Control Architecture for Service Grids in a Federation of Utility Data Centers

Sven Graupner, Vadim Kotov, Artur Andrzejak, Holger Trinks
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-235
August 21st , 2002*

E-mail: {sven_graupner, vadim_kotov, artur_Andrzejak, holger_trinks}@hp.com

grid computing, utility computing, utility data center, control architecture, virtual data center

Growing complexity and cost of system deployment, ownership and operation pushes to look for economical, yet limitless ways to organize and manage large-scale computing in science, technology and businesses. The two most prominent examples are the concepts of the Utility Data Center [1] and the Grid [2].

One of the hard problems in system management is the distributed resource allocation problem. We assume that no global information about resource availability and demands for resources can be provided due to the scale and dynamism of large systems. The paper introduces an architecture of an automated service demand-supply control system that is part of a large-scale Grid infrastructure comprised of a federation of distributed Utility Data Centers.

* Internal Accession Date Only
Approved for External Publication

Control Architecture for Service Grids in a Federation of Utility Data Centers

Sven Graupner, Vadim Kotov, Artur Andrzejak, Holger Trinks

Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA

{sven_graupner, vadim_kotov, artur_Andrzejak, holger_trinks}@hp.com

Abstract

Growing complexity and cost of system deployment, ownership and operation pushes to look for economical, yet limitless ways to organize and manage large-scale computing in science, technology and businesses. The two most prominent examples are the concepts of the Utility Data Center [1] and the Grid [2].

One of the hard problems in system management is the distributed resource allocation problem. We assume that no global information about resource availability and demands for resources can be provided due to the scale and dynamism of large systems. The paper introduces an architecture of an automated service demand-supply control system that is part of a large-scale Grid infrastructure comprised of a federation of distributed Utility Data Centers.

1 Introduction

As reaction to concerns about growing complexity and, as result, the potential ineffectiveness and insufficient manageability of large-scale systems, new approaches to system design, use, and management are emerging:

- the aggregation and consolidation of system and application components into larger building blocks,
- systematic and standard ways of their integration and communication,
- sharing of distributed resources, and
- automated system management and operation control.

Two the most prominent and practical examples are the concepts of the Utility Data Center [1] and the Grid [2]. The Utility Data Center (UDC) consolidates computing resources in order to significantly reduce deployment and operation costs. Grids are large networks of computing resources that can be transparently shared and utilized for solving complex tasks or providing computing services.

These two concepts can be combined into a concept of Virtual Data Centers [3] that consolidate resources of a federation of distributed Utility Data Centers into virtual resources that are shared using Grid–type mechanisms.

Grid computing emerged in the scientific supercomputing in the early 1990's by tapping into underutilized resources available in organizations and making them available for solving complex computations. A software layer provides the coordinated, transparent and secure access to shared resources across geographically distributed sites. Resource virtualization allows transparency and security. The software layer also provides "grid membership" of a machine or a device making its resources discoverable and allocatable to other entities in the system. Many Grid projects in research and industry are based on the Globus toolkit [4], a widely used public domain software. Commercial Grid products are offered by IBM [5], Platform [6] and Sun [7]. An overview of Grid resource management systems can be found in [8].

Another source of the "grid trend" is the utility model of resources. Access to resources is aimed to be as simple and efficient as accessing power or other utilities. Resource markets are envisioned where resources used in information processing can be traded and exchanged as commodities. Resource commoditization also helps to overcome the diversity and complexity of IT landscapes making it attractive for both IT vendors and customers.

Resource capacities should also be provided locally to where demands occur avoiding cross-network traffic. Since demands are fluctuating over time and locations, service capacities need to be adjusted accordingly, ideally automated without human intervention. Such an automated service grid control system transparently regulates service demands and supplies.

So far, most integrated management systems are limited in regard to functioning in virtualized environments across organizational boundaries. Besides automated fail-over techniques in high-availability systems, management systems typically automate monitoring and information collection. Decisions are made by human operators interacting with the management system. Major service capacity adjustments imply manual involvement in hardware as well as in software. Systems need to be adjusted, re-installed and reconfigured.

A new type of data center infrastructures provides immediate support for these tasks. The HP offering is the Utility Data Center (UDC) [1], [9]. Its capabilities allow a whole new approach to automate adjustment processes and by thus set the foundation for 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 with utility data center (UDC) capabilities providing immediate support for service demand - supply control.

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

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 meta-system. We then briefly discuss distributed decision-making algorithms performed in that overlay meta-system with their tradeoffs between quality of solutions and reactiveness. This architecture is general enough for a variety of large-scale distributed systems such as a federation of distributed Utility Data Centers.

2 Utility Data Center Platform

The reason for developing a utility data center platform was that deployment and operational costs dominate the balance sheets of enterprise IT customers. Platforms and management solutions are emerging reducing service deployment times and operational costs. Those platforms support the deployment of services (installation and configuration of software and data), the virtual wiring of machines into application environments, here referred to as virtual server environments, independently of the physical wiring in a data center. They allow programmatic rearrangements of services' applications among machines, the dynamic sizing of service capacities, and the isolation of different environments hosted in the same data center. Besides those capabilities, utility data centers are enablers for automated control systems as discussed in this paper.

A major characteristic of an utility data center is resource virtualization. The storage virtualization fabric with the storage area network attaches storage elements (disks) to processing elements (machines). The network fabric links processing elements together in a private virtual LAN.

Two types of resources are virtualized in the UDC:

- network resources: by permitting the programmable rewiring of server machines and devices to create a virtual LAN network. Virtual wiring is achieved by programming network switches connecting machines and programmatically connecting or removing machines to or from virtual networks,

- storage resources: by containing whole disk images with all persistent states of application environments, file systems, bootable operating system images, application software, etc. Given the programmability of the storage fabric, storage images can be made appearing on SCSI interfaces of machines from where machines obtain boot images and further data.

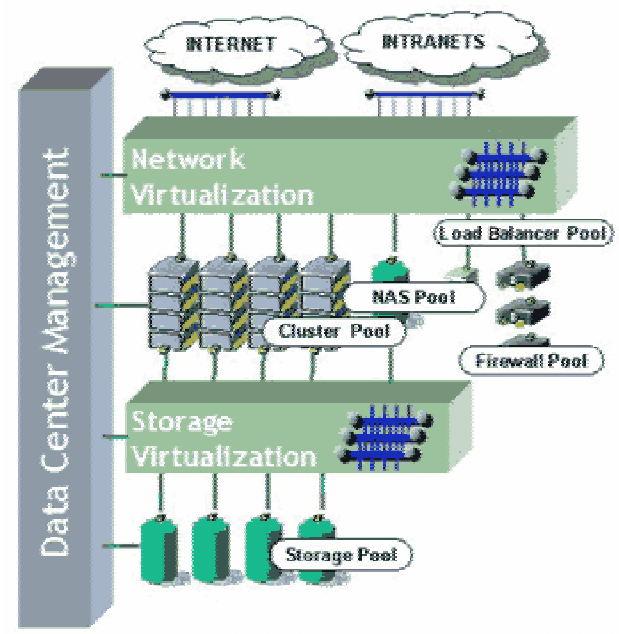

Figure 1: Utility Data Center (UDC) [1] with the two main components: the fabric for network and the fabric for storage virtualization.

A UDC platform has two major benefits:

- *automated services deployment:* achieved by entirely maintaining persistent services' states in the separate storage system and conducting programmatic control over attaching storage to machines, and
- *dynamic capacity sizing of services:* achieved by the ability to automatically launch additional service instances absorbing additional load. Service instances are launched by first allocating spare machines from a pool maintained in the data center, then virtually wiring them into the specific environment of the service with attaching the appropriate storage to those machines and launching the operating systems and applications obtained from the attached storage.

We leverage the benefits of the utility data center as execution platform for control decisions. The research presented in this paper extends the scope of control beyond the walls of one data center assuming a multitude of them as infrastructure for next-generation services grids encompassing seamless service demand-capacity control.

3 Towards OGSA Service Grids

The Global Grid Forum (GGF) [2] has the goal to coordinate activities and to establish standards for emerging service grids. The most visible project from the scientific community is The Globus Project [10]. This project is recently gaining significant attention and support from hard- and software vendors. A related academic project with similar goals of securely and transparently sharing resources has been NOW, Network of Workstations from the University of Berkeley [11]. The currently largest effort is the SETI@home project [12] with more than 3.7 million Internet users that have donated a total of 1 million hours of processing time translating into $1.6*10^{21}$ ($\approx 10^9$ Tera) Flops, by far the largest supercomputer that ever existed.

Though the need for compute power seems infinite in scientific supercomputing, resource sharing comprises more than distributing compute tasks transparently among heterogeneous compute nodes. From a commercial point of view even more important is data sharing as transparently and securely among organizations, including software and services needed to access, maintain and process this data. A grid in this context stands for a collaborative domain spanning multiple networks and organizations to securely and efficiently connect different organizations for sharing data and services needed for collaboration [3], [13]. Such grid domains comprise the whole spectrum of resources, documents and all other data including accompanying applications – all diversity consolidated under a uniform view of *services* [14]. Terms like collaborative virtual environments or virtual organizations are used as well.

The goal is to provide access to resources and services seamlessly, transparently and securely across organizational boundaries in a yet controlled and secure manner. Recent publications from Globus reflect this in the *Open Grid Services Architecture (OSGA)* [15].

In order to categorize grids, we refer to the first-generation grids as *resource grids* and to the second-generation as *service grids*. Services grids may differ in scale and scope: enterprise grids (within one organization), peer or partner grids (across defined sets of organizations) and global grids (publicly accessible).

A major aspect of service grids is to keep service supply in balance with demands for services. Service demand needs to be met by service supply. Service supply then induces demand for resources in the underlying infrastructure that provides resources. The logical chain is: service demand to service supply translating into resource demand to be met by resource supply. Grids are basically resource supply infrastructures. Supply and

demand are quantitative aggregates meaning that services do not only provide a required functionality (and most service description methods and frameworks only capture functionality, often in form of API specifications such as WSDL [16]), services also have to provide their functionality in sufficient quantity such that occurring demands can be satisfied in desired quality.

4 Service Control

Control of the demand-supply balance can be exercised by various instruments, on the demand side as well as on the supply side.

Demand control instruments can be established in various ways, for instance, by admission control (refusing further demands coming into services), by redirecting demands in the system to where capacity is still available, or even by 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 in the system [17].

Since the storage system is separated from the machine resources in the UDC, multiple images of a service can be maintained each representing different capacity configurations. During low demand, the control system will activate the low-capacity configuration of the service and during high-demand the high-capacity configuration. "Virtual server" capacity can be adjusted in the UDC by programming the resource allocations in the utility data center. Control instructions are described in a special language FML (Farm Markup Language) that 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 the 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.

5 Architecture of an Automated 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 traces of workloads. 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 introduces the general architecture of the control system. It consists of three layers:

1. an infrastructure layer consisting of resources offered from data centers with utility capabilities performing a (virtual) "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 dynamic server capacities and service demands.

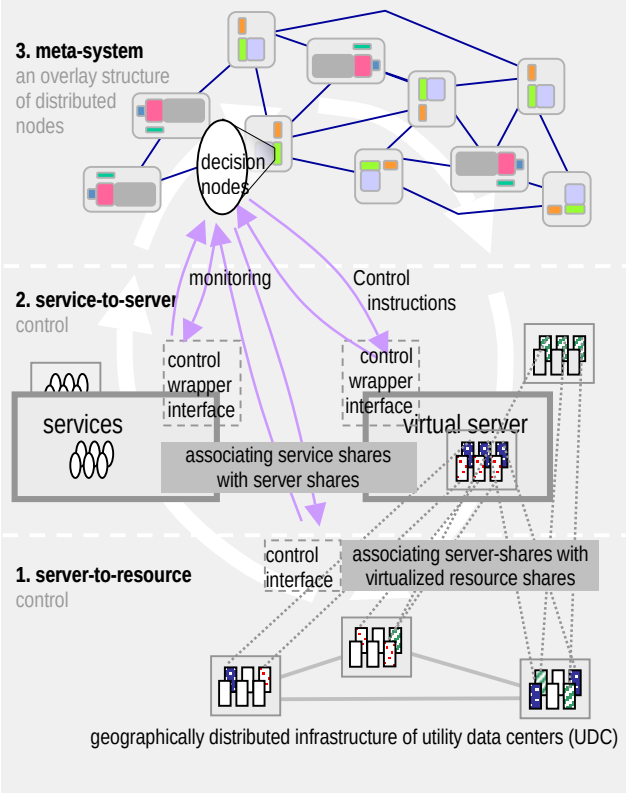

Figure 2: Three-layer architecture of the control system.

The architecture is based on a 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 needed for hosting and performing one or a multitude of services.

OGSA uses the term hosting environment [15]. A "virtual server" in this sense consists of a set of virtual 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 derives direct hardware support from the utility data center in form of representing a "virtual server" by a so-called "UDC farm", a programmable set of resources plus binary images in the storage system for service's software and data.

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. Technology for automating these processes is being developed, for instance by [18].

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 [19] 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 kept in balance 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) in the system.

Figure 3 shows a distributed federation of Utility Data Centers each represented by a node in the meat-system. Nodes in the meta-system are interconnected. Nodes in the meta-system represent virtual server environments (server descriptors), services (service descriptors) or, summarized, entire data centers as shown in the figure.

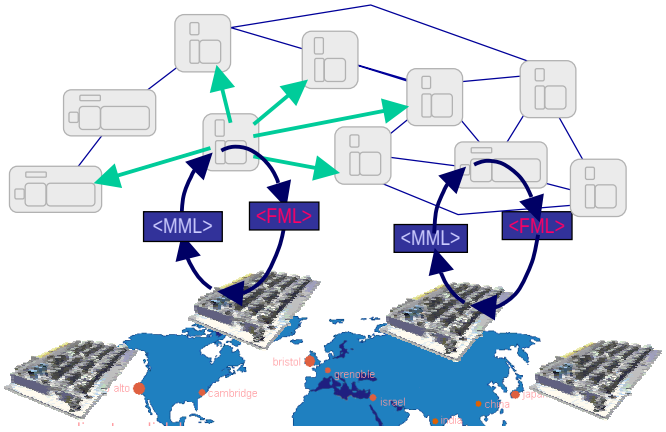

Figure 3: Federation of UDC control systems with meta-system with monitoring, decision-making and actuation.

Each node contains static descriptive data about the entity it represents as well as dynamic parameters about the current condition. Dynamic parameters need to be updated. This task is performed by the monitoring subsystem using an event or time-triggered push method. Data are transmitted between the entity and the node in MML (Monitoring Markup Language) format.

The node then observes dynamic parameters and initiates action (decision and actuation) when conditions require. Since nodes are interconnected, nodes can communicate based on neighborhood relationships. Constant communication takes place in the meta-system by distributed control algorithms.

The federative structure of the overall service demand-supply control system thus is materialized in form of communicating nodes in the meta-system, with each node representing one member of the control system that itself is a control system for the entity it represents (illustrated by the two control loops in Figure 3.

After a node (in conjunction with distributed control algorithms) has made a decision, it is translated into an action represented in FML (Farm Markup Language), the control language of the Utility Data Center, which is sent to the control interfaces in the underlying system closing the entity's control loop.

5.1 Metrics: Formalizing Service Demands and Capacities

Another precursory for automated decision-making is how service demands and capacities can be properly described and formalized such that algorithms can be applied for deriving decisions.

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 benchmark measures: machine configuration X is capable of processing load Y of application or service Z. For example, a specific machine configuration can handle 100 transactions per second (TA/s) of a business application or an industry-standard benchmark. Given another machine environment equipped with more resources, it 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 that is suitable to characterize the service. Server shares are normalized to a chosen *base unit*, for instance, to the first example with 100 TA/s. This server configuration would represent a server share of 1.0. The second configuration capable of handling 250 TA/s would then 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.

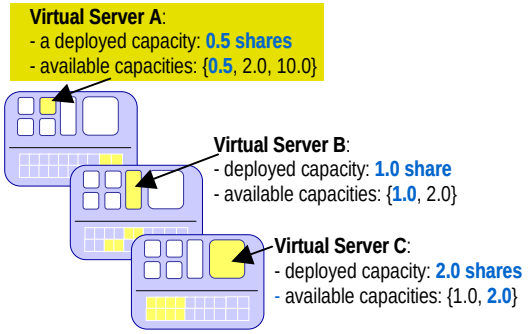

Figure 4: Three virtual server environments with different deployed and available capacities expressed in terms of server shares.

The approach of formalizing server capacities relatively 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, etc. Our approach allows to summarize the aggregated behavior of all inner diversity 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 4 shows 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. And the lower data center offers two server configurations with capacities {1.0, 2.0} with 2.0 being deployed.

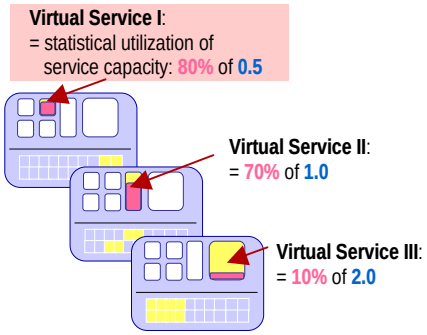

Figure 5: Three deployed services with service shares.

Figure 5 shows three services (or three instances of one 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 of 0.5. The middle service utilizes 70% of the capacity 1.0, and the bottom service only utilizes 10% of the respective capacity 2.0. 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 2). Decision-making algorithms then operate in this meta-system of distributed nodes, each node representing a server capacity, service allocation and

utilization. Nodes publish information about themselves (including capacities, utilization etc.) in XML documents called *descriptors*. Descriptors are used to communicate information with other nodes in the meta-system. Two main types of descriptors exist: a *server descriptor* and a *service descriptor*. More detail can be found in [14].

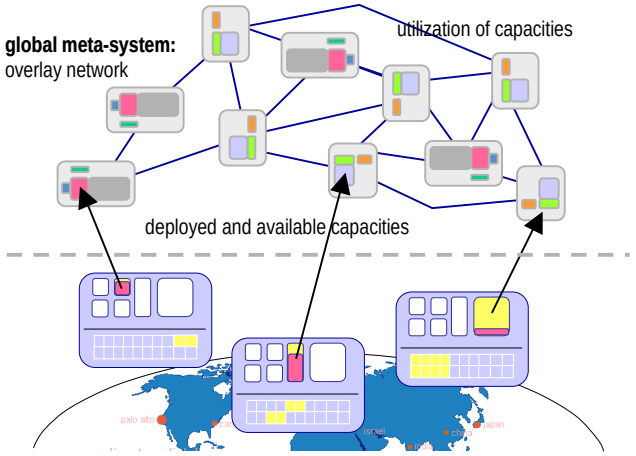

Figure 6: Extracting information to the meta-system.

Since large scales of systems are anticipated in the future, we do not propose a global hierarchical structure for the meta-system, rather a loosely coupled, federative structure of nodes, each representing server capacities and service utilizations. Nodes can freely join or disappear. We leverage recently emerged overlay network technology and apply it 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 [20] 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 utilizations.

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 the meta-structure of nodes.

5.2 Distributed Control Algorithms

For making decisions about managing resource demands and supplied capacities, 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 reactiveness of an algorithm must be traded against the quality of a solution. Reactiveness is understood as the time between detecting an abnormality, for instance a sudden peak demand, and the final computation of a decision how the situation can be dealt with. Thus, reactiveness constitutes one parameter of the design space. Another parameter is the degree of distribution of the control system, ranging from centralized to completely decentralized. 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.

The next figure classifies six distributed algorithms in regard to solution quality versus reactiveness [21] and relates them to Integer Programming (blue):

- Agents in Overlay Networks,
- Ant Colony Algorithms [21], [23],
- Broadcast of Local Eligibility (BLE) [24],
- Local Random / Local Round Robin,
- Local Greedy Distribution.

Three time scales are considered: the design stage of an initial service placement, in longer periods reiterated as long-term adjustment process in the system; a mid-term period for periodic operational adjustments, and a shorter-term period for discharging sudden hot spots.

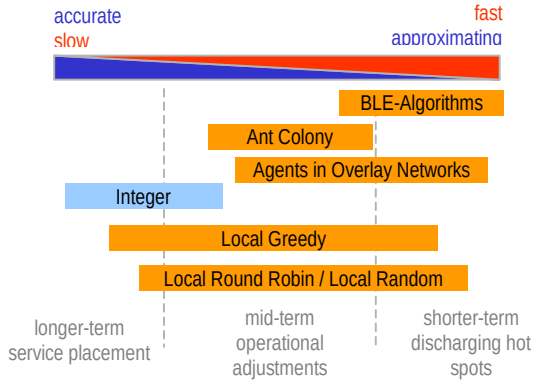

Figure 7: Tradeoffs of decision-making algorithms.

The last two algorithms are characterized by simplicity and statelessness. Pretty much like random or round robin scheduling, the load distribution algorithm pushes load from an overloaded node to a randomly or in a round robin fashion chosen neighbor that may absorb that load if it has the capacity, or it pushes the load further on to another node chosen in the same fashion. Once a place has been found where the load can be absorbed, the actual load migration is then initiated in the underlying system.

The advantage of this algorithm is its simplicity and statelessness (efforts to maintain states can be avoided).

The disadvantages are unpredictability and insufficient (random) convergence.

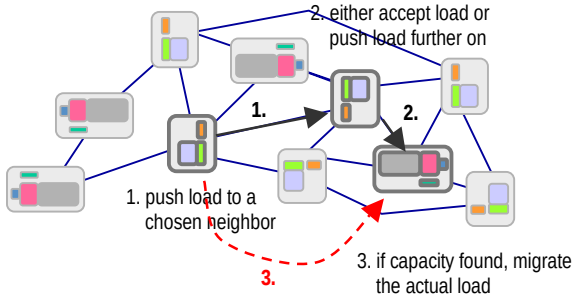

Figure 8: Load Distribution Algorithms.

The termination problem of the algorithm can be addressed by limiting the number of hops. Cycles cannot be avoided due to the statelessness of the algorithm.

Control Goals

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).

6 Adjusting Virtual Server Capacity in the UDC

Figure 9 shows a UDC with three available server configurations materialized as 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)
- Farm C: 400 TA/sec, normalized to 2.0.

(Transactions per second (TA/sec) has been chosen as an example to represent a capacity measure for a service. Different service types may require different measures.)

In Figure 9, all farms are passive. This means server configurations basically only consist of various regions in the storage system. No machine or network resources are assigned to any of these passive farms. The figure also shows a block called the UDC Utility Controller, a software component of the UDC management system that allows activating any configured farm in the UDC.

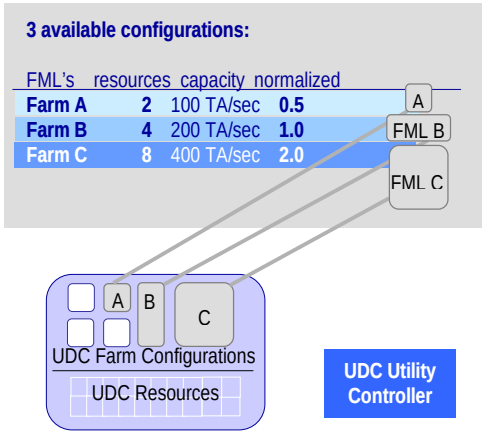

Figure 9: Three available server capacity configurations materialized as farms configured in the storage system.

Multiple farms may coexist in a UDC, not only as passive configurations in the UDC storage system, but also as activate farms with resources allocated and services running. A farm is allocated by sending a control instruction in form of a FML (Farm Markup Language) document to the UDC Utility Controller. The controller will verify the correctness of the document and will allocate needed resources (machines, private networks, storage) from its resource pools and virtually wire these components as described in the FML document. During this process, the UDC Utility Controller will attach one of the farm's configurations held in the storage system to the allocated machines.

Machines then will bootstrap from storage and launch all needed applications as configured as persistent states in the storage system. At the end, the activated farm will perform the service in the chosen configuration. Figure 10 shows the activation of farm B representing a capacity configuration of 1.0 for the service.

The respective FML document will be sent from the meta-system to the UDC Utility Controller.

An active farm can be terminated by sending a respective FML document to the UDC Utility Controller. After the farm has been shut down properly, and all the persistent service's states have been written out to the storage system, the UDC Utility Controller will release all allocated machine and network resources and return them to the pool.

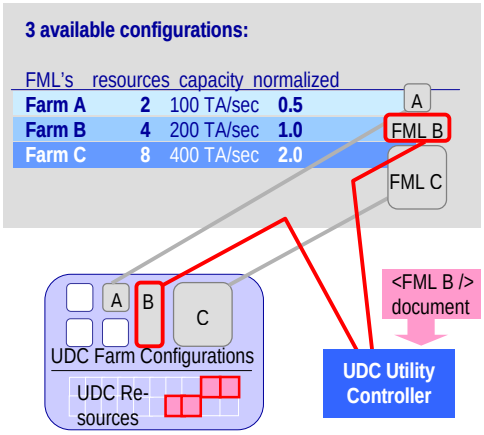

Figure 10: Activating farm B with capacity 1.0.

Obviously, after one farm representing a specific capacity configuration has been shut down, another farm representing a higher or lower capacity configuration of the same service (such as farm A or C in the example) may be activated. This leads to the effect that the capacity of a server and with it the capacity of the service can be adjusted. Figure 11 shows a capacity switch from 1.0 to 2.0, or from 200 TA/sec to 400 TA/sec, respectively.

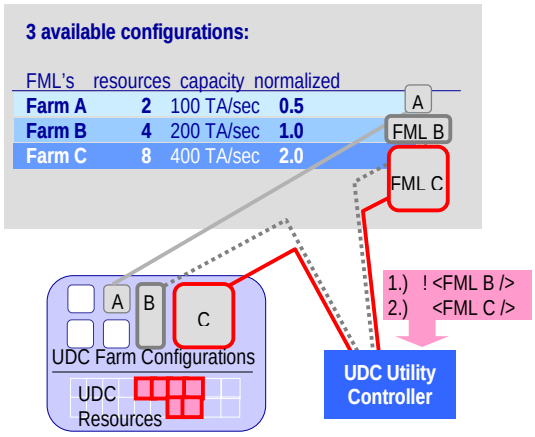

Figure 11: Switching capacity from 1.0 to 2.0.

Adjusting capacity can be achieved within one UDC by the shown procedure. The idea can be extended also between UDC by adjusting service capacities at different sites. It may even be considered that persistent service states residing in the storage system of one UDC may be migrated to a different UDC with the effect that the service may be activated at the new location as described.

The overall effect of this system allows building a control system with the decision-making layer in the meta-system based on distributed decision-making algorithms that provides service capacity as well as demand control in a large-scale service grid.

7 Capacity Control in a Homogeneous Grid Cluster

Implementing the demand/supply control system in a homogeneous cluster environment (we assume an Intel/Linux cluster) requires a different approach. Clusters do not have the full-fledged virtualization capabilities of a Utility Data Center. Storage and networking resources cannot be virtualized. The notion of a farm representing a hosting environment for a service does not exist.

In order to coordinate compute tasks from different customers, clusters are typically operated under a cluster management system such as openPSB. These management systems allow users to allocate machines from the cluster for a certain time. Using Grid technology, and specifically the Globus toolkit, a Globus Resource Allocation Manager (GRAM) can be implemented that performs the task of coordinating different customers' machine allocations. Users can be direct customers to a GRAM, or a hierarchy of GRAMs can be established such that higher-ordered GRAMs request machines from underlying GRAMs.

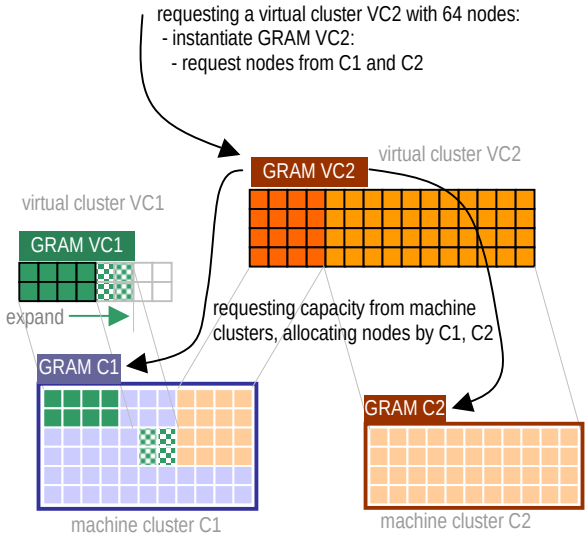

Figure 12: Establishing virtual clusters by hierarchically instantiating GRAMs.

This then can be seen as establishing a "virtual cluster" with machines that are under the control of the higher-ordered GRAMs. "Virtualization" here refers to allocating machines for customer use only. It does not include protection or isolation of application environments as in the UDC.

Figure 12 shows two virtual clusters established in two underlying machine clusters each of which under the control of one GRAM: GRAM C1 for machine cluster C1 and GRAM C2 for machine cluster C2. Both GRAMs accept requests for allocating machines from their pools.

The figure also shows two virtual clusters VC1 and VC2, under the control of GRAM VC1 and VC2, respectively. Virtual clusters consist of sets of machines allocated from underlying clusters, such as from only one cluster (as in case VC1) or from multiple clusters (case VC2). Machines allocated to virtual clusters can be managed by the virtual cluster GRAMs and can then be requested by customers in the same way as they would have been requested from underlying GRAMs.

Virtual clusters basically materialize in form of an instance of a GRAM that manages a set of machines represented by IP addresses and obtained form a variety of underlying clusters. Customers allocating machines from virtual clusters obtain these IP addresses and will be given the right to use these machines for an agreed time. GRAMs are acting as control points coordinating customer requests and adjusting virtual clusters capacities.

Machine allocations to virtual clusters can also be adjusted by adding machines to or removing machines from the virtual cluster. Machines are requested from or released to one of the underlying GRAMs. This provides the capability to dynamically adjust a virtual cluster's capacity in terms of the number of machines belonging to that cluster.

Though machine allocations from virtual clusters can be made transparent from actual machine locations in underlying environments, there is obviously a significant difference in connectivity between machines from different underlying machine clusters. This situation should be recognized and avoided by the GRAM. The GRAM is aware of machine locations it has requested from underlying clusters. It can then determine allocations that are physically within one cluster domain in order to satisfy a customer request. Virtual cluster GRAMs should determine sets of machines originating from the same machine cluster. If this is impossible, the GRAM may chose to allocate machines from different underlying clusters or may choose to expand its capacity by requesting more machines from underlying clusters. These machines may later be released again. These dynamic allocation processes and flexibility remain hidden to customers, which provide the value and purpose of such a resource allocation schema.

8 Related Work

IBM's Autonomic Computing vision [26] aims to provide self-managing systems. The intent is to create systems that respond to capacity demands and system failures 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** [27] is an ongoing effort under the Autonomic Computing vision for creating servers that automatically respond to unexpected capacity demands and system glitches. The goals are increased reliability, availability and serviceability while decreasing 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 [28], 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 [1] have been focused on distributed supercomputing where schedulers make decisions about where computational tasks will be assigned. Typically, schedulers are based on simple policies such as round-robin due to the lack of a feedback infrastructure reporting load conditions back into schedulers. More sophisticated approaches are in planning for grids [15]. However, it is currently not foreseeable whether Globus will evolve into such a comprehensive service demand-supply control system as discussed in this paper.

Other grid approaches such as the Sun's Grid Engine [7] basically only provide a resource sharing capability with the focus on making compute resources available to other users. Automated decision-making or even an integrated control system appears to be beyond the current capabilities of the Grid Engine.

References

[1] HP, *Utility Data Center*, http://www.hp.com/go/hpudc, http://www.hp.com/go/always-on, November 2001.

[2] The Global Grid Forum, http://www.gridforum.org/.

[3] Kotov, V.: *On Virtual Data Centers and Their Operating Environments*, HP Labs Technical Report[1], HPL-2001-44, March 2001.

[4] The Globus Toolkit, http://www.globus.org/toolkit.

[5] IBM, http://www.ibm.com/grid.

[6] Platform Inc., http://www.platform.com.

[7] Sun Microsystems, *The Sun Grid Engine*, http://wwws.sun.com/gridware.

[8] Krauter, K., Buyya, R., Maheswaran, M.:*A Taxonomy and Survey of Grid resource Management Systems,* Software-Practice and Experience, 32(2):135-164, 2002.

[9] Rolia, J., Singhal, S., Friedrich, R.: *Adaptive Data Centers*, Proceedings of SSGRR 2000 Computer and eBusiness Conference, L'Aquila, Italy, August 2000.

[10] The Globus Project, http://www.globus.org.

[11] The Berkeley Network of Workstations (NOW) Project, http://now.cs.berkeley.edu.

[12] SETI@home project, University of Berkeley, http://setiathome.ssl.berkeley.edu.

[13] 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.

[14] Kotov, V.: *Towards Service-Centric System Organization*, HP Labs Technical Report, HPL-2001-54, March 2001.

[15] 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, May 2002.

[16] W3C, *Web Services Description Language (WSDL)*, http://www.w3.org/TR/wsdl, March 2001.

[17] 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.

[18] HP Labs, *Serrano Project and the SmartFrog Language*, http://www.dcs.ed.ac.uk/home/dcspaul/wshop/SmartFrog.pdf.

[19] 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.

[20] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., *A Scalable Content-Addressable Network*, SIGCOMM 2001, San Diego, August 27-31, 2001.

[21] Andrzejak, A., Graupner, S., Kotov, V., Trinks, H.: *Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems*, HP Labs Technical Report, to be published in September 2002.

[22] Dorigo, M., Maniezzo, V., Colorni, 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.

[23] Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: *Ants for Load Balancing in Telecommunications Networks*, Adaptive Behavior 2:169-207, 1996.

[24] 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.

[1] HPL-TR are available: http://lib.hpl.hp.com/techpubs.

[25] 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.

[26] IBM, *Autonomic Computing*, Manifesto, http://www.research.ibm.com/autonomic/manifesto.

[27] IBM, *eLiza*, http://www-1.ibm.com/servers/eserver/introducing/eliza.

[28] IBM, and University of Berkeley, *Oceano Project*, http://www.research.ibm.com/oceanoproject.