An Application Requirement-based Framework for Adaptive Middleware

Qing Wu, Danzhen Wang
Hangzhou Dianzi University
Institute of Computer Application Technology
Hangzhou, Zhejiang
P.R.China, 310018

Ying Li
Zhejiang University
College of Computer Science
Hangzhou, Zhejiang
P.R.China, 310027

Abstract: High dynamic computing environments make QoS guarantee more important for component-based distributed system. Software system should possess self-tuning capacity for reacting to external environment variation. This paper firstly analyzes a model of component resource requirement and dependence relations among them that affect the quality of service. Then we propose an adaptive middleware framework that automatically tune server configuration parameters and react to workload changes to preserve the quality of service the application requires. The key of this framework is adopting backtracking algorithms, and it guides to search for the best combination of configuration parameters to satisfy application requirement and QoS requirement. Finally, it is prototyped on tomcat application server and validated using an information query system by comparing the performance requirement satisfaction with and without this framework. The results show that the application performance can be improved through this adaptive framework’s regulation when the workload increases.

Key–Words: Adaptive Middleware, Requirement model, QoS

1 Introduction

With the background of flourishing Internet, software systems operate under highly unpredictable and changeable conditions including users’ requirement and running environment, which need software system to be aware of dynamic context and reconfigure its resource to maintain the best application performance in the face of changes in system load and available resources. In other words, software system must be adaptive.

Distributed component middleware system used as web application server such as EJB, CORBA, .NET have developed as fundamental software in web computing environment. These kinds of middleware system provide a bundle of services that applications need such as security, transaction, and tremendously simplify the developing of large-scale complicated distributed system through shielding underlying platform’s heterogeneity. In addition, with excellent modeling, lots of COTS components can be deployed on any middleware system platform compatible with standards, which enhance the reuse of software.

Nowadays, EJB and .NET technology implement of middleware configuration through XML document configuration and application deployment. This technology performs well on the side of supporting application’s functional requirements, but lacks corresponding support for application with stringent QoS requirements. Middleware systems with static configuration lack of QoS guarantees mechanism and can hardly adapt to changeable load.

When application performance becomes main criterion, numerous dynamic factors must be taken into consideration. In this way, advanced middleware platform must support further developing applications and be able to adapt them to change in resource availability to meet real-time QoS requirements.

2 Requirement Model

Requirement model includes user requirement model and resource requirement. In addition, QoS is the outcome of the interaction of user requirement, and system behavior and resource requirement under that system behavior. Uncontrollable QoS will be caused by unknowing about any one of the three factors. Only accurately describe these factors and their dependency relationship, can the best application performance be achieved.

2.1 User requirement

User constraints can be divided into soft constraint and hard constraint[1]. Hard constraint emphasizes
certain performance index must be achieved. It is the bottom line that user can tolerate. The soft constraint stands for the anticipated application performance. The more the application is close to it, the greater utility this application will achieve, and user will get more contentment from it. The service supplied based on context, mainly focuses on two universal performance indexes:

**Definition 1** The average service-response time $R^s$. The average time spent on requesting a service from server and responding to client after processing service.

**Definition 2** The average service throughput $T^s$. The amount of the service having been performed in the unit time.

Among these two performance indexes, what the users are concerned about is the maximum average response time $R^s_{\text{max}}$ and the minimum average throughput $T^s_{\text{min}}$. $R^s_{\text{max}}$ appoints the maximum average service-response delay time which user can tolerate. $T^s_{\text{min}}$ appoints the lowest capability of service execution which users can tolerate. Thus both $R^s_{\text{max}}$ and $T^s_{\text{min}}$ describe the hard constraint of the application performance. It is described by formula as follows.

$$ R^s \leq R^s_{\text{max}} \land T^s \geq T^s_{\text{min}} $$

It defines performance deviation which expresses the deviation extent of the actual value of performance index to the value of hard constraint. So response time deviation here can be defined as follows.

$$ \Delta R^s = (R^s_{\text{max}} - R^s)/R^s_{\text{max}} $$

Simultaneously throughput deviation here is defined as follows.

$$ \Delta T^s = (T^s - T^s_{\text{min}})/T^s_{\text{min}} $$

On the basis of performance deviation, it defines function Utility to describe the soft constrain. Utility here be defined as follows.

$$ Utility = \sum_{s \in S} w_s \times (w^R_s \times \Delta R_s + w^T_s \times \Delta T_s) $$

From the function Utility which is described by simply using the linear relationship of performance deviation, it can be found that when the response time become less, the throughput will become more, the performance deviation will become bigger and the value of Utility will be greater according to Utility formula above. This analysis shows that the value of Utility can directly reflect the satisfaction degree of the soft constraint. In formula $Utility$, $S$ denotes the services set which component-based application can provide, and $w_s$ denotes the weight of different service.

$$ \sum_{s \in S} w_s = 1 $$

For each service $s$, $w^R_s$ and $w^T_s$ respectively reflect different preference on throughput and response time.

$$ w^R_s + w^T_s = 1 $$

### 2.2 Resource requirement

For the component technology based on container architecture, each component runs in corresponding container. Application server use container to manage the execution of components. Container processes all component behaviors including interaction with external system and manages all kinds of system resources. Therefore the performance of the component-based application not only depends on itself, but also depends on the middle system where the application is deployed. Meanwhile the performance of middleware system depends a great extent on correct resource parameters configuration. The performance of component behavior depends on resource configuration which its container have applied for it. Concurrently access resource can lead to their competition, and this will have influence on requesting processing. As a result, the most important part of describing system performance is modeling their relationships.

**Definition 3** Assume that a composite service is composed by $n$ service components, then $DG = (SC, E)$ [2] denote the function relationship of composite service. Among them, $SC = \{sc_i, 1 \leq i \leq n\}$, $E = \{(sc_i, sc_j, 1 \leq i, j \leq n)\}$ [3].

One example is shown in figure 1. Every directed edge between two nodes represents existing control-flow or data-flow. In addition to meet function dependency between components, each composite service still has to meet QoS requirement between each component. In general, QoS characteristic of component can be described by triple as $QoS = (QoS_{in}, QoS_{out}, Re)$. $QoS_{in} = (q^1_{in}, q^2_{in}, \ldots, q^n_{in})$ denotes the $QoS$ list whose adjacent component instance have to be achieved when running a component instance. $QoS_{out} = (q^1_{out}, q^2_{out}, \ldots, q^m_{out})$ denotes the $QoS$ list whose adjacent component instance provides when running a component instance. $Re$ denotes the demand of system resource like CPU, memory, network bandwidth and so on, when running a component instance. The relationship among $QoS_{in}$, $QoS_{out}$, and $Re$ is that the attribute value of $QoS$ in
Definition 4 Re = (r1, r2, . . . , rk) is a resource requirement list of a service component. Among them, k denotes k different styles of available resource like CPU, memory and network bandwidth and so on. ri (1 ≤ i ≤ k), a specific value, describe the quantity needed for resource of number i like rcpu = 20% or rmemory = 2 MB.

Definition 5 Assume that n service components are running in system and each component has k different styles of resource requirement, then the total of resource of these n components request can be described in following formula[4].

\[ \sum Re = (\sum_{i=1}^{n} r_{i,1}^i, \sum_{i=1}^{n} r_{i,2}^i, \ldots, \sum_{i=1}^{n} r_{i,k}^i) \quad (7) \]

Definition 6 Assume that n service components are running in the system, available resources can be described like RA = (ra1, ra2, . . . , rak). And the needed resource of service component \( \sum Re \) is calculated according to formula (7). If these n service components can be instantiated and can run in the system, following formulas holds.

\[ \sum Re \leq RA \cap (\forall j \cdot 1 \leq j \leq k \cdot \sum_{i=1}^{n} r_{i,j}^i \leq ra_j) \quad (8) \]

In other words, only resource requirement of running service component in equipment be satisfied as above inequality, this service could be deployed successfully.

Definition 7 A service composed by n service components has k different styles of resource requirements, and then SRC is defined as follows.

\[ SRC = \sum_{i=1}^{k} w^i r^i + \sum_{1 \leq i,j \leq n} i \neq j w^i r_{scij}^k \quad (9) \]

Among them, \( w^i \) denotes the resource weight which means importance degree of i resource. This indicates that the value of \( w^i \) become bigger, this kind of resource will have more difficulty in applying. \( \sum_{i=1}^{k} w^i = 1 (0 \leq w^i \leq 1) \). \( sc_{ij} \) denotes dependency relationship between component sc_i and sc_j. If sc_i and sc_j are deployed on same equipment then \( r_{sc_{ij}}^k = 0 \), otherwise \( r_{sc_{ij}}^k \) denotes the network bandwidth occupied by these two components for their communication. The value of SRC depends on \( r^i \), weight \( w^i \) and \( r_{sc_{ij}}^k \) according to definition 7.

In general, the higher demand level of QoS, the better quality of the service, the more demand quantity for resource, the bigger value of SRC. But the distributed resource is often very limited and therefore the amount of distributed resource is often less than the best quality of service demands. From the above analysis, it can easily be found out that when the value of SRC becomes bigger, the time waiting for satisfying with needed amount of resource will be longer. Waiting time become longer means the response time of service increases and the quality of service decreases. As a result, the value of SRC can be used as a reference point for assessing resource allocation strategy.

3 Adaptive Middleware Framework

In this section, we will introduce the architecture of adaptive middleware framework and the configuration algorithm of server parameter.

3.1 Architecture of adaptive middleware framework

Figure 2 shows an adaptive middleware framework[5] including load monitor, configuration selector, and QoS monitor.
QoS monitor is responsible for calculating the amount of completed service requests, each service’s response time, the average service-response time and throughput in every adjusted time interval. In addition to that, it also checks whether the QoS requirement is violated or not. All these statistic data will together determine when the application server needs re-configuration. When application need re-configuration, QoS monitor will inform configuration selector. After received re-configuration order, configuration selector will search configuration parameters in parameter table and select the most satisfactory configuration after considering each candidate configuration and dependency relationship between these parameters. Load monitor is in charge of calculating the amount of each service requests and partly control configuration selector according to user demand.

3.2 Configuration algorithm of server parameter

The goal of adaptive middleware framework is that under changeable load, when the hard constraint and resource constraint of each service has been satisfied, making soft constraint achieve the highest utility value through adaptively adjust the related server configuration parameters. About this adjusting, the key is how to determine the correct configuration parameters. In this paper, this framework uses following algorithm to search a point in configurable parameters table of server in order to meet hard constraint and maximize the utility of QoS. K dimensional vector $Re = (r_1, r_2, \ldots, r_k)$ denotes a kind of configuration defined Re contains k parameters and each parameter value has a given range. $r_i \in (QM_i^1, QM_i^2, \ldots, QM_i^n)$ means that each component demand for QoS model is not necessarily same. Thus here the value of n is also not necessarily same. Adjacent configuration set of $Re$ can be described as follows.

$$M_{Re} = \{Re + t_i\} \cup \{Re - t_i\} \quad (10)$$

$t_i(1 \leq i \leq k)$ is k dimensional vector equals $\{0, \ldots, 1, \ldots, 0\}$. The value of $i$ element is 1. And the remaining is 0. $Re_0$, a configuration parameter before adjustment, is the initial focal point in searching. In iteration, this algorithm will select the best configuration among adjacent configuration to focal point and obtain component interaction relationship which mainly contains dependency relationship including call sequence, call amount and degree of concurrency of call. This algorithm will check resource constraint according to definition 6. If resource constraint is met as formula (7), then the component will be instanced, else the algorithm continue to search another configuration. In the search process, the algorithm calculates the average service response time $R^s$ and the average service throughput $T^s$ in each candidate configuration. Vector $R = (R_{s1}, R_{s2}, \ldots, R_{sn})$ is the average service-response time of $s_1, s_2, \ldots, s_n$. Vector $T = (T_{s1}, T_{s2}, \ldots, T_{sn})$ is the average service throughput of $s_1, s_2, \ldots, s_n$. The utility value based on this configuration, is calculated by function Utility. $HC$ denotes the bottom line of performance user can tolerate. Function satifyser is used for judging whether $< R^s, T^s >$ meets $HC$ constraint. $HU$ denotes needed resource used for completing component deployment. Function satifye source is used for judging whether $\sum Re \leq RA$ meets resource constraint. $RA = (ra_1, ra_2, \ldots, ra_k)$. Function Predict is used for calculating service performance. The algorithm is described as follows.

START;

$Dres = 1$;
$Re_{cur} = Re_0$, $Re_{new} = Re_0$;
REPEAT

Improved = FALSE;

$< R^s, T^s > =$ Predict($Re_{cur}$);

MaxUtility = Utility $< R^s, T^s >$;

FORE Each Rein $M_{Renew}$

$< R^s, T^s > =$ Predict($Re$);

IF( satifyser($< R^s, T^s >$, $HC$) AND satifye source($Re$, $HU$) AND Utility($< R^s, T^s >$) > MaxUtility)

THEN

MaxUtility = Utility($< R^s, T^s >$);

$Re_{new} = Re$;

Improved = true;
ENDIF
ENDFOR

$Re_{cur} = Re_{new}$;

$Dres = Dres + 1$;
UNTIL( Improved = FALSE OR $Dres = MaxDres$);
END.
It’s easy to realize using this algorithm, and the convergence speed is almost same as other intelligent search methods, when the state space is relatively small. The operation efficiency is affected by amount of the equipment and component. After configuration, which is evaluated by SRC value, the paper sets up the weight according to the actual resource.

4 Experiments

4.1 Case study

We realize the self-adaptive frame using JRE1.5 on Tornet5.5. In order to test the performance guarantee effect of the frame, the application of information inquiry system is taken as the example. With load changing, this method compares the difference between using and not using this frame. Here, difference mainly include average response time of system, through-put satisfaction extend to hard constraint and utility value defined by soft constraint. For easy experiment, application only provides inquiry service to find out matched information. As this kind of service, response time is a important sector for valuing the quality of service. We assume the response time weight is 1, through-out weight is 0, service weight is 0.8, and meanwhile change the quantity of requesting service to simulate the change of the load. According to formula (4), 

\[
\text{Utility} = \sum_{s \in S} w_s \times \left( w_s R_s^R \times R_s^R \right)
\]

The maximum response time is 500 ms, i.e. \( R_s^R < R_{max} \), \( R_{max} = 500 \text{ms} \).

4.2 Experiment results

The Fig. 3 shows the comparison on with and without adaptive frame as the quantity of request increases. As Fig. 3 shows, when load increases, the service response time without adaptive frame soars, with no guarantee on quality of service, while the result is still favorable with this adaptive frame.

5 Related work

At this moment, there are two major methods in guaranteeing the performance based on self-configuring, one is based on control theory, the other is based on performance model. There are several methods to guarantee the control strategies, such as non-linear control, heuristic control and fuzzy algorithm control. Feedback control is mostly applied in the control-based methods, which compares the performance deviation between measured value and expected value and accordingly provides adjustment strategy. To control-theory-based method, system is a black box, which has no way to judge the effect of the strategy, only to adjust in line with the external changes can the result be satisfactory. Performance-model-based method establishes various performance models concerning the system, in this way to assess the strategy to enhance the accuracy. Self-configuring framework discussed in this paper establishes a holistic performance model according to the relationship and dependence between the components.

6 Conclusions

This paper comes up with a middleware self-configuring framework based on the performance-demanding model, which adjusts the system parameters dynamically during the operation. When determining the strategy to be adopted, the framework can make use of the performance model to predict the effect of every optional setting. In order to choose the optional configuration, which enhances the accuracy of control. The self-adaptive middleware framework coming up in this paper can be applied into other component platforms, such as EJB, .Net and CCM. Our next work aims at two aspects. One is expanding this self-adaptive framework. The other is to apply the framework to more component middleware platforms and applications.

Acknowledgments

This research was supported by National Natural Science Foundation of China under Grant No. 60703088.
References:


