AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications

Stefan Seltzsam, Daniel Gmach, Stefan Krompass, Alfons Kemper
Technische Universität München
D-85748 Garching bei München, Germany
firstname.lastname@in.tum.de

Abstract

Future database application systems will be designed as Service Oriented Architectures (SOAs) like SAP's NetWeaver instead of monolithic software systems such as SAP's R/3. The decomposition in finer-grained services allows the usage of hardware clusters and a flexible service-to-server allocation but also increases the complexity of administration. Thus, new administration techniques like our self-organizing infrastructure that we developed in cooperation with the SAP Adaptive Computing Infrastructure (ACI) group are necessary. For our purpose the available hardware is virtualized, pooled, and monitored. A fuzzy logic based controller module supervises all services running on the hardware platform and remedies exceptional situations automatically. With this self-organizing infrastructure we reduce the necessary hardware and administration overhead and, thus, lower the total cost of ownership (TCO).

We used our prototype implementation, called AutoGlobe, for SAP-internal tests and we performed comprehensive simulation studies to demonstrate the effectiveness of our proposed concept.

1. Introduction

Database systems are not used as stand-alone software systems. Rather, they are accessed via application services. Therefore, automatic administration concepts for database applications cannot only concentrate on the database alone, but have to comprise the entire service architecture, of which the database is only one–albeit very important–component. There exists a large research and development effort in automatic administration concepts for databases [3]. Concerning the administration of Service Oriented Architectures (SOA) for database applications, such as SAP's new technology platform NetWeaver [20], three objectives can be identified: Low administration effort, low total cost of ownership (TCO), and a high degree of service performance, i.e., ensuring that a predefined number of clients (customers and employees) can be handled by the infrastructure. To achieve the last objective, a highly scaled hardware architecture can be used. Obviously, this counteracts the second objective of low TCO. To balance these three objectives, services have to be assigned to the available servers in an optimized way. In cooperation with the SAP Adaptive Computing Infrastructure (ACI) group we develop such capabilities. Figure 1 shows the layering of the SAP NetWeaver technology stack. NetWeaver itself is an integration and application platform, i.e., a platform for an SOA. On top of NetWeaver, the mySAP business suite and other business applications are realized. The self-management capabilities are provided by the adaptive computing infrastructure layer below. For this layer, we are developing new concepts for self-organizing infrastructures.

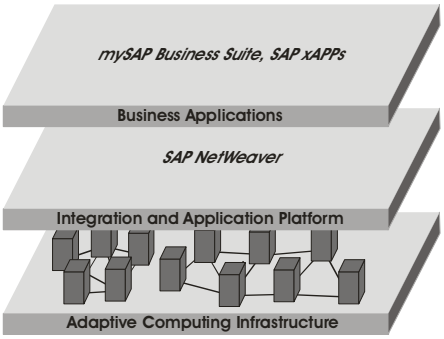

Figure 1. SAP NetWeaver Architecture

The term "self-organizing infrastructure" refers to self-management capabilities including self-configuration, self-optimization, self-healing, and self-protection. While existing implementations of these concepts are oftentimes vendor-specific, we focus on a generic approach for heterogeneous hardware and software landscapes. Our approach analyzes the current load induced by service instances as well as their resource consumption and reacts on exceptional situations. For this dynamic adaption, services are supervised by a fuzzy logic based controller. For example, in case overloaded service instances are detected, the situation is remedied by either starting new service instances or by moving instances to more powerful servers. With this approach of automatic runtime adaptions, AutoGlobe reduces administrative overhead and achieves a reduction of TCO as either more users can be handled with the existing hardware or because less hardware is required initially.

The allocation decisions depend on the capabilities and constraints of the application services and the hardware environment. These are described using a declarative XML language. Among other constraints the maximum and minimum number of instances of a service can be defined, the performance of hosts can be related to each other, and the rules for the fuzzy controller can be specified. One basic aspect of AutoGlobe is that services are virtualized, i.e., they are not running on a fixed server. Thereby, available resources are shared between all services as appropriate for a particular situation.

The remainder of the paper is organized as follows: In Section 2 the architecture of AutoGlobe is presented, which is based on our ServiceGlobe platform for location-independent execution of Web services. The foundations of fuzzy controllers are described in Section 3. A detailed description of the fuzzy controller of AutoGlobe is presented in Section 4. Then, simulation study results follow in Section 5. Finally, in Section 6 we present related work prior to a conclusion and discussion of future research in Section 7.

2. Self-Organizing Infrastructure

AutoGlobe is based on our distributed and open Web service platform ServiceGlobe [15, 16]. ServiceGlobe is fully implemented in Java Version 2 and is based on standards like XML, SOAP, UDDI, and WSDL. The key innovation of ServiceGlobe is its support for mobile code, i.e., services can be distributed and instantiated during runtime on demand at arbitrary servers participating in the ServiceGlobe federation. Those servers are called service hosts. Of course, ServiceGlobe offers all the standard functionality of a service platform like a transaction system and a security system [23]. The goal of the AutoGlobe project is to add an active control component for automated service and server management to ServiceGlobe.

Services managed by the AutoGlobe platform are virtualized by the use of service IP addresses, i.e., every service has its own IP address assigned. This IP address is bound to the physical network interface card (NIC) of the host running the service. Thus, if a service is moved from one host to another, the virtual IP address is unbound from the NIC of the old host running the service and afterwards bound to the NIC of the target host. Consequently, services are decoupled from servers. This service virtualization is a vital requirement for AutoGlobe.

The benefits of AutoGlobe can be experienced best on a flexible infrastructure like a blade server environment that we investigated, though not being restricted to this kind of hardware infrastructure. Blade server are relatively cheap compared to traditional mainframe hardware and the processing power can easily be scaled to the respective demand by varying the number of blades on the fly. Blade servers normally store their data using a storage area network (SAN) or a network attached storage (NAS). Thus, CPU power and storage capacity can be scaled independently and services can be executed on any blade because services can access their persistent data regardless of the blade on which they are running.

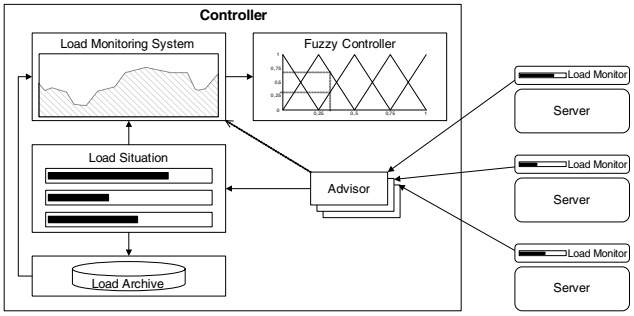

Figure 2. Architecture of the Controller Framework

The architecture of our controller framework is shown in Figure 2. *Load monitors* run on every server and report their measurements to *advisors*. These measurements are used to maintain an up-to-date local view of the load situation of the system. Imminent overload situations are reported to a *load monitoring system* which observes the load changes for a while and triggers a *fuzzy controller* in case of a real overload situation. This fuzzy controller initiates actions to prevent critical load situations. For example, if a CPU overload on a service host is detected, the controller can move services from this overloaded host to currently idle hosts. The controller also reacts upon idle situations. As the proceeding is quite analogous we will focus on overload situations in the following. Failure situations like a program crash are remedied for example with a restart. A *load archive* stores aggregated historic load data. These modules are described in more detail subsequently.

Load Monitors and Advisor Modules. Every server and every service is monitored by a *load monitor* service, which is a specialized service for resource monitoring of service hosts and of resource usage of services, respectively.[1]

Load Monitoring System. In real systems short load peaks are quite common. Immediate reaction on these peaks could lead to an unsettled and instable system. Thus, if load values exceed a tunable threshold, the advisor passes the load data to the *load monitoring system* module for further observation. Then, the load data is observed for a tunable period of time (*watchTime*). If the average load during the watch time is above a given threshold, a real overload situation is detected and the fuzzy controller module is triggered.

Fuzzy Controller. The fuzzy controller identifies appropriate actions to remedy overload situations. For this purpose,

[1] Figure 2 only shows the load monitors and advisors responsible for the servers. For simplicity of the illustration, services running on the servers and their load monitors and advisors are omitted.

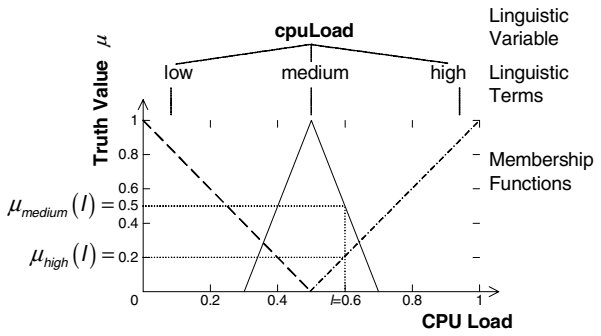

Figure 3. Linguistic Variable cpuLoad

it is initialized with information about the current load situation of the affected services and servers. After that, the fuzzy controller calculates the applicability of all actions. If required, e.g., for a move action, the fuzzy controller afterwards calculates the score of all suitable target service hosts. The action with the highest applicability is executed and the host with the highest score is selected as target host of the action, see Section 4.

Load Archive. The load archive stores a persistent aggregated view of historic load data. This data is used to calculate the average load of services during their watchTime and to initialize all resource variables of the fuzzy controller.

3. Fuzzy Controller Basics

In general, fuzzy controllers are special expert systems based on *fuzzy logic* [17]. Fuzzy controllers are used in control problems for which it is difficult or even impossible to construct precise mathematical models. In the area of a self-organizing infrastructure, these difficulties stem from inherent nonlinearities, the time-varying nature of the services to be controlled, and the complexity of the heterogeneous system. Contrary to classical controllers, fuzzy controllers are capable of utilizing knowledge of an experienced human operator as an alternative to a precise model. This knowledge is expressed using intuitive linguistic descriptions of the manner of control.

Fuzzy logic is the theory of *fuzzy sets* devised by Zadeh [26]. The membership grade of elements of fuzzy sets ranges from 0 to 1 and is defined by a membership function. Let X be an ordinary (i.e., crisp) set, then

$$A = \{(x, \mu_A(x)) \mid x \in X\} \quad \text{with} \quad \mu_A : X \rightarrow [0,1]$$

is a fuzzy set in X. The membership function μ_A maps elements of X into real numbers in $[0,1]$. A larger (truth) value denotes a higher membership grade.

Linguistic variables are variables whose states are fuzzy sets. These sets represent *linguistic terms*, such as low, medium, or high. A linguistic variable is characterized by its name, a set of linguistic terms, and a membership function for each linguistic term. An example for the linguistic variable *cpuLoad* is shown in Figure 3. The figure shows the three linguistic terms *low*, *medium*, and *high* with their assigned trapezoid membership functions.

Figure 4 shows the general architecture of a fuzzy controller according to [17]. The controller works by repeating a cycle of three steps. First, measurements are taken of all variables representing relevant conditions of the controlled infrastructure. These measurements are converted into appropriate fuzzy sets (input variables) in the fuzzification step. After that, these fuzzified values are used by the inference engine to evaluate the fuzzy rule base. At last, the resulting fuzzy sets (output variables) are converted into a vector of crisp values during the defuzzification step. The defuzzified values represent the actions taken by the fuzzy controller to control the infrastructure and the target hosts of the actions, respectively. We will now explain the fuzzy controller mechanisms by way of an example from the automatic administration context in more detail.

During the fuzzification phase, the crisp values of the measurements (e.g., CPU load of a host) are mapped onto the corresponding linguistic input variables (e.g., cpuLoad) by calculating membership rates using the membership functions of the linguistic variables. For example, according to Figure 3, a host having a measured CPU load $l = 0.6$ (60%) has 0.5 medium and 0.2 high cpuLoad.

In the inference phase, the fuzzy rule base is evaluated using the fuzzified measurements. The form of the rules is exemplified by the two sample rules[2]

```
IF cpuLoad IS high AND
    (performanceIndex IS low OR
     performanceIndex IS medium)
THEN scaleUp IS applicable

IF cpuLoad IS high AND performanceIndex IS high
THEN scaleOut IS applicable
```

with `cpuLoad` and `performanceIndex` (specifying the relative performance of a server) being the input variables and `scaleUp` and `scaleOut` being the output variables. Typical fuzzy controllers have dozens of rules. Currently, our AutoGlobe fuzzy controller currently comprises about 40 rules. The first sample rule states that it is reasonable to move a service to a more powerful host (scale-up) if the host running the service has a high load and a low or medium performance index (the higher the performance index of a host, the more powerful it is). The second rule states that it is reasonable to start an additional service instance, if the host running the service is highly loaded despite it being very powerful.

Conjunctions of truth values in the antecedent of a rule are evaluated using the minimum function. Analogously, disjunctions are evaluated using the maximum function. Given a performance index i and a CPU load of $l = 0.9$, the membership grades for the linguistic variable cpuLoad

[2]These simple rules are only used to explain the inference phase. The rules used in our self-organizing system are generally more complex.

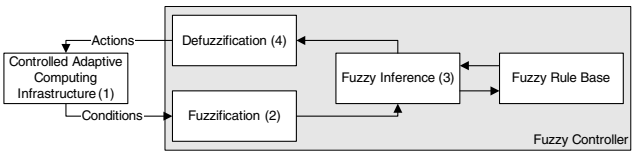

Figure 4. Architecture of a Fuzzy Controller

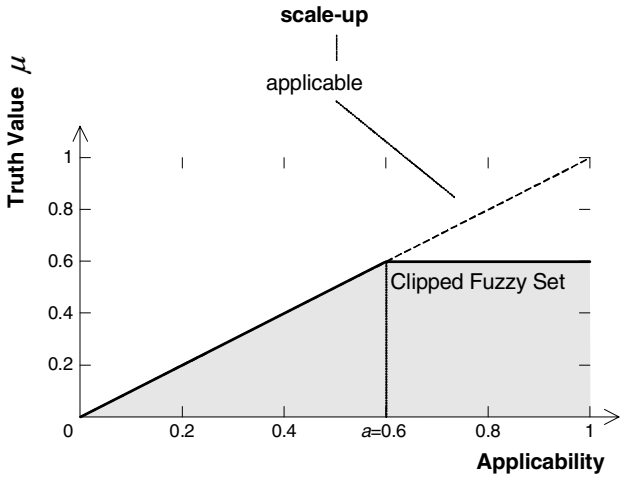

Figure 5. Max-Min Inference Result

are $\mu_{low}(l) = 0$, $\mu_{medium}(l) = 0$ and $\mu_{high}(l) = 0.8$. We assume for this example that the membership grades for the linguistic variable performanceIndex are $\mu_{low}(i) = 0$, $\mu_{medium}(i) = 0.6$ and $\mu_{high}(i) = 0.3$. Thus, the truth value of the antecedent of the first rule evaluates to $min(0.8, max(0, 0.6)) = 0.6$ and the truth value of the antecedent of the second rule evaluates to $min(0.8,\ 0.3) = 0.3$.

In classical logic, the consequent of an implication is true if the antecedent evaluates to true. For fuzzy inference, there are several different inference functions proposed in the literature. We use the popular max-min inference function. Using this function, the fuzzy set specified in the consequent of a rule (e.g., applicable) is clipped off at a height corresponding to the rule's antecedent degree of truth. After rule evaluation, all fuzzy sets referring to the same output variable are combined using the fuzzy union operation:

$$\mu_{A \cup B}(x) = \max\left(\mu_A\left(x\right), \mu_B\left(x\right)\right) \quad \text{for all } x \in X$$

The resulting combined fuzzy set is the result of the inference step. Figure 5 shows the result of the inference for the linguistic output variable scaleUp.

During the defuzzification phase, a sharp output value is calculated from the fuzzy set that results from the inference phase. There are several defuzzification methods described in the literature. We use a *maximum* method, such that the result is determined as the leftmost of all values at which the maximum truth value occurs. Regarding our example shown in Figure 5, the crisp value for the action scale-up is 0.6, i.e., the action is applicable to a degree of 0.6. The linguistic variable scaleOut is defined analogously. Thus, the action scale-out is applicable to a degree of 0.3. Therefore, the controller will favor the scale-up action for execution.

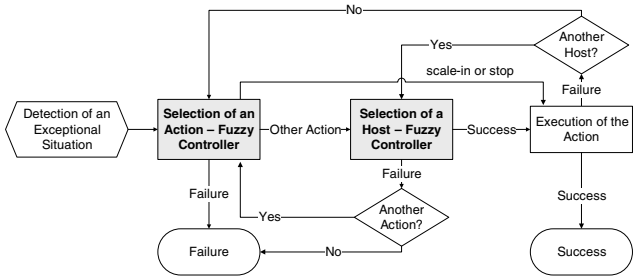

Figure 6. Interaction of Fuzzy Controllers

Variable	Description
cpuLoad	CPU load of a server
memLoad	memory load of a server
performanceIndex	performance index of the server
instanceLoad	load of a service instance
serviceLoad	average load of all instances of a service
instancesOnServer	number of services running on a server
instancesOfService	number of instances of a service

Table 1. Input Variables for Action-Selection

4. Fuzzy Controller for Load Balancing

The fuzzy controller module in AutoGlobe consists of two separate fuzzy controllers. The first one reacts on exceptional situations and determines an appropriate action. If the selected action requires a target host, e.g., scale-out, a second fuzzy controller is triggered to determine a suitable service host. Figure 6 shows the interaction of the two fuzzy controllers *selection of an action* and *selection of a host*. After a rearrangement has taken place, the involved services and servers are protected for a certain time, i.e., they are excluded from further actions. This protection mode prevents the system from oscillation, e.g., moving services back and forth.

4.1. Action-Selection Process

First, the input variables of the fuzzy controller are initialized. Table 1 shows the input variables of our controller. All variables of the fuzzy controller regarding CPU or memory load are set to the arithmetic means of the load values during the service specific watchTime. The other variables are initialized using the current measurements or using available meta data, e.g., for the performanceIndex.

The fuzzy controller distinguishes between exceptional situations induced by a service and exceptional situations induced by a server (see Figure 7). If a service triggered the controller, it decides on the basis of information from the considered service, the service instance, and the server on which it is executed. Other services running on the considered host are not considered. If a server triggered the fuzzy controller, it takes the information of all services running on the considered host into account.

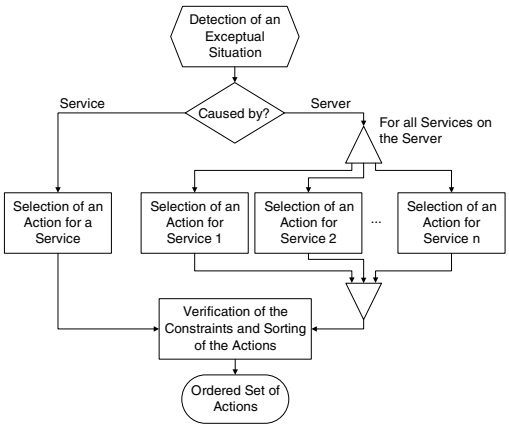

Figure 7. Flowchart of Action-Selection

Variable	Description
start	starting of a service
stop	stopping of a service
scaleIn	stopping of a service instance
scaleOut	starting of a service instance
scaleUp	movement of a service instance to a more powerful host
scaleDown	movement of a service instance to a less powerful host
move	movement of a service instance to an equivalently powerful host
increase-priority	increasing the priority of a service
reduce-priority	reducing the priority of a service

Table 2. Output Variables for Action-Selection

Since the action-selection process depends on the specific situation, our controller is able to handle dedicated rule bases for different exceptional situations (triggers). We distinguish between four different triggers: *serviceOverloaded*, *serviceIdle*, *serverOverloaded*, and *serverIdle*. Further, our controller facilitates dynamic adaptations. For example, an administrator can add service-specific rule bases for mission critical services, e.g., to favor powerful servers for these services. A rule base comprises dozens of rules each consisting of an antecedent and a consequent.

The fuzzy controller evaluates the appropriate rule base and calculates crisp values for the output variables. Table 2 shows the output variables. These output variables represent the actions executed by the controller to control the infrastructure.

The fuzzy controller only considers actions that do not violate any given constraint, e.g., a traditional SAP database service does not support a scale-out. Thus, the action scale-out is not possible for such a service. These constraints are defined using a declarative XML language. The result of the fuzzy controller is a list of actions along with their ratings between 0% and 100%. These ratings determine the applicability of the actions in the current situation. In case that a server triggered the controller, we execute the fuzzy controller for each service running on the server and subsequently collect the possible actions of all services.

Variable	Description
cpuLoad	CPU load on the server as average load over all CPUs
memLoad	memory load on the server
instancesOnServer	number of instances on the server
performanceIndex	performance index of the server
numberOfCpus	number of CPUs of the server
cpuClock	clock speed of the CPUs of the server
cpuCache	cache size of the CPUs of the server
memory	memory size of the server
swapSpace	size of the available swap space
tempSpace	size of the available temporary disk space

Table 3. Input Variables for Server-Selection

Afterwards, the actions are sorted by their applicability in descending order. Actions whose applicability value is lower than an administrator-controlled minimum threshold are discarded. The first action of the list is selected and verified once more. This is necessary, because the fuzzy controller is able to handle several exceptional situations concurrently. Thus, for example, if now the maximum number of instances of a service are running, the controller cannot start another one and, therefore, cannot perform a scale-out.

4.2. Server-Selection Process

In the case of a scale-out, scale-up, scale-down, move, or start, an appropriate target server where the action should take place must be chosen. The selection of a server proceeds analogously to the selection of an action. First, a list of all possible servers is determined. Initially, these are all servers on which an instance of the service can be started and that are not in protection mode. For each server the fuzzy controller is executed with the input variables initialized to the current values. Table 3 shows the input variables for the server-selection.

Since the server-selection process depends on the specific action, our controller is able to handle different rule bases for different actions. With these rules we determine how proper a server is for the problem. In the defuzzification phase, the controller calculates a crisp value for every possible host and selects the most applicable server.

4.3. Execution of the Controller's Decision

The controller can operate in two different modes: In the *automatic mode*, the actions are logged and then executed. In *semi-automatic mode*, the human administrator is contacted to confirm the action before execution. If there are no possible hosts and actions with a sufficient applicability, the controller requests human interaction by alerting the system administrator.

For this purpose, our controller offers a graphical controller console which displays the monitored state of the system. Using this console the administrator can manually execute the actions that are normally triggered by the fuzzy controller. Figure 8 shows the GUI of the controller

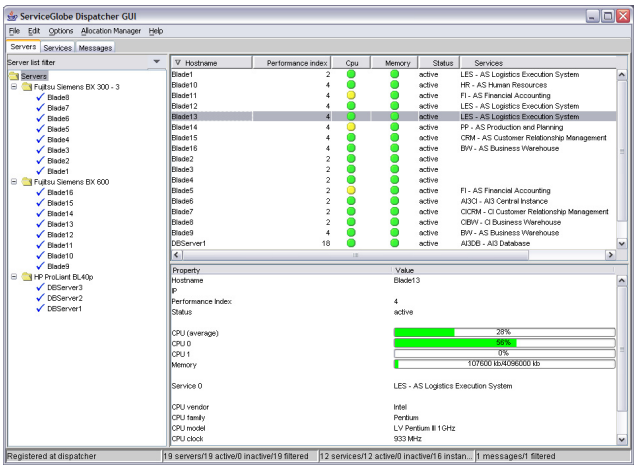

Figure 8. Administrator Controller GUI

console. There are three different views: the server view displays information about the controlled servers, the service view is analogously displaying information about the controlled services and the message view lists administrative messages and notifications. The presented screenshot shows the server view. The panel on the left-hand side shows a list of all controlled servers grouped by category. The upper right-hand panel displays overview information about all servers belonging to the selected category. Finally, the lower right-hand panel displays detailed information about the selected server.

5. Simulation Studies

We performed comprehensive simulation studies using our prototype implementation AutoGlobe to assess the effectiveness of our autonomic computing infrastructure. They have been conducted using a simulation environment that models a realistic SAP installation.

5.1. Description of the Simulation Environment

The simulation environment models a realistic SAP system with the corresponding hardware. The simulated services and servers are described using our declarative XML language, just like real existing services and servers. Figure 9 shows the architecture of our simulated SAP installation, which is—like real SAP systems—divided into three layers: the presentation layer, the application server layer, and the database layer. End-users communicate with the SAP installation using clients in the presentation layer. The end-users' clients themselves do not affect the system, thus we only simulate the number of users connected to services of our simulated SAP installation. Our installation comprises three subsystems in the application and database layer: *Enterprise Resource Planning (ERP)*, *Customer Relationship Management (CRM)*, and *Business Warehouse (BW)*, each running its own dedicated database and central instance. The central instance application servers (CI) are

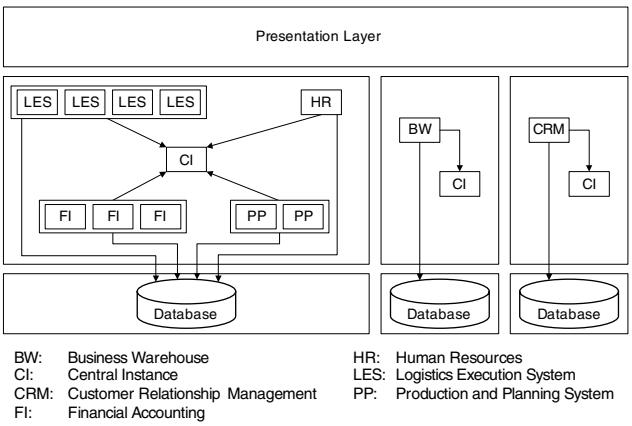

Figure 9. Example of an SAP Environment

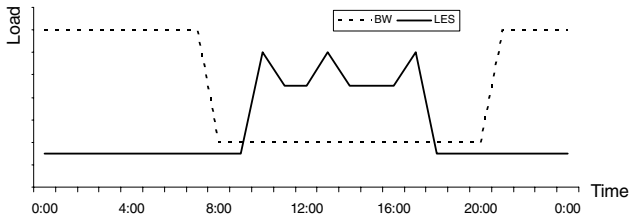

Figure 10. Load Curves of LES and BW

responsible for the global lock management of their particular subsystem. The other application servers (BW, CRM, FI, HR, LES, PP) execute the application logic, i.e., process user requests. Our controller supervises these application servers, databases, and central instances.

In a real system, there is a lot of communication between the individual services. In our simulation environment, we neglect communication costs, because we assume a local high-bandwidth network connection. This is realistic in blade server environments, which are normally equipped with, e.g., Gigabit Ethernet or Infiniband.

Our system simulates a varying number of users which are generating user requests. As observed in existing SAP installations, the course of a request is simulated as follows. First, a request increases the load of the affected service host for a short period. Before handling the request in the database, the lock management of the central instance (CI) is requested. Finally, the database sends the answer back to the application server. Since the load caused by a single request depends on the specific service (e.g., an FI request produces lower load than a BW request) our simulation system uses service-specific parameters to simulate the impact of requests. In addition to the load produced by user requests, every application server itself induces a basic load.

The load curves generated by simulated services follow predetermined patterns that can be observed in many companies running SAP software. Figure 10 shows example load curves for an LES and for a BW application server over one day. At eight o'clock, when the employees start

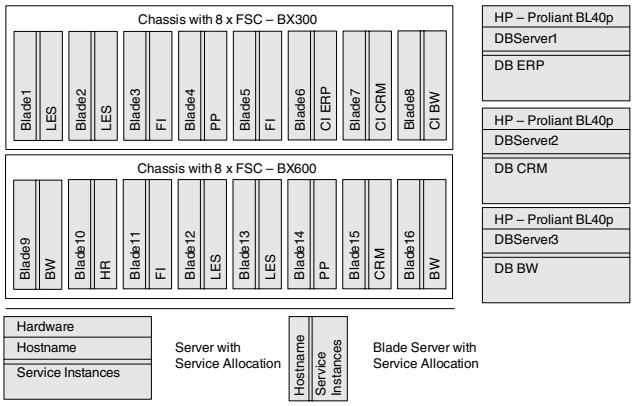

Figure 11. Simulated Hardware and Initial Allocation

to work, the number of requests sent to the LES application servers increases. There are three peaks, one in the morning, one before midday and one before the employees leave. The load curve of a BW application server is different. During the night, several heavy-load batch jobs are processed. During the day, only few user requests have to be processed based on the aggregated data.

We assume a hardware environment that is scaled for peak load as it is quite common in today's computing centers. A standard single processor blade in our simulation (performance index = 1) is dimensioned to handle at most 150 users of one service. The CPU load of the blades is between 60% and 80% during main activity in order to retain reserves for unpredictable load bursts. Figure 11 shows the simulated hardware and the initial static allocation of the services. The simulated servers are[3]:

- 8 FSC-BX300 blades with one Intel Pentium III 933 MHz processor and 2 GB main memory each (performance index = 1).
- 8 FSC-BX600 blades with two Intel Pentium III 933 MHz processors and 4 GB main memory each (performance index = 2).
- 3 HP-Proliant BL40p servers with four Intel Xeon MP 2,8 GHz processors and 12 GB main memory each (performance index = 9).

Table 4 shows the maximum number of users per service and the number of instances that are started initially. These numbers are reasonable for a medium-sized company running SAP, e.g., most departments use the LES application servers while only the staff department uses the HR application servers.

Every simulation starts with the same reasonable initial allocation of the services shown in Figure 11. We run different simulation series and always increase the number of users by 5% until the system becomes overloaded. The BW is an exception, because it processes batch jobs instead of interactive requests. Thus, we increase the load per batch job by 5% and leave the number of jobs constant.

[3]The performance index values stated are based on estimations and do not necessarily reflect the real performance of the servers.

Service	Number of Users	Number of Instances
FI	600	3
LES	900	4
PP	450	2
HR	300	1
CRM	300	1
BW	60	2

Table 4. Initial Number of Users

Service	Conditions	Possible Actions
database ERP	exclusive	–
	min. perf. index 5	
database BW, CRM	min. perf. index 5	–
central instances	–	–
application server	min. 2 FI instances	scale-in,
	min. 2 LES instances	scale-Out

Table 5. Services in the CM Scenario

We perform simulation series using three different scenarios. In the *static* scenario, a computing environment with all services being static is simulated. Today, this is the standard environment used in most computing centers. In the *constrained mobility* (CM) scenario, we simulate a SAP environment supervised by our controller, with some but not all services supporting move, scale-in, and scale-out actions (Table 5 gives an overview). The exclusive condition states that no other service may be executed on the host. The *minimum performance index* defines the minimum performance requirements of a service. The *minimum instances* condition defines the minimum number of instances of a service allowed. In this scenario, all databases and central instances are static. Application servers support scale-in and scale-out. After a scale-out, the system does not dynamically redistribute the users, i.e., users are logged in at one service instance during their complete session. We simulate a fluctuation of the users, i.e., users infrequently log themselves off of the application server they are connected to and reconnect to the currently least-loaded server. This behavior can be observed in real systems, too. In the *full mobility* (FM) scenario, we simulate a system where the BW database can be distributed across several servers. The central instances and the other application servers can be moved from one host to another, see Table 6 for details. Furthermore, if a new instance of a service is started, the users are equally redistributed across all instances.

Today, the movement of services is not yet supported by all services, because services must explicitly assist the movement. This requires that the service is able to store its internal state before it is stopped and that the newly started instance can restore the old state. Furthermore, it must be guaranteed that the users are reconnected automatically to the newly instantiated service instance. In the future, we expect, that more services support dynamic relocation and,

Service	Conditions	Possible Actions
database ERP	exclusive min. perf. index 5	–
database CRM	min. perf. index 5	–
database BW	min. perf. index 5	scale-in, scale-out,
central instances	–	scale-up, scale-down move
application server	min. 2 FI instances min. 2 LES instances	scale-up, scale-down, scale-in, scale-out, move

Table 6. Services in the FM Scenario

thus, we consider them in the full mobility scenario.

To prevent the system from reacting too late, we set the threshold value for a CPU overload to 70%, i.e., if a server has more than 70% CPU load it is considered overloaded. In this case, the controller monitors the load values of the service for 10 minutes (watchTime) in order to prevent the system from over-reacting upon short load bursts. After an action took place, the affected services and servers are protected for 30 minutes. The threshold value for an idle situation depends on the server and is 12.5% divided by the performance index of the server. An idle situation is recognized after a watchTime of 20 minutes.

All simulation runs have been carried out in 40-fold acceleration and are simulating a system for 80 hours. The shown time intervals correspond to simulated time.

5.2. Results of the Simulation Studies

Figures 12, 13, and 14 show simulation results with the number of users increased by 15% compared to the user numbers shown in Table 4. These results demonstrate how the SAP installation handles an increasing number of users. The figures show the load curves of all servers and the average load of the whole system, indicated by the thick line.

In the static scenario, several servers become overloaded, i.e., have a CPU load of more than 80% for a long time, at regular intervals, thus a non self-organizing infrastructure cannot handle this situation satisfactorily. If a host running an interactive service is overloaded, the service requires more time to process the requests and, therefore, delays new requests. Thus, users cannot perform all their requests in a given period, e.g., a working day, and requests will be delayed till next day.

The situation already improves in the constrained mobility scenario. The controller reacts on arising overload situations by automatically starting additional instances of services. Because the users are not dynamically redistributed after a scale-out has taken place, the original servers remain quite loaded. Due to user fluctuations, the load of the initially overloaded services slowly decreases. Altogether, the overload situations are on average shorter than in the static scenario, but due to the restrictions of the static user distribution, the overload situations cannot be prevented completely.

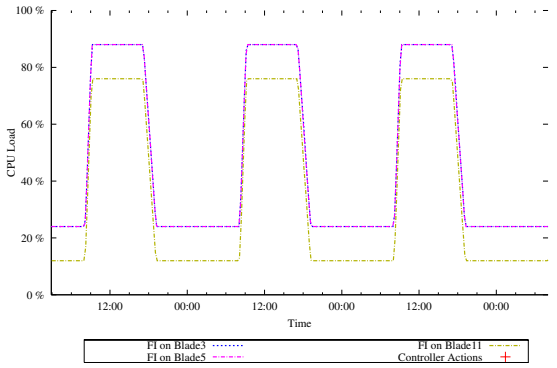

Figure 15. CPU Load of the FI (Static)

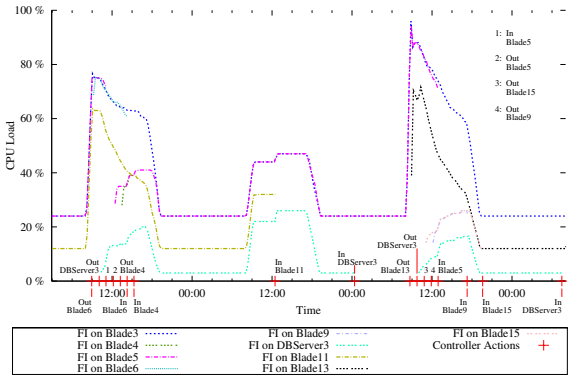

Figure 16. CPU Load of the FI (CM)

In the full mobility scenario, the results are significantly improved. Idle resources are efficiently used to relieve the load on heavily used resources. Thus, the utilization of the hardware is well-balanced. Due to the dynamic redistribution of users across all service instances, the effects of controller actions are observable almost instantly. Another advantage of the full mobility scenario is that the controller can react more flexibly on overload situations. The remaining short overload peaks at the beginning stem from the watchTime. If the instances of a service become overloaded, the controller monitors the instances for 10 minutes before starting a new instance. Therefore, for a short time, the existing instances stay overloaded. After the first day, there are normally more instances of every application server running than in the beginning. Thus, if the controller does not stop too many instances, the load can be distributed across a sufficient number of instances, and overload situations can be avoided.

In order to demonstrate the behavior of our controller in more detail, we present the FI application servers' load curves of the above described simulations.

Figure 15 shows the load curve of the FI application servers in the static scenario. There are three instances running on Blade3, Blade5, and Blade11. As services are static, the controller cannot remedy the overload situations. Thus, the service instances running on the less powerful

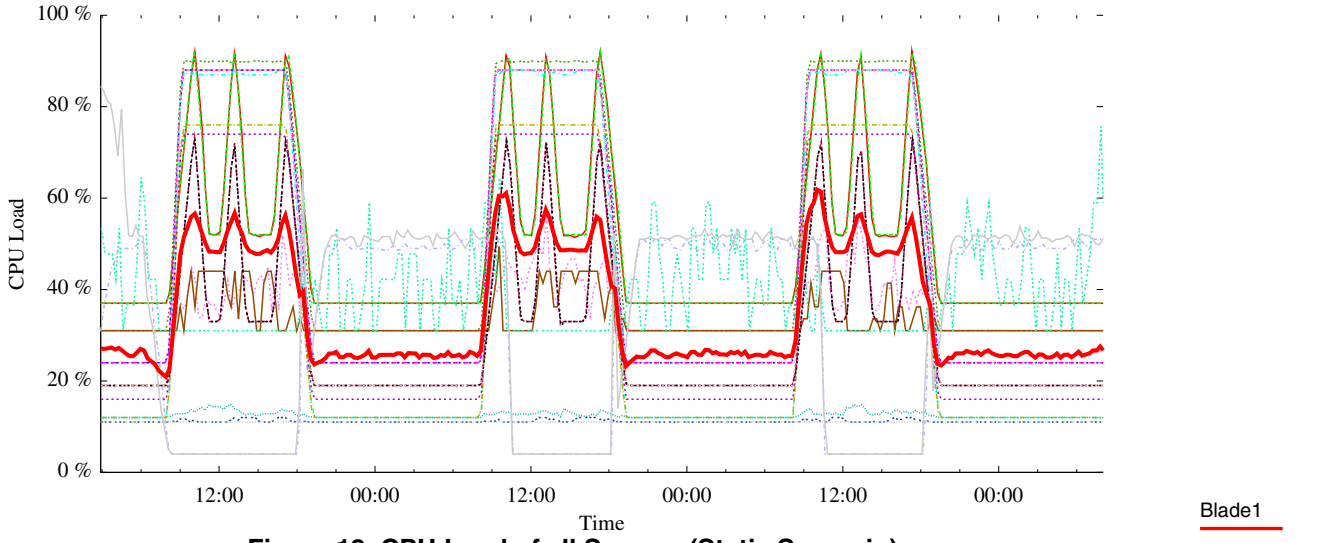

Figure 12. CPU Load of all Servers (Static Scenario)

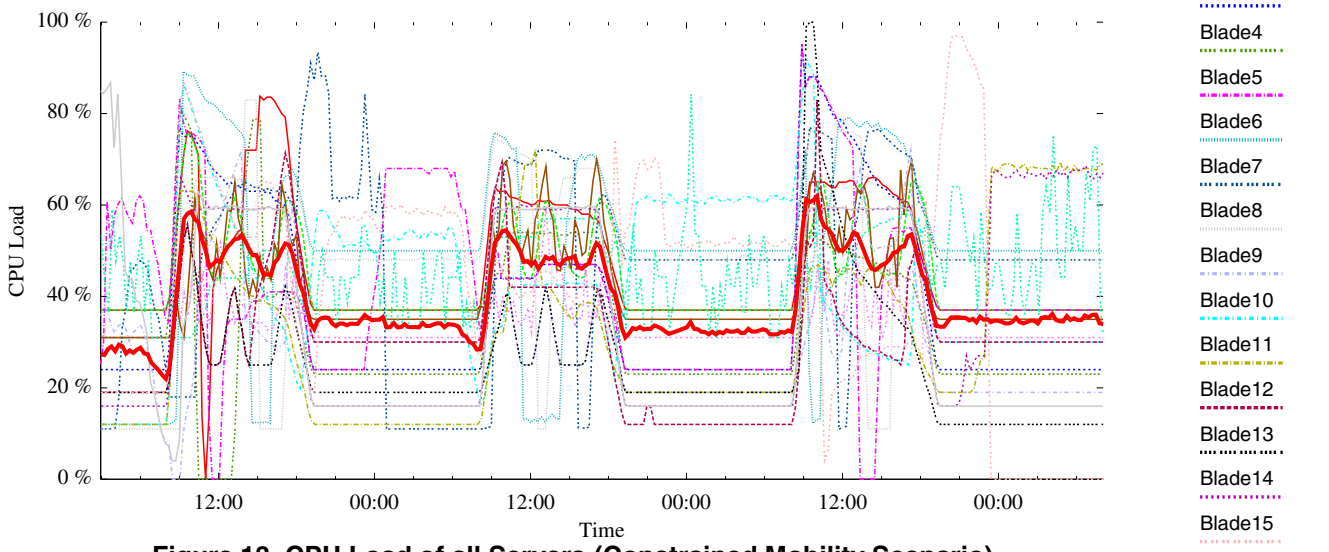

Figure 13. CPU Load of all Servers (Constrained Mobility Scenario)

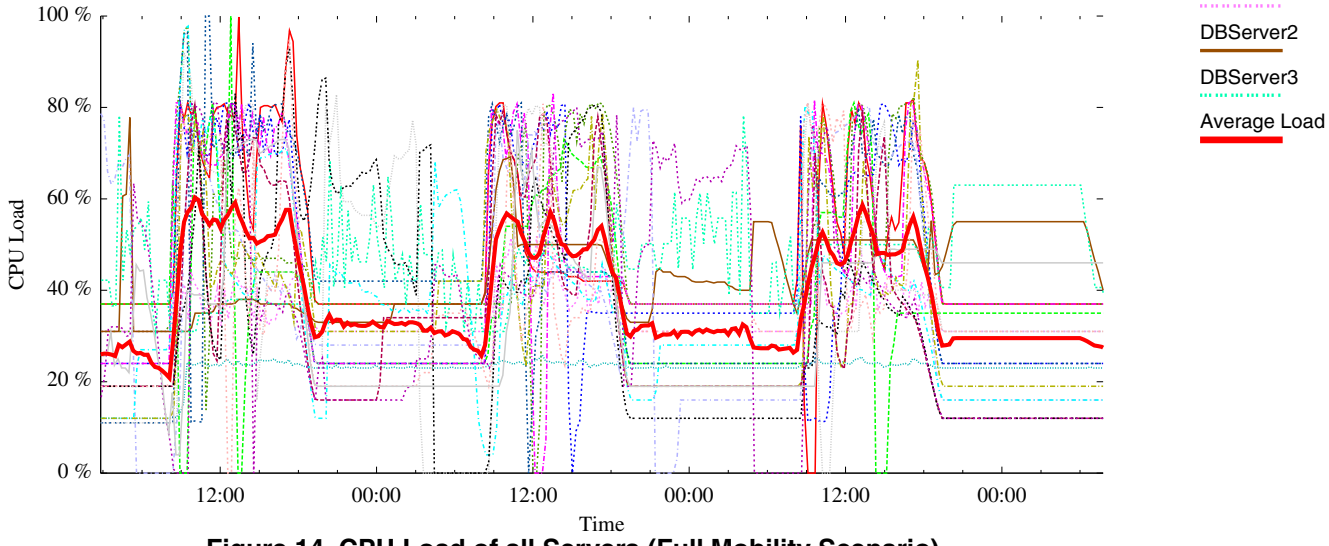

Figure 14. CPU Load of all Servers (Full Mobility Scenario)

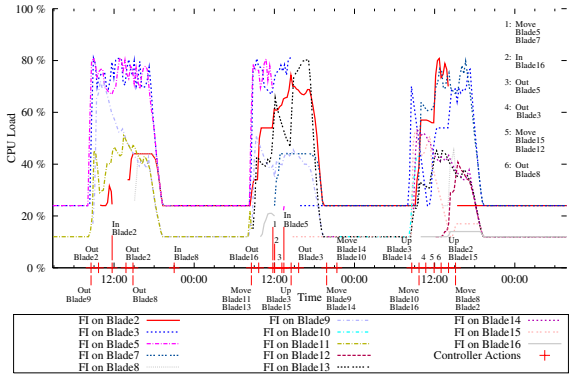

Figure 17. CPU Load of the FI (FM)

blades become overloaded periodically. If a service or a server is overloaded, it can no longer be used in a reasonable way because the processing of mission critical OLTP-requests is slowed down.

Figure 16 shows load curves in the constrained mobility scenario. When the employees begin to work, the instances on Blade3 and Blade5 become overloaded. The controller's reaction is to start an additional instance on Blade6 ("Out Blade6"). Since users are not redistributed dynamically, the load of Blade3 and Blade5 only decreases slowly. These two hosts are still overloaded after the protection time, thus the controller starts another instance on DBServer3 ("Out DBServer3"). Because these actions do not remedy the overload on Blade5 fast enough, the controller decides to stop the instance running on Blade5 ("In Blade5") to protect the host from a continuous overload situation. This FI instance is started again ("Out Blade5") after a short period of time due to an overload situation on Blade6. Another FI instance is started on Blade4 ("Out Blade4"). Further on, the controller starts new FI instances as required and stops instances running on overloaded blades and idle instances. During the second day, the controller needs only to execute one scale-in action because the FI instances running on Blade3, Blade5, and DBServer3 can handle the load. The FI instance on Blade11 is stopped ("In Blade11") because Blade11 is additionally running a CRM instance and, thus, is overloaded. The FI instance running on DBServer3 is stopped ("In DBServer3") in the night because the database of the BW subsystem uses the resources of DBServer3 heavily. Thus, at the beginning of the third day, the remaining FI instances become overloaded. To remedy this overload situation, the controller starts new FI instances as required. In summary, the controller can avert most imminent overload situations from the FI. The remaining overload situation periods are short.

Figure 17 shows load curves in the full mobility scenario. Again, the controller adds and stops instances as required. Additionally, service instances are moved from heavy loaded servers to other servers. In this scenario, users are dynamically redistributed, thus the effects of controller actions are observable instantly and overload situation can be averted completely.

5.3. Summary of Simulation Assessment

We ran simulation series for the three scenarios and each time increased the number of users by 5% until the system became overloaded. Table 7 shows the maximum numbers of users that can be handled by the existing hardware in the different scenarios relative to the number of users stated in Table 4.

Scenario	static	constrained mobility	full mobility
Max. Users	100%	115%	135%

Table 7. Maximum Possible, Relative Number of Users

In the static scenario, the hardware is sized for the initial number of users. Thus, if we increase the number of users by 5%, the installation immediately becomes overloaded, i.e., batch jobs are not processed in time and the response time of interactive requests increases. Thus, the working schedule is screwed up. Eventually, this might lead to an instable system. Using our controller in the constrained mobility scenario, the SAP installation can already handle 15% more users, because available idle resources are used to remedy overload situations. Due to the restrictions of the static user distribution and of the available actions, idle resources cannot be used as efficiently as in the full mobility scenario. Nevertheless, our controller already works quite well for the constrained mobility scenario. In the full mobility scenario, our controller can push the number of users that can be handled by the SAP installation to 135% compared to the static scenario. The number of users is higher than in the constrained mobility scenario, because more services are mobile and dynamic relocation is supported.

The conclusion of our studies is, that our controller can improve the capability of current IT-infrastructures if static services like databases and central instances are deployed well. Additional degrees of freedom and dynamic user redistribution result in much more effective controller actions and, thus, a higher number of users that can be handled by the available hardware.

6 Related Work

IBM coined the term of autonomic computing [11] in October 2001 and triggered several research projects of global industrial players in this area. An autonomic computing system provides self-managing capabilities, i.e., it handles self-configuration, self-healing, self-optimization, and self-protection. Weikum [25] motivates the automatic tuning concepts in the database area and concludes that it should be based on the paradigm of a feedback control loop which consists of three phases: observation, prediction, and reaction.

Modern database systems are increasingly realizing automatic administration features. [18] presents IBM's autonomic query optimizer—based on a feedback control loop—that automatically self-validates its cost model without requiring any user interaction to repair incorrect statistics or cardinality estimates. [2] proposes the application of a fuzzy controller for Quality of Service (QoS) management of query execution plans in distributed query processing systems. Actions like *alterNetServiceQuality*, *movePlan*, *useCompression*, or *increasePriority* are triggered by the fuzzy controller to enforce QoS parameters like query result quality, query execution time, and query evaluation cost. In [3] new administration and management concepts from IBM DB2, Microsoft SQL Server, and Oracle 10g were presented. All these self-administration concepts from the DBMS manufacturers and from researchers consider the database system isolated and integrate specialized self-management capabilities. Nevertheless, these concepts are not sufficient to provide comprehensive automatic administration for self-organizing infrastructures. Therefore, the complete system, i.e., the database applications and the databases themselves, must be regarded.

There are first results in the area of autonomic infrastructures, e.g., *IBM Director 4.1* [12]. Using this tool, administrators can view and track the hardware configuration of remote systems and monitor the usage and performance of critical components. Further, it contains tightly integrated tools, e.g., for monitoring the availability of hardware and software and distributing system resources according to administrator-defined policy entitlements, to optimize performance and maximize availability. HP researchers [14, 22] discuss load-balancing technologies for services. Their basic objective is to make sure all system resources work effectively. While the commercial products depend on vendor-specific hardware-features, our solution is independent from the underlying hardware. Further, our adaptive controller supervises and controls a complex self-organizing infrastructure without human interaction and/or supports administrators by giving recommendations.

Currently, a shift from Web services to Grid services can be observed. Basically, a Grid Service is a Web service implementing some standard interfaces. Thus, our administration concept can be used within grids as well. The Condor project [4] provides a flexible framework for matching resource requests with resources available in Grid environments. Its goal is to harness wasted CPU power from otherwise idle desktop workstations. The Enterprise Grid Alliance [5] is a consortium formed to develop enterprise grid solutions and accelerate the deployment of grid computing in enterprises. Major hardware and software vendors are members of the EGA, e.g., Fujitsu Siemens, HP, Intel, Oracle, and Sun. Commercial infrastructures, e.g., Fujitsu Siemens's FlexFrame [6], IBM's Dynamic Infrastructure [13], and Sun's N1 Advanced Architecture for SAP [24] offer a flexible infrastructure and virtualized services like AutoGlobe, but are tailored and limited to the mySAP Business Suite. The automatic administration of these infrastructures is mostly rule-based and not as flexible as our fuzzy controller. At this stage, there are already some large scale deployments of, e.g., FlexFrame within enterprises like T-Systems [19] and Hella [10] showing the benefits of such infrastructures.

For the description of the servers and services we use a proprietary XML language that is based on preliminary versions of an XML language for the description of servers and services from the project group *Scheduling and Resource Management* of the *Global Grid Forum* [7]. If this XML language becomes a standard, we will adopt it. In [1] the concepts and terminology of load balancing are explained pragmatically. This book shows the complexity of load balancing in computing infrastructures.

In [8], we presented feed-forward control techniques for adaptive infrastructures which are orthogonal to the techniques presented in this paper: exploitation of administrator given hints and short-term load forecasting for services with periodic behavior. Using these techniques, adaptive infrastructures can react proactively on imminent overload situations. There are overlaps between some aspects of the administration concepts presented in this paper and the German paper [9]. In [9], we put the main emphasis on the optimization of the static allocation of services to servers.

7. Conclusion: Status and Future Work

We presented a novel fuzzy controller based automatic administration concept which is hiding the ever increasing complexity of managing IT-infrastructures. We introduced our service platform ServiceGlobe, which forms the basis of AutoGlobe. After that, we described the architecture of AutoGlobe and its controller framework, which enhances ServiceGlobe with functionality to generate an up-to-date view of the load situation of the system. This view is used by a fuzzy controller to generate actions to remedy imminent overload, failure, and idle situations.

We implemented a research prototype of AutoGlobe and field tested it on blade server environments run by the ACI group of SAP using a rule base comprising about 40 rules. Up to now, the largest environment used for testing was a blade server system with 160 processors overall (with 2 and 4 processors per blade, respectively). Using our prototype we demonstrated the effectiveness of our proposed solution by performing a set of comprehensive simulation studies. The results of these studies confirm the applicability of a fuzzy controller for the supervision of a self-organizing infrastructure and the benefits of such an infrastructure even for already existing complex environments.

A demonstration of the self-management capabilities at

the Sapphire 2003 [21] gained huge interest among the SAP customers. SAP NetWeaver and new architectures like the Fujitsu Siemens blade server infrastructure FlexFrame can profit from the presented techniques by supplementing the service virtualization features with self-management capabilities.

We plan to improve the decisions of the controller even more. First, we will enhance the controller in such a way that it can manage explicit reservations, i.e., that an administrator can register mission-critical tasks along with their resource requirements. Second, we work on predicting the future load of services based on historic data stored in the load archive using pattern matching and data mining techniques. First encouraging simulation studies have already been conducted. The reservations and load prediction can be used to improve the action and host selection process of the controller. Additionally, based on explicit reservations for mission critical services and on the predicted load situation of services, we plan to develop a landscape designer tool. This tool calculates a statically optimized preassignment of all services to improve the dynamic optimization potential of the fuzzy controller. Eventually, we plan to enhance AutoGlobe towards QoS management for self-organizing infrastructures. The actions will then be used to enforce Service Level Agreements.

8. Acknowledgments

We would like to thank Wolfgang Becker, Ingo Bohn, and Thorsten Dräger of SAP's Adaptive Computing Infrastructure group for their cooperation. Also, we would like to thank Tobias Brandl for helping in the implementation of AutoGlobe.

References

[1] T. Bourke. *Server Load Balancing*. O'Reilly & Associates, Sebastopol, CA, USA, 2001.

[2] R. Braumandl, A. Kemper, and D. Kossmann. Quality of Service in an Information Economy. *ACM Transactions on Internet Technology (TOIT)*, 3(4):291–333, 2003.

[3] S. Chaudhuri, B. Dageville, and G. Lohman. Self-Managing Technology in Database Management Systems. Tutorial at the International Conference on Very Large Data Bases (VLDB), Toronto, Canada, Sept. 2004.

[4] Condor. http://www.cs.wisc.edu/condor/, 2005.

[5] Enterprise Grid Alliance. http://www.gridalliance.org.

[6] FlexFrame für mySAP Business Suite. http://www.fujitsu-siemens.de/le/solutions/it_infrastructure_solutions/sap_infrastructure/index.html.

[7] Scheduling and Resource Management Area of the Global Grid Forum. https://forge.gridforum.org/projects/srm/.

[8] D. Gmach, S. Krompass, S. Seltzsam, M. Wimmer, and A. Kemper. Dynamic Load Balancing of Virtualized Database Services Using Hints and Load Forecasting. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05) - Workshops Vol. 2*, pages 23–37, June 2005.

[9] D. Gmach, S. Seltzsam, M. Wimmer, and A. Kemper. AutoGlobe: Automatische Administration von dienstbasierten Datenbankanwendungen. In *Proceedings of the GI Conference on Database Systems for Business, Technology, and Web (BTW)*, pages 205–224, Feb. 2005.

[10] S. Hofmeyer. Flexibler geht's nicht. *SAP Info – Das SAP-Magazin*, (121):60–62, Oct. 2004.

[11] P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, 2001.

[12] IBM Director 4.1. http://www.ibm.com/servers/eserver/xseries/systems_management/director_4.html.

[13] IBM Dynamic Infrastructure for mySAP Business Suite. http://www.ibm.com/solutions/sap/us/en/xslpage/xmlid/25044.

[14] L. Jin, F. Casati, M. Sayal, and M.-C. Shan. Load Balancing in Distributed Workflow Management System. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, Las Vegas, NV, USA, March 2001.

[15] M. Keidl, S. Seltzsam, and A. Kemper. Reliable Web Service Execution and Deployment in Dynamic Environments. In *Proceedings of the International Workshop on Technologies for E-Services (TES)*, volume 2819 of *Lecture Notes in Computer Science (LNCS)*, pages 104–118, Berlin, Germany, Sept. 2003.

[16] M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1047–1050, Hong Kong, China, Aug. 2002.

[17] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1994.

[18] V. Markl, G. M. Lohman, and V. Raman. LEO: An Autonomic Query Optimizer for DB2. *IBM Systems Journal*, 42(1):98–106, 2003.

[19] U. Petersen, S. Dreyer, and N. Paira. A Flexible Framework. *SAP INFO – The SAP Magazine*, (120):64–65, Sept. 2004.

[20] SAP NetWeaver. http://www.sap.com/solutions/netweaver/.

[21] SAP Keynote: Turning Vision into Reality: Customer Roadmaps to Lower TCO. http://www.sap.com/community/events/2003_orlando/keynotes.asp, 2003. SAPPHIRE'03.

[22] M. Sayal, F. Casati, U. Dayal, and M.-C. Shan. Business Process Cockpit. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, Aug. 2002.

[23] S. Seltzsam, S. Börzsönyi, and A. Kemper. Security for Distributed E-Service Composition. In *Proceedings of the International Workshop on Technologies for E-Services (TES)*, volume 2193 of *Lecture Notes in Computer Science (LNCS)*, pages 147–162, Rome, Italy, Sept. 2001.

[24] N1 Advanced Architecture für SAP. http://de.sun.com/solutions/solution_sales/sap_erp/n1aa/index.html.

[25] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 20–31, Hong Kong, China, Aug. 2002.

[26] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.