Adaptive Self-optimization in Distributed Dynamic Environments

Wolfgang Trumler, Andreas Pietzowski, Benjamin Satzger, Theo Ungerer
Department of Computer Science
University of Augsburg
Eichleitnerstr. 30, 86159 Augsburg, Germany
{trumler, pietzowski,satzger,ungerer}@informatik.uni-augsburg.de

Abstract

Grid and ubiquitous computing systems generally consist of a large number of networked nodes with applications implemented as distributed services or processes, respectively. A crucial point is the distribution of the services to balance the load within the system during runtime.

In a former work we developed a self-optimization mechanism which shows outstanding performance in static environments where the services do not change their resource consumptions (e.g. CPU, memory, communication bandwidth). In this paper we present simulation results for the self-optimization within dynamic environments where the services change their load during runtime.

1. Introduction

Large distributed computing environments are state of the art in many areas. Grid computing uses networked servers all over the world to build massively parallel and powerful computing environments. Ubiquitous systems are rising up to be widely deployed in everyday environments like smart offices and smart homes. All of these systems are comprised of a large number of networked nodes, thus it is impossible for an administrator to manually fine tune the system for optimal performance. Furthermore, if the optimization parameters change due to application requirements the administrator would have to optimize the system again to meet the new requirements.

It is obvious that these systems should self-optimize and adapt to changing environments and application requirements at runtime. Self-optimization is one of the key features of autonomic [2] and organic computing systems [3]. Both initiatives propose systems that expose life-like features to be more robust and reliable even in unexpected situations.

The middleware system OC\textmu [6] was developed to offer self-configuration, self-optimization, self-healing, and self-protection capabilities. It is based on the assumption that applications are composed of services, which are distributed to the nodes of the network. The services are distributed to the nodes during the initial self-configuration phase considering the available resources and the resource requirements of the services. At runtime the services’ resource consumptions are monitored. A service can be relocated or transferred to another node to balance the resource consumption (load) between the nodes.

In a former work we developed a self-optimization mechanism [7] which shows outstanding performance in static environments where the services do not change their resource consumptions (e.g. CPU, memory, communication bandwidth). In this paper we present some evaluation results of our former proposed self-optimization in dynamic environments.

The remainder of this paper is structured as follows. The next section gives a brief overview of the metrics we used for the self-optimization. In section 3 we describe different scenarios of a dynamic environment as we expect them in a typical OC\textmu setup. The simulation results are presented in section 4. Section 5 gives an overview of related work and the paper closes with a conclusion and future work in section 6.

2. Transfer Strategies

The load of a distributed system based on services can be balanced by transferring services between the nodes. The metrics we use to decide whether the load between two nodes of the network should be balanced are named transfer strategies because they decide on the transfer of a service.

All the relevant information of the message exchange between the nodes and a detailed description of the employed Transfer Strategies (TS) can be found in [6, 7].

We developed four TS. The Simple Transfer Strategy transfers a service any time a load difference is encountered between two nodes. The Transfer Strategy with Load Estimator tries to estimate the average load of the network
and transfers a service only if one node is beyond and the other is below the estimated load. This reduces the amount of service transfers to 40% to 50% of the former TS. The Transfer Strategy with adaptive Barrier employs an adaptive barrier, which is calculated out of the load difference between the nodes. This value gives a hint about the optimization potential of the load in the network. If the load difference of the nodes is high the barrier decreases and vice versa. This suppresses the transfer of services which would add only little gain to the overall optimization. In fact, the adaptive barrier further reduces the amount of service transfers to 40% to 50% of the former TS. The Hybrid Transfer Strategy combines the advantages of the TS with Load Estimator and the TS with adaptive Barrier in one TS.

3. Dynamic Behavior of Services

It is very hard to describe the dynamic behavior of a system in terms of resource consumptions because there might be more than one influencing factor. On the other hand to gain meaningful evaluation results about the capabilities of the TS in dynamic environments it is crucial to differentiate between the possible load changes of the system.

We identified two different ways the load of a system can change. With combinations of both it should be possible to describe most of the dynamics in terms of resource consumption. The first is the periodic peak load and the second is an upswelling and decongesting load, respectively.

3.1. Periodic Peak Load

A typical dynamic behavior of a distributed system based on services or tasks which respond to incoming requests is a periodic peak load. If a request arrives at the service it begins to compute the response consuming resources like memory or CPU. After the computation, the service waits for the next request. Figure 1 shows such a dynamic behavior of a single service.

![Figure 1. Parameter definition of the periodic peak load for the simulation](image)

At the beginning the service is on a normal load level either sleeping or doing some background processing which will produce a certain base load to the system. For the simulations we assume an increase of the peak load of \( \Delta b \) if a request arrives. The computation requires \( bp \) simulation steps. After that time the load of the service is reduced to the normal load level. The service waits for the next request \( rp \) simulation steps which ends one duty cycle.

The load increase is proportional to the duty cycle of a periodic peak load. The cycle time is \( bp + rp \) and the portion responsible for the load increase is \( bp \). In the simulations the duty cycle time can be defined as an amount of simulation steps and \( bp \) can be defined as a percentage of the duty cycle time.

The overall load change of the system can be calculated with formula 1, where \( \Delta s \) is the percentage of services that behave dynamically.

\[
\Delta B_{\text{peak}} = \Delta s \ast \Delta b \ast \frac{bp}{bp + rp}
\]

3.2. Upswelling and Decongesting Load

The second kind of a load change is an upswelling and decongesting load. A number of services or tasks increase their load during a given time, stay at the higher load level for a while, and return to the normal load level during another interval. This kind of dynamic behavior can occur if a large number of services are needed to fulfill a certain job like a batch processing at night or a data mining run at certain times. Also cron-jobs known from Unix systems can produce a similar behavior. Figure 2 shows the upswelling and decongesting load with the parameters used for the simulations.

![Figure 2. Parameter definition of the upswelling and decongesting load](image)

The load of the services is increased by \( \Delta b \) during \( b \) simulation steps starting from their normal load level. They stay at the higher load level for \( bp \) simulation steps and return to the normal load level within \( b \) simulation steps. For the sake of simplicity we chose the interval of the upswelling load and the decongesting load to be the same. Formula 2 expresses the load increase of the overall system.

\[
\Delta B = \frac{s}{S} \ast \Delta b
\]
The overall load increase is defined by the additional load $\Delta b$ and the portion of the services that are subject to the load increase. In this case $s$ is defined as an absolute value of the interval $[0...S]$, while $S$ is the total amount of services.

4. Evaluations

To evaluate the dynamic behavior of the services we extended our simulator to change the loads of the services as explained in the last section.

All simulations were done a hundred times to avoid misinterpretations due to good starting conditions. Every simulation was reinitialized with the same parameter setting but the initial service assignment of the nodes was created randomly within the given parameters.

The default simulation setup consists of 1000 nodes and about 8300 services. The initial load of the services is chosen randomly between 1% and 10% of a node’s load. During the initialization every node is filled up with new services until the next generated service would exceed the node’s capacity. Then the next node is initialized the same way. The overall capacity of a node is also generated randomly prior to service creation. This guarantees that there are all kinds of nodes in the network from heavy loaded to unloaded nodes. Due to the randomized initialization it is further guaranteed that every simulation run starts with a different initial setup of services.

4.1. Periodic Peak Load

The periodic peak load has three relevant simulation parameters that can be varied: (1) the load ratio $bp : rp$, (2) the load change $\Delta h$, and (3) the amount of dynamic services. The dynamic behavior of the services starts right from the beginning which means that there is no initialization time for the TS to find a good optimization.

Figure 3 shows the results of the four TS for 100% dynamic services. The left chart shows the amount of service transfers and the right chart shows the average error. The average error is a measure for the quality of the optimization. If the average error is 0 all the nodes have exactly the same load. It can be observed that the TS with dynamic Barrier and the Hybrid TS are able to adapt to the dynamic behavior of the network even with 100% of dynamic services.

4.2. Upswelling and Decongesting Load

The upswelling load has also three parameters that can be varied to evaluate the TS: (1) the height of the load increase, (2) the amount of simulation steps for $b$ and $bp$, and (3) the amount dynamic processes.

The setup of the simulations is the same as for the periodic peak load. The only difference is that the simulation runs take 3000 simulation steps and that the upswell of the load starts after 1500 simulation steps. For the simulations we defined 25% of dynamic services with a load change of 100%.

Figure 4 shows the simulation results for $b = 50$, $bp = 500$. All TS try to balance the increasing load, which results in an increase of transferred services. The Hybrid TS needs the lowest amount of transfers.

As a conclusion to all the simulations we have done so far (more than 50 different settings beside those shown here) we can state that the Hybrid TS is the best in terms of the amount of service transfers. The best concerning the average error is the TS with Load Estimator.

5. Related Work

The presented self-optimization differs from state of the art load balancing mechanisms [5, 9] in two important points: first, no additional messages are used for the self-optimization except those needed for the transfer of the services and second, the load information is collected only locally. Furthermore there is no central instance that coordinates the self-optimization process. Thus, we want to compare it to systems that employ the same approach.

A very similar system, the Digital Hormone Model (DHM) [4], uses the Reaction-Diffusion-Model investigated by Alan Turing in 1952 [8]. The main idea is that the flow of information is modeled by the diffusion process of hormones. The DHM achieves interesting results with robot swarms. The main disadvantage is the high computation power needed to do the calculations of the differential equations. This would not be suitable for ubiquitous or embedded systems.

6. Conclusion and Future Work

In this paper we present the evaluation results of our self-optimization mechanism gained from simulations of dynamic environments. The metrics (Transfer Strategies) used to decide whether a service should be transferred to another node or not perform good even in large networks with 1000 nodes and more than 8000 services. The results show that the TS with adaptive barrier and the Hybrid TS are able to adapt to the average dynamics and to suppress unwanted transfers. One of the most outstanding issues about the self-optimization is that it is completely distributed without any central control and that no extra messages are used except for those needed to transfer a service to another node.

We plan to further evaluate the Hybrid TS in different real systems, especially in embedded systems. Therefore
we need to improve the measurement of the resource consumption to get more accurate load values.

References