to be formalized using appropriate metrics.

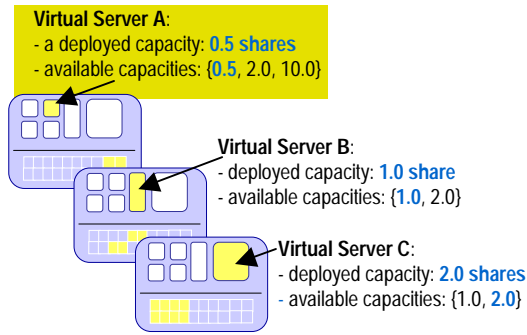
## 3 Metrics: Formalizing Service Demands and Capacities

We use an approach of characterizing a virtual server capacity for a class of services in terms of server shares. Server shares represent the capability of a server configuration to handle a certain maximum load of a service when the service would be deployed in that server environment. We apply here a similar approach of how processing capacity is expressed today in terms of higher application measures (benchmarks) such as: machine configuration X can process load Y of application Z. For example, a specific HP machine configuration can handle 100 transactions per second (TA/s) of a business application or preferably an industry-standard benchmark. Given another machine equipped with more resources, this configuration might be capable of handling 250 TA/s of that application. A *server share* represents the normalized measure expressing a virtual server's capability of handling a maximum amount of load related to a particular benchmark application. Server shares are normalized to a chosen base unit, for instance, to the first example with 100 TA/s. As base unit, this server configuration would represent a server share of 1.0. The second configuration capable of handling 250 TA/s would have 2.5 server shares expressing that this server configuration is 2.5 times more powerful (or has 2.5 times the capacity) than the reference configuration of the considered benchmark.

The approach of formalizing server capacities relatively to benchmarks provides an "outside-the-box" perspective rather than aggregating internal server parameters influencing its capacity, such as numbers of CPUs, cache sizes, etc. It allows to summarize the aggregated behavior into one, consolidated number: server share.

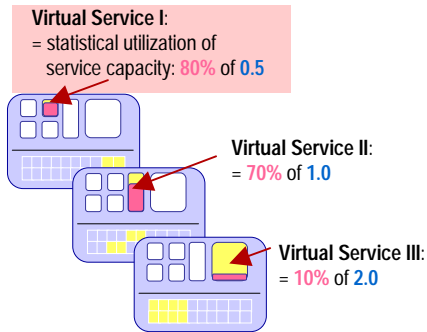
Respectively, service demands can be expressed relatively to utilizations of server capacities among the same class

of services. This measure is called a *service share*.



**Figure 2:** Three virtual server environments with different deployed and available capacities as server shares.

Figure 2 represents three data centers with various offerings of server environments for hosting services. The upper 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 this capacity has allocated resources in the data center. The two other configurations currently do not have resources assigned, but those may be deployed and activated later caused by a control command issued to the data center. Those capacities are referred to as available capacities. The data center in the middle offers two server configurations with capacities {1.0, 2.0} with 1.0 currently being deployed. The lower data center offers two server configurations with capacities {1.0, 2.0} with 2.0 being deployed.

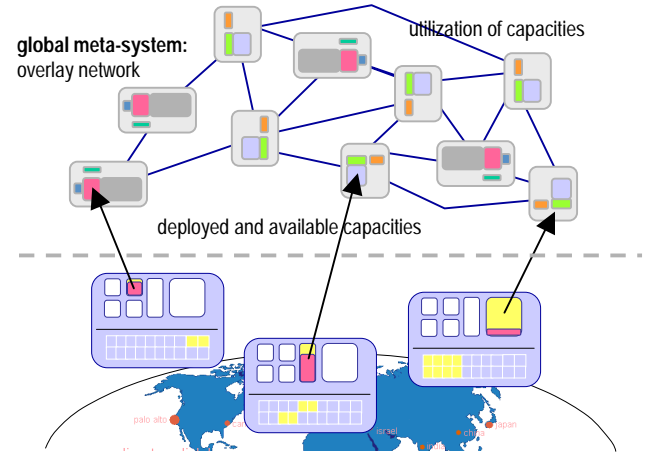


**Figure 3:** Three deployed services with service shares.

Figure 3 shows three services (or three instances of the same service) being allocated to the three deployed server environments. Service demands are formulated in terms of service shares representing the current utilization of the hosting server capacity. In the figure, the upper service has a utilization of 0.8 (80%) of the server capacity, 0.5 in this case. The middle service utilizes 70% of the capacity 1.0, and the bottom service only utilizes 10% of the respective capacity. Since service shares are expressed in terms of utilizations of normalized capacities of hosting

server environments, service shares are indirectly normalized as well and can be correlated.

Server capacities (expressed in server shares) and service utilizations of those capacities (service shares) can be extracted from the real system and be placed into the context of a meta-system (layer 3 in the Architecture shown in Figure 1). Decision-making algorithms then operate in the meta-system of distributed nodes, each representing server capacity and service allocation and utilization.



**Figure 4:** Extracting information to the meta-system.

Since large scales of systems are anticipated, we do not propose a global management hierarchy, but a loosely coupled, federative structure of nodes, each representing capacity and utilization descriptions that can freely join or disappear the structure. We leverage recently emerged overlay network technology and apply them in a slightly extended fashion. Extensions are needed since conventional overlay networks are specialized for searches of rather static content (they are often referred to as content addressable networks, CAN [11], or distributed hash tables, DHT). However, mechanisms for self-establishing structures and keeping relationships among nodes have been leveraged in our system. Extensions primarily refer to the separation of static attributes describing server environment capabilities and dynamic attributes needed for capturing utilization data.

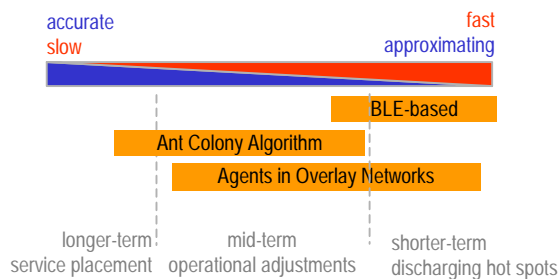
Next, we discuss the decision-making process that builds upon the established overlay structure and automatically adjusts service capacity according to demand fluctuations in a global service grid. Taking the large scale into account, it becomes obvious that centralized approaches are inappropriate. We thus present distributed algorithms that operate in a node overlay.

## 4 Distributed Control Algorithms

We now focus on decision-making for managing resource demand and supplied capacity. Optimal placement of services on servers is the primary factor for the economic utilization of underlying resources, preventing overloading server environments or the communication infrastructure, keeping resource utilization and response times in balance, and achieving higher availability and fault-tolerance.

The biggest challenge is to find algorithms that are both reactive and deliver high-quality solutions for the control scale we are dealing with. In practice, the responsiveness of an algorithm must be traded against the quality of a solution. Thus, responsiveness constitutes one parameter of the design space. Another parameter is the type of the control system, ranging from centralized to completely distributed. Since it is unrealistic to find one algorithm, which can be parameterized in both dimensions, we look at several approaches covering most of the design space.

Figure 5 classifies three algorithms in regard to solution quality vs. reactivity. Since being part of a control system, reactivity of decisions is important. Reactiveness is understood as the time between detection an abnormality, for instance a sudden peak demand, and the final computation of a decision how the situation can be dealt with. Three time scales are considered: the “design” stage of an initial service placement, in longer periods reiterated as long-term adjustment process; a mid-term period for periodic operational adjustments, and a shorter-term period for discharging hot spots.



**Figure 5:** Decision-making algorithm tradeoffs.

One approach is a centralized heuristic algorithm based on integer programming. This algorithm yields high-quality solutions but at a cost of longer running time and limited scalability. For improved responsiveness and larger scale, we explore agent-based and distributed algorithms described below. Such algorithms are composed of several simple decision-making instances, sometimes also referred to as agents. Decision-making instances perform their algorithms in a distributed structure of nodes as shown in Figures 1 and 4 in the

meta-system of the control architecture. They communicate with each other directly or indirectly in order to approximate a solution. Each decision-making instance has, in general, only partial knowledge of the system. This facilitates scalability of such approaches. Furthermore, failure of any of the decision-making instance does not make the overall algorithm fail. The algorithms assume a hierarchical overlay control structure, which determines how each decision-making instance of the distributed algorithms is placed and how they communicate with one another.

One agent-based approach is based on the “ant-based” control algorithms [12], [13]. This fully distributed algorithm has medium responsiveness and can be used for periodical reassignments of services onto servers.

As an alternative approach, we evaluate an agent system based on a paradigm known as Broadcast of Local Eligibility (BLE), used for coordination of robot teams [14]. This partially distributed algorithm allows faster rebalancing of the managed services for the price of potentially lower-quality assignments.

### 4.1 Control Objectives

As discussed in the beginning of this section, the goals for optimal placement might vary in general. Therefore, the following algorithms are designed to be generic enough to support new objectives without fundamental changes. However, we focus on only few aspects to be achieved by control decisions. These are:

1. Balancing the server load such that the utilization of each server is in a desired range.
2. Placing services in such a way that communication demand among them does not exceed the capacity of the links between the hosting server environments.
3. Minimizing the overall network traffic aiming to place services with high traffic close to each other on nearby servers (nearby in the sense of a low number of communication hops across nodes).

### 4.2 Ant-Based Control Algorithm

In the classical Ant Colony Optimization [12], the path taken by an ant on its way between objects (e.g. cities in the Traveling Salesman Problem) represents a possible solution to the optimization problem. In our case, the objects would be both servers and services, and the alternating path would represent an assignment of services to servers. However, this approach is centralized and not really scalable for the following reasons:

1. The ant must “remember” the whole path it has taken; this information might become very large in the end.

2. The ant must visit all objects on its tour. In a dynamic system, this could pose a serious drawback.
3. Finally, each solution (path) must be evaluated against others. This requires central knowledge.

For these reasons, we evaluate another approach resembling the ant-based control for network management described in [13].

In our system, for each service  $s$  to be placed on a new server due to an overload condition or the arrival of a new service, an ant (“agent”) is created. This ant knows the requirement attributes of the service it represents, such as processing and storage requirements. It also carries a list of services cooperating with  $s$  together with their communication requirement attributes.

The ant then travels from one server to another choosing the servers along the path based on a probability computed locally as described below. The ant then finally makes a decision based on the knowledge it has accumulated on its travel on which server the service will finally be placed.

On each server, the ant evaluates the score of the server in respect to  $s$ . This score expresses how well the currently visited server is suitable for the placement of  $s$ . Once evaluated, a data structure of the current server called multimark is updated with the score of  $s$  and with flags indicating services cooperating with  $s$ . The multimarks of the nearby servers are updated as well, both with the score for the placement of a cooperating service and with related flags for  $s$ .

The score for a server in respect to  $s$  is computed by:

- How well does the server meet the requirement attributes of  $s$  based on recent utilization history?
- Can cooperating services be placed on nearby servers?
- What is the amount of the weighted traffic between the current server and the servers potentially hosting cooperating services? Weight factors are the distances between all server pairs.

The choice of an ant which server to visit next is based on the current utilization of the considered server (the probability decreases with higher utilization) as well as on the value of the multimark for  $s$  (or for one of the cooperating services, if the multimark for the targeted server has no entry for  $s$ ). Here, we use the fact that multimark records contain both the score of the ants, which visited this server as well as information of the ants corresponding to its cooperating services.

The termination of the walk of an ant is determined by a parameter set upon its creation – the maximum number of servers to be visited. It gives us partial control over the

trade-off between responsiveness and the solution quality. Upon termination, the ant determines the server with the highest score from an internal (limited length) priority list. It then sends a message to the managing agent of this server with the suggestion to install  $s$  on it.

### 4.3 BLE-Based Control Algorithm

We adapt the concept of the Broadcast of Local Eligibility used for coordination of robots [14] for the placement of services. This concept can be used to create highly fault-tolerant and flexible frameworks for coordination of systems of agents. However, the originally proposed framework has a drawback of limited scalability. To overcome this problem, we use a hierarchical control structure discussed below.

We consider a cluster of servers with a distinguished server called cluster head. Each member of the cluster has the ability to broadcast a message to all other members of the cluster. This can be done either directly or via the cluster head. The placement of services in this cluster is periodically re-evaluated by arbitration between peer servers in so-called decision cycles. The time between two cycles is determined by the required responsiveness to fluctuations in server utilization and by the induced communication between cluster members.

In each decision cycle, the following actions take place:

1. Each server broadcasts the list of services it hosts with all new arrived services and simultaneously collects a list of all services in the cluster.
2. Each server evaluates its own suitability to host each service and sorts the list according to the computed score. The criteria are similar to those for the ant-based control system. In addition, a service already deployed on a server highly increases the score.
3. Each server broadcasts a list ordered by scores of those services the server can host simultaneously without exceeding its capacity.
4. When a server receives a score list from a peer, it compares the score with its own score for a service. Each server now knows whether it is the most eligible one for hosting a particular service.
5. The changes in the service placement are executed. Notice that each server knows already whether it has to install new or remove current services. In addition, the cluster head compares the initial list of the services with those, which will be hosted at the end of this decision cycle. The remaining services are passed on to the next hierarchy level.

Obviously, the scalability of this approach is limited by the size of the cluster, the communication capacity in the cluster and the processing capacity of the cluster head.

We propose a following hierarchical approach to extend the scalability. Basically, the cluster heads of the clusters at level  $k$  are treated as "normal" members of a cluster of level  $k+1$ . However, they compete only for services, which could not be installed in their own cluster (see step 5. above). After a decision round in the cluster of level  $k+1$ , these pending services are possibly moved to another peer, which is a cluster head for a cluster of level  $k$ . (The cluster head evaluates the eligibility of the servers in its own cluster, not its own eligibility). In the cluster of level  $k$ , these services become part of the list of services to be installed and participate in the normal decision cycles.

The cluster size is essential for the balance between the responsiveness of the system and flexibility. Identifying a correct hierarchical structure can be done similarly to clustering algorithms used in sensor networks [15].

#### 4.4 Agents in Overlay Networks

Each service is represented in multiple service groups carried in agents that move around in a overlay network of nodes. Each agent evaluates its current position for a potential new placement decision on the node it currently visits. It also observes neighbored agents for services sent by them for deployment on that node. It finally makes a decision to place its service group on that node or move on to one of the neighbors. The topology is a CAN-based overlay network [11] that maps server (node) capacities into key attributes. Since being structured by node capacities, the overlay network allows agents to move faster towards nodes with capacities they assume. Overlay networks also have a self-organizing capability in regard to including new members into the network and rearranging structures. However, since server capacity is seen as basis for structuring the network (keys in the index structure), capacity dynamism is hard to achieve. The assumption is that server capacities do not change frequently and allocation is binary meaning that one service (agent) deciding for a node utilizes 100% of that node for the time it requires that capacity.

Agents evaluate on their current locations demands from all services they represent. They also investigate all neighbors for available capacity. When an agent cannot place demand on either the node it currently resides or on one of the neighbor nodes, it decides to move on to one of the neighbor nodes and repeats the algorithm. The decision where to move next depends on so-called neighborhood density values, aggregates all neighbors collect from their neighbors about capacities. Asking all neighbors for their neighborhood density values allows an

agent to decide for the node with the highest capacity density value leading to a hill climbing behavior of all agents moving towards locations with most available capacities and by doing so filling these capacities. This algorithm therefore has an inherent balancing character.

Decentralized control algorithms appear to be promising for decision-making in large-scale virtual data centers as part of their control systems.

## 5 Related Work

IBM's Autonomic Computing vision [16] aims to provide self-managing systems. The intent is to create systems that respond to capacity demands and system glitches without human intervention. These systems intend to be self-configuring, self-healing, self-protecting and self-optimizing. We share this vision extending it beyond data center boundaries into planetary-scale service grids.

IBM's Project **eLiza** [17] is an ongoing effort to create servers that respond to unexpected capacity demands and system glitches without human intervention. The goal are new highs in reliability, availability and serviceability, and new lows in downtime and cost of ownership. Project eLiza has made self-management capabilities possible throughout IBM system families. Traits shared by xSeries, iSeries, zSeries and pSeries servers include:

- Support for dynamic clustering.
- Support for dynamic partitioning.
- EZSetUp wizards, allowing for self-installation.
- User authentication, directory integration and other tools to protect access to network resources.
- Heterogeneous enterprise-wide workload management.

The **Océano** project [18], joint work between IBM and the University of Berkeley, is designing and developing a pilot prototype of a scaleable, manageable infrastructure for a large scale "computing utility powerplant" that enables multi-customer hosting on a virtualized collection of hardware resources. A computing utility infrastructure consists of a "farm" of massively parallel, densely packaged servers interconnected by high-speed, switched LANs. This project aims to address many of the open technical issues in these powerplant environments. Hosted customers increasingly require support for peak loads that are orders of magnitude larger than what they experience in their normal steady state. Thus, a hosting environment needs a faster turnaround time in adjusting the resources (bandwidth, servers, and storage), assigned to each customer to the dynamically fluctuating workload. The objectives of the Océano project include:

- Implement an infrastructure that enables large numbers of hosted customers over Linux servers.
- Reduce the costs of setting up and operating the hosting farms by automation.
- Dynamically assign resources to accommodate planned and unplanned fluctuation of workloads.
- Offer a wide variety of services levels to customers.
- Secure sharing of resources across multiple customers.
- Provide adequate reliability through massive redundancy, and automated re-provisioning.

Océano will develop middleware and infrastructure that provide composition of hosting services, including monitoring of Service Level Agreements, Dynamic Resource Allocation, and High Availability. This middleware and infrastructure will enable the development of powerplants that can handle multiple customer applications and large surges in workload traffic.

Océano as well as eLiza are both targeted to “inside the data center” solutions. They do not encompass virtual environments and planetary-scale distributed services grids as proposed in this paper.

Traditional grid approaches such as Globus [4] have been focused on distributed supercomputing where schedulers made decisions about where computational tasks will be assigned. Typically, schedulers were based on simple policies such as round-robin due to the lack of a feedback infrastructure reporting load conditions back to schedulers. More advanced approaches are in [11].

## References

- [1] Foster, I., Kesselman, C., Nick, J.M., Tuecke, S., *The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration*, DRAFT, <http://www.globus.org/research/papers/ogsa.pdf>, 2002.
- [2] HP, *Utility Data Center*, <http://www.hp.com/go/hpudc>, <http://www.hp.com/go/always-on>, November 2001.
- [3] The Global Grid Forum, <http://www.gridforum.org/>.
- [4] The Globus Project, <http://www.globus.org>.
- [5] Graupner, S., Kotov, V., Trinks, H.: *Resource-Sharing and Service Deployment in Virtual Data Centers*, IEEE Workshop on Resource Sharing in Massively Distributed Systems (ICDCS-2002), July 2, 2002, Vienna, Austria.
- [6] Kotov, V.: *On Virtual Data Centers and Their Operating Environments*, HP Labs Technical Report<sup>2</sup>, HPL-2001-44, March 2001.
- [7] Kotov, V.: *Towards Service-Centric System Organization*, HP Labs Technical Report, HPL-2001-54, March 2001.
- [8] W3C, *Web Services Description Language (WSDL)*, <http://www.w3.org/TR/wsdl>, March 2001.
- [9] Graupner, S., Kotov, V., Trinks, H.: *Recursive Deployment of Management Agents in Planetary-scale Control Systems*, HP Labs Technical Report, to be published in December 2001.
- [10] Andrzejak, A., Graupner, S., Kotov, V., Trinks, H.: *Self-Organizing Control in Planetary-Scale Computing*, IEEE International Symposium on Cluster Computing and the Grid (CCGrid), May 21-24, 2002, Berlin.
- [11] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., *A Scalable Content-Addressable Network*, SIGCOMM 2001, San Diego, August 27-31, 2001.
- [12] Dorigo, M., Maniezzo, V., Colomi, A.: *The Ant System: Optimization by a Colony of Cooperating Agents*. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41, 1996.
- [13] Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: *Ants for Load Balancing in Telecommunications Networks*, Adaptive Behavior 2:169-207, 1996.
- [14] Werger, B. B., Matarić, M.: *From Insect to Internet: Situated Control for Networked Robot Teams*, to appear in Annals of Mathematics and Artificial Intelligence, 2000.
- [15] Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: *Next century challenges: Scalable coordination in sensor networks*, Proceedings of MOBICOM, pp. 263-270, Seattle, USA, August 1999.
- [16] IBM, *Autonomic Computing*, Manifesto, <http://www.research.ibm.com/autonomic/manifesto>.
- [17] IBM, *eLiza*, <http://www-1.ibm.com/servers/eserver/introducing/eliza>.
- [18] IBM, and University of Berkeley, *Océano Project*, <http://www.research.ibm.com/oceanoproject>.

---

<sup>2</sup> HPL-TR are available: <http://lib.hpl.hp.com/techpubs>.