An Integrated Adaptive Management System for Cluster-based Web Services

Ying Jiang[2], Dan Meng[1], Chao Ren[2], Jianfeng Zhan[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100039
[2]Graduate School of the Chinese Academy of Sciences, Beijing 100080
{ cylinder, md, renchao, jfzhan }@ncic.ac.cn

Abstract

The complexity of the cluster-based web service challenges the traditional approaches, which fail to guarantee the reliability and real-time performance required. In this paper, we present an Integrated Adaptive Management System (IAMS) for such service. The issues we discuss address to efficiently allocate resources and provide more effective QoS support under a wide range of load conditions. For the global resource level, we introduce spare instance and corresponding management strategy as a supplemental adaptive mechanism. The spare instances hosted on shared node afford better resource utilization and more effective QoS support in the case of overload or workload fluctuation. Further, it can relax the influence of the fault recovery from the hardware and software failure. For the local level, we apply a multi-purpose linear-quadratic regulator (LQR) as basic adaptive element. The control scheme using reject time ratio as control input is able to provide guarantees for overload protection, resource control, Qos control, performance isolation, and effective management for spare instances.

Results of experiments on both static and dynamic web sites illustrate the efficiency and robustness of the multi-purpose LQR.

1. Introduction

Over the last few years, cluster systems have been gaining in popularity for providing web service such as commercial sites, financial services, education sites and so on. The internet application differs from traditional parallel jobs in several ways. A significant challenge is the workloads for web services tend to be bursty and fluctuate dramatically. For example, the daily peak-to-average load ratio at Internet search service Ask Jeeves (www.ask.com) is typically 3:1 and the peak loads can be an order of magnitude larger than the average and unpredictable in the presence of extraordinary events. Over-provisioning system resources for a service site to accommodate the potential peak will not be cost-effective[1]. In addition, the web services require the service-level agreements (SLA), which guarantee reliability, availability, and quality of the service that the businesses pay for. Furthermore, multiple service classes may be hosted on shared nodes to afford better resource utilization, which introduces the issue of performance isolation.

As an important and challenging topic, the adaptive mechanisms for managing cluster-based internet service have received wide attention in the research field. Some papers[2,3,4] address to dynamically allocate the cluster resources to guarantee contracted (SLAs). A few mechanisms have been proposed to support admission control[5,6,7,8]. Recently, control theory has been applied as an adaptive mechanism[9,10,11] in the context of web servers. For the limitation of space, we will not describe the related work in detail.

Although cluster-based web services have been widely deployed we have seen limited research in the literature on comprehensive adaptive mechanism for resource management with QoS support.

In this paper, we present an integrated adaptive management system (IAMS) for cluster-based web services. The main adaptive mechanisms in our system are:

- The resource adaptation based on the SLA event-driven mechanism
- A supplemental adaptive mechanism based on spare instances and the corresponding strategy.
- A multi-purpose control scheme incorporated in the local manager and/or application is used as the basic adaptive element to provide guarantees for overload protection, resource control, Qos control, performance isolation and management for spare instances.

The main contributions in our paper are the supplemental adaptive mechanism and the multi-purpose control scheme, which provide more adaptive

QoS support and improve the performance of cluster based web services during overload or instance failure.

The aim of our supplemental adaptive mechanism is to (1) provide additional resources for the service that is experiencing light overload; (2) offer temporary resources to the service which is enduring short-term severe overload and (3) supply a few shared resources for instance failure. As we know, in the case of light overload or short-term severe overload (the severe overload situation lasts in a short time) situation, simply rejecting excessive requests will lose customers and revenue of the websites, and overbooking resources is not a cost-effective way. Further, in the case of hardware or software failure or the SLA violation, the fault recovery may result in a gap of service. To resolve the above problems, our system provides an additional instance for the web service on few shared nodes. When encountered with the instance failure or overload, all or parts of requests will be redirected to the additional instance. The additional instance is the so-called spare instance. In this way, our system offer better resources utilization and improve the QoS performance for the web service on global resource level. The performance isolation in the shared node is also discussed in the paper.

Second, the multi-purpose control scheme functions as the basic element, combined with the spare instances to enforce policies for interrelated metrics. Our controller provides further adaptation and is applicable to various applications without changing the source code.

The features of our multi-purpose feedback control scheme are:

(1) Use reject time ratio (*RTR*) as control variable. For multi-purpose, the control input should be capable of affecting a variety of performance metrics in a meaningful way. *RTR,* the rejection time interval in control period, is proved to be the right choice.

(2) LQR based control design. Developing the controller based on the classical theory could not avoid the cumbersome trial and error approach. In addition, it is difficult to design MIMO system with classical theory. LQR approach based on modern control theory is directly performance oriented and can be used to design complex MIMO system easily. A QoS differential service scheme using multi-variable system is presented in the paper. Moreover, the input(s) in our scheme is closely related to the throughput which is an important QoS metric. LQR design approach offers negotiating trade-off between the control object and the input(s), which can avoid too many requests rejected and keep a desired throughput. This is another important reason for employing LQR

(3) System identification based modeling. By employing system identification, we obtain the service model without the knowledge of inner-mechanism about the web services. Further, the model is on a per-system basis, which may be more accurate than that obtained by theoretical approach.

For providing a full-scale evaluation, we present some experiments including overload protection, resource utility and QoS performance control both on Tomcat server and a 3-tiered web site. The results illustrate our multi-purpose control scheme can offer satisfied performance for both static and dynamic websites. The results also indicate the control mechanism can support QoS differential service.

The cluster operating system prevails in that it is becoming more and more mature and abundant basic services have been provided. Our management system is implemented on the Dawning 4000A super server, which are the biggest cluster systems for scientific computing in China.

The rest of this paper is organized as follows. Section 2, 3 introduce the architecture of IAMS and the SLA-based adaptive mechanisms briefly. Section 4 discusses the spare instance and its management. The multi-purpose feedback control scheme and the implementation of QoS differentiated service are discussed in section 5. Section 6 and 7 present the experiment results on Tomcat and a 3-tiered web site to illustrate the efficiency of our control scheme respectively. Section 8 summarizes our work and discusses the future work.

2. IAMS architecture

IAMS architecture features a hierarchical design with multiple tiers. Being the first tier, global resource manager focuses on adaptation properties of the global computing environment. The global manager selects corrective actions when a service level threshold is exceeded or when server failures occur. The second tier is local manager, which maintains system-wide properties of a single server, such as adaptation agility, system utility, or the fairness between the independent applications. Besides launching and terminating the applications as the global manager instructs, the local manager performs the resource allocation. The third tier is a feedback control system and formed by the service instance, the controller and the detect agent. This tier focuses on application-specific adaptation choices and meets the needs of individual applications.

A broad architecture overview is given in Fig. 1. A、B、D each represents an application.

The detector agent is a basic service provided by the Dawning 4000A. Detect agent local to each server collects the application resource utility and the performance metrics. Besides, the detect agent issues the meta-event when the system resource metric or the service level threshold is exceeded.

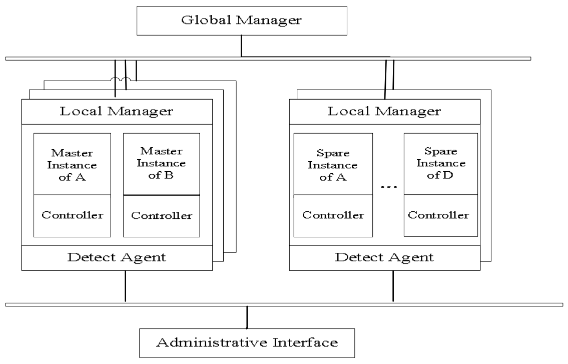

Fig. 1. IAMS architecture

On the right hand of the Fig. 1, there are some spare nodes specialized for the spare resource pool. While the application starts up, an additional instance which is called spare instance is also started in the spare resource pool simultaneously. If the corresponding master instance fails or the SLA violation occurs, requests are redirected to the spare instance. The details of the spare instances will be explained in section 5.

3. Adaptive resource scheduling

The global and local resource managers perform the resource scheduling all together. Dynamic resource configuration provides scalability, high-availability, increased fault tolerance, when encountered with the server failure, the change of the SLA requirement, etc.

3.1. SLA-Based global resource manager

The global resource manager is event driven. When a service level threshold is exceeded, or server failures occur, an event will be sent to the global manager. Then the unit may choose a proper actions include modifying server-set assignments, throttling incoming request streams, initiating recovery actions, and issuing Administrator alerts.

The preferred remedies to the given problem are listed in detail:

1. On detecting the server failure, the warnings are issued to the global manager. For every service instance on that server, the global scheduler inquires the resource configuration database about other potential server, which could offer the same type of service. If any is found, the original instance will be migrated to the new server, and the IP sprayer automatically will bind with new instance. In addition, when the fault recovery procedure is in process, part of the requests will be redirected to the spare instance to avoid the service breakdown and the decline of SLA.
2. The global manager will instruct the corresponding local manager to restart the instance, once the service instance crashes or behaves abnormal. Also the requests will be sent to the spare instance during the fault recovery.
3. On detecting the SLA violation, the warnings are issued to the manager and penalties as stated in the SLA are collected. Some requests are redirected to the spare instance instantly. A timeout event will be sent if the spare instance exceeds the time limit. The manager then determines the spare instance as the long-term overload, and reassigns an underutilized server to launch a new instance. On the other hand, the manager may cut down a running instance, if the load is rather light.
4. In another scenario, the SLAs with a component service are not violated. But, in order to keep up with the demand of its customers, who requires a higher level of service, the global manager dynamically negotiates a new SLA (i.e. different attributes, perhaps at an increased cost).
5. On detecting the resource capacity change (for example, a new server is added to the cluster), the global manager updates the configuration database.
6. When a resource allocation request is accepted, the manager pre-allocates servers and other global resources based on SLA requirements. It reclaims the resources if the application terminates.

In addition, the global resource manager provides interface for the customized policy.

3.2. Local resource management

Local manager combined with detector agent and resource controller(s) provides local resources management. The resource controller(s) (or application controller) can achieve overload protection, performance guarantees, and service differentiation in the presence of load unpredictability. Further, it ensures performance isolation for shared node.

The local manager is the link between the global resource manager and the service instance. The main functions are listed below:

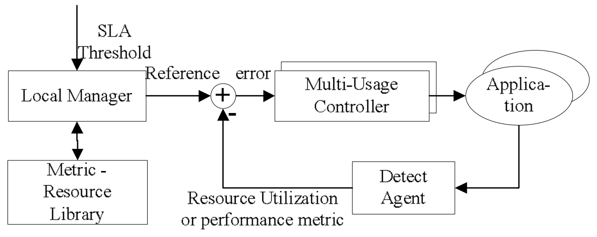

Fig. 2. Local Manager and control loop

- The local manager launches and terminates the instance and the corresponding controller according to the instruction of the global manager
- In order to control the application resource and guarantee the application performance, the local manager sets the set point of the controller shown in Fig. 2. In IAMS, the set point may be the resource quota or the contracted performance limit. The set point can be modified during the life cycle.
- The local manager sends an event-driven signal to global manager when the SLA violation occurs.

4. Spare instances and schedule strategy

4.1. Spare instance

On detecting the hardware or software failure or the SLA violation, the traditional policy adopted is choosing a free server to recover or replicate the service instance. However, for the server failure it may bring on a gap of service during the fault recovery. For the SLA violation caused by workload surge, it may result in frequent instance start/stop and unnecessary system overhead. Moreover, the fluctuating workloads always cause service in light overload state, which would lead to performance drop.

Hence, we introduce an additional instance to which the requests will be immediately redirected, in case of the related service failure or overload. The additional instance launches as soon as the related service starts up and has same life-cycle with this application. The additional instance is just the spare instance, while the other common instance of the service is called the master instance. Since only a part of web services may be overloaded and few will be serious at a time, spare instances for different service can be allocated on a shared node for making better use of system resource. The shared node on which spare instances host is called spare node.

As afore-mentioned, our aim is to provide addition resources for the lightly overloaded services and offer temporary resources for the service that is being short-term severe overload or instance failure, in the term of supplying few shared resources (spare nodes). We should guarantee the fairness and resource availability in the spare node, thus none of the spare instance is allowed to occupy excessive resource for long term. IAMS defines a *quota limit* for every spare instance. The quota is set evenly or according to their weights.

Three spare instance states are defined according to the resource utilization during the run time:

SPARE: the basic state of the spare instance, in which no requests are redirected to the instance and the instance consumes little resource.

NORMAL: The instance is marked as NORMAL when its resource consumption is below the quota limits.

EMERGENT: The instance is placed into EMERGENT when the resource consumption exceeds the quota limit. It is a temporary state.

Both of the EMERGENT and NORMAL state are working state.

4.2. Spare node manager

The structure of spare node is similar to other node, but a resource utility based controller must be embedded in every spare instance to ensure performance isolation. Spare node manager is a special local manager. The additional responsibilities are:

Dynamically allocate and set the reference (resource utility) for spare instance controllers according to the number of working instances and their workloads in the spare node, so as to afford higher resource utilization and performance isolation. The reference value could be larger than its quota. A single spare instance can even occupy almost whole node resources temporarily, if no other instance is working.

Spare instance is allowed to occupy the excessive resource in a short time because of severe overload. But if the time is out, the local manager will set the controller reference to its quota, imposing the resource utility reduced to its quota. Meanwhile, the local manager will send an event to the global manager asking more resources for the corresponding service.

4.3. Spare instance state transition

Originally the spare instance is in SPARE state. When the master instance fails or server overload occurs, some requests are redirected to the spare one, which leads to the state transition. And whether the state changes to NORMAL or EMERGENT depends on the resource occupied. Further, the instance in NORMAL state may transfer to EMERGENT if the situation is even worse. Similarly, the instance may transfer from EMERGENT state to NORMAL or SPARE if the fault recovers or the overload disappears. All the resource is freed when the state finally returns to SPARE. Sometimes the situation may not turn better for a long time and excessive resources keep occupied. To assure the availability of the spare resources, we limit the interval that the instance can be in EMERGENT state. In breach of the time limit, it will be imposed to NORMAL by the local manager. Fig. 3 gives state transition diagram.

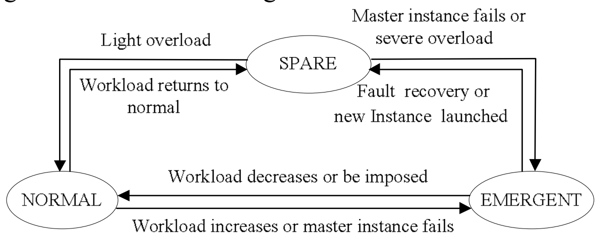
Light overload
Master instance fails or severe overload
SPARE
Workload returns to normal
Fault recovery or new Instance launched
Workload decreases or be imposed
NORMAL
EMERGENT
Workload increases or master instance fails

Fig. 3. State transition diagram of spare instance

We have made a simple comparison. In our system the elapsed time of putting the spare instance into use is about 0.12 sec, while the time taken for the traditional policy is 1.05 sec for the Apache service and for Oracle 10g it is 7.69 sec.

In summary, the spare node(s) contributes to a spare resource pool. In the case of the master instance's failure, the spare instance avoids service breakdown and the decline in SLA, it improves the fault tolerance and contributes to provide non-stop service. For short term severe overload, the available resource could be enlarged instantaneously, much quickly without launching a new instance temporarily. And it can supply additional resource when slightly overloaded for a long time. By introducing the spare instance, the adaptive capacity of the application would be greatly improved in the case of overload and the workload surge.

5. The basic adaptation element —— multi-purpose control scheme

In this section we discuss the design and implementation of the multi-purpose feedback control scheme. In the end of this section, we extend it to implement QoS differential service.

The multi-purpose control scheme using reject-time-ratio (*RTR*) as control input is shown in Fig. 4. In this figure, the output is performance metric measured periodically by the detector and reference is the desired value of output. The scheme consists of LQR, AC actuator and the web service itself. The actuator is responsible for rejecting or redirecting incoming requests in rejection time interval according to the *RTR* given by the LQR. The LQR produces optimal control signal *RTR*, so as to ensure the output meet the desired value and make suitable trade-off between performance error and the throughput.

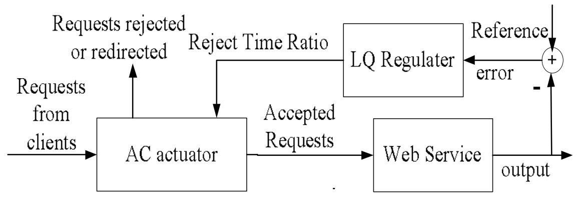

Fig. 4. Block diagram of control scheme

As *RTR* can affect a variety of performance metrics in a meaningful way, this scheme can be used as multi-purpose controller. When regarding resource utility as the output, it would be a resource utility based controller which can be used to prevent overload or performance isolation. In other case, when stressing on a specified performance metric or lacking of the knowledge of the bottleneck of resources, the performance metric, such as throughput, response time should be used as output and then it could be a QoS metric based controller aiming to control QoS metric.

5.1. Admission control using reject time ratio

Admission control is an effective technique to afford QoS support under overload condition. The idea is reducing the amount of work required when faced with overload by dropping a portion of the requests. By this way, the server can service the accepted requests faster and meet the QoS performance. However, dropping too many requests would cause revenue loss. An online feedback based admission control scheme is illustrated in Fig. 4. A controller periodically takes resource utility of the node (bottleneck for service) or performance measurements of the Web service from a detect agent, compares it with the reference (desired value), and adjusts the admitting probability (*P*) to meet the control goal. The changes to the admitting probability can be actuated through an admission control (AC) module. Through the AC module, a request is being accepted with probability P_a, and being dropped with probability $P_d = 1 - P_a$ [12]. Based on the fact of the ratio of the time for rejecting requests to the control period (Reject Time Ratio) is proportional to drop probability P_d, it is used as an input for admission control in our system.

We define reject time ratio as follows

$$RTR = t_{rjt} / T_{ctrl} \qquad (1)$$

where T_{ctrl} is the control period. t_{rjt} is the time interval in which all requests are rejected, and in the left time, requests are admitted.

If in rejection time interval, only the requests of new sessions are rejected or redirected to keep the session integrity, it would be a session based admission control applied for the session based applications such as commercial services.

5.2. Modeling

The dynamic model, that describes the mathematical relationship between input(s) and output(s), is the basis for controller design. For lack of detailed knowledge about the inner workings of the web service, we adopt system identification approach to establish the models. The open-loop service is modeled as a difference equation with unknown parameters. Then we stimulate the web service with pseudo-random signal as inputs and sample output data. After collecting the data, the least square estimation of model parameters can be solved via MATLAB.

5.2.1. Model structure. Difference equation with unknown parameters is a common used dynamic model. In most cases the first order or second order Difference model is exact sufficiently. We adopt first order difference equation

$$y(k) + ay(k-1) = bu(k-1) \quad (2)$$

to describe the web service.

5.2.2. Input signal. The input signal should be persistently exciting to satisfy the identifiable condition. Pseudo-random digital white noise and pseudo-random binary sequence (M-sequence) have been widely used for system identification. In our experiments, M-sequence is used as input signal to randomly switch RTR.

5.3. Linear quadratic optimal design

To design linear quadratic optimal controller, state equations are required to describe the control system. A state representation and state feedback equation are shown as (3) and (4) respectively.

$$x(k) = Ax(k-1) + Bu(k-1) \quad (3)$$

$$u(k) = Kx(k) \quad (4)$$

Here, *x(k), u(k)* is state vector and control vector at instant *k* correspondingly, ***K*** is control gain matrix. The aim of controller design is to find the proper gain matrix ***K*** to minimize the cost function (5)

$$J = \sum_{k=0}^{\infty}\left[x^T(k)Qx(k) + u^T(k)Ru(k)\right] \quad (5)$$

Q is a positive semi-definite state weight matrix, and *R* is the control weight matrix which is positive definite. *Q* and *R* are usually diagonal matrices. This approach involves following main steps.

(1) Select state variables and obtain state equations for the system. The state variable should be relative to performance metric.

(2) Determine weight matrices *Q* and *R* to get satisfied trade-offs among different states (performance metrics) and the magnitude of inputs. A basic principle for selecting weight matrix is: the more important the state, the larger the weight.

(3) Solve the Riccati equation to determine the optimal control gain matrix.

(4) Check whether the stability and dynamic character of the close loop system are satisfied. If not, repeat step (2).

The main problem for LQR design is to select state variables and weight matrices carefully.

5.4. Implementation of the controller

5.4.1. Selection for state variables. Assume that the open loop model has already been obtained and shown as (2). Since the close loop system works as Fig. 4, we select the states as

$$x(k) = [x_1(k), x_2(k)]^T$$

$$x_1(k) = e(k), x_2(k) = \sum_{j=1}^{k} e(j) \quad (6)$$

Thus, the control input *rtr* (reject time ratio) would be

$$rtr(k) = k_1x_1(k-1) + k_2x_2(k-1) \quad (7)$$

The corresponding cost function is

$$J = \sum_{k=0}^{\infty}\left[x^T(k)\boldsymbol{R}x(k) + rtr^T(k)\boldsymbol{Q}rtr(k)\right] \quad (8)$$

The state equation translated from difference equation (2) is shown as (9).

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} -a & 0 \\ a & 1 \end{bmatrix}\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} b \\ -b \end{bmatrix} rtr(k) \quad (9)$$

5.4.2. Selection for weight matrices. As the transient error is more important than accumulate error, and the input *rtr* is direct proportion to the rejected requests, too smaller weight for it would result in excessive reduction of throughput which is an important performance metric. We select a larger weight for state x_1 (transient error), and same weight for x_2 and *rtr*, then we find the optimal control gain using MATLAB. If the stable and dynamic indexes are satisfied, the controller has been designed successfully.

5.5. Extending the controller to differentiated service

In this paragraph we discuss how to extend our controller designed by LQ optimal approach to support service differentiation. We consider three classes of quality of services. All clients are classified into VIP, premium and basic classes. The requests from VIP (few clients with the highest priority) will never be rejected, i.e. the *RTR* for VIP is zero for ever. The premium client who has the middle priority gets better quality of services than the basic one. The percentage of premium requests or sessions that completed successfully should be far more than that of basic during overload. That means the *RTR* of basic should be (much) larger than that of premium. The differentiated service can be implemented by differentiated control. As we know, in the LQR design the trade-off between different inputs can be achieved by adjusting the weight matrix of inputs and in cost function (5), an input with a heavier weight would has a smaller control gain. Therefore we select a larger weight for the input of premium, then the control value for the premium is sure to be (far) less than that of the basic, and the throughput of premium would outweigh that of basic. A differentiated service control scheme based on above idea is shown in Fig. 5.

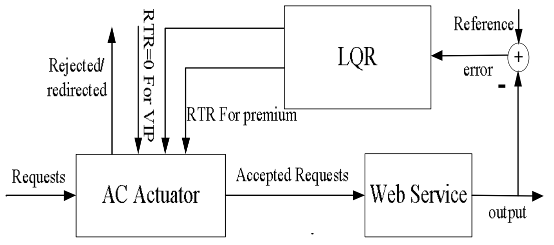

Fig.5. Differentiated services control

In this scheme, two LQ regulators are employed for controlling the premium and the basic requests respectively. The web service with two inputs and single output can be described with a MISO difference equation, which is expressed as

$$y(k)+ay(k-1)=b_1u_1(k-1)+b_2u_2(k-1) \quad (10)$$

The unknown parameter in (10) can be achieved by system identification and the state representation can be translated from (10). Selecting appropriate weight for *Q* and *R* in cost function (5), satisfied QoS differentiated services can be obtained. By changing the weights in *R*, the completed percentage ratio of the two classes is tunable. An experiment of differential QoS service is presented in next section.

6. Experimental results on Tomcat

In this section, we provide some experiments including overload protection, resource utility and QoS performance control on Tomcat. Since most commercial web services are session based, all experiments are the session-based.

6.1. Testbed

The testbed consists of three Pentium IV computers as client running the workload generator. Tomcat 5.5 runs on a Pentium III 500MHz, with 512MB RAM as server machine. All the machines run Linux Kernel 2.4.21 and are connected through a LAN of 100Mb/s. In such circumstance, the capacity of Tomcat is less than 20sess/sec, for in that time the CPU utility is larger than 0.95.

6.2. Workload

The workload was generated by Httperf[13], a workload generator and performance measurement tool, which support to generate HTTP/1.1 requests and manage user sessions. We use a simulation model with following parameter:

- The session length is exponentially distributed with a mean of 15,
- A timeout — the time client waits for a reply before resending the request — is set to 10 seconds,
- Think time between the requests of the same session is exponentially distributed with a mean of 5 seconds,
- The number of retries to resend the request after timeout is 1.

Throughout this section, we consider a file mix as defined by SpecWeb99[14]. Since the web pages are all static page, a series of experiments in the workload can prove the adaptive effect for static website.

6.3. Experiments for CPU utility based controller

6.3.1. Overload Control. The CPU utility based controller is used in the experiment because CPU is the bottleneck of resources in our experiment environment. CPU utility is regarded as output and reference is set to 0.9. The load-performance comparison given in Fig. 6 indicates the throughput in completed session of augmented Tomcat keep in a higher level and the average response time maintains within an acceptable extent (82~133ms), when workload changes greatly. If the request is rejected, we send back a clear message of rejection to prevent clients from unnecessary retries. Since issuing an explicit rejection message incurs an additional load on web server, the throughput of augmented Tomcat is a little less than the max throughput of the original one.

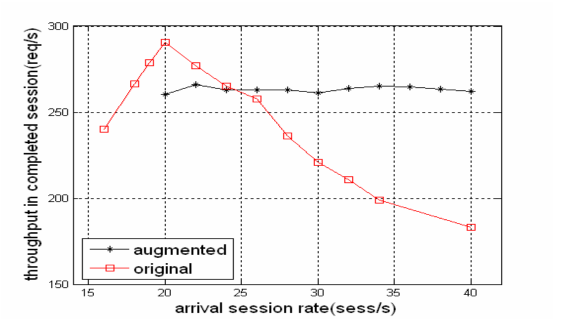

Fig. 6(a). Comparison of throughput in completed sessions

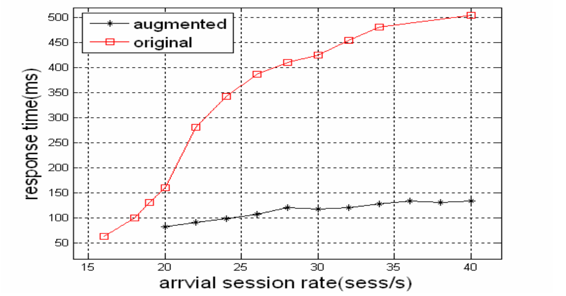

Fig. 6(b). Comparison of response time

6.3.2. CPU utility control. In the experiments below, Tomcat and other applications run on the same server. The desired CPU utility of Tomcat is 50%

Under a varying workload shown in Fig. 7(a), the result shown in Fig. 7(b) illustrates the CPU utility of the original Tomcat fluctuates with the load in a large scale, and that of the augmented Tomcat fluctuates around 0.5 with a mean of 0.49 and a mean square variance of 0.004.

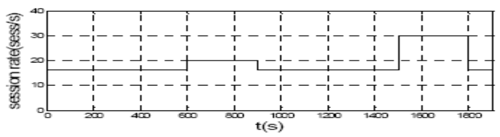

Fig. 7(a). Workload pattern

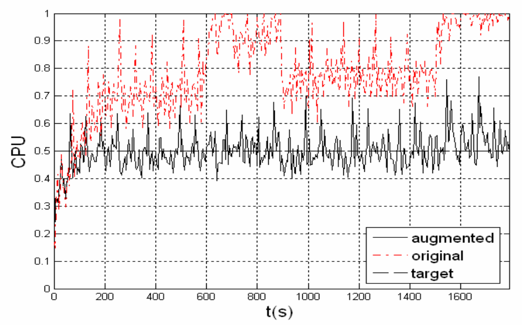

Fig. 7(b). Comparison of CPU utilization

6.4. Experiment for response time based controller

In the experiment we set reference value 100ms. Fig. 8 illustrates the average response time of augmented Tomcat bound at about 100ms. Even if the workload is high up to 40sess/s (about twice the capacity), the response time being 105ms is a little more than the desired value.

Fig. 9(b) shows the response time of augmented Tomcat fluctuates near the 100ms under varying work-load shown in Fig. 9(a), while that of original Tomcat changes with workload and is extremely greater.

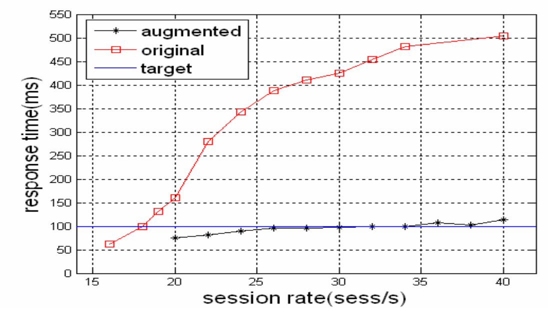

Fig. 8. Average response time comparison

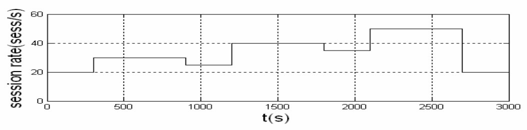

Fig. 9(a). Workload pattern

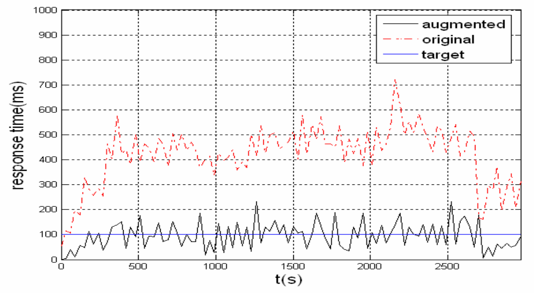

Fig. 9(b). Response time control in fluctuating workload

6.5. QoS differential service

In this paragraph, we provide the result of QoS differential service experiment, applying control scheme shown in Fig. 4. For simplicity, only two classes (premium and basic) are considered. In this experiment, CPU utility based control is used and its reference is set to 0.9. The weight matrices are selected as *Q=diag([1,0.1]),R=diag([0.1,0.001])*, the weight of premium is 100 times of that of basic. The premium client sends the requests at the same session rate as that of the basic client. Fig. 10 demonstrates the comparison of percentage of completed session. When in the normal load, the completed percentage are both close to 100%. As the load rises up, the percentage of the premium descends slightly, but that of basic drops drastically. When the load reaches 40sess/s, the percentage of the premium is about 75% which is four times of that of the basic. The result indicates the control scheme can provide differentiated service effectively. By changing the weights of elements in *R*, the percentage ratio of the two classes is tunable.

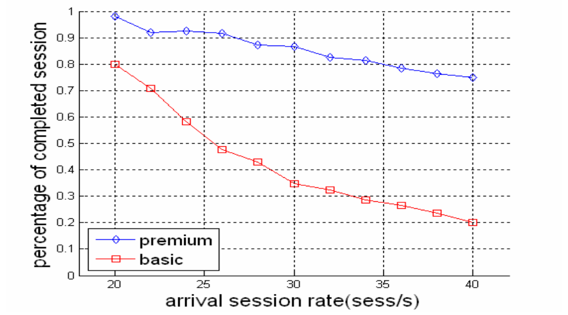

Fig. 10. Percentage of completed session

7. Experimental results in 3-tiered web site

To provide a full-scale evaluation of our multi-purpose control scheme, this section will show some experiment results in a 3-tiered web site. The experiments prove the control effect for dynamic website.

7.1. Testbed

The structure of our testbed is shown in Fig. 11. It consists of a client node and two server nodes. Each node is dual Opteron processor 1.5 GHZ, 2G RAM with Gigabit Ethernet connected point-to-point full duplex with the switch. One server node runs the Web server and application server software, while the other contains the database. The client node drives the system with the standard workload generator TPCW. All nodes run Red Hat Linux with the Linux kernel 2.4.21. We use Apache v2.0 for the front-end Web server, Jakarta Tomcat v5.5.12 as the application server and MySQL v5.0.18 for the database server. The controller is place in the proxy.

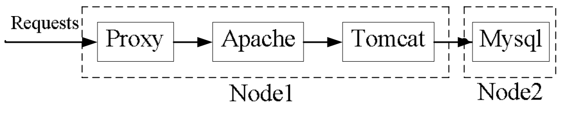

Fig. 11. Structure of 3-tiere web site

7.2. Workload Generator

We use TPC-W[15] for our e-commerce Web site testbed. It implements all functionalities that typical e-commerce Web sites provide. A Java implementation of TPC-W from the PHARM group at the University of Wisconsin [16] is modified to make it compatible with the newest version of Tomcat and MySQL installed in our testbed. It implements all functionalities in TPC-W specification. The database is configured to contain 10,000 items and 288,000 customers. Think time is exponentially distributed with a mean of 0.7 seconds and bounded at a maximum of 7 seconds.

7.3. Experimental result

We present the results both of the CPU utility based overload control and the response time based control experiments on 3-tiered website. For the limitation of space we will not give detail description for them. The advantage and efficiency of the multi-purpose control scheme is obviously.

7.3.1. CPU utility based overload control. In our experiment, the bottleneck of resources is the CPU utility of the database MYSQL. We use it as output and set the reference to 0.85. Fig. 12(a) and 12(b) is the throughput and response time comparison for the controlled and un-controlled 3-tiered web site.

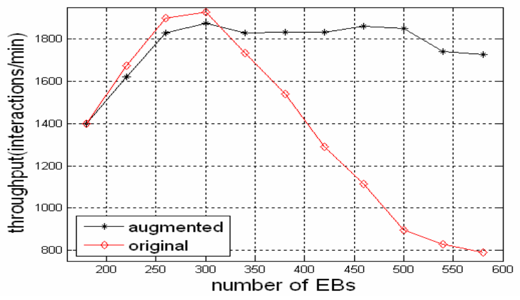

Fig. 12(a). Throughput comparison

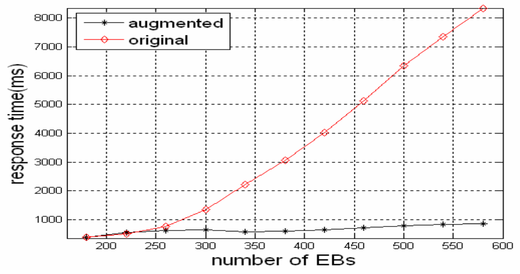

Fig. 12(b). Comparison for response time

7.3.2. Response time based control. Here we present the results of two response time control experiments. Fig. 13 is for average response time control, and Fig. 14 is for transient response time control under a varying workload. The setting value is 600ms. The results show the response time of controlled web site can track the setting value well.

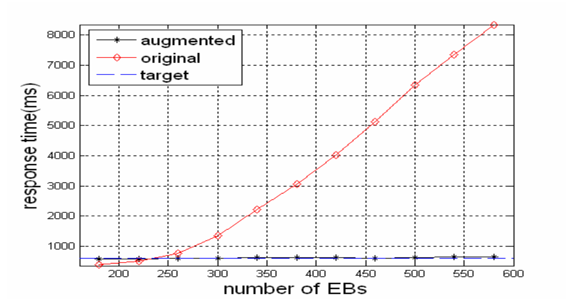

Fig. 13. Comparison for average response control

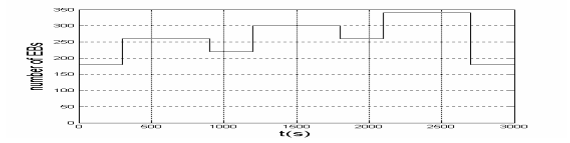

Fig. 14(a). Workload pattern

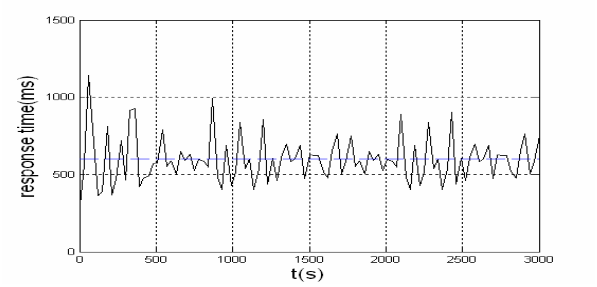

Fig. 14(b). Controlled 3-tiered web site

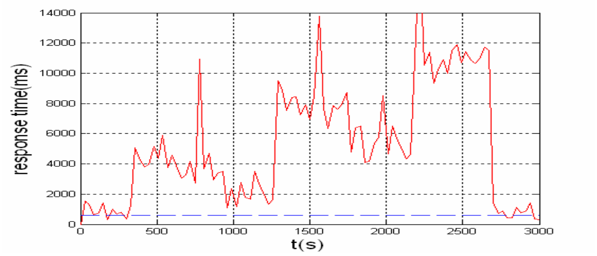

Fig. 14(c). Response time of un-controlled 3-tiered web site

As shown, the control scheme can be flexibly configured to the dynamic website and achieve satisfied result.

8. Conclusion and future work

In this paper we present a SLA event-driven based integrated adaptive management system (IAMS) for cluster-based web services. The system introduces spare instances as a supplemental adaptive mechanism and a multi-purpose feedback control scheme as the basic element to enforce the policies for interrelated metrics. The controller provides flexible adaptation and can be extended to support QoS differentiated service. In addition it can be easily configured for various applications, without changing the source code.

We have presented rich experiments including overload protection, resource utility and QoS performance control both on static and dynamic website. The results illustrate our multi-purpose control scheme can offer effective QoS differential service and satisfied performance. Now the controller as the adaptive element only provides throughput different-tiation. In the future, we will make attempts to improve the differential service, such as providing response time differentiation in more complicated environment.

9. References

[1] K. Shen, H. Tang, T. Yang, and L. Chu, “Integrated Resource Management for Cluster-based Internet Services”, In OSDI 2002.
[2] Sara E. Sprenkle, “Exploring Availability and Usage Guarantees in Resource Allocation Through Leases”, Duke University Computer Science, 2004.
[3] K. Applby, S. Fakhouri, L. Fong, *et al.*, “Oceano -- SLA Based Management of a Computing Utility”, the IFIP/IEEE Symposium on Integrated Network Management, 2001.
[4] K. Hartig, D. Reedy, “Associating Service Level Agreements to Applications in a Dynamic Environment”, http://rio.jini.org/.
[5] T. Voigt, R. Tewari, D. Freimuth, and A. Mehra, “Kernel Mechanisms for Service Differentiation in Overloaded Web Servers”, the USENIX Annual Technical Conference, 2001.
[6] N. Bhatti, R. Friedrich, “Web Server Support for Tiered Services”, IEEE Network, vol. 13, no. 5, pp. 64–71,1999
[7] S. Elnikety, E. Nahum, J. Tracey, W. Zwaenepoel, “A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites”, In Proc. of the Int'l WWW Conf, 2004
[8] L. Cherkasova, P. Phaal, “Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites”, IEEE TRANS. COMPUTERS, vol. 51, no. 6, 2002
[9] Y. Diao, N. Gandhi, J.L. Hellerstein, *et al.*, “Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web”, NOMS 2002.
[10] C. Lu, T.F. Abdelzaher, J.A. Stankovic, and S.H. Son, “A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers”, RTAS, 2001
[11]A. Kamra, V. Misra, E. Nahum, “Yaksha: A Self-tuning Controller for Managing the Performance of 3-Tiered Websites”. IWQoS 2004
[12] Xue Liu, Jin Heo, Lui Sha,Xiaoyun Zhu, “Adaptive Control of Multi-Tiered Web Application Using Queueing Predictor”, NOMS 2006
[13] D. Mosberger, T. Jin, “httperf: A Tool for Measuring Web Server Performance”, in First Workshop on Internet Server Performance, 1998
[14] “The Workload for the SPECweb99 Benchmark”, http://www.spec.org/web99/, 2002.
[15] D.A. Menasce, “TPC-W: A Benchmark for E-commerce”, IEEE Internet Computing, 2002
[16] T. Bezenek, H. Cain, R. Dickson, *et al.*, “Characterizing a Java implementation of TPC-W”, 3rd Workshop On Computer Architecture Evaluation Using Commercial Workloads, 2000