Sessions Design and Management for Java Autonomous Web Applications

Patrick Bellot

Institut Telecom, Telecom ParisTech, CNRS LTCI-UMR 5141
46 rue Barrault, 75634 Paris Cedex 13, France

bellot@telecom-paristech.fr

Abstract. In [11], we presented SAW, a Web application development framework that can be efficiently applied to the programming of a wide range of Web services and applications. SAW is not an open Web server. It is designed to easily program a single Web application using pre-defined software patterns and components. We have implemented SAW in Java. It has been successfully used to propose a fully functional online conference management Web application. After an overall description of the server, we give the original design guidelines concerning the session objects that allow the server to have high performances. Finally, we explain how we plan to provide scalability, reliability and availability.

Keywords: Web applications and services, autonomous server, availability reliability, scalability, Java.

1 Introduction

Web applications that are not implemented using a Web Application Servers (WAS) such as GlassFish, IBM WebSphere, BEA Weblogic, JOnAS, JBoss, and so on, are built using LAMP architectures [1]. The L (Linux) can be optional but A (Apache), M (MySql) and P (PHP) are widely used. LAMP architectures are powerful and efficient tools with many available extensions. A possible option is to use the above-mentioned WAS. They are heavy tools providing powerful development environment (IDE) and multiple easy-to-use functions implementing most of Web 2.0 and Web 3.0 advances. This approach is well suited for Enterprise Information Systems (EIS) because all enterprise services and applications, internal and external, can be integrated in a single framework that becomes corporate culture. Finally, developers can use a Content Management System (CMS) usually implemented on the top of LAMP architectures. Many such tools allow an easy conception of Web sites with elegant designs following the numerous templates available on the Web. We have used this solution for the institutional Web site of our organization http://www.infres.enst.fr. In such a system, it is easy to have a living web site, to produce new pages, to organize them, to integrate community tools such as blogs. Integrating business logic in a CMS web site is of course possible but not always straightforward and performances are not very high because of the use of several software layers.

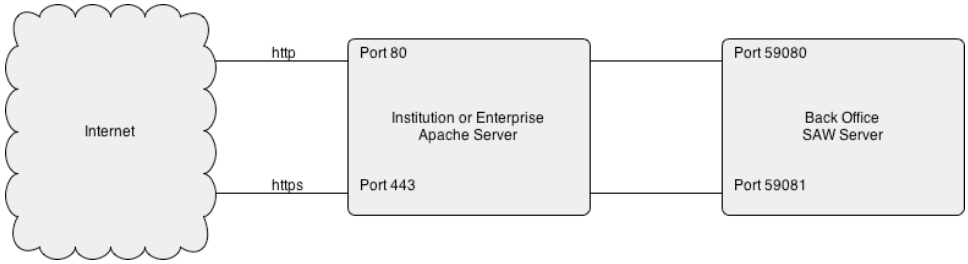
Our proposal, presented in [11], is to build Java dedicated standalone Web applications based on our framework without using any external components except the file system. SAW provides many tools allowing to easily develop efficient Web applications. Our technology cannot be used in place of LAMP or WAS architectures. A LAMP or a WAS architecture is open: everyone who is administratively authorized can implement new applicative pages and develop new Web sites without practically impacting the server performances. Many powerful options, languages, specific modules are available on LAMP architectures such as an Apache server and its extensions. SAW does not allow that. A SAW server is not intended to be an open server where authorized users can deposit static or dynamic contents. Of course, it is possible to add new pages or functions as far as it does not require new Entity objects classes or modifications of the kernel but this is not one of the goals of SAW. The server is designed to be dedicated to a single Web application.

SAW is the concatenation of many interesting paradigms and tools that make it more efficient and easier to program than a classical LAMP architecture. SAW stands for Stand Alone Web server and is written in Java. A lot of Web servers proposals are written in Java and available as Open Source software [2]. They are able to deliver static contents and sometimes, they implement Servlet and JSP technologies. SAW tries to avoid WAS weight and complexity while providing the well known WAS triplet: Container, Session beans and Entity beans [7] and numerous design patterns for implementing application components. Session beans implement the business logic and are responsible for answering the requests of the clients. Containers provides an execution context to the Session beans such that the software developer does not have to care about technical aspects such as handling TLS/SSL, decoding the request, managing the GET and POST variables or managing the user's session environment. The programmer of a Session beans only focus on the business logic. Its container must provide everything required to implement the business logic. Entity beans classically contain the data of the Web application and are managed using hibernation mechanisms [10]. SAW Entity objects and their management have been described in [15]. Entity objects are persistent data. They allow shared access that can be synchronized or not depending on the data intended lifecycle. An Entity object contains serializable data. They are loaded from disk files at the start of the server and saved to disk files at the shutdown the server using Java serialization. Backups are done at regular intervals of time because a power cut is always possible. Collections of Entity objects may be used by a Web application, for instance the collection of the registered users of the Web site. They have primary keys and may participate in relationships with other Entity objects. They can be grouped in collections named Catalogs. We provide the indexing primitives. Index keys can be integers or objects. There exist indexing tools where the key is unique as a primary key, for instance the e-mail of the user. And there exist indexing tools where the key is not unique, for instance the last name of users. These indexing tools, based on hash tables, allow very fast retrieval of data in memory.

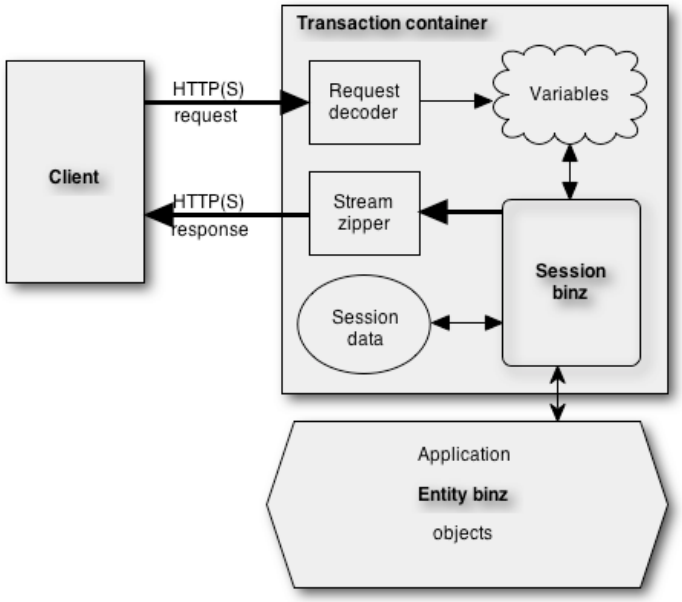
We have used SAW to implement a fully functional and very complete online conference management system [5]. It is operational since January 2010 and used for the Asian conference IEEE-RIVF in Vietnam with around 800 registered users, 110 program committee members, 150 submitted papers and at least 3 reviews per paper.

It covers every aspect of a conference: papers submission, papers reviewing, papers selection, mailing, registrations, management of committees, etc.

Today, because of the rapid emergence of B2C and B2B, there is a real need for high performance and scalable web servers [9]. A SAW server is intended to be executed by any reasonable today computer and proxied by an Apache server if it does not have a direct Internet access. An Apache server is able to efficiently proxy many SAW Web applications. The Apache server and the SAW applications do not need to be executed on the same computer. That means that a single Apache server can be a front-end for many back-end SAW applications or services that are distributed on a farm of low cost backend computers. This decreases the load of the front-end server. This ensures horizontal scalability of Web application servers across multiple computers with a reasonable economical impact. That is the way we deployed our online conference management application that was running on a dedicated single core Intel computer with 2 GB of memory.



We detailed in [11] the design of SAW, see figure below. One of the reasons of its efficiency is that it is dedicated to a single application, thus allowing memory caching of Session and Entity objects. For instance, the data memory size of our online conference management application does not exceed 5 MB. Memory caching can always be replaced by file caching for Entity objects with big memory size or Session objects with big output. Using Java classical hibernation mechanisms such as JPA (Java Persistence API) or directly accessing databases is also a solution when big amount of data are required by the application but it requires explicit programming.



A SAW application has to be seen as a pure Java program where Entities objects are the global data of the application and Sessions objects are concurrent entry points. Current computers in 2010 have high computational capabilities, around 3 GHz processors and several cores, big memory caches in processors and very big RAM memories (8 GB is now a common memory size). That is why the approach we propose is realistic. SAW is implemented in Java but a C++ implementation has been realized for evaluation purposes. Meanwhile it is significantly more efficient than the Java implementation, we gave up the C++ implementation because it was really too difficult to debug and it required a lot of work to port it to various OS.

In section 2, we describe our containers, named Transaction objects, and how they process incoming requests and call a Session object handle to answer the request. In section 3, we explain how session objects output can be programmed and compiled to provide high efficiency. Section 4 is dedicated to the optimized delivery of static contents. Section 5 and 6 describes the caching mechanism proposed by SAW. Finally, section 7 explains how we plan to implement scalability, reliability and availability.

2 Transaction objects

The notion of container is probably the most innovative and crucial paradigm of Web Application Servers. In SAW, containers are called Transaction objects. They have been carefully designed and efficiently implemented because they are the core support of all transactions. Transaction objects are Java threads waiting for an incoming request. After processing the request, the Transaction object returns to its wait state. A pool of Transaction objects is created at the launch of the server and these Transaction

objects are kept ready to handle request connections. When a HTTP or HTTPS request reaches the server, a Transaction object is responsible of its handling. It is must manage TLS/SSL encryption if necessary. It decodes the lines of the HTTP request header, the GET parameters provided in the request header path and the POST content if any. If SAW is proxied by an Apache server, there is no need to support SSL connections.

According to the header request path, the Transaction object selects the appropriate Session object that will answer to the request. It calls the handle method of the Session object with itself as a parameter so that the Session object can always access the data and functions of its associated Transaction object. The Transaction object is responsible for dynamically loading the Session object class in memory and creating the Session object. Session objects are stored in a hash table where the keys are the request paths. If the Session object class is recompiled, the Transaction object will renew the Session object in memory after reloading the new Session object class. That's also a technical reason for the Session objects to be stateless. This dynamic loading of Session objects is mainly used to debug the Web application; it could be disabled when the Web application is in exploitation.
Transaction objects provide many output functions to the Session objects:

- to output various standard HTTP response headers so that the developer does not have to build them ; these response headers include the session cookie necessary to maintain a session environment.
- to output many types of data such as strings, arrays of bytes, numbers, etc.
- to output many standard pre-defined HTTP responses such as page not found, internal error, and so on.

These three types of functions can be extended to output any type of object in the Transaction object class depending on the Web application requirements. The Transaction object is also responsible for zipping the output content (after the response header) if the client claims to accept zipped content as most of today's browsers do. Concerning the Web application data, the Transaction object gives access to:

- the Entities objects that contain the data of the Web application, these objects are entirely application dependant as explained in [15] ; they are usually implemented with classical Java Collections data structures ;
- the GET and POST variables in a Transaction variables object which is no more than an optimized hash table where keys and values are strings ;
- the session environment: this an object containing the data which are remanent from one request to another like PHP session variables.

Of course, the session environment contains session variables and a few mandatory data such as user identifier, client IP, session identifier and last access time. Client IP is used to avoid session identifier fraud attempts. Last access time is used to manage session timeout. The Web application designer, if needed, can freely extend it with new attributes. Any serializable Java object can be part of the extended session

environment. The handle method of a Session object is its business method receiving the Transaction object as a parameter. Typically, it is programmed as follows:

- If necessary, the Session object handle method looks at the session user, inside the session object, to validate it. For instance, an already logged user may be required. Sometimes, the method may have to check the user rights in the context of the Web application.
- Then the handle method looks at the Transaction variables (GET and POST variables) to validate them. If they are not sound, the Session object handle method can raise an exception asking the Transaction object to log the error, to issue a bad request answer (suspecting a fraud tentative for instance) and then terminates.
- And the method does its business, preparing its response. The Session object has access to the Web application Entity objects containing the application data. As all Session objects have *a priori* access to all Entity objects, some kind of synchronization must be required depending on the application. The use of a non-coherent object may result in an exception caught by the Transaction object.

The steps described above can be mixed but it is a safe programming rule to program the business method as described, especially to get a coherent response in case of error. For instance, if an error is detected after the output has begun, then the error message will be mixed with the beginning of the normal output. That is why all checks have to be done before the output begins. If an error occurs during the processing of the request by the Session object, the Transaction object that called the Session object catches it. According to the exception associated with the error, the Transaction can automatically output a selected HTTP response. That means that the programmer of a Session object does not have to care too much about errors. If an error occurs during the execution of the handle method of a Session object, the error is logged and the associated Transaction object replies with an error page.

As presented, the system may look as a LAMP system where PHP has been replaced by Java. Indeed, we can use Java in the same way we use PHP. For instance, one may only use output functions only to produce an XHTML page. However, we may appreciate to be able to use Java without any restriction and the numerous SAW tools that will be described in this article. They allow very fast processing of requests while keeping programming of Session objects functions very easy. These tools allows to pre-compile static content of pages and to cache most of the outputs. From the point of view of efficiency, Entities objects are also very interesting. They can be accessed in memory and manipulated using all the power of Java libraries because they are ordinary Java Collections data structures. Entities objects replace databases as backend application data holder.

3 Elements and compilation

XML elements are generically represented by Java classes provided by SAW. All XHTML elements are implemented this way so that the output of a Session object can be described by a SAW Element. For instance:

```
Element object =
```

```
new Html(new Head(new Title(…))),
         New Body(new H1(…),
                  new P(…),
                  new Table(new Tr(…),…),
                  …)) ;
```

The programmer can easily implement dynamic elements which outputs are dynamically computed. It is sufficient to extend the `DynamicElement` abstract class and to implement the `writeTo(…)` method. For instance, a dynamic element can compute and output the current date:

```
public class DateElement
   extends DynamicElement
{
   public String getCurrentDate()
   {
      return … ;
   }

   public void writeTo(TransactionOutput out)
   {
      out.write(getCurrentDate()) ;
   }
}
```

Then, it can be used as any predefined element for producing a page:

```
Element object =
   new Html(new Head(new Title(…))),
            New Body(new H1(…),
                     …,
                     new DateElement(),
                     …)) ;
```

To be more efficient, elements are compiled before using them. The result of the compilation of an element is an array of atomic compilations that will be executed one after the other. An atomic compilation can be simply an array of bytes corresponding to a static content. This array of bytes has to be output as is by the Transaction object. Or an atomic compilation can be a call to the `writeTo(…)` method of a dynamic element that will compute its output as `DateElement` above. Several consecutive arrays of bytes in a Compilation array are merged into a single array. If we consider the previous example and assume that the date element is the only dynamic element in the page, the result of the compilation will be an array of three atomic compilations: the array of bytes corresponding to the text before the date element, a call to the date element and the array of bytes corresponding to the text after the date element.

As can be seen, this kind of compilation is optimal because all parts of the output that can be pre-computed become array of bytes ready to be output. This efficiency can be enhanced with an automatic memory or file caching mechanism provided by SAW classes. Caching is applicable only if the element does not access the Transaction variables or the session environment. For instance, in our online conference management system, we use a country selection widget. The HTML code for this country selector is provided by a dynamic element that computes it from the Countries Entity object. However, this element is a cached element. That means that this country selector HTML code is computed once and then cached to be delivered when required. This is obtained by inheriting a special Entity class provided by SAW.

The Observer-Observable design pattern described in section 6 is used to renew the caches when appropriate.

4 Delivering static content

The server is of course able to deliver static content such as HTML, JavaScript, CSS, images, PDF documents, Office documents, zipped files, etc. SAW distinguishes four categories: HTML, JavaScript, CSS and other data. Each category has dedicated Session classes. When a static content is requested, a Session object is automatically created to deliver it according to the category of the static content. This Session object manages two caches; one for the original static content and one for the zipped static content for the case where the client accepts zipped content. The caches can be in memory if not too big. Otherwise, it is in a disk file. The programmer set the threshold under which the cache is in memory. In our online conference management application, this threshold has been set to 128 KB meaning that probably all static contents (HTML, CSS, JavaScript, images, etc.) are kept in memory.

For structured language content such as HTML, JavaScript and CSS, the dedicated Session object automatically compacts the original content before caching to reduce the delivery size. Implemented compaction rules depend on the language: for instance removing end of line characters or replacing adjacent space characters with a single one when not in a string, removing comments, etc. That means that the developer does not have to compact the files.

A well-known performance tip when writing HTML content is to avoid too much external files such as linked CSS or JavaScript files. This is for not having too much parallel requests when loading a page. One solution, for instance, is to join all CSS files into one file on the server side but this is against modularity of code and it is more difficult to handle by the programmers. To solve this problem, SAW provides specialized Session objects that collect several files but deliver them as a single file to the client. This can be done for HTML, CSS and JavaScript. Of course, this Session objects also implement the caching mechanism and the compaction techniques described above. In our online conference management application, we provide a site wide global CSS file implemented with the Session class that collects several CSS files into one delivery. Then each page uses this global CSS, which is loaded once by the browser, and a page specific CSS file when required.

5 Cached Session objects

When a Session object does not use the Transaction variables or the session environment, its output can be cached either in memory if not too big or in a disk file. If a Session object output cannot be cached, it can nevertheless be represented by a XML element that is compiled and the static content of the page is stored in array of bytes ready to be output as explained in section 3.

In order to cache a Session object output, SAW provides specialized Session classes that automatically cache their outputs. The programmer chooses if the cache must be in memory or in a disk file depending on the expected output size. As shown below, it defines its Session class by inheriting the appropriate memory or file caching class and it provides a Compilation object for the Session object as described in section 3. The first time the Session object produces its output, the original output is cached and a zipped version of it is also cached in the case where the client accepts zipped content. The next section describes how caches are refreshed. Here follows an example of class cached in memory:

```
public class  ImportantDates
   extends SessionWithMemoryCache
   implements Session
{
   Compilation compilation ;

   public Compilation getCompilation()
   {
     return compilation ;
   }

   public ImportantDates()
   {
      compilation = … ;
   }
}
```

It is easy to understand that the caching of Entity and Session objects increases efficiency, much more when the caches are in memory. It can be asked if too much memory caches can exhaust the available program memory. Our online conference management application is not a toy Web application. With more than 800 users, 150 papers and at least 3 reviews per paper, it uses less than 25 MB of memory. That is far less than the available physical memory.

6 Refreshing caches

When a Session object is cached, its cache may need to be renewed. This is also applicable to the cached dynamic elements of section 3. For instance, in our online conference management application, there is a cached page displaying the Program Committee (PC) of the conference. Each time the PC is modified, the normal and zipped caches of this page have to be renewed. There exists an Entity object containing the list of the PC members. A modification of this PC Entity object must trigger the caches refresh of the PC page Session object.

The Observer-Observable design pattern [16] provides a straightforward solution. This pattern is sometimes used in the Model-View-Controller (MVC) design pattern for graphical interfaces. The model, that is the data, is the Observable and the different graphical views of the data are the observers. When the model is modified,

the views must be updated. In our case, the PC Entity object is the Observable. The Observers are all the cached Session objects or dynamic elements that depend on the PC Entity object. Observers and Observable have a publish-subscribe relationship. Observers subscribe to receive notifications from the Observable. After the Observable is modified, it notifies all the Observers. That is an obvious way to automatically trigger the refresh of the caches.

7 Scalability, reliability and availability

Due to the increasing usage of Internet B2B and B2C applications, the problems of scalability, reliability and availability are crucial. Scalability means that we are always able to enhance the server capacity when the traffic increases. Everyone with an idea can develop a Web application and install it on the Web with the help of an Internet Service Provider. However, a single server that handles all incoming requests does not have the capacity to support high traffic volumes. Increasing the server capacity in terms of processor speed, number of cores or memory size is only a short-term solution [17]. Moreover, this server is a single point of failure and cannot be declared as reliable and available.

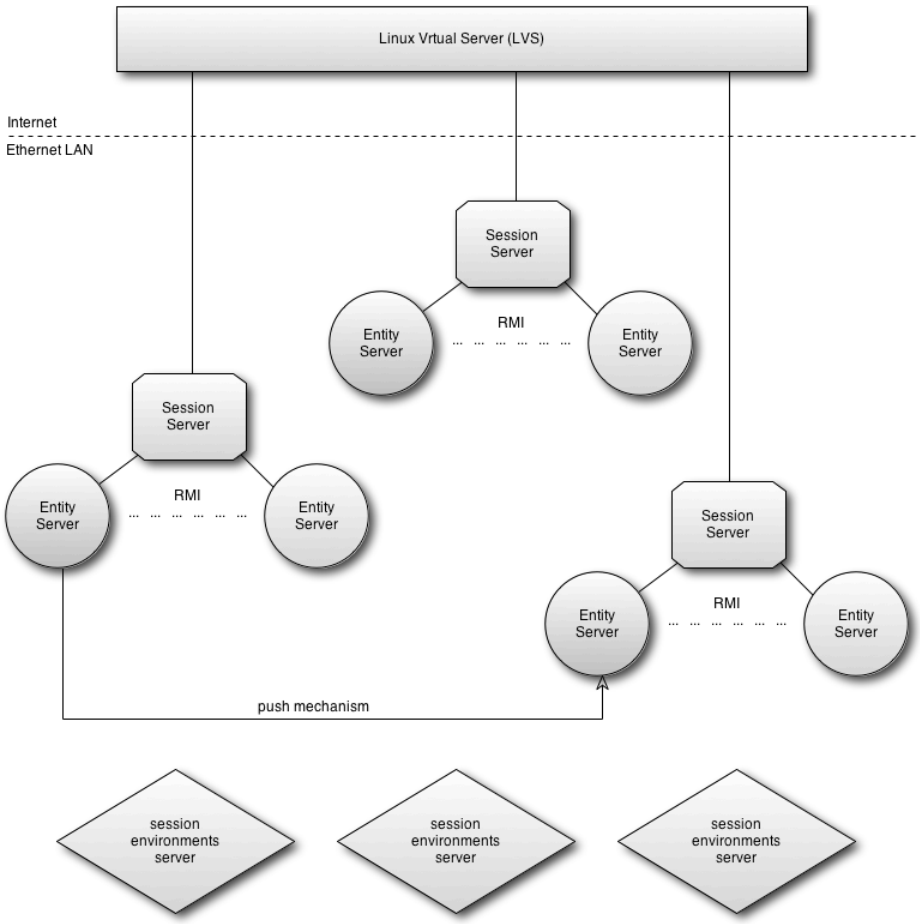
The classical solution to provide scalability and availability is to have a farm of servers with a load balancer front end such as Linux Virtual Server (LVS). LVS is a very efficient and reliable front-end for a farm of servers. It can be doubled so that if one occurrence of LVS fails, the second one is automatically activated. It supports Network Address Translation (NAT) routing. That means that LVS appears as a single server and the server farm is behind it on a local network, possibly a very fast Ethernet network. As stated in [17], a server farms can be made with low cost computers for economical reasons. The main challenging problem of this approach arises because SAW transactions are stateful: the session environments must be restored from one transaction to the other. Another problem occurs because the servers must share the data represented by the Entity objects and because we have adopted the Observers-Observable approach for the caches maintenance.

Our solution consists in externalizing all Entity objects on distinct servers. Thus, we will have Session servers and Entity Servers. All communications will be transparently done using Remote Method Invocation (RMI). That means that all Entity objects and all Observers objects must be remote objects in the sense of RMI. When a Session object requires a data, it asks to the appropriate Entity server that delivers a serialized copy of the data through RMI. If the Session object modifies its copy of the data, it must push it back to the Entity server. All communications within the server farm are handled by RMI.

This does not entirely solve the problem of reliability and availability because the Entity servers remain single points of failure. But as we duplicated Session servers, it is also possible to duplicate Entity servers. Two solutions are possible for this redundancy organization. The first solution is to have a complete set of Entity servers associated to each Session server. The second solution is to have a fixed number of occurrences of each Entity server and let the Session servers choose the Entity servers to work with using a very simple algorithm such as round-robin. When a data

modification is pushed on one Entity server, it must be forwarded to all its sibling Entity servers. To avoid a heavy load for the first server, the propagation of the data modification must be propagated in a delegated manner.

Finally, there is the so-called problem of “user affinity”, that is related to the fact that application session environments must be retrieved. The solution of keeping them in the Session servers implies that the front-end server must analyze the requests to forward the same session requests to the same Session server holding its session environment. This has a cost and it is a duplication of the work because the Session server will redo the same analysis. That’s why we decided to have several session environment servers in the same way that we have several duplication of each Entity server. The Session server chooses the session environments server according the numerical session identifier that is transmitted with the request as a cookie. This is as simple as choosing the session environments server which number is equal to the session identifier modulo the number of session environments servers. Thus, each session will have a dedicated session environments server and the LVS can distribute the requests to the Session servers using the very simple round-robin scheduling algorithm that distributes the requests to the Session servers in circular order. With this approach, the work of the front-end load balancer is reduced to the minimum. The hardest task of the front-end load balancer is to handle TLS/SSL.



We have realized a proof of concept of this approach. We did not use LVS as the front-end load balancer but a homemade Java load balancer that does the minimal tasks described above: handling TLS/SSL if necessary and relaying communications between the client and the selected Session server. We did not implement the redundancy of Entity servers but we implemented the duplication of session environments servers. The difference between Entity servers and session environment servers is that the later do not need to save their data at shutdown. We did not find a convincing way to properly recover from a general crash of Entities server. In case of a crash, each Entity server will have its own copy of the data on its local file system. But this can lead to an inconsistent situation where the data copies may not agree. This is a real problem that needs to be carefully studied.

8 Conclusions

In our approach, requests are answered using Session objects dynamically loaded, kept in memory and renewed if necessary. Application persistent data are stored in memory in Entity objects. Delivered static data (HTML, CSS, JavaScript, images, etc.) are also cached in memory if not too big, renewed if necessary. This is made possible only because the SAW server is dedicated to a given Web application. If the server were an open server such as an Apache server, this would not have been realistic. In this case, we would have to implement some kind of Entity objects hibernation technology [10].

SAW technology has proved its viability. Our online conference management has been successfully experimented in a real situation. The server successfully passed the connections peaks that occur at the conference deadlines. The application has been easy to maintain, to debug and to extend on demand. If compared to a previous LAMP implementation, it was more efficient. SAW is twice faster because of its memory caching mechanisms. The core of SAW is about 25,000 lines of code and 10,000 lines of Java, CSS, JavaScript, HTML have been necessary to implement the Web application at the top of SAW.

We have implemented well-known servers schemes, thus we have reproduced well-known vulnerabilities. A DoS (Denial of Service) attack is possible. There is no protection against DoS attacks. Such an attack is able to exhaust, for instance, the available session environments table because we cannot release the session environments before they expire and we cannot extend this table indefinitely. As pointed out by Apache team [8], this is a problem inherent to servers: if you want to serve, you have to accept all clients and, if then intent to block you, they will.

Now that the core of SAW is written and we think that future performance enhancement will be marginal, we plan to work on scalability and reliability as explained in section 7. And we are still looking for innovative concepts that would ease the work of the Web application developers.

References

1. S. A. Walberg. “Tuning LAMP systems’. IBM Developer Works, March 2007.
2. http://java-source.net/open-source/web-servers
3. B. Tindale. “Hash Tables in Java”. The Linux Gazette, n° 57, Sept. 2000.
4. K. Zorgdrager. “Tips and tricks for better Java performance”. IBM Developer Works, May 2001.
5. IEEE-RIVF Conference. http://www.rivf.org
6. H. Garcia-Molina, J. D. Ullman and J. D. Widom. Database Systems: The Complete Book. Prentice Hall, 2nd edition, ISBN-13: 978-0131873254, June 15, 2008.
7. E. Roman, R. P. Sriganesh and G. Brose. “Mastering Enterprise JavaBeans, 3rd edition”, Wiley, 2005.
8. C. Folini. “Apache attacked by a slow loris”. http://lwn.new/Articles/338407/
9. S. Souders. “High performance Web Sites”, O’Reilly Media, Sept. 2007.
10. W. Iverson. “ Hibernate: A J2EE™ Deveoper’s Guide”. Addison-Wesley, 2004.

11. P. Bellot, L. Baud. “A Dedicated Architecture for Efficient Web Server Technology”. The 3rd International Conference on Theories and Applications of Computer Science (ICTACS 2010), Can Tho (Vietnam), Sept. 2010.
12. R. Bayer, E.M. MacCreight. “Organization and Maintenance of Large Ordered Indexes”. Acta Informatica 1, pp. 173-189, 1972.
13. R. Fieldings. “Architectural Styles & the Design of Network-based Software Architecture”. PhD Dissertation. University of California, USA.
14. “JavaScript Programming Language, Standard ECMA-262 3rd Edition”, Dec. 1999.
15. P. Bellot, L. Baud. “Entities Design and Management for Java Autonomous Web Applications”, submitted to WebTech 2011, Malaysia, March 2011.
16. T. Sintes. “Speaking on the Observer pattern”. Java World, May 2005.
17. G. Roth. “Server load balancing architectures”. Java World, October 2008.