Appendix

Realistic and Tractable Modeling of Multi-tiered E-business Service Provisioning

G. Ferrari†, P. Ezhilchelvan† and M. Little‡
†School of Computing Science, University of Newcastle upon Tyne, UK
‡Arjuna Technologies Ltd., Newcastle upon Tyne, UK

March 11, 2005

Abstract

This appendix models the activities within an E-business site by representing them in an abstract yet realistic manner and also by postulating a set of rules which govern the interaction between them. The objective is to be able to derive an analytical model that is *accurate* in representing site operations and also analytically *tractable*. The Analytical model can then be used for controlling the Quality of Service (QoS) offered by sites of commonly encountered architectures. This will lead to the site administrators to dynamically selecting appropriate configuration parameters that would match the user's SLAs.

1 Introduction

E-business sites provide specific services to their customers with agreed Qoality of Service (QoS) attributes such as end-to-end response time, site response time, throughput in requests/sec and so forth. 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.

In this appendix we introduce a model that represents the activities within an E-business site in an abstract yet realistic manner and also postulates a set of rules which govern the interaction between them. The objective is to be able to derive an analytical model that provides a trade-off between the *accuracy* in representing site operations and computational *tractability*.

The model can then be used for estimating and controlling the QoS offered by sites of commonly encountered architectures. This will in turn give support to the site administrators for dynamically selecting appropriate

configuration and parameter tuning that would match the application level load and, at the same time, meet the agreed QoS.

The appendix is organized as follows. Next section describes the common E-business site architecture. The following section highlights the activity occurring on the tiers and the tier interactions. Finally, we present the model representing the E-business site core that is accurate and analytically tractable. In the last section some related works are reported.

2 E-business Site Architecture

An E-business site is conventionally structured into logical components called *tiers*. (See Fig.1.), a tier comprises one or more servers capable of running a particular set of applications. Usually we ca expect to have the following:

Presentation Tier: the first tier is the front end of the enterprise, and runs a Web Server (WS, for short) which maintains the presentation logic of the application in the form of *static* read-only HTML pages. It receives the client requests coming from the outside network and passes them on to be processed at the lower tier.

Business Tier: It is responsible for business-specific computations. The Application Server (AS) within it maintains java objects and associated business logic and is therefore capable of dealing with more complex, *dynamic* queries.

Data Tier: This tier is represented by databases (DB) which maintain the persistence data. It is usually physically separated by the other two tiers in order to protect sensitive data by being accessed by unauthorized users. The users cannot directly connect to this tier, but only through the front-end ones.

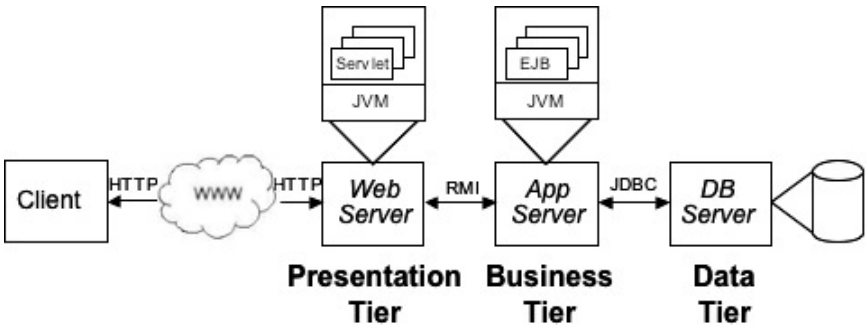

Figure 1: Structure of an E-business site

Note that some authors, such as [1], additionally define a *Client Tier* containing all potential clients accessing the site using the Internet. Very little computation is done at the Client Tier, other than a client either composing a request or 'thinking' over a response received and possibly submitting a subsequent one after some 'think-time'. We however focus here on those tiers that make up the site.

2.1 Tier Implementation Alternatives

For reasons of scalability, responsiveness and availability, a tier within an E-business site is implemented by a *cluster* of server nodes. A *cluster* is a logical group of nodes running Web applications simultaneously and appearing as a single server to the world [2]. *Load balancers* are used to distribute the load between the cluster nodes by redirecting web traffic to an appropriate cluster member.

Tier-Cluster mapping. E-business sites can differ by the way the tiers mapped to the clusters that implement them. In a *single-tier clustering*, all three tiers are implemented by servers of a single cluster. A load balancer, placed in front of the Presentation Tier, distributes requests, hence the processing load, evenly to all servers in the cluster. A *multi-tier clustering* extends the logical separation between tiers into physical separation: every tier runs on a distinct cluster, with a load balancer at each tier. If the tier functionality is *replicable on-demand*, the number of servers within a cluster, and thus the provisioned tier capacity, can be varied dynamically.

DB Replication. With regard to the Data Tier, traditionally database servers have employed a *shared-nothing* architecture that does not support replication. However, some databases use the approach *shared-everything*, but the complexity of its distributed architecture is shielded by providing a transparent view of a single database image to external applications, even if it is composed of two or more servers[3].

Technologies. E-business sites are developed using the component-oriented technologies that promote the use of *containers* to host component instances, such as J2EE, CORBA, COM+/.NET. Using these middleware technologies, developers of E-business applications are free from explicitly handling issues such as transactions, database interactions and concurrency, which are handled instead by the Business Tier, the one providing the business logic.

Selection of the architecture or the technology for E-business site implementation largely depends upon usage pattern and the type of the application.

3 Tier Operations and Interactions

A client on the Internet sends a request to the E-business site using, for example, a browser and thus connecting to the site web pages.

The request is always routed to the Presentation Tier, where a WS instance, or a thread, serves the request and generates either a static or a dynamic content response. In the latter case, the Business Tier is contacted to execute a program containing the necessary business logic. The interaction is in the form of a blocking call being made to an AS thread, thus the WS thread awaits until the AS thread returns a response.

The AS thread, on its behalf, may have all the data necessary for the computation in the local cache. In that case it swiftly returns its response to the WS. If this is not the case, the data are retrieved from the database issuing one or more queries. Here again the AS thread is blocked while awaiting the DB thread to serve the request. As a call returns, the caller may do further processing; at the end the Presentation Tier synthesizes the response that is returned to the client.

In the simplest case each request is processed exactly once by the tier and possibly it is directly returned to the caller; otherwise it is forwarded to the tier below for further processing. More complex processing at the tiers is also possible, in such scenarios, each request can visit a tier multiple times.

These interactions between the tiers are highlighted in Fig.2.

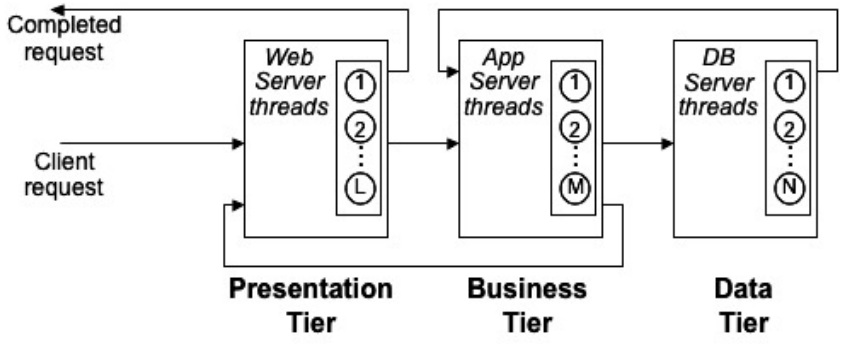

Figure 2: Tier interactions in the E-business site

3.1 Example: a J2EE Application

Let's focus on a commonly used middleware architecture, the Java 2 Platform Enterprise Edition (J2EE)[4], and consider the case of a specific E-business application, an auction.

An auction application implements the functionalities of an auction site,

such as selling items, browsing and bidding. Selling and bid operations are usually restricted to the registered users whose credit data have been stored on the site database.

In a typical request pattern for a bid operation, the client sends a HTTP request to place a bid on a given item. A Java Server Page (JSP) or a servlet at the Presentation Tier processes the request; this triggers a series of invocations at the Enterprise Java Beans (EJBs) [5] deployed in the Business Tier. EJBs are the components that implement the application business logic. They are executed in the EJB containers, which provide a set of ready-to-use services including security, object persistence and transactions.

There are two types of transactions possible. *Local transactions* are those that run purely in the scope of the database and commit or rollback within a single database interaction. *Global transactions* are those that are controlled within the application server and may encompass other application servers and databases. A Local transaction that is used within a Global transaction is automatically subordinate to the Global transaction, i.e., it cannot commit or roll back independently.

In order to display information on an item, the EJBs retrieve the data from the DB, if they are not already cached; before registering the new bid placed, it may also check the registration status of the bidder and her credit requirements. The EJB contacts the database using the JDBC service[6] of the container, which provides a connection pooling mechanism to facilitate connection reuse across requests. Then data is returned to the EJB that elaborates the response.

There are operations that require more than one access to the database. For instance, placing the winning bid entails to check of client's credit details, to finalize the purchase, to close the auction notifying the other bidders. All these are SQL operations at the database, to search, read or write data on the database tables. Some of the accesses, such as the final purchase, can be performed using a *transactional database access*: the database checks for conflicts as part of acquiring a write lock, and, if there are conflicts, i.e. read or write locks, then it won't get the lock.

In case lock conflicts occur on the database, the transaction being attempted is automatically rolled back and an exception will be throw to the application. Another possible implementation is to block the thread when a conflict is detected, until the current locks are released, due to commit or roll back of the other transactions. The behavior after a transaction failure depends on the type of implementation: in the *Container Managed Persistence* (CMP) the transaction is controlled by the AS, which immediately aborts the transaction; on the contrary, in the *Bean Managed Persistence* (BMP) the bean controls the transaction, handling the returned exception, thus the effect of a failure can be other than an error response, e.g. a retry.

The final computation of the EJB is transmitted to the calling JSP page that synthesizes the HTML response sent to the client, and even contacts

the other bidders to declare the winner of the auction.

The given example points out the specifics of the three tiers:

- *Specifics of the Presentation Tier:* the only task of the Presentation Tier is to handle HTTP requests, contact the business components and eventually generate HTML pages. In [10], the authors stated that the amount of work performed by the front-end servers, e.g. the WSs, is low enough that over-provisioning is a cheap solution to meet QoS requirements.

- *Specifics of the Business Tier:* this tier implements the business logic using the container components that manage the creation and execution of the EJBs and handle issues such as EJBs lifecycle, security, transactions, database interactions, concurrency control, messaging, naming and management support. The AS controls the behavior of the EJBs, their concurrency, and their caching, as well as the interactions of the EJBs with the data sources. The AS is connected to a back-end DB where enterprise data are persisted [7].

- *Specifics of the Data Tier:* the database system maintains the persistent data, and it executes the SQL queries as it is required by the EJBs. The complexity of this tier is usually hidden from the E-business application, infact the issues such as security, database open connections and database transactions are services managed by the container components.

It is worth noticing that the type of workload for E-business traffic is basically different from the general web traffic. The main differences lie in:

- presence of high level of online transaction processing activity

- large proportion of dynamic requests

- great number of requests in secure mode.

which trigger higher response times at this tier, caused by increased computation and communication times combined with the time spent to access the database.

AS reduces the burden on the DB by not requiring the DB to directly manage session information, and it allows more complicated session data than is practical to pass with every web request. It provides communication with both back-end DB and front-ent web clients in addition to providing a framework to connect application specific "business rule" that govern the interaction between the two[8].

It is possible to improve DB ability to meet its imposed QoS requirements. Since there are methods to maximize profits in the Presentation and

in the Data Tiers, an E-business site profit depends on how well the AS resources are used to serve requests. If not configured properly, a customer can suffer numerous service misses, and it is responsibility of the E-business site to prevent or minimize the possibility of violation of the agreement with its customers. Then it is significant to be able to predict and improve performance of the Business Tier [9].

Therefore, the model we present below will be primarily AS-centric, focusing on AS operations and AS-DB interactions.

4 The Model

4.1 Preliminaries and Assumptions

The model is expressed using two primitive data structures: *threads* and *queues*. The former process the client requests and each tier is modeled as a collection of threads of distinct type. The queues hold those requests that are awaiting to be processed by a thread.

A thread is in the *active* state while it is processing a request; as a part of processing, it may make a 'blocking' call to another thread and enter the *blocked* state. When the call returns, the caller thread re-enters the active state. Finally, a thread is in the *available* state when it is neither active nor blocked. When a request arrives at a tier, it is either handed to an available thread (if any) of that tier which subsequently becomes active, or kept in a queue until a thread becomes available to get active on it. A queue can thus be either *empty* or *non-empty*.

We assume that the threads are reliable, they complete processing of any request in some finite time, and a remote call always returns with a response that can be *normal* or *exceptional*. Thus, threads cannot remain forever in the *active* state over a given request and in the *blocked* state over a given call.

We consider a special class of queues, called the *Timed Queues*, characterized as follows. If an entry in a timed queue is not dequeued by a thread within Δ seconds, it is deleted and the thread that enqueued the deleted entry receives an exceptional response. Thus, a timed queue models a request waiting on a timeout for a resource to become available and an available resource being allocated to a waiting request in the FIFO manner. We note here that the notion of *queues with reneging* can be used to model and analyze the behavior of requests in timed queues.

A queue, timed or otherwise, is assumed never to become full, and an incoming request is always enqueued. The rationale for this assumption is two-fold.

1. We assume that an admission control policy is operative for requests entering the business site, and that an aspect of this policy is to admit

requests only if the queue in the Presentation Tier is not full.

2. Since a remote call is blocking in nature, the number of requests awaiting to be processed in the second or the third tier cannot exceed the number of threads in the previous tier which is finite and often known. A careful provisioning of buffer space avoids queues at Business and Data Tiers from becoming full.

For the reasons stated in the previous section, it is the performance of the *Business and Data Tiers* which crucially influence the site performance. So, the *System Model* is concerned only with these two tiers; the clients accessing these tiers are represented by *proxy clients* (typically the servlets of the Presentation Tier) which provide dynamic web pages to actual clients.

4.2 Description

We model the Business and Data Tiers as sets of M and N threads, respectively. Each tier is associated with a queue that holds the requests while no thread is available within the tier. The queue of the Data Tier is a timed one - modeling the fact that waiting for a DB thread is on a timeout. The model is depicted in Fig.3.

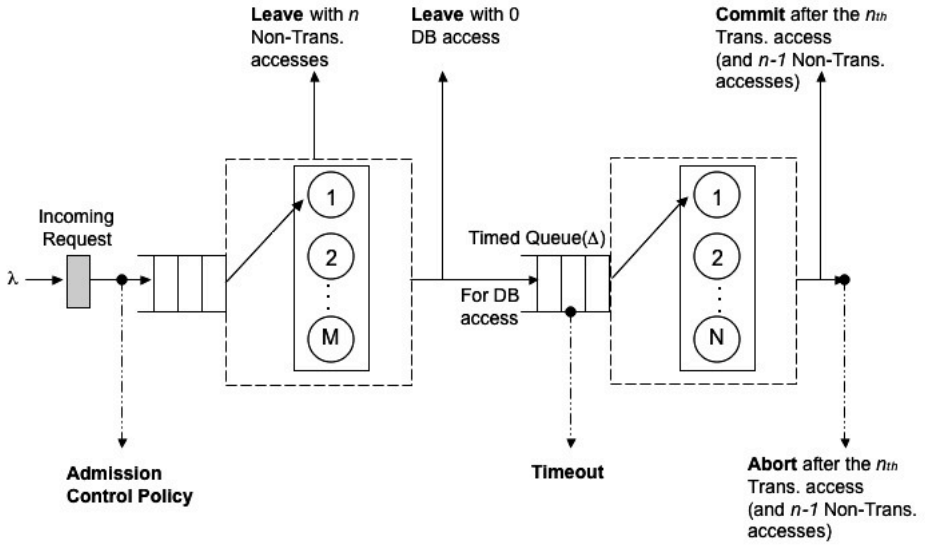

Figure 3: AS model

The Business Tier threads process the requests at the average service rate μ requests per second, if no (blocking) call were to be made to the Data Tier during processing. A request however requires a business thread to make an average of $n \geq 1$ calls to the Data Tier. After each call is made,

the Business Tier thread is blocked. A call is queued in the timed queue and if it is not pick-up be an available thread within seconds, the business thread receives an exception and returns to 'available' state.

With probability p, the nth call can be a *transactional* one and, if so, processing of that call terminates with either a *commit* or an *abort* result. The processing of a non-transactional call is similar, but is less time consuming. Infact a transactional call may take longer (at least 10 times) to be processed compared to a non-transactional one. So, two service rates are defined for Data Tier threads: ν (non-transactional) and $\hat{\nu}$ (transactional).

The pattern of a *successful request* is described as following.

The request joins the Business Tier queue and awaits until it gets served by one of the M threads of this tier. In serving the request, the thread may need to get, or read, some stored data. There are two cases: all the data are in the local cach; or the data need to be retrieved from the database. In the former case, there is no database access: the computation may finish at this point, a response is sent back and the AS thread is released. In the latter case, the AS thread awaits to get an open connection to the database, the AS thread holds and a thread at the Data Tier serves the request, which is either a read or a write operation. The result is returned to the AS thread and the DB thread is released.

In accessing the Data Tier, there are again two cases: the read or write operation at the database requires a simple function; or the operation is a write operation that requires a transactional call. In the former case the Data Tier thread quickly returns the data to the Business Tier, which may finish here and send the response back. Besides, it may require some further accesses to the Data Tier then it has to compete again for the database connection. In the latter case, the transactional call is the last write operation of the given request, before the response is successfully completed by the Business Tier. Thus the database table are upgraded and the Data Tier thread is released. The AS thread performs the final computation and sends the response back; now it is available to pick another request up from the waiting queue.

4.3 Related Work

The modeling of E-business site operations are generally carried out in the context of performance modeling and QoS control (e.g., [15], [16], [17], [18], [19], [20], [21], and [22]). Of these, we specifically focus on the model aspects (not the analytical aspects) of [15] and [16] since they, like TAPAS, are concerned with applications hosted by component-based middleware.

The authors of [15] simplify the modeling by considering only a single tier and expressing the whole system as a composition of multiple instances of the single-tier model. The main difference between our model and this one is in the accuracy of the model. Since the three tiers are described in the

same manner, the overall model is made of a composition of three identical tiers. Moreover, their model ignores the careful provisioning that can be done to avoid overflowing of queues by knowing the number of threads in the previous tiers (see section 4.1); instead we have noticed that the number of requests awaiting to be processed in the second or the third tier cannot exceed the number of threads in the previous tier.

Finally, we have introduced a timed queue in front of the Data Tier, where requests are dropped in case of timeout, on the contrary [15] uses a queue of finite size which can lose requests when full.

The main purpose of [16] is to show that the performance can be predicted by deriving accurate models from the system design, when the latter is expressed formally. The authors have chosen the EJB architecture for demonstrating their claim. Since the design is assumed to have been expressed in detail, they work with a very detailed knowledge of the deployed application. The resulting model characterizes the operations of service components and the frequency of each operation, in order to produce a performance prediction of the system in the form of an equation with performance profile parameters. In a second phase, they deploy their specific application on the target platform and measure the performance. Finally, the parameter values describing the performance profile of the platform are determined solving the model corresponding to the given application, and used to calculate the required quantitative prediction of performance of that system.

This model is strictly correlated to the software structure of the application. In fact, there is considered a specific implementation of a trade application, which is used it to derive the parameters of the equations. The accuracy is high because of the availability of a very detailed knowledge of the deployed application, particularly in terms of characteristics of the implemented transactions and number and type of EJBs. Because of the specific implementation of the application, It does not distinguish between transactional and non-transactional access: every call to the Data Tier is a non-transactional access, resulting in a normal response, justified by the limited number of DB entries considered.

References

[1] A. I. Kistijantoro, G. Morgan, S. K. Shrivastava, M. C. Little,"Component Replication in Distributed Systems: a Case study using Enterprise Java Beans", In Proceedings of the 22nd International Symposium On Reliable Distributed Systems (SRDS), Florence, Italy 2003

[2] R. Tyagi, "Enterprise application clustering architecture", http://builder.com.com/5100-6387-1045317.html

[3] B. Zeller, A. Kemper, "Experience Report: Exploiting Advanced Database Optimization Features for Large-Scale SAP R/3 Installations". In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002), Hong Kong, China, August 2002

[4] Sun Microsystem, Java 2 Platform Enterprise Edition v 1.4, http://java.sun.com/j2ee/

[5] Sun Microsystem, Enterprise JavaBeans Technology, http://java.sun.com/products/ejb/

[6] Sun Microsystem, JDBC Technology, http://java.sun.com/products/jdbc/

[7] M. Raghavachari, D. Reimer, and R. D. Johnson,The Deployers Problem: Configuring Application Servers for Performance and Reliability, ICSE 2003, Portland, OR, 2003

[8] M. Karlsson, K. Moore, E. Hagersten, and D. Wood, "Memory characterization of the ECperf benchmark", In Proc. of the 2nd Annual Workshop on Memory Performance Issues (WMPI 2002), held in conjunction with the 29th International Symposium on Computer Architecture (ISCA29), Anchorage, Alaska, May 2002

[9] U. Vallamsetty, K. Kant, P. Mohapatra , "Characterization of E-Commerce Traffic", Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02), p.137, June 26-28, 2002

[10] D. Villela, P. Pradhan, and D. Rubenstein - "Provisioning Servers in the Application Tier for E-Commerce Systems", Proc. of IWQoS'04, Montreal, Canada, June 2004

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

[12] E. Cecchet, J. Marguerite,W. Zwaenepoel, Performance and scalability of EJB applications, 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (Oopsla 2002), Seattle, November 2002

[13] D. Krishnamurthy and J. Rolia, "Predicting The QoS of an Electronic Commerce Server: Those Mean Percentiles". In Proceedings of the Workshop on Internet Server Performance, Madison, Wisconsin, June 1998

[14] 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

[15] 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

[16] Y. Liu, A. Fekete and I. Gorton, "Design Level Performance Modeling of Component-based Applications", Technical Report N.543, School of Information Technologies, University of Sydney, November 2003

[17] S. Kounev, A. Buchmann, "Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets". In Proc. of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'03), March 2003

[18] Abhinav Kamra, Vishal Misra and Erich Nahum, "Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web sites". In International Workshop on Quality of Service (IWQoS), Montreal, Canada, June 2004

[19] T. Liu, A.B. Behroozi and S. Kumaran,"A Performance Model for a BPI Middleware",Proc. 4th ACM Conf. Electronic Commerce. ACM. 2003, p. 222-3,June 2003

[20] Paul Reeser, Rema Hariharan, "Analytic model of Web servers in distributed environments". In Proc. of WOSP 2000: Second International Workshop on Software and Performance, Ottawa, Canada, September 17-20, 2000

[21] Tarek F. Abdelzaher and Chenyang Lu, "Modeling and Performance Control of Internet Servers," Invited Paper, 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000

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