RC22100 (W0106-030) June 25, 2001
Computer Science

IBM Research Report

Enterprise JavaBean Response Time Analysis

Te-Kai Liu, Santhosh Kumaran
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

ENTERPRISE JAVABEAN RESPONSE TIME ANALYSIS

Te-Kai Liu and Santhosh Kumaran
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
tekailiu@us.ibm.com

This paper presents two methods for analyzing the response time of method calls of Enterprise JavaBeans (EJB). The first method breaks response time measured by a single client into two sets of parameters: one set relating to the beans themselves, and the other set characterizing the EJB run time environment including the EJB container, the underlined persistent store, and the remote method calling overhead. A system of linear equations can be used to solve for the values of the parameters. The second method involves measuring the response time of EJB method calls under different workload intensities and estimating the CPU and disk demands of a method call. As examples, the response time of method calls of several primitive EJBs deployed on an EJB server is measured and analyzed. The two methods are useful in understanding how much time is spent in the major components of the EJB run time environment and in estimating the resource demands of EJB method calls for further developing capacity planning models.

1. Introduction

The Enterprise JavaBeans (EJB) architecture combines the merits of distributed object technologies and traditional transaction processing monitors (TP monitors). EJB is a server-side component model, which greatly simplifies the development and deployment of enterprise business applications that are transactional, secure, and scalable. EJB capitalizes on the success of Java and is adopted by the industry at a rapid pace. Today there are more than 25 commercial EJB servers [EJB1] and 4 OpenSource EJB servers available. EJB 1.0 and 1.1 specifications are already released and EJB 2.0 specification is currently in the final draft review stage [EJB2].

The present study is motivated by the desire to understand the Enterprise JavaBeans (EJB) technology and its performance characteristics so that one can further develop performance models for performance prediction and capacity planning. Being a newly developed technology, the performance of an EJB based system is not well understood yet. Thus, one objective of the present study is to identify the major components in the response time of a method call on an EJB. Another objective is to extract the resource (i.e. CPU and disk) demands of EJB method calls in order to develop capacity planning models.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to EJB. Section 3 identifies the major components in the response time of a method call on an EJB, and the method for estimating these parameters. Section 4 describes a method for estimating the resource demands of EJB method calls from the response time. Section 5 reports on the experimental setup, the measurement data, and the analysis results. Section 6 concludes the paper.

1. Enterprise JavaBeans

The idea behind the EJB architecture is to have application developers make use of the beans that are provided by bean developers thereby speeding up the development cycle. The completed applications can then be deployed on any application server that supports the EJB specification (called EJB servers). EJB servers provide the development and run-time support for transaction, security, and resource management.

There are two basic types of EJBs: entity beans and session beans. Entity beans model data objects; these objects usually correspond to persistent records in a database. Session beans model business objects; they typically work with entity beans or other resources to implement the business logic. The EJB technology has a notion of remote interfaces and home interfaces. A remote interface defines the exposed business methods of an EJB, whereas a home interface defines the life cycle methods of the bean typically used by the EJB container for locating, creating, and removing the beans.

In the EJB architecture, session beans are further divided into two classes: stateless and stateful beans. Stateful session beans keep a session context throughout a client's session, whereas stateless session beans do not keep a session context from method calls to method calls. Entity beans are also classified into 2 types: container managed persistence (CMP) and bean managed persistence (BMP). The difference between the two lies in who takes care of the persistent fields of the entity beans. From the viewpoint of an application developer, CMP beans are simpler to use since the container synchronizes the persistent fields with the database, which is transparent from bean developers.

EJB architecture uses distributed object protocols such as Java RMI and CORBA IIOP for the communication between distributed objects. These protocols provide infrastructure services such as error and exception handling, parameter passing, and the passing of transaction and security context. To hide the complexity of the underlined distributed object protocols, client-side stubs and server-side skeletons are typically generated by EJB tool vendors automatically.

Before a client can invoke a call on the business methods of an EJB, it needs to first get a remote reference (stub) to an EJBObject (skeleton), which intercepts the call and invokes the corresponding method of a bean instance inside the container. The EJBObject works with the EJB container to execute resource management strategies such as instance pooling and provides the support for transaction, persistence, and security.

The way a client gets a remote reference to an EJBObject is by first looking up the home interface of the EJB via JNDI (Java Naming and Directory Interface) and then invoking the create method defined in the home interface. If the bean the client wants to call already exists, which is possible for entity beans, the client can invoke one of the find methods defined in the home interface (typically the findByPrimaryKey method), after the JNDI lookup. An EJB server typically provides a naming server for clients to look up the EJBs deployed on it. A good introduction to the EJB technology can be found in [MONS00].

1. Contention-Free Response Time Analysis

This section identifies two sets of parameters, one intrinsic to the beans and the other related to the container and the persistent store. The method for estimating those parameters is also described.

When a client application invokes a business method call on a session bean of an EJB application deployed on an EJB server, the client actually invokes a corresponding method call on a local proxy (i.e., a stub in Java RMI terms) of the session bean deployed at the EJB server. The local proxy serializes the parameters of the call and sends them to a server side proxy (i.e., a skeleton or an EJBObject). The EJBObject, implemented by the container, will collaborate with the container to provide middleware services such as persistence support, transaction concurrency control, and EJB instance pooling. Subsequently the call will be delegated to a bean instance, which completes the method call and returns the results back to the EJBObject, the stub, and to the client eventually. While a session bean instance processes a call, it is quite likely that it will invoke the method calls of other entity beans for reading or updating business data persistent in databases. Figure 1 illustrates the above calling sequence with stubs and skeletons not shown explicitly.

In Figure 1, a client application running on one JVM invokes a method call on a remote session bean, which in turn calls an entity bean deployed on the same JVM as that of the session bean. Suppose the entity bean is a CMP bean that has simple getter and setter methods for its persistent fields. Further assuming that its deployment descriptor specifies a transaction attribute of "transaction required", the container will access the requested data from the database or its cache, depending on the database access attribute (i.e., "shared" or "exclusive"). The attribute of "shared" means that the database is shared with other applications that also have access to the persistent fields of entity beans. The attribute of "exclusive", on the other hand, means that the database is used exclusively by the container. In such a case, beans' persistent fields cached by the container are always up-to-date.

The total response time is broken down into several elapsed time parameters as follows:

I. t1 = elapsed time for the client to call the null

method of the session bean, which simply returns immediately.

II. t2 = elapsed time the session bean to process the method call requested by the client, excluding the time waiting for the response from the entity bean.

I. t3 = elapsed time for the session bean to call the null method of the entity bean, which returns immediately.

I. t4 = elapsed time for the container to access (read or update) the requested data from the database.

I. t5 = elapsed time for the container to access (read or update) the requested data from its cache.

I. t6 = elapsed time for the entity bean to process the call from the session bean, excluding the time spent by the container managing its persistent data. t6 is typically close to zero for getter and setter methods.

From the above definition, we see that t2 and t6 are independent of the container and are determined by bean developers (and of course by the processor speed in part). On the other hand, t1, t3, t4 and t5 are parameters characterizing the runtime environment of the beans. Both t1 and t3 include the time spent by the stub, skeleton, and the container's overhead for executing its instance pooling strategy. Moreover, t1 may further include the network delay if the client JVM is running on a different machine than the server JVM.

Figure 2 shows a client application calling an entity bean deployed on a separate JVM directly. The figure and the elapsed time parameters are similar to Figure 1, except for the absence of t2 and t3. Figure 3 shows a client application calling a session bean, which is the same session bean as in Figure 1 except that it does not call any entity bean.

To facilitate the discussion, we designate the scenarios in Figures 1, 2, and 3 as the two-bean, single-entity, and single-session scenarios, respectively. Let Ttb, Tse and Tss be the total response time of the two-bean, single-entity, and single-session scenarios, respectively. We have

Ttb = t1 + t2 + t3 + t4 + t6, when data is accessed from database, or
t1 + t2 + t3 + t5 + t6, when data is accessed from container's cache.

Tse = t1 + t4 + t6, when data is accessed from database, or
t1 + t5 + t6, when data is accessed from container's cache.

Tss = t1 + t2.

The above system of linear equations has 6 unknowns and 5 equations which has a degree of freedom of 4. In general, the system of equations has no unique solution. However, by deploying a primitive session bean that does nothing but call the entity bean, t2 becomes zero. For simple getter and setter method calls of the entity bean, t6 can be approximated by zero. We thus can solve for t1, t3, t4, and t5. Note that after t1 is obtained, t2 of another session bean of interest can be estimated by (Tss - t1). In Section 5 we will present the experiments for measuring the total response time for the 3 scenarios, and the numerical results obtained.

1. Estimating Resource Demands from Response Time

This section describes a method for estimating the resource demands of EJB method calls from response time measurements. Slightly different from Section 3, this section deals with response time of a method call measured in a contention mode.

We assume that the CPU and disk demands for a method call of an EJB are Dc and Dd, respectively. In the contention-free mode (i.e., single client scenario), the response time of the method call is the sum of Dc and Dd. When there are multiple client making the method call concurrently, the response time can be obtained by solving the queueing system in Figure 4, which shows a closed queueing network with N customers and two servers (i.e. CPU and disk). The response time of a method call is the sum of the time a customer spends in the two servers.

Let Rn be the response time of the method call when there are n customers in the system. We have R1 = Dc + Dd. Now using analytical methods such as MVA [MENA94] one can predict R2, R3, R4, etc. for a given Dc in the range of (0, R1). By matching against the measured R2, R3, R4, etc., a reasonably good value of Dc and Dd can be obtained. Alternatively, one can perform regression analysis to find the optimal values for Dc and Dd.

1. Measurements and Results

We have deployed three simple EJBs on one test application server, which runs on a Windows 2000 machine with a 700MHz Pentium III processor and 256MB RAM. A database server, functioning as the persistent store of the application server, is also installed on the same machine. Among the 3 EJBs, the first one is a CMP entity bean, which has a getter and a setter method for accessing an integer instance variable. The second bean is a stateful session bean,

which also has a getter and a setter. But the getter and setter of the session bean simply call the getter and setter of the entity bean. The third bean is a stateful session bean with a null method, which does nothing and but returns immediately. The entity bean has the transaction attribute of "transaction required", whereas the two session beans have the transaction attribute of "transaction not supported". The entity bean further has the isolation-level attribute of "read committed" in its deployment descriptor. "Read committed" means that a transaction will not read uncommitted data used in other concurrent transactions.

For each scenario described in Section 3, a test client was used to measure the total elapsed time for making 10,000 calls from the client. The total elapsed time was measured by taking the difference of the returned values of two System.currentTimeMillis() calls inserted at the beginning and ending of a loop wherein 10,000 calls are made by the client. In the present study, the client is running on another JVM, which runs on the same machine as the EJB server.

The following shows the measured average response time Ttb, Tse and Tss, and the obtained elapsed time t1, t3, t4, and t5. Note that t2 and t6 are zero as explained in Section 3.

Ttb = 4.84 ms
Tse = 4.52 ms
Tss = 1.83 ms
t1 = 1.83 ms
t3 = 0.32 ms
t4 = 2.69 ms
t5 = 1.50 ms

From the result that t1 is 5 times more than t3, we see that EJB method calls across two JVMs take much longer than calls within a JVM (i.e. t1 >> t3). This effect is expected to be even more salient when the two JVMs are running on different machines connected by a network. By comparing t4 and t5, we see that the response time of a call on an entity bean's accessors (getter or setter) is shorter if the container can bypass the database in loading a bean's instance fields in the beginning of each transaction. But this benefit is only possible when the container is the only application that has access to the database, i.e., exclusive access.

To illustrate the method described in Section 4, another experiment is performed. The response time of calling the create() method of an entity bean's home interface experienced by n concurrent clients is measured for n = 1, 2, and 4. The measured (R1, R2, R4) in milliseconds is (6.49, 12.49, 24.99). The measured R1 is broken down into 4 cases whose predicted response time curves are plotted in Figure 5. It can be seen that (Dc = 6.24, Dd = 0.25) gives the best match among the 4 cases.

1. Conclusion

We have demonstrated two methods for analyzing the response time of EJB method calls. The first method involves measuring the response time of method calls of EJBs in different scenarios and solving a system of linear equations for the model parameters. The values of the parameters help us understand where the time is spent in the system when a method call is invoked by a client. The second method uses response time data to estimate CPU and disk demands of EJB method calls. The method will help in constructing capacity planning models for EJB applications. Work is underway to develop a capacity planning model for a real world EJB application.

1. References

[EJB1] Industry momentum on EJB. http://java.sun.com/products/ejb/tools1.html.

[EJB2] EJB specifications 1.0, 1.1, and 2.0. http://java.sun.com/products/ejb/docs.html.

[MONS00] R. Monson-Haefel, *Enterprise JavaBeans*, 2nd Ed., O'Reilly, 2000.

[MENA94] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy*, Capacity Planning and Performance Modeling, Prentice Hall*, 1994.

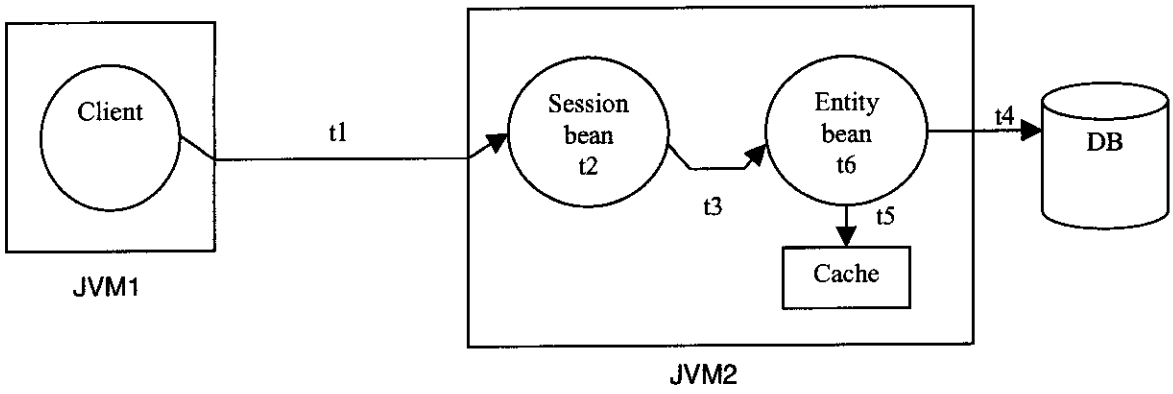

Figure 1. The two-bean scenario.

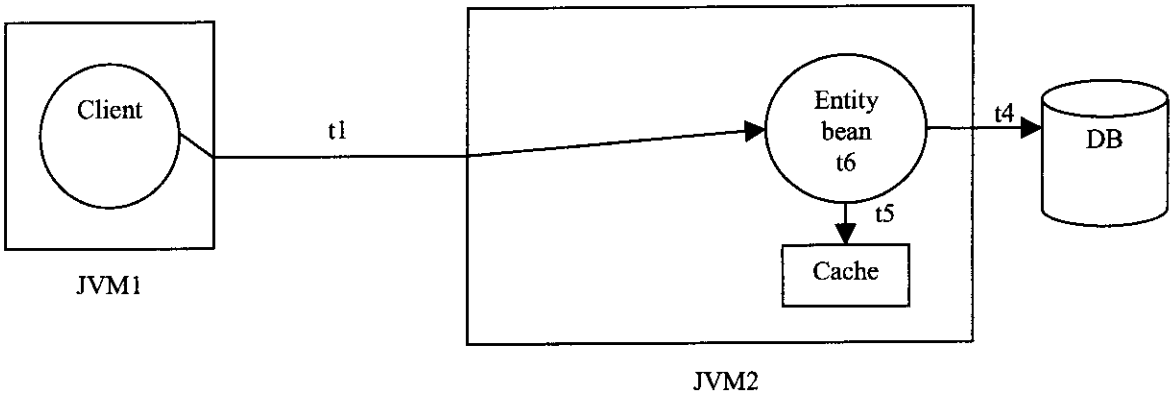

Figure 2. The single-entity scenario.

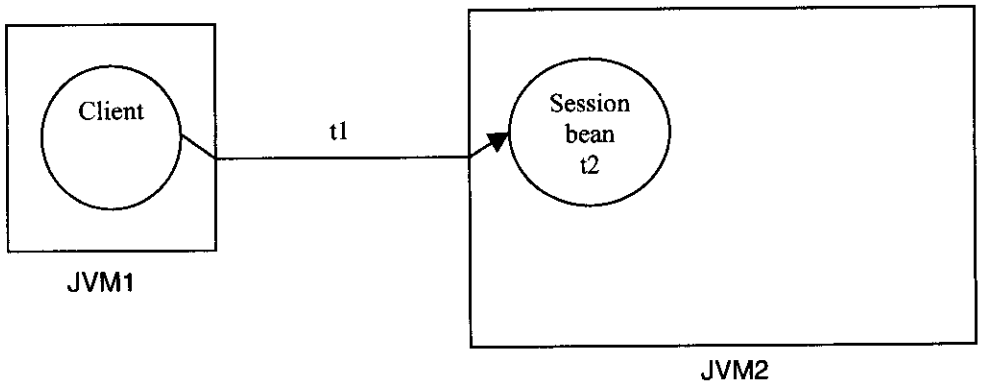

Figure 3. The single-session scenario.

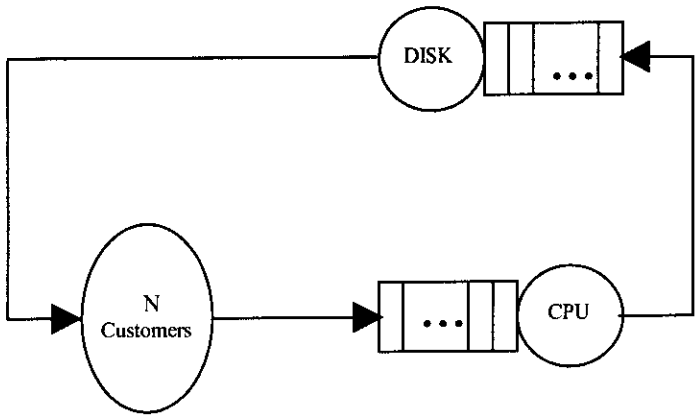

Figure 4. A closed queueing network of 2 servers and N customers.

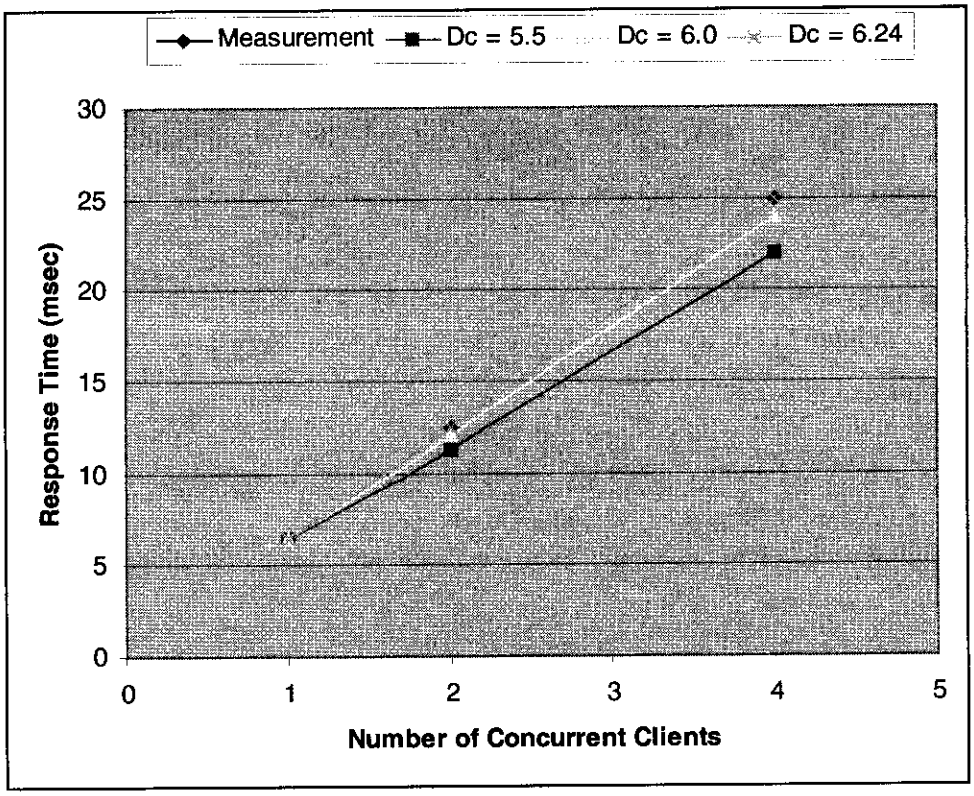

Figure 5. Measured response time compared to 4 sets of predicted response time.