An Approach to Adaptive Performance Tuning of Application Servers

Giovanna Ferrari,
School of Computing Science
University of Newcastle, UK
giovanna.ferrari@ncl.ac.uk

Santosh Shrivastava,
School of Computing Science
University of Newcastle, UK
santosh.shrivastava@ncl.ac.uk

Paul Ezhilchelvan
School of Computing Science
University of Newcastle, UK
paul.ezhilchelvan@ncl.ac.uk

Abstract

Controlling Quality of Service (QoS) offered by application servers involves selecting appropriate configuration and parameter tuning that would match the application level load. This is a challenging problem, since application servers have many parameters at different levels that often influence the QoS in complex ways. This paper presents an empirical approach to keep the QoS offered close to the level specified by the hosted applications, using on-line configuration tuning. Based on this approach, an automated tuning system is currently under development to monitor and adapt the application server performance under variable load conditions.

1. Introduction

An e-Business site should provide specific services to its customers with agreed QoS attributes such as end-to-end response time, site response time, throughput in requests/sec and so forth. However, the unpredictability of the environment complicates the problem of maintaining adequate QoS levels. In fact the workload of an e-Business site could be highly dynamic and may have load spikes that far exceed the average load; besides, there can be variations in the resource requirements, due to node failures or troubles with the internal software. When workload surges appear, performance degrades significantly.

In the work presented here, we have addressed the problem of maintaining adequate QoS levels under such conditions by implementing an automated tuning system based on an adaptive approach. It monitors performance of the application server, which can be considered the bottleneck of the e-Business platform. In case of variations of the load conditions, it applies an adaptive strategy by dynamically changing configuration parameters of the application server. The values to assign to each parameter are based on pre-computed experimental results. A benchmark test suite has been used to exercise the system and measure its behavior with different configuration settings.

Data has been collected on the most appropriate set of configuration parameters under a given loading condition; this data is used at runtime to change configuration settings as variations in load are detected.

A major challenge in performance tuning occurs because e-business IT activities exhibit an inherent complexity which developers find hard to quantify. While component-based technologies provide flexible infrastructure solutions for developing e-Business applications, they also introduce a multi-layer structure which admits various architectural trade-offs to be made and design patterns be used. Consequently, the task of performance tuning becomes complex, as the effects of many possible combinations of trade-offs need to be understood.

The problem is exacerbated by the fact that an e-Business site typically includes a cluster of application servers, which are not necessarily homogeneous. For tractability, we distinguish two levels of performance tuning: a *macro-level adaptation* that regulates controls at the cluster level (e.g., load balancing); and a *micro-level adaptation* that deals with performance tuning at a finer granularity of resources internal to an application server (e.g., server queues and connection pools). In this paper, we focus only on the latter.

The outline of the rest of this document is as follows. Section 2 provides some background on middleware technologies used within e-Business sites. Session 3 explains the main issues on configuration tuning of application servers. Section 4 describes the implementation of the automated tuning system. Related works are presented in section 5. Section 6 concludes the paper.

2. The E-Business Site

E-Business sites provide services to their customers using middleware technologies to deploy, host and manage enterprise applications. At peak loads, these systems are susceptible to large volumes of transactions and concurrent users. And yet they are expected to maintain the offered QoS metrics while scaling appropriately to handle different bursts of traffic in a predictable manner.

If problems arise in the service provisioning at application run time, it is the responsibility of the e-Business site to adapt the system platform, in order to prevent or minimize the possibility of violation of the agreement with the customers. The main issue is to design a system with confidence that it will perform well enough to meet Service Level Agreements, the legally binding contract between the e-Business site and the customer [1].

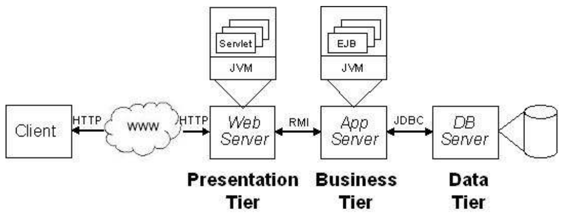

Fig.1: Structure of an e-Business site

Fig. 1 shows a conventional structure for e-Business sites. It is a multi-layered structure, logically divided in several *tiers*. Usually we can expect to have the following: *Presentation Tier* that runs within a web server that hosts a number of components based on web technologies such as HTML for static contents, or JSP and Servlet for dynamic contents; *Business Tier* that runs within an application server and provides for the case of Java component middleware (J2EE), EJBs, RMI classes, JDBC pool, and their associated business logic; and *Data Tier* that consists of one or more databases and a database server that manages data persistence.

Technologies behind e-Business solutions are the current component-oriented architectures that promote the use of containers to host component instances, such as J2EE, CORBA, COM+/.NET. Using these middleware architectures, developers of e-Business applications are free from explicitly handling issues such as transactions, database interactions, concurrency; all of them handled by the application server.

It has been demonstrated that application servers are often the bottleneck of the e-Business site platform [2]. The main problem of predicting application server performance is not straightforward. The tight coupling of application components and component infrastructure introduces a high level of complexity in predicting the effects of various architectural trade-offs. For instance, it has been observed that the same technology with different vendors [3], or different classes of e-Business applications [4], or even the same application type, but implemented using different design patterns [5] perform differently from an instance to another. Furthermore some of the differences become even more significant as the same test case is scaled to run on more application server nodes in a clustered platform, in attempt to increase the overall application throughput [3].

Nevertheless application servers are designed to be flexible and application deployers are provided with 'knobs' to tune several components of the application server. But in many cases application servers are configured using a mixture of rules-of-thumb, intuition and trial-and-error approaches [4].

The solution we are investigating attempts to find the 'best' configuration settings by monitoring of the application server and extracting runtime performance characteristics, under experimental conditions with an adequate set of benchmarking application. The information collected in this way is employed at runtime to adapt the system to the variations of the workload conditions, and automatically tuning the configuration parameters along the lines discovered with testing.

For our work, we have focused on specific middleware architecture, the Java 2 Platform Enterprise Edition (J2EE) [8], and, more precisely, on JBoss [9].

3. Tuning Configuration Parameters

Performance of application servers depends heavily on appropriate configuration, but to choose the correct settings of parameters is a difficult and error-prone task, given that application servers may have more than a hundred parameters that can be modified and many of them may present complex interactions [6].

An important distinction among the overall setting of configuration parameters is that the modifications can be *statically* or *dynamically* carried out. In the former case, parameters are assigned only at server set-up time; in the latter case the parameters can be tuned at run-time. In the rest of the paper we shall consider only the dynamically tuned parameters, since our aim is to provide on-line performance tuning, without shutting down the application server.

Generally, the administrator of the server nodes manually tunes these parameters to optimize the performance, using best-practice guides and profiles. There are cases in which the best configuration of the server is found to be in conflict with the best practice suggestion [4].

Modification of the configuration can be applied at different level of the single node.

At the process level, the Java Virtual Machine (JVM) can be started with the `–Xmx <size>` option, which sets the maximum JVM heap size, that is the maximum amount of memory allocated to the JVM in which the application server executes [7].

At the server level it is possible to specify the group of services for the application server start-up, i.e. the minimum number of services required to start the server; the default J2EE; or the configuration containing all the available services, included clustering.

At the component level, examples include the configuration of multiple thread pools, queues, cache size, and timeout and retry values. In JBoss, for instance, there is the *Backlog* parameter, which sets the maximum queue length for incoming connection requests, so that if a connection request arrives when the queue is full, the connection is refused; the *Database Connection Pool Size* (DCPS) parameter, which sets the size of the pool of live database connections that can be concurrently handled, or reused in order to reduce the overhead of opening new connections; the *Thread Pool Size* (TPS) parameter, which represents the number of live server threads maintained in a pool, accepting requests from the waiting queue and handling simultaneous client sessions.

In particular, TPS is a critical parameter, because it dictates the concurrency level at the application server. A small number of threads may work well for providing good response time, but there is higher probability of rejecting client requests, leaving the server under-utilized; on the contrary, a high number of concurrent threads increases utilization but slows down response time.

Our experience in tuning this parameter during benchmarking tests shows that, at a fixed load, increasing TPS leads to an improvement of both response time and throughput; but this is true until only a certain value, after which the application server saturates. The trade-off is that there is a point at which the overhead associated with context-switching, i.e. giving the CPU to each of the threads in turn, becomes so costly that performance dramatically degrades. Therefore it is important to find the best setting at a given workload, even applied to more than one parameter, which may influence each other.

Still one could argue that it is sufficient to size the system so that it can provide the best resource availability for serving the maximum number of requests under peak of loads. For instance, it could be possible to set TPS at the highest value, even if the load is low. When the workload intensity increases, there will already be the maximum available number of threads waiting for serving a high number of incoming requests.

It happens instead that balanced configuration setting can avoid build up queues of jobs waiting to be served.

A client request, as shown in figure 2, can encounter waiting either at the application server door, waiting to be served by a server thread, or at the database server door, waiting to write/read data in the database for its computations. If the number of threads serving requests is too high, the queue at the database increases and the database become congested. On the other hand a low number of threads can build up queues at the application server door, leaving the database under-utilized.

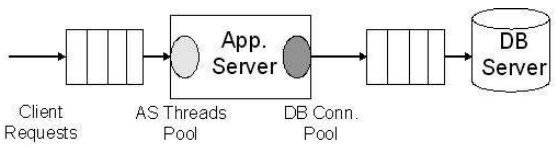

Fig.2: Queues at Different Tiers

Another important reason for applying configuration tuning is that a shared resource is optimally accessed without setting the configuration parameter at the highest. For example, let's consider the case of a database shared among different application server instances (a cluster). Each application server maintains its pool of database connections, and the size of the pool is the maximum number of connections to the database that can be concurrently open. It is worth keeping the number of connections at a low level when there is no high workload, so that the access on the connection table is faster for the overall set of nodes in the cluster.

As result, it is worth using configuration tuning that can support the achievement of the best performance of overall system at any time, without leaving resources under-utilized.

4. Automated Tuning System

To monitor and adapt the application server to the variation in the workload we have realized an automated tuning system, fully integrated on the application server platform. We have focused on an open source implementation of the J2EE architecture, the JBoss application server [9]. However a similar strategy can be applied to any other application server technology, the only requirement is the presence of server knobs to use for automatically tuning the configuration parameters.

JBoss is an open-ended middleware, in the sense that users can extend middleware services by dynamically deploying new components into a running server. The foundation of JBoss middleware components is the JMX specification [10], which provides a lightweight environment where components can be dynamically loaded and updated, and which makes JBoss manageable. JMX provides a common software bus that allows the integration of components, such as modules, containers, and plugs-in, declared as *MBeans services*, where the MBean is the Java object representing manageable resource (any device, application or Java object. On top of it, JBoss introduces its own model for the components, centered on the service components, the modules that implement every key feature of the J2EE technology.

As it is suggested in [12], the basic architecture for an automated tuning system is made of the *target system*, which is the JBoss application server being managed, and two kinds of interfaces with the target system. The first is represented by the *Monitor* component, which provides access to performance data. The second is represented by the *Controller* component, which provides access to the tuning knobs that control the performance of the target system.

The metrics that we have chosen to control are: *server Throughput*, i.e. the number of requests per second that complete execution from the system; and *server-side Response Time* (RT), i.e. the time elapsed since a request arrives at the system until it is completely processed and a reply is sent back. No network time is included. So, the clients of the application server are the effective "proxy-clients".

The Monitor component periodically collects the data on server QoS, which are maintained in the *Actual QoS* object. The object is updated by the *Invocation Data* component that detects client method calls to the EJBs of the running application, using an *interceptor* to detect the inbound and outbound times. It exploits the interceptor stack of stateless components, in which every call proceeds through the stack from first to last, until the target EJB component is called. After the EJB has finished with its method, the call will unwind through the stack in reverse order [11].

Whenever a method call is issued on the client-side proxy, the RMI call is routed by an Invoker MBean at the server side, where is routed through the chain of interceptors associated to the container of the target EJB, among which there is our monitor interceptor.

The Controller component has the duty of maintaining performance at an acceptable level, defined by *Agreed QoS*. Therefore it periodically compares this Agreed QoS with the values of the Actual QoS provided by the Monitor. If the evaluation is above a fixed threshold, then there is an early warning and the adaptive strategy is applied. The adaptation is carried out tuning the application server configuration, and directly modifying the configuration parameters exposed in the MBean interfaces of the manageable resources.

The new server configuration is selected among a set of optimum configurations. This set is originated by an *off-line testing phase* that precedes the on-line run of the hosted application. During the testing phase, the application server performance is measured with a benchmarking application; the best configurations at different loads are selected. At application run-time, the parameters are automatically tuned, without requiring any human action.

Both the Monitor and the Controller components are started and managed by the *Micro Resource Manager* MBean.

These are the components that provide QoS control, the shaded ones depicted in Fig.3. At application server startup, the MBean server starts the Micro Resource Manager MBean, which runs the Monitor and the Controller component. The former gets the Actual QoS object from the Invocation Data component, which is updated by the interceptor. The latter examines the acquired values and decides to tune the configuration of selected MBeans.

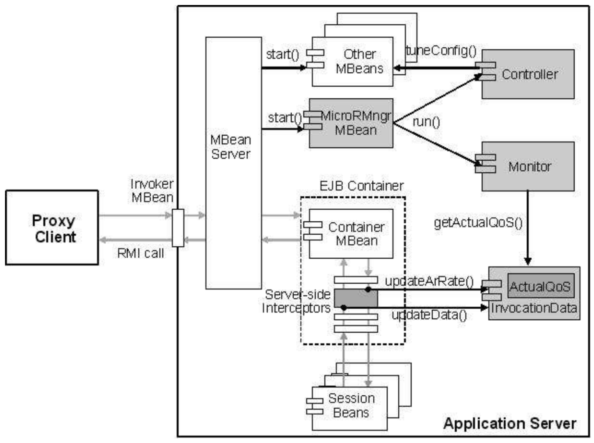

Fig.3: Automated Tuning System

During the testing phase, the evaluation of the performance is carried out with the use of ECperf [13]. It is a benchmark application for measuring performance and scalability of e-Business systems built by Sun in conjunction with J2EE application server vendors. The series of simulated events that represent the business problem modeled are based on manufacturing, supply chain management, and order-inventory, all of them requiring the use of many middleware services, which are stressed and measured by the benchmark. The workload generator is simulated by a multithreaded application that spawns several agents to simulate the clients. The activities of the agents are related to the chosen *injection rate*, the rate at which business transactions requests are sent to the server.

The result of one of the tests that has been conducted is shown in the graph in Fig.4. The graph represents the variation of RT during the steady state of an ECperf run, at a constant injection rate.

Throughout the simulation time, the values of the configuration parameter TPS and DCPS were increased, from the default values, in *conf_1,* to five times these values, in *conf_4*.

The RT obtained improves at each configuration change. The conf_1 shows a quite uneven RT plot since many of the ECperf requests to the database were rejected, because of the limited number of database connections. The high error rate decreased with the increment of DCPS. The best configuration shown in this example is the conf_4, which has as side effect a higher consume of node and database resources, which should be taken in account in case of shared resources.

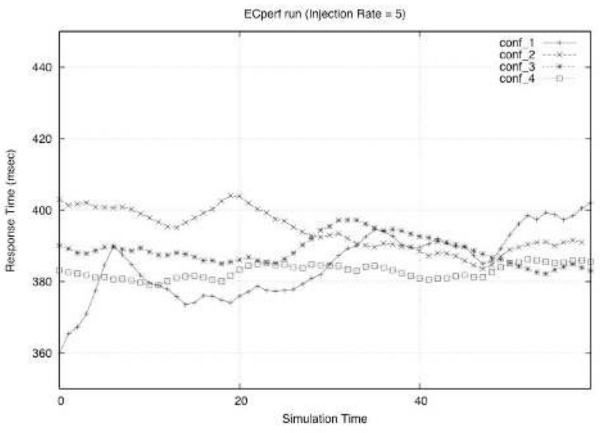

Fig.4: ECperf test: Application Server RT

Adaptation at run time can start before the violation point of the QoS metric is reached. For instance, it may occur that the QoS metric is violated for a short time, which denotes that the violation point is not reached yet and that configuration tuning can be effective.

The described adaptation policy is operative at node-level and it is defined as *micro-level management*. We introduced this term for tractability, to make a distinction between a single node and a collection of application server nodes.

It is worth considering that, to achieve application server scalability in the enterprise environment, the client requests can be distributed among multiple cooperating server node of a cluster. Therefore, in case high workload burst, also due to node failures, for instance the crash of some of the nodes in a cluster may overload the others, the violation of the agreement, as stated in the Agreed QoS, is detected and moved forward to the cluster level, defined as *macro-level.* A different adaptive policy can be applied at this level, such as admission control, service differentiation, or adjustment of the cluster configuration integrating new resources or server nodes. The last strategy is the one enforced by the *macro-level management* [17].

5. Related Works

Performance analysis and configuration tuning have been generally investigated in the context of web servers, which have less complex interactions than application servers. An example is the work done in [18] that describes the on-line optimization of an Apache web server setting the optimal value of a single parameter, at different loads.

Performance tuning of application servers is attracting increasing attention now; so far two approaches have been followed.

In an *empirical approach*, performance analysis is led by a series of tests that identify bottleneck and system behavior with different configuration setting or usage patterns.

This is the case for [4], in which tests are driven using a simple methodology to explore the configuration space, and [6], in which several sampling and search algorithms are studied for finding the best configuration setting with a small numbers of test runs.

These works highlight significant correlations among the parameters to tune and present best effort algorithms that can be used to increase the efficiency in selecting the configuration parameters. However they do not provide a solution for online optimization of the application server performance.

On the other side, the experimental evaluation has been used to validate the correctness of an *analytical approach.* The analytical foundations in designing system model are provided by mathematical techniques such as Queuing Network (QN) modeling, like in [14] and [15], or Control Theory, like in [16].

In [14] the authors use QN modeling to analyze the performance of an e-Business site and determine the tuning of some configuration parameters for adapting the e-Business site to the load. In [15] there is described a methodology to determine the optimal concurrency level of an application server. A simple benchmarking application is used to derive a QN model of the server, which is strictly related to the design pattern of the running application as well as the application server implementation.

Another method that can be applied is the Feedback Control Theory, as it is used in [16]. The authors build an analytical model of an Apache web server in order to enforce policies even in the presence of interactions between the controls.

The work presented here involves building a general purpose monitoring and control sub-system for online control. It is in conjunction with the empirical approach. The information acquired off-line is employed to decide the best configuration setting in case of load variations, when the e-business application is running and SLAs are regulating the client-server interactions. In the worst case here, if adaptation is not effective, the violation warning is moved to the macro level, where the configuration of the e-Business site cluster is managed.

6. Concluding Remarks

In this paper we addressed the problem of controlling the QoS of modern e-Business sites.

We note that technologies behind e-Business solutions, which are based on component-oriented architectures, exhibit an inherent complexity that can be hard to quantify, even in simple models. Services such as caching, pooling, replication, clustering or JVM optimizations, provided by the application server, contribute to an improved, but, at the same time, highly unpredictable runtime environment. The search for performance improvements of the underlying middleware platform, under variable load conditions, leads to increased complexity.

The solution presented in this work is based on a monitoring component, which at runtime extracts performance data such as method execution time of the e-Business application, and a controller component that periodically compares the actual performance data with the expected ones. If any violation is detected, the system can improve its performance with an automatic fine-tuning of the application server, choosing the best configuration to apply among a set of adequate configurations selected by a testing-phase. Besides, there is the possibility of an escalation of the violation warning, till to the cluster level, where the control of the overall e-Business site cluster is managed by a Macro Resource Manager component, which can apply a different adaptation policy.

7. Acknowledgements

This work has been supported in part by the European Union funded project TAPAS: Trusted and QoS Aware provision of Application Services, EU Project IST-2001-34069.

8. References

[1] Molina-Jimenez, C., Shrivastava, S., Crowcroft, J. and Gevros P. "On the Monitoring of Contractual Service Level Agreements" IEEE International Workshop on Electronic Contracting (WEC), July 2004, San Diego, CA,

[2] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, W. Zwaenepoel "A Comparison of Software Architectures for E-business Applications". *Technical Report TR02-389*, Rice University, 2001.

[3] I. Gorton et al., "Middleware technology Evaluation Report" *CSIRO Technical Report,* 2002. www.cmis.csiro.au/adsat/mte.htm

[4] Raghavachari, M., Reimer, D. and Johnson, R. D. "The Deployer's Problem: Configuring Application Servers for Performance and Reliability", *ICSE 2003*, Portland, OR, 2003.

[5] E. Cecchet, J. Marguerite,W. Zwaenepoel, "Performance and scalability of EJB applications", *Oopsla 2002*, Seattle, November 2002.

[6] B. Xi, Z. Liu, M. Raghavachari, C. Xia and L. Zhang, "A Smart Hill-Climbing Algorithm for Application Server Configuration" In Proceedings International WWW Conference, New York, 2004.

[7] E.Ort, "Virtual Machine Performance", Java Sun Technical Articles, February 2001 http://java.sun.com/developer/technicalArticles/Programming/JVMPerf/

[8] Sun Microsystem, "Java 2 Platform Enterprise Edition v 1.4", November 2004

[9] http://www.jboss.org/

[10] Sun Microsystems, "Java Management Extensions. Instrumentation and Agent Specification v1.1", 2002. http://java.sun.com/jmx/

[11] M. Fleury, F. Reverbel, "The JBoss Extensible Server", *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, 2003.

[12] J.Hellerstein, "Automated Tuning Systems: Beyond Decision Support", *in Proceedings of Computer Measurement Group,* Orlando, Florida 1997.

[13] Sun Microsystems, "The ECperf 1.0 Benchmark. Specification", June 2001. http://java.sun.com/j2ee/ecperf/

[14] [MBD01] D. A. Menasce, D. Barbara, R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: a performance model approach", *in Proceedings of the 3rd ACM conference on Electronic Commerce*, Florida, 2001.

[15] Y. Liu, S. Chen, I. Gorton, A. Fekete, "A methodology for predicting the performance of component based systems", *Poster session of ACM ICSE*, Edinburgh, May 2004.

[16] N. Gandhi, J. Hellerstein, S. Parekh, D. Tilbury and Y. Diao, "MIMO Control of an Apache Web Server: Modeling and Controller Design", *Proceedings of American Control Conference*, May 2002.

[17] TAPAS Deliverable report D7, "TAPAS Architecture: QoS Enabled Application Servers", March 2003, http://www.newcastle.research.ec.org/tapas/index.html

[18] Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J. L. and Parekh, S. "Online Response Time Optimization of Apache Web Server". *IWQoS* 2003.