Testing J2EE clustering performance
and what we found there*

Davide Rossi, Elisa Turrini
Dipartimento di Scienze dell'Informazione, Università di Bologna
{rossi|turrini}@cs.unibo.it

Abstract

In this paper we investigate issues about clustering performance of JBoss, an open source J2EE compliant application server. We evaluate: (I) how clustering affects the performance of the system; (II) which is the best setup to improve clustering performance; (III) if it is more convenient to manage session information at the Web tier or at the Business tier.

1. Introduction

In the last few years the Java 2 Enterprise Edition (J2EE) platform [1] has become the most popular technology for the design and the development of enterprise applications. J2EE is a distributed multitiered architecture: a typical application is composed by a client tier (usually composed by a web browser but it can be a specific application as well), a Web tier (present only when the clients are browsers or software components that support web services), a Business tier that contains the components implementing the business logic of the application, and an Enterprise Information System (EIS) tier that stores the persistent data used by the application. The EIS is usually a relational database that connects with the application server using specific technologies (such as JDBC).

In this paper we investigate the clustering performance of JBoss[2], an open source J2EE compliant server, by comparing different load balancing and high availability strategies that can be implemented at Web tier and at the Business tier. Specifically, we aim to find out the weak and strong points of a clustering architecture. We also investigate (I) how clustering affects the performance of the whole system, (II) which is the best setup to improve clustering performance, (III) if it is more convenient to manage session information at Web tier or at Business tier. Our studies show that clustering should be considered with care, mostly if session data replications is involved. Due to the severe impact session data replication has on system performance, it should be applied only to the (restrict) class of applications that cannot afford to lose session information.

The paper is organized as follows: in the next session a background of the J2EE architecture is provided, in Section 3 we present some interesting related work in this area. In Section 4 and 5 we describe the test environment and the experimental results. Finally, in Section 6, we present the conclusion and the future work.

2. Background

2.1 J2EE architecture

A J2EE compliant application server is composed by the Web container (or servlet container) and the Business container (or EBJ contatiner); the Web container hosts presentational components (the servlets) whereas the Business container hosts business logic components (the Enterprise Java Beans, EJBs). In the logical architecture of J2EE the web container implements the web tier whereas the EJB container implements the business tier.

Clustering cross-cuts these two logical tiers in multiple physical locations. This means that both the component container of the Web tier and the component container of the EJB tier can be hosted in multiple nodes.

This leads to several possible solutions when taking clustering into account:

- Web tier and EJB tier co-located, no clustering;
- Web tier and EJB tier physically split, no clustering;
- Web tier and EJB tier co-located, with clustering;

* This work was partially supported by the EU project TAPAS (IST-2001-34069) and the WebMinds FIRB project of the Italian Ministry of Education, University and Research

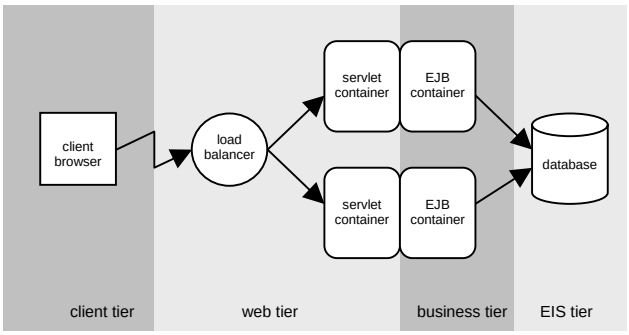

Figure 1: JBoss suggested configuration for clustering

- Web tier and EJB tier physically split, with clustering.

All of these alternatives are possible and a number of issues have to be taken into account when choosing the better one.

Web tier and EJB tier co-located, no clustering.

When the tiers are co-located and there is no clustering the entire application server runs on a single node. The invocation of a component in a different tier can be executed in process, there is no need from remote invocations: a single request from the client tier can be satisfied by the application server inside a single node (apart from the access to the EIS). Of course the system cannot scale and we are in the presence of a single point of failure.

Web tier and EJB tier physically split, no clustering.

The application server is hosted in two different nodes; the first one hosts the servlet container, the second one hosts the EJB container. Each access from the client to the application thus involves (at least) one network call between components in the two tiers (plus the potential access to the EIS). The components in the Web tier and in the EJB tier draw resources from two different machines but we now have two single points of failure.

Web tier and EJB tier co-located, with clustering.

The application server is hosted in multiple nodes: each node hosts both a servlet container and a EJB container. Requests from the client tier can be satisfied inside a single node (apart from the access to the EIS). The nodes can be used for load-balancing and for high availability purposes.

Web tier and EJB tier physically split, with clustering.

The application server is hosted in multiple nodes: each node hosts either a servlet container or a EJB container.

Each access from the client to the application thus involves (at least) one network call between components in the two tiers (plus the potential access to the EIS). Both the nodes that host the Web tier and the nodes that host the EJB tier can be used for load-balancing and for high availability purposes.

2.2 Clustering

What are the advantages of using a clustered solution?

A cluster provides to the client a unified view of the services that each node offers individually. Basically, the clustering has been introduced to improve availability and throughput. With availability we mean the percentage of time the system is available for use by its client. In a cluster, if a node crashes (for an hardware or a software failure), the requests originally meant for the failed node can be redirected to other nodes belonging to the same cluster. More nodes are present in a cluster, higher the availability offered by that cluster will be. The throughput is the number of requests that the system can satisfy in a given amount of time. Another advantage of clustering is improved throughput: when a request arrives, it can be redirected to whichever node in the cluster by applying a load balancing policy thus sharing the load among all the nodes of the cluster.

In the J2EE architecture, the load balancing policy can be applied either at the Web tier or at the Business tier, according to which architecture is used. In the co-located tiers architecture, the load balancing of the requests should be done at the Web tier. We present two possible solutions for load balancing at the Web tier:

Apache and mod_jk

The cluster is front-ended by a load balancer that redirects the requests to the back-end nodes according to the Round-Robin policy. This solution can be implemented using Apache and mod_jk[3]. Note that in this architecture the web server running Apache with mod_jk is a single point of failure, for this reason, one should configure other machines that can on the fly substitute the crashed server.

HTTPLoadBalancer

If the J2EE server is JBoss, HTTPLoadBalancer[4] can be an alternative solution. HTTPLoadBalancer is a package deployable in JBoss. It acts as reverse proxy: it redirects the requests to the nodes and collects the responses. HTTPLoadBalancer can be set to apply a load balancing policy that takes into account the real load of every node.

At the Business tier, JBoss implements three load balancing policies: *Round Robin*, *First available*, and *AvailableIdenticalAllProxies*. These policies are automatically disabled for local calls, as it will be too much expensive, in terms of performance, to invoke a component running on a different Java Virtual

Machine (JVM) when an in process solution is possible. Because of this JBoss embedded load balancing techniques can be applied in the case of Web tier and EJB tier physically split architecture only. It is worth noting that no one of the current load balancing JBoss policies takes into account real nodes load conditions.

Both clustering options described previously (co-located tiers and split tiers) allow for load-balancing and high availability; it should however also be noticed that, while load-balancing comes essentially for free, when enabling high availability there is a (big) price to pay caused by the necessity of maintaining a coherent session state. Typically, when a user visits a given site a session is established. During a session, information concerning a given user, or the action he/she performs, are maintained on the server. These information are typically discarded when the user session is over hence they are not committed to stable storage as persistent data. In an e-commerce site, a typical example of this information is the shopping cart.

Since the HTTP protocol has not being specifically designed to manage sessions (it is a session less protocol), a small chunk of information (a cookie) is included by the client with its requests in order to allow the application server to associate a user with their session data. On the server-side, session information can be maintained at Web tier (in a shared component accessible by the servlets) or at EJB tier, using the stateful session beans (SFSB) that are state-aware components. In both cases, if a node containing a stateful components crashes and we want another node to take over, the new node has to carry a state-aware copy of the stateful components that were in the crashed node or should be able to dynamically create new state-aware copies when needed (this second option, typically implemented via checkpointing and recovery, is not usually a viable one for J2EE servers so in the rest of the paper we will focus on the first option: active replication).

Note that if session information are not replicated, when a node crashes only the sessions data stored in that node are lost. This implies that users belonging to those sessions have to initiate a new session (e.g. to login again into the system); no consequences impact other users with sessions data stored into different machines. These considerations imply that only critical applications need session replications. When replication is needed all the J2EE compliant application servers that support clustering achieve high availability by having active, synchronized replicas of the state-aware components instantiated in multiple nodes of the cluster.

Unfortunately, maintaining a coherent state among nodes in a cluster can be quite expensive as it requires messages exchange among nodes. This augments the load of the nodes, and reduce the resources that a node can use to satisfy user requests, affecting negatively the performance of the application.

Since J2EE is a transactional architecture in which all clients requests that get into the EJB tier are enclosed in a transaction, the replicated components have to synchronize their state before the transaction is over. This is commonly achieved by using a reliable multicast layer that ensures that the new state of a component is correctly distributed to its replica in a synchronous way before committing the transaction. This implies that keeping a correct synchronization between distributed replicas of state-aware components in a J2EE architecture introduces a not-negligible overhead: this is the price to pay we were referring to above.

The motivations of having a cluster is to improve both availability and throughput, but, as previously explained, improving the availability by sessions data replication can worsen the throughput. In the next sections we measure with experimental tests how sessions data replication affects the throughput.

3. Related Works

The performance of J2EE technologies is a hot issue in the e-business community. This wide research area can be divided in three sub-areas:

- comparing the performance of the different J2EE-compliant servers in order to find which is the best;
- find the setup, for each J2EE-compliant server, that allows to achieve better performance;
- analyzing the design patterns that can be applied in the development of a J2EE application in order to find the ones that ensure the better performance.

One of the main testing performance problems is to decide which application should be used for testing, as the results can change considerably according to the characteristics of the application and to the adopted methodology.

ECperf[11] is a widely used tool, it consists of a benchmark and implementation for measuring performance and scalability of J2EE-compliant servers. In [6][7] the authors use ECperf to identify and discuss the factors that have the most relevant performance impact on J2EE applications. They also examine the point that are crucial for scalability and that could often turn into system bottlenecks. The authors then propose a list of optimization techniques

that could be applied to boost the performance of any arbitrary J2EE application.

The main drawback of using ECperf is that all the tests are performed on a specific, although realistic, application.

As an example of a different approach, in [5] the authors perform a comparison between two J2EE servers, Jonas [12] and JBoss, by comparing six versions of the same application implemented using different strategies. Specifically, the versions differ in the business logic location (servlets or EJBs) and in the methods used to interface with the persistent storage (CMP and BMP entity bean). To some extent this paper is similar to ours as it evaluates different design strategies applied to a specific application; the differences with respect to our work are mainly two: (I) we are interested in specifically evaluating clustering performance; (II) we do not compare implementation choices but session data replication strategies.

An alternative approach to performance evaluation for J2EE applications is proposed in [8], where the authors model the characteristics of the application server and the application itself in order to predict the performance of the application once deployed. The performance model allows the system designer to make decisions among alternative implementation strategies. It is interesting to note that the authors also explore the performance of an application deployed over a two-nodes server cluster obtaining results similar to those of our tests.

4. Experimental Environment

Several J2EE compliant servers are available. We chose JBoss because, at the time of this writing, it was the only open source application server that supported clustering at the Business tier level. In order to stress JBoss, we implemented a typical e-business application: a virtual book store.

Client emulator

In order to emulate client load and take measurements we used JMeter[9]. We emulated a typical user session (home page, login, visualize items, add to cart, delete from cart, confirm order, buy). In each experiment, we ran for a total of 60 users at the same time, each user performed 20 sessions (we chose these figures after different tests since they assure a request rate high enough in order to maximize the throughput but not so large to overload the server).

Server Environment

We used JBoss 3.2.3 in standard configuration. The chosen servlet container was Jakarta Tomcat v. 4.1 (embedded in JBoss). We made up a cluster of two machines with similar hardware and software characteristics (Pentium IV 2GHz processor, 1 GB of RAM, Debian Linux OS).

For dispatching HTTP requests we installed Apache Web server with mod-jk2 (as proposed in JBoss documentation) on a different machine; the load balancing policy applied was Round-Robin. An alternative solution to the use of Apache+mod_jk2 is HTTPLoadBalancer, but at the time we deployed our tests only a Beta version was available.

In the implementation of the standard cluster configuration shown in the Figure 1, we used MaxDB [10] as BDMS and located it in a dedicated host. We also tested a setup with HSQL [15] (an in-memory database) in place of MaxDB.

5. Experimental results

In this section we describe the experiments we performed and the results we obtained.

Single node

First of all, we ran the benchmark against the application deployed in a single node obtaining the results shown in Table 1.

Response time (ms)	Throughput (res/sec)
507	80,4

Table 1

Two nodes: standard configuration

After that, we ran the benchmark against a cluster set up as suggested in the JBoss documentation: apache+mod_jk, two machines running JBoss instances and the database on a dedicated host (see Figure 1). We obtained a performance degradation: as shown in Table 2, the response time is more than tripled, while the throughput has become a fourth. Since the application was deployed with no replication at all, the performance degradation could be caused by the overhead introduced by mod_jk or by the one introduced by the database when accessed by multiple clients (e.g. the database could turn out in a bottleneck). To better understand, we evaluated separately the overhead caused by these two entities.

Response time (ms)	Throughput (res/sec)
1726	24,6

Table 2

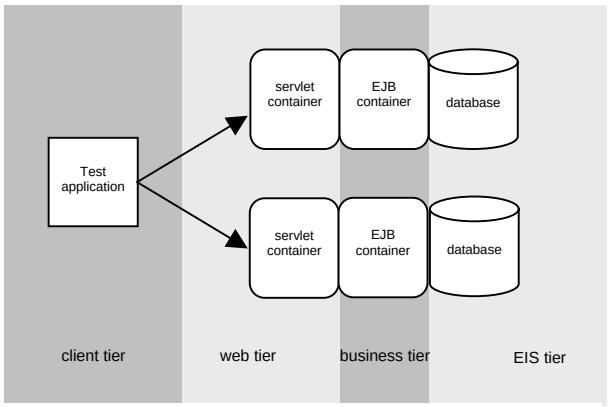

Figure 2: our test configuration

Mod_jk2

We evaluated the overhead introduced by mod_jk. We ran the benchmark against a single node with mod_jk at the front-end. The results in Table 3 show that the performance degradation is about 20%, both for the response time and for the throughput.

Response time (ms)	Throughput (res/sec)
610	67

Table 3

Database

In order to evaluate the overhead introduced by the database, we tested the performance of a single node when another node is accessing to the database with similar load. The results in Table 4 show that the response time doubled while the throughput becomes one half.

Response time (ms)	Throughput (res/sec)
1020	47

Table 4

It is easy to see that both mod_jk and the bottleneck at the database introduce large overheads. Then, in order to insulate the performance of the clustering we decided to change our architecture: as shown in Figure 2, we used an instance of HSQLDB in each node in order to avoid all possible resource contention at the database level and we load-balanced the requests directly at the client level (of course this is a meaningless setting in a production systems, but it makes sense for our testing purposes).

In order to test the effectiveness of different configurations where session information are stored or in the Web tier or in the EJB tier, we implemented two versions (functionally equivalent) of our test application: one that uses servlet's sessions and one that uses stateful session beans. Note that the only way to find the stateful session bean that is holding the data relative to a single client is to store the reference for this bean in a such a way the client can find it across subsequent invocations. But if we store this reference in a single node we have a new single point of failure. There is no easy workaround for this problem but cookies can be used to force the client itself to store a serialized version of this reference and send it along with its requests: the new single point of failure is the browser but this is a single point of failure anyway for most web applications.

For our stress tests we chose the following configurations:

- C1: single node-application server; application with session state handling at the Web tier;
- C2: double node-application server; application with session state handling at the Web tier; no state replication;
- C3: double node-application server; application with session state handling at the Web tier; Web tier (servlet sessions) state replication;
- C4: double node-application server; application with session state handling at the Business tier; EJB tier (stateful session beans) state replication;
- C5: double node-application server; application with session state handling at the Business tier; both Web tier and EJB tier state replication.

The results obtained with these different configurations are shown in Table 5 and, graphically in Figure 3.

Configuration	Response time (ms)	Throughput (res/sec)
C1	507	80,4
C2	256	145,9
C3	1207	45,8
C4	853	58,2
C5	2251	40,1

Table 5

Comparing the results of C2 and C3, it is easy to see how expensive the synchronization of the replicated servlet sessions is. The response time is about five (!) times the one obtained with the two independent nodes and the throughput is one third.

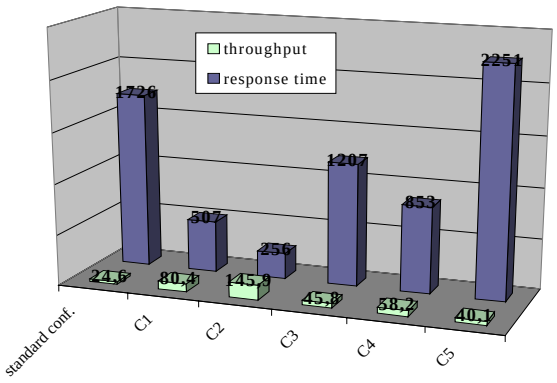

Figure 3: test results

Comparing C3 and C4, it seems that replication at the EJB tier level is less expensive than the one at the Web tier level. But, as we explained above, we have the problem of storing the reference to the stateful session bean that holds the session data for a specific user.
It is easy to see how bad these results are. It is even worse if we think that C3 is probably the most used clustered configuration[13][14]. Session replication should only be used for the most critical application where re-login and lose your session data is not an option.

6. Conclusions and Future work

Clustering is an exciting feature of the J2EE architecture: it promises both load balancing and high availability. The two things, however, do not go along very well. Load balancing can be achieved quite easily with an hardware load balancer in front of the application server cluster (while software solutions like mod_jk have inherent limitations and should be deployed with care) but high availability imposes a huge price to pay both in terms of response time and throughput. In our tests it turns out that enabling state replication between different nodes leads to performance that are worse than the one of the single node by two/three times (at least) voiding all the possible advantages of load balancing.

It should be noted that the results we obtained are strongly application dependant. We tried to run our tests against a very standard application. By avoiding contention on the database we probably inflated the advantages of load balancing with no replication, but our aim was measure the overhead of high availability solutions inside the application server; for real world applications both the overhead of replication and database contention has to be taken into account.
We are currently setting up other test applications and test suites in order to better take into account real-world applications and load patterns. In the stress tests we presented in this paper all the simulated users access one resource after another with no pause whatsoever. This is meaningful in order to test the overhead of architectural solutions in the application server but in order to test an applications under realistic load we have to model the users: their path between the views dispatched by the web applications, their think time and so on. To run this kind of tests we are implementing an extension of JMeter to ease the simulation of a group of “real” users. Once the tool is ready we expect more interesting results about J2EE clustering in real world applications.

7. References

[1] http://java.sun.com/j2ee/
[2] http://www.jboss.org/index.html
[3] http://jakarta.apache.org/tomcat/tomcat-3.3-doc/mod_jk-howto.html
[4] http://www.jboss.org/wiki/PageInfo.jsp?page=HTTPLoadbalancer
[5] E. Cecchet, J. Marguerite and W. Zwaenepoel, “Performance and scalability of EJB applications”, in Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (Oopsla 2002), Seattle, WA, USA, November 2002.
[6] S. Kounev and A. Buchmann, “Performance Issues in E-Business Systems”, in Proceedings of the International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet (SSGRR-2002w), L'Aquila, Italy, January 2002
[7] S. Kounev and A. Buchmann, “Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services”, in Proceedings of the 28th International Conference on Very Large DataBases (VLDB'02), Hong-Kong, August 2002.
[8] Y.Liu, A. Fekete, I. Gorton, “Design Level performance modelling of Component-based Applications”, Technical Report n. 543, November 2003
[9] http://jakarta.apache.org/jmeter/
[10] http://www.mysql.com/products/maxdb/
[11] http://www.spec.org/jAppServer2004/
[12] http://jonas.objectweb.org/
[13] Sacha Labourey and B. Burke, “JBoss Clustering”, JBoss official documentation series.
[14] Howto: Clustering with JOnAS, available at http://jonas.objectweb.org/current/doc/howto/Clustering.html
[15] http://hsqldb.sourceforge.net/