Dynamic Load Balancing of Virtualized Database Services Using Hints and Load Forecasting

Daniel Gmach Stefan Krompass Stefan Seltzsam
Martin Wimmer Alfons Kemper

Technische Universität München
D-85748 Garching/München, Germany

⟨*firstname.lastname*⟩`@in.tum.de`

Abstract. Future database application systems will be designed as Service Oriented Architectures (SOAs), in contrast to today's monolithic architectures. The decomposition in many small 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 are necessary. It monitors the system, reacts automatically on exceptional situations, e.g., overload of a server, and features self-optimizing capabilities. In the context of distributed services it takes some time until the reaction on an exceptional situation becomes effective. During this time the server stays overloaded which leads to a poor performance of services running on this server. In this paper, we present a novel concept to improve self-organizing infrastructures to react proactively. For this purpose we present two techniques: Short-term load forecasting for services with periodic behavior and exploitation of hints from administrators, e.g., resource consumptions, for irregular events. With these techniques our system reacts proactively on imminent overload situations before they actually appear, thus avoiding overload situations. The quality of higher-level services, like enterprise resource planning (ERP) systems, can be improved by running on this proactive platform. We used our prototype implementation to perform comprehensive simulation studies, which demonstrate the effectiveness of our approaches.

1 Introduction

Concerning the administration of Service Oriented Architectures (SOA) for database application systems, three objectives can be identified: Low administration effort, low total cost of ownership (TCO), and a high degree of service availability and performance, i.e., ensuring that a predefined number of clients (customers and employees) can be handled by the infrastructure. To guarantee high performance, an oversized hardware environment could be used. Obviously, this counteracts the second objective of low TCO. To balance these objectives, services have to be assigned to the available servers intelligently and monitored continuously. Thus, we developed our self-organizing infrastructure AutoGlobe

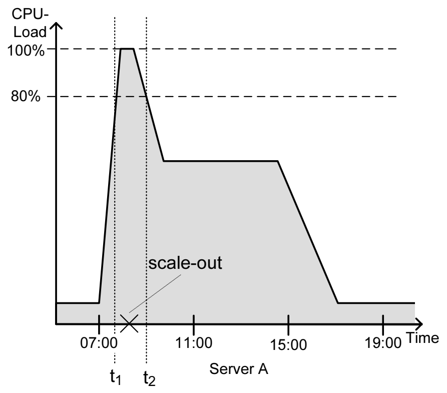


Fig. 1. Standard Controller

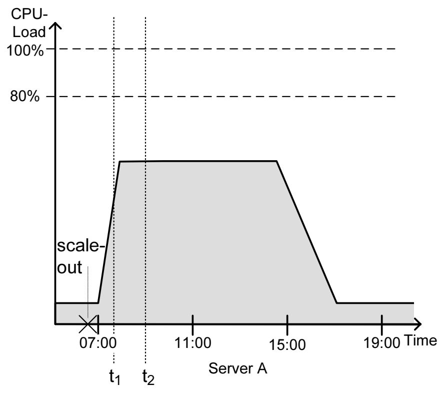


Fig. 2. Proactive Controller

[1] that is based on a feedback control loop. AutoGlobe monitors the current state of the system and reacts on an exceptional situation by, e.g., moving an instance of a service to another less loaded server during runtime. Figure 1 shows the load curve of a typical interactive service. In the morning at 7 AM when users start to work the load increases rapidly. The self-organizing infrastructure detects the exceptional situation at time t_1 and reacts after a verification phase in which the service is monitored for a certain time to filter out short load peaks. Furthermore, until the reaction – indicated by the start of an additional instance on another server (scale-out) in Figure 1 – takes effect, some time goes by (t_2). During the period $[t_1, t_2]$ the server is overloaded – we assume a server with load above 80% as overloaded – and the services running on the server exhibit poor performance. With the proactive control concept proposed in this paper, we improve our self-organizing infrastructure to administrate the system proactively, i.e., executing actions before load actually increases (see Figure 2). This concept is based on two techniques, which we integrated into our self-organizing infrastructure AutoGlobe [1]. Short-term load forecasting for services with periodic behavior and exploitation of hints for irregular events. Empirical studies show that most ERP-systems exhibit periodic load patterns. We use this knowledge in our first technique, that is based on short-term load forecasting. The controller forecasts the load of a service in the near future using extracted load patterns and the current system state. With short-term load forecasting we achieve a well balanced system by avoiding most exceptional situations without administrator interactions. The second technique is based on hints specified by administrators. For example, an administrator may specify the necessary number of service instances running during the course of a working day. Based on these hints, the controller starts additional instances before the load actually increases, thus avoiding exceptional system states.

The remainder of the paper is organized as follows: In Section 2 the architecture of our self-organizing infrastructure AutoGlobe is presented. Section 3 describes the exploitation of hints. Then, methods for pattern extraction from historical data and short term load forecasting follow in Section 4. 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.

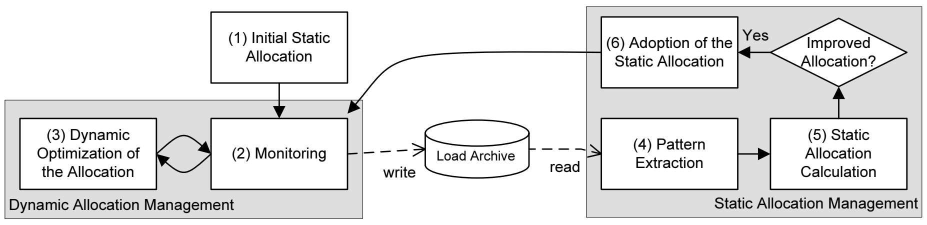

(1) Initial Static Allocation
(6) Adoption of the Static Allocation
Yes
Improved Allocation?
(3) Dynamic Optimization of the Allocation
(2) Monitoring
Load Archive
write
read
(4) Pattern Extraction
(5) Static Allocation Calculation
Dynamic Allocation Management
Static Allocation Management

Fig. 3. The AutoGlobe Framework

2 Self-Managing IT-Infrastructure

AutoGlobe [1] is based on our distributed and open Web service platform ServiceGlobe [2, 3]. 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 the support of 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 the entire standard functionality of a service platform like a transaction system and a security system [4]. 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.

Though not being restricted to this kind of hardware infrastructure, the benefits of AutoGlobe can be experienced best on a flexible infrastructure like a blade server environment. The advantages of blade servers compared to traditional mainframe oriented hardware are that they are relatively cheap 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.

2.1 Dynamic and Static Service Deployment

Figure 3 shows the basic AutoGlobe concepts, i.e., the interplay of dynamic and static service-to-server allocation management. The dynamic allocation management monitors services and servers (2). Exceptional situations, like failures and overload situations are detected and remedied by a fuzzy logic based controller (3).

	Static	Dynamic
Scale-Out Capable	• increase-priority • decrease-priority • scale-out • scale-in	• increase-priority • decrease-priority • scale-out • scale-in • move • scale-up • scale-down
Not Scale-Out Capable	• increase-priority • decrease-priority	• increase-priority • decrease-priority • move • scale-up • scale-down

Fig. 4. Classification of Services and Possible Control Actions

In conjunction with service monitoring, the resource consumption of each service is recorded and stored in a load archive. This aggregated historic load data is afterwards evaluated and service specific load patterns are determined (4). The pattern extraction constitutes a prerequisite for static allocation optimization (5): Knowing the load characteristics of services, those with complementary characteristics can be allocated on the same servers. For example, a service that mainly operates at night can be executed on the same server as a service that is mainly used during the day. Static allocation is used to review the current service deployment, i.e., regularly optimized allocation designs are calculated and compared with the actual one. The newly computed allocation design serves as new initial allocation on which the dynamic allocation management can set up. The initial allocation design (1) can be calculated in the same way, if load characteristics of services are given in the form of a time-dependent cost model.

The relation between the two approaches can be characterized as follows: Dynamic allocation management addresses local optimizations, i.e., it induces fast and small online deployment modifications. In contrast to this, global reorganizations induced by the static allocation management component might require (parts of) the system to be halted in the meantime. This, for example, applies to traditional database applications that do not support flexible moves between servers.

2.2 Service Model

In AutoGlobe services can differ in their level of flexibility. First we describe the characterization of services and afterwards we demonstrate how administrators can give hints to the self-organizing infrastructure.

Level of Flexibility. Services can be characterized by their level of flexibility, denoting the operations that can be performed to remedy overload situations. We distinguish between static and dynamic services. Instances of static services are assigned to designated servers. In contrast to this, the deployment of dynamic services can be modified during runtime. Furthermore, we characterize services regarding their scale-out capability, i.e., whether more than one instance of a service can be executed, or not. Figure 4 illustrates the possible combinations with the respective actions that are supported. For example, Web services, which can be replicated and instantiated on arbitrary service hosts, constitute the highest

Action	Description
start	starting a service
stop	stopping a service
scale-in	stopping a service instance
scale-out	starting an additional service instance
scale-up	moving a service instance to a more powerful server
scale-down	moving a service instance to a less powerful server
move	moving a service instance to an equivalently powerful server
increase-priority	increasing a service's priority
reduce-priority	reducing a service's priority

Table 1. Supported operations (depending on the level of flexibility)

level of flexibility. In contrast to this, traditional central database instances are oftentimes static and not scale-out capable. An overview of the actions is given in Table 1. These operations are initiated by the controller that manages the dynamic allocation.

Moving an instance requires that its context, i.e., the state information of all users currently utilizing the service instance, has to be materialized and a new instance has to be restarted that restores this context. Thus, the necessary time for a movement depends on the service. Obviously, such reorganization must be transparent to clients. Thus, a sophisticated dispatcher that supports a smooth transition onto a new instance is needed. Some of the operations depicted in Table 1, like moves, are not supported by all available services yet. Currently, the trend away from monolithic applications towards ERP systems that consist of aggregated, distributed, and autonomous services can be observed. Thus, applications are supposed to achieve a higher level of flexibility in the near future.

Hints. The features of services, e.g., their level of flexibility, are described using an XML description language. We extended this language such that administrators can give hints about services to the infrastructure, e.g., resource requirements. Table 2 gives an overview of the possible hints. For example, the hint `instances` denotes the minimum and maximum number of instances that should be executed for a service. With the hints `pinned` and `exclude` the administrator specifies on which servers the service may or must not be executed. AutoGlobe supports two kinds of hints: `instances` and `overallPIES` are service-oriented hints, all other hints describe the properties of every instance of the service.

Furthermore, the administrator can specify hints that are valid in periodic intervals. The following example shows a temporary resource hint for a financial accounting (FI)-service.

```
<service id="FI">
  <temporaryHints>
    <series name="example" startDate="20050201" endDate="20050301">
      <time startTime="07:00:00" endTime="18:00:00"/>
      <hint>
        <instances hintName="between two and four">
          <min>2</min>
          <max>4</max>
```

Hint	Description
instances	minimum and maximum number of instances
memory	minimum and maximum of main memory of the server
tempSpace	minimum and maximum of temporary space of the server
performanceIndex	minimum and maximum performance index of the server
exclusive	exclusive allocation of the service
pinned	list of possible server for the service
exclude	list of excluded server for the service
overallPIES	minimum and maximum for the sum of the performance indices

Table 2. Supported Hints

```
        </instances>
        <overallPIES hintName="at least 10 PIES">
          <min>10</min>
        </overallPIES>
      </hint>
    </series>
  </temporaryHints>
</service>
```

The hint is valid between February 1st, 2005 and March 1st, 2005. During this time the hint should be adhered every day from 7 AM to 6 PM and states that the service should be executed on at least 2 and at most 4 servers. Furthermore, the sum of the performance indices (the performance index of a server describes its overall capacity) should be at least 10 PIES, whereas one PIES corresponds to the capacity of a server with performance index one.

3 Hints about the resource consumption

In many cases the administrators know situations that claim for plenty resources in advance and can help the system by giving hints (see Section 2.2).

Figure 5 shows the architecture of the hint controller. The hint controller becomes active at the start of a service and every time a temporary hint becomes effective. First, it checks for inconsistencies and consolidates all hints that are effective at the moment. Afterwards, the hint controller enforces the consolidated hints by activating the fuzzy controller. Figure 6 shows the embedding of the hint controller in the dynamic controller. The hint controller chooses an appropriate action and triggers the fuzzy controller that chooses an appropriate target server for the action. Finally, the dynamic controller executes the action on the target server and, thus, achieves adherence of the hints.

3.1 Detection of Inconsistencies

In a first step the hint controller checks if several hints valid at the moment contradict each other, e.g., one hint claims at least 4 instances and another at most 2 instances. In case of inconsistencies, the hint controller sends a warning to the administrator and ignores inconsistent hints. Afterwards, it consolidates the remaining hints. For Example, if we have two hints, the first one claims at least 2 and at most 6 instances and the second one at least 4 instances, the consolidated hint claims at least 4 and at most 6 instances for the service.

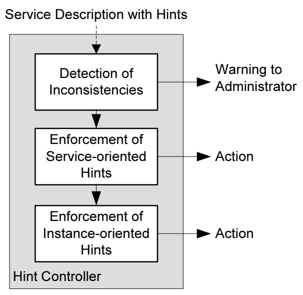

Service Description with Hints
Detection of Inconsistencies
Warning to Administrator
Enforcement of Service-oriented Hints
Action
Enforcement of Instance-oriented Hints
Action
Hint Controller

Fig. 5. Hint Controller

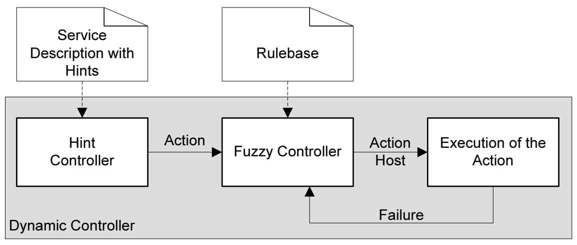


Fig. 6. Dynamic Controller

3.2 Exploiting Hints

The exploitation of the consolidated hints proceeds in two steps. In the first step the hint controller checks the service-oriented hints (see Section 2.2), i.e., number of instances and the total sum of performance indices (overallPIES). If the considered service has less instances than required, the hint controller will trigger the fuzzy controller with scale-out actions and specify the performance indices of the target servers regarding the total sum of performance indices. Thus, we increase the chance that the hint for the total sum of performance indices is adhered. Of course, the fuzzy controller only selects target servers for the action that fulfill all conditions. The case that too many instances of a service are running is treated analogously. Additionally, the hint controller checks the sum of performance indices and potentially corrects the situation by triggering scale-ups or scale-downs, respectively.

In the second step the hint controller checks for every instance of the service if all instance-oriented hints are satisfied. The different hints are checked sequentially for every instance, starting with resource hints like performance index, main memory, and temporary space, followed by the exclusive flag and the list of pinned and excluded servers. If one of these hints is violated, the hint controller triggers actions like move, scale-up, or scale-down – depending on the current situation and on the operations supported by the service.

4 Dynamic Load Balancing Based on Short-term Load Forecasting

We use short-term load forecasting to improve the dynamic load balancing of our system. Therefore, we extract patterns from services revealing periodic behavior to gain information about their future load consumption.

4.1 Pattern Extraction

The load induced by ERP applications often shows a periodic characteristic. Considering an LES service for example, typical load peaks can be recognized caused by the work day of the employees. In our experiments we analyzed the CPU and memory requirements of services. As the algorithms are not restricted

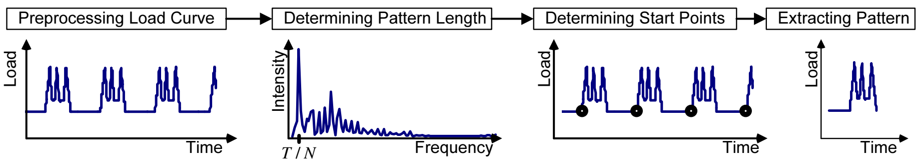


Fig. 7. Pattern Extraction

to CPU and memory, we use a generic notion of load in the following. Figure 7 gives an overview of the pattern extraction.

First, historic load information of the distinct service instances is aggregated to one equidistantly sampled time series representing the overall load induced by the respective service. Thereby, the performance of the host machines the instance is executed on is considered through a normalization factor.

According to the classic additive component model [5], a time series $(x_t)_{1\leq t\leq N}$ is composed of trend component, cyclical component, and remainder. The trend models the long-term monotonic change of the average level of the time series while the remainder represents noise effects. Service specific load patterns are characterized by the cyclical component. Time series can be represented as an overlay of harmonics that can efficiently be computed via Fourier Transformation. The function $I(\lambda)$, with

$$I(\lambda) = N \cdot \left[C(\lambda)^2 + S(\lambda)^2\right], \text{ with } \lambda \in \mathbb{R} \text{ and}$$

$$C(\lambda) = \frac{1}{N} \sum_{1\leq t\leq N} (x_t - \bar{x}) \cdot \cos 2\pi\lambda t, \quad S(\lambda) = \frac{1}{N} \sum_{1\leq t\leq N} (x_t - \bar{x}) \cdot \sin 2\pi\lambda t$$

defines the intensity, with which the harmonic of frequency λ is present in the time series that is normalized by the mean value $\bar{x}$. The correlation between the pattern length T and I is that I is maximized at T/N. Thus, finding the dominant frequency provides hints about the length of a pattern. Subsequently, we determine distinctive start points for the particular pattern occurrences of length T within the time series. Finally, the representative load pattern p_s for service s is computed as the mean of these reoccurrences. For further details see [1] where pattern extraction is used in the context of static allocation management.

The extraction process is done under the assumption, that the series actually reveals a periodic characteristic. Whether this holds is evaluated in a subsequent classification phase. Therefore, the distance of the extracted templates to the original time series and the evaluation of the periodogram are used to estimate the quality of the extracted patterns. Depending on these estimation factors, clusters of services are calculated – in the simplest case, services with relatively high probability of being periodic are separated from those that are supposed to show irregular load development. To sum up, it is noticeable that the pattern extraction proceeds with a minimum of parameterization, which is a prerequisite for its applicability in the context of adaptive computing.

4.2 Short-term Load Forecasting

The dynamic controller uses patterns to forecast load for the monitored services in the near future. Figure 8 shows a typical load pattern for an ERP service,

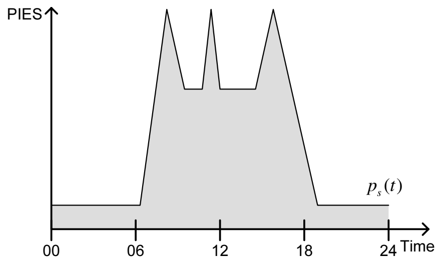


Fig. 8. Extracted Load Pattern

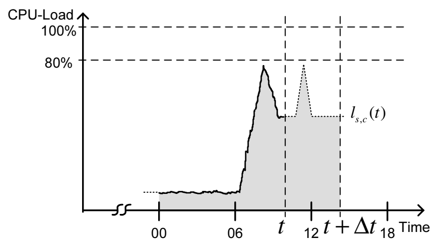


Fig. 9. Load Forecasting

where load increases in the morning and reaches peaks in the morning, before midday, and in the evening before users leave off work. The load is measured in PIES and one PIES corresponds to the capacity of a server with performance index one. Figure 9 shows the situation at time t. The solid line exhibits the load of the monitored instance and the dotted line exhibits the forecasted load in the near future. Due to performance the controller only forecasts the load l_s of the service s for the time $t + \Delta T$, but for illustration purposes, we show the complete curve. The assumption

$$\frac{l_{s,c}(t)}{l_s(t)} = \frac{l_{s,c}(t + \Delta t)}{l_s(t + \Delta t)}$$

is necessary to forecast future load. It states that the fraction of instance load $l_{s,c}$ of service s on server c by the service load l_s is assumed constant for the forecasting period. This denotes that the distribution of the users or requests to the service instances between t and $t + \Delta t$ remains constant. Under the precondition that the load of the service $p_s(t)$ extracted from the pattern for the time t is greater than zero the equation

$$l_{s,c}(t + \Delta t) = \frac{1}{pi_c} \cdot l_{s,c}(t) \cdot \frac{p_s(t + \Delta t)}{p_s(t)}$$

calculates the future load value $l_{s,c}$ for the time $t + \Delta t$. If the pattern is correct $p_s(t)$ equals $l_s(t)$ which is the monitored value. The current load of the instance $l_{s,c}(t)$ multiplied by the relative change of the service load (last fraction) is the forecasted load of the instance in PIES. To obtain the load caused by the instance on the server c, we divide this value by the performance index pi_c of the server c.

In case $p_s(t)$ is zero we need information about how the dispatcher distributes users or user requests. Then, the controller can estimate the future load of the service instances from the forecasted load of the service.

4.3 Reacting Proactively

The dynamic controller uses the forecasted load values and reacts if an exceptional situation is forecasted. Therefore, the interval Δt should be chosen carefully. If Δt is very small, the effect of the reaction probably is too late. If Δt is very large, on the one hand forecasted values are inaccurate because the above assumption is likely not to hold and on the other hand the controller reacts

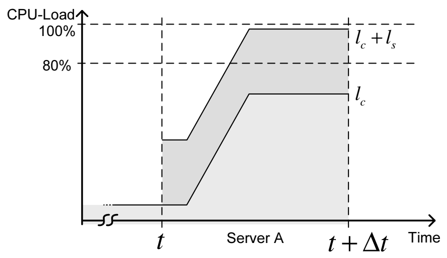


Fig. 10. Scale-out on Server A

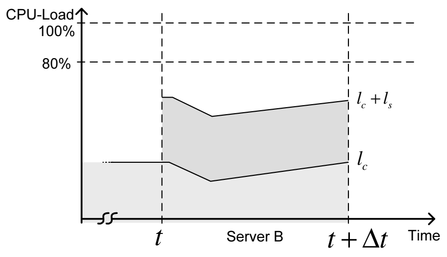


Fig. 11. Scale-out on Server B

very early and potentially wastes resources. A good value of Δt is the time that passes from the detection of an exceptional situation until the reaction becomes effective. In case the controller forecasts an exceptional situation it has to choose a proper reaction. Of course, for the decision making process we use forecasted load information to select a proper action and target server. Figure 10 and 11 show two possible target servers for a scale-out action at time t. In Figure 10 the scale-out is executed on server A and in Figure 11 on server B. The bright grey area is the load caused by other services and the dark grey area is the load caused by the instance started through the scale-out. If the fuzzy controller only knows the historic load values it will choose server A because it is currently less heavily loaded than server B. When considering the future load this is a suboptimal decision as load caused by other services on server A increases and A will become overloaded. If the dynamic controller considers current and forecasted values then it makes optimal decisions for the near future. In this case it will choose server B for the scale-out.

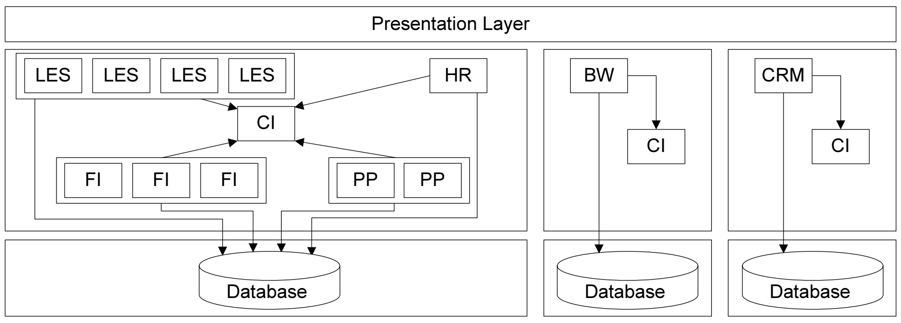
5 Simulation Studies

We performed comprehensive simulation studies to demonstrate the effectiveness of our proactive infrastructure. They have been conducted using a simulation environment, that models a realistic ERP installation (see Figure 12).

5.1 Description of the Simulated Environment

The installation is divided into a database layer, an application server layer, and a presentation layer. Furthermore, it comprises three subsystems in the application and database layer: Classical *Enterprise Resource Planning (ERP)*, *Business Warehouse (BW)*, and *Customer Relationship Management (CRM)*, each supplied with its own dedicated database and central instance (CI). The central instance applications are responsible for the lock management. The other application servers (BW, CRM, FI, HR, LES, PP) execute the application logic, i.e., process user requests.

On this system different user requests are simulated, affecting the respective application server, central instance and database. On each involved component representative load is simulated. For example, an LES request produces lower load on a database than a BW request. In order to simulate a realistic ERP



BW:	Business Warehouse	HR:	Human Resources
CI:	Central Instance	LES:	Logistics Execution System
CRM:	Customer Relationship Management	PP:	Production and Planning System
FI:	Financial Accounting		

Fig. 12. Simulation System – Architecture

system, the load curves of the simulated services follow predetermined patterns that can be observed in many organizations running ERP systems. For example, a BW application is mainly working during the night, while the LES's peaks are in the morning, before lunch and in the evening. For further details on the simulation system see [6].

We simulate an ERP environment supervised by the dynamic controller. The application servers and central instances support scale-in and scale-out actions and the database services are static and not scale-out capable. The dispatcher distributes users to the instances of a service according to the performance indexes of the servers running the instances. After a scale-out of an application, users are not dynamically redistributed. They remain logged on until they complete their session. We simulate a fluctuation of users, i.e., users infrequently log themselves off and reconnect to the least-loaded application server – a kind of behavior that can be observed in real systems, too.

We simulated 120% Users and used the following settings: To prevent the system from reacting too late, we set the threshold value for a CPU overload to 80%, i.e., if a server has more than 80% CPU load it is considered to be overloaded. In this case, the controller monitors the server for 10 minutes (watchTime) in order not to react on short load bursts. After executing an action, the affected services are protected – no actions can be executed on this service – for 30 minutes and the affected servers for 60 minutes.

5.2 Results of the Simulation Studies

Figures 13, 14, and 15 demonstrate the effectiveness of our proposed concepts. They show the load curves for all servers and the average load of the whole system indicated by a thick line. Figure 13 shows the load curves of the system managed by the dynamic controller that we proposed in [1]. This system has problems when the load increases rapidly at 8 AM. After some time it manages to remedy the overload situations. To prevent the system from exceptional situations we gave the system hints that are shown in table 3. The result of the simulation with exploiting hints is shown in Figure 14 where we manage to avoid nearly all

Service	6 AM - 6 PM Instances	6 AM - 6 PM PIES	6 PM - 6 AM Instances	6 PM - 6 AM PIES
CRM	> 2	> 8	$= 1$	> 3
LES	> 6	> 20	$= 1$	> 3
FI	> 5	> 18	$= 1$	> 3
PP	> 3	> 12	$= 1$	> 3
HR	> 2	> 8	$= 1$	> 3
BW	$= 1$	> 3	> 4	> 8

Table 3. Given Hints

overload situations. At 6 AM the controller starts additional instances to satisfy the temporary hints. Thus at 8 AM there are enough instances running to take on the increasing load. In the evening the additional instances are stopped and the resources are freed for other services.

In Figure 15 we have the same initial situation like in Figure 13 but now we use our proposed load forecasting technique to prevent the system from overload situations. We monitored the simulated system for some days and extracted patterns from the historic load data. Using these patterns our proactive infrastructure forecasts load for 180 minutes in order to have enough time to instantiate more additional instances. In the morning the controller detects that the load will increase in three hours and starts additional instances of the services. Furthermore, it considers forecasted load values for the choice of target servers. Thus, the dynamic controller manages to avoid the overload situations.

5.3 Comparison

Both techniques – administrator given hints and short-term load forecasting – provide a well-balanced system and manage to avoid nearly all predictable overload situations. Furthermore, they are complementing each other. Short-term load forecasting should be used for services with cyclical behavior and hints should be used additionally for mission critical services with irregular load consumptions as they cause more administration effort. Finally, temporary hints can be used to execute services exclusively on a server, e.g., the BW-database should be executed exclusively on the server during night while during day other services can run on it as well.

The conclusion of our studies is that reacting proactively improves the performance of self-organizing infrastructures. Using short-term load forecasting for periodic services supplemented with administrator given hints for irregular situations that are known in advance, we achieve to prevent the system from nearly all overload situations. Of course, if an unpredictable exceptional situation appears, the dynamic controller will still recognize and remedy it.

6 Related Work

Previous work in the AutoGlobe project is described in [1]. It explains the architecture, the fuzzy controller, and the concept of static and dynamic allocation management in detail. [6] shows the results of the corresponding simulation

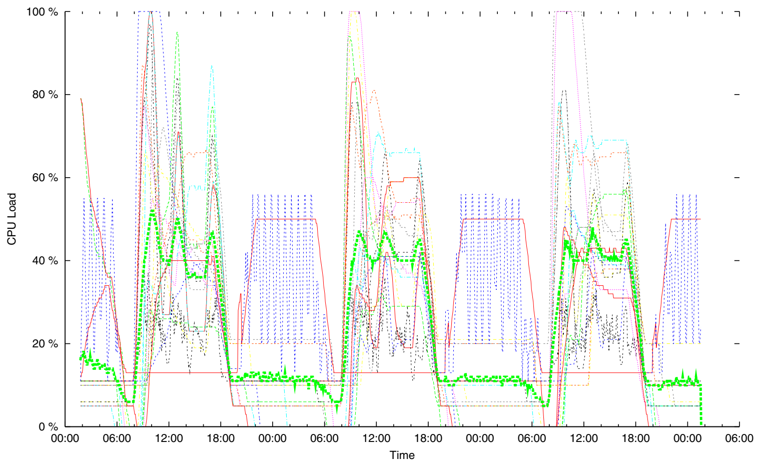


Fig. 13. CPU Load of all Servers

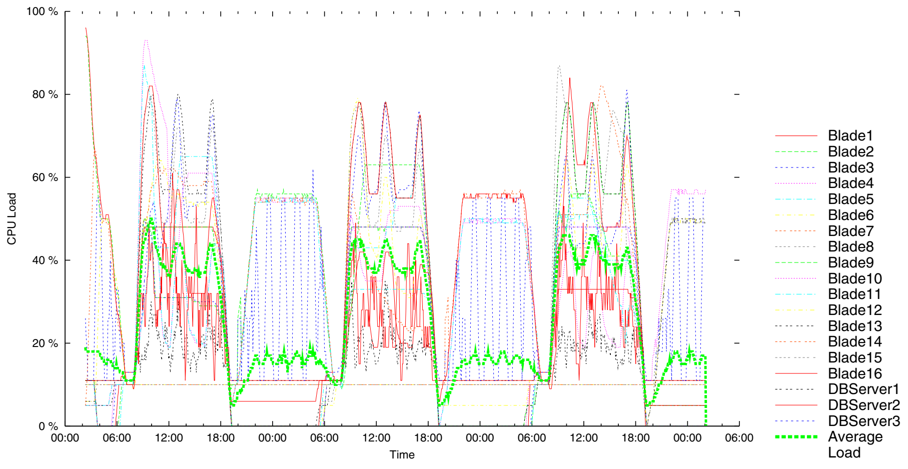


Fig. 14. CPU Load of all Servers with Exploitation of Hints

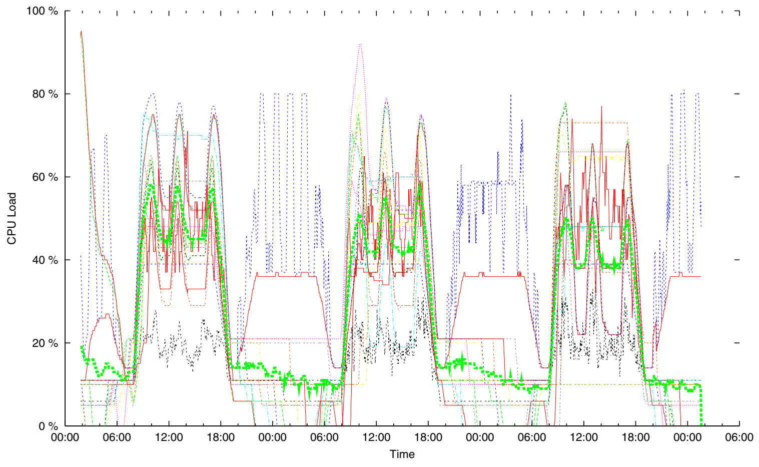


Fig. 15. CPU Load of all Servers with Load Forecasting

studies. The proactive control concept presented in this paper improves the dynamic allocation management of AutoGlobe to react proactively. Weikum [7] motivates the automatic tuning concept 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. [8] 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.

Since IBM coined the term of autonomic computing [9] in October 2001 several global industrial players initiated research projects in this area. An autonomic computing system provides self-managing capabilities, i.e., it handles self-configuration, self-healing, self-optimization, and self-protection.

The author of [10] pragmatically explains the concepts and terminology of load balancing. This book shows the complexity of load balancing in computing infrastructures. [11] presents an architecture that combines resource reservations and application adaptations in the context of network applications. The authors examine the interaction of reservations and adaptions in detail for the resource network bandwidth. [12], [13], and [14] present storage systems that provide self-organizing capabilities and support administrators during the design of storage systems. [15] develops an architecture for multi-platform working, based on an autonomic computing concept, where the autonomic elements communicate their 'vital signs' to achieve self-healing capabilities. [16] is a framework for enabling autonomic grid applications. They use a decentralized deductive engine, that provides core capabilities for supporting automatic compositions, adaptations, and optimizations. While these projects monitor the system and react on exceptional situations our concept forecasts exceptional situations and reacts proactively.

7 Conclusion

We presented a novel concept for self-organizing infrastructures that are based on a feedback control loop to react proactively and thus improve the performance of the system. Therefore, we introduced two techniques: short-term load forecasting and exploitation of administrator given hints. Furthermore, we improved our dynamic controller to regard forecasted load values for the choice of a target server. We implemented the two techniques within the scope of our prototype AutoGlobe, which is based on a fuzzy controller that determines an action and a target server to remedy the exceptional situation. Using our prototype we demonstrated the effectiveness of our proposed concept by performing comprehensive simulation studies. The results of the studies confirm the applicability of our two techniques.

We are currently investigating concepts to improve quality of service (QoS) in self-organizing infrastructures. For this purpose administrators specify service level agreements (SLA). Using the SLAs and the system state the controller can calculate reservations, such that the SLA are adhered. Furthermore, the controller monitors the QoS-Parameters and reacts on imminent SLA-violations.

References

1. Gmach, D., Seltzsam, S., Wimmer, M., Kemper, A.: AutoGlobe: Automatische Administration von dienstbasierten Datenbankanwendungen. In: Proceedings of the GI Conference on Database Systems for Business, Technology, and Web (BTW), Karlsruhe, Germany (2005) 205–224
2. Keidl, M., Seltzsam, S., Kemper, A.: 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)., Berlin, Germany (2003) 104–118
3. Keidl, M., Seltzsam, S., Stocker, K., Kemper, A.: ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In: Proceedings of the International Conference on Very Large Data Bases (VLDB), Hong Kong, China (2002) 1047–1050
4. Seltzsam, S., Börzsönyi, S., Kemper, A.: 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)., Rome, Italy (2001) 147–162
5. Schlittgen, R., Streitberg, B.: Zeitreihenanalyse. 9 edn. R. Oldenbourg Verlag (2001)
6. AutoGlobe: Simulation studies. `http://www-db.in.tum.de/research/projects/AutoGlobe` (2005)
7. Weikum, G., Mönkeberg, A., Hasse, C., Zabback, P.: 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), Hong Kong, China (2002) 20–31
8. Markl, V., Lohman, G.M., Raman, V.: LEO: An Autonomic Query Optimizer for DB2. IBM Systems Journal **42** (2003) 98–106
9. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf` (2001)
10. Bourke, T.: Server Load Balancing. O'Reilly & Associates, Sebastopol, USA (2001)
11. Foster, I., Roy, A., Sander, V.: A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In: 8th International Workshop on Quality of Service. (2000)
12. Ganger, G.R., Strunk, J.D., Klosterman, A.J.: Self-* Storage: Brick-Based Storage With Automated Administration. Technical report CMU-CS-03-178, Carnegie Mellon University, Pittsburgh, USA (2003)
13. Uttamchandani, S., Yin, L., Alvarez, G.A., Palmer, J., Agha, G.: CHAMELEON: A Self-Evolving, Fully-Adaptive Resource Arbitrator for Storage Systems. USENIX Annual Technical Conference (2005) 75–88
14. Alvarez, G.A., Borowsky, E., Go, S., Romer, T.H., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., Wilkes, J.: Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems. ACM Transactions on Computer Systems (TOCS) **19** (2001) 483–518
15. Sterritt, R., Bantz, D.F.: Pac-men: Personal autonomic computing monitoring environment. In: 15th International Workshop on Database and Expert Systems Applications (DEXA 2004), IEEE Computer Society (2004) 737–741
16. Agarwal, M., Bhat, V., Liu, H., Matossan, V., Putty, V., Schmidt, C., Zhang, G., Zhen, L., Parashar, M., Khargharia, B., Hariri, S.: AutoMate: Enabling Autonomic Applications on the Grid. In: Proceedings of the International Workshop on Active Middleware Services (AMS), Seattle, WA, USA (2003) 48–59