3rd International Conference on Theories and Applications of Computer Science (ICTACS 2010)
Can Tho (Vietnam), September 2010
&
Journal of Science et Technology, Special Issue on Theories and Appliction of Computer Science
vol. 48, n° 4, pp. 26-37, September 2010

A Dedicated Architecture for Efficient Web Server Technology

Patrick Bellot, Loïc Baud

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

{bellot,baud}@telecom-paristech.fr

Abstract. This article presents SAW, an efficient web server architecture that can be applied to a wide range of services implementations. SAW can very simply and efficiently implement dedicated Web sites or Web applications. However, it is not intended to act as a full and open Web server. It has been implemented by the authors in Java and also in C/C++ for comparison, testing and evaluation purposes. SAW has been used by the authors to implement a fully functional Web application related to online conference management. After an overall description of SAW, we give the technical details of its implementation and we provide our conclusions resulting from an analysis of performances, ease of deployment and ease of programming and debugging Web applications.

Keywords: Web server, Web services, Web application, Java, C/C++.

1 Introduction

Web sites, when they do not rely on a well-known Web Application Server such as GlassFish, IBM WebSphere, BEA Weblogic, JOnAS, JBoss, are built using LAMP architecture [1]. The L (Linux) may be optional but the A (Apache), the M (MySql) and the P (PHP) are widely used. LAMP architecture is a widespread, powerful and efficient tool for implementing Web sites and Web services. Another option is to use the above-mentioned Web Application Servers. They are rather heavy tools providing powerful development environment (IDE) and multiple easy-to-use functions implementing most of Web 2.0 and Web 3.0 advances. Such an approach is well suited for Enterprise Information Systems because all enterprise services, internal and external, can be integrated in a single framework which becomes a corporate culture. Finally, developers can use a Content Management System (CMS). There exist many such tools allowing to easily build elegant web sites following the many templates available on the Web. Our team has used this solution for the web site of our department: 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 blogs and other community tools, and so on. However, integrating an application in such a web site is possible but not straightforward and performances can be seen as not attractive.

What we propose is an efficient technology named SAW, for Stand Alone Web server, currently in version 7. This technology allows to build dedicated Web applications without using any external components such as a database. Only Java, or C/C++, and the file system are required to develop a Web application or a Web service. SAW is not a software revolution but the concatenation of several interesting paradigms that make it more efficient than a LAMP architecture for the implementation of a comparable service.

SAW technology cannot be used in place of a LAMP architecture or a Web Application Server (WAS). A LAMP or a WAS architecture is open: everyone who is authorized can implement new pages and develop new Web sites on a given server without practically impacting the server. And many powerful options, languages, specific modules are available on a LAMP architecture such as Apache server and its extensions. SAW does not allow that. SAW allows implementing a Web site or a Web application and, of course, one can dynamically add new pages or services but SAW is not intended to support that on a large scale. SAW technology is not intended to be able to support it, mainly because most of the server outputs and Web application data are cached in memory.

Many web servers have been written in Java or C. They are available as open-source software; see [2] for a Java list. Most of them are light web servers able to deliver pages, images and so on. Some of them implement the Servlet and JSP technologies. SAW is not very different from them. However, SAW has adapted some of the main concepts of Enterprise Java Beans [7] and delivers them under the name of Binz that has a special meaning in French slang.

SAW reproduces the triplet "Container – Session – Entity". Session binz provide the services. They answer the HTTP requests with business methods. Containers provide a runtime environment for the Session binz such that Web application programmers only focus on the business implementation. Entity beans represent the persistent data of the Web application.

We have implemented SAW in Java and also in C/C++ for comparison, evaluation and testing purposes. The Java implementation is complete and fully operational meanwhile the C/C++ implementation is only a proof of concept. Each language has its pros and its cons as explained later. Now, we decided to focus on the C/C++ implementation. It is a matter of performance and programmers feeling. We mention C/C++ instead of C++ because we did not use the C++ libraries. For instance, our strings are C strings (null terminated array of bytes) and not object of the C++ `String` class. Here again, it is a matter of performance and programmers taste.

One of our concerns was to avoid the complexity of the current powerful Web Application Servers (WAS) while maintaining some of their very interesting concepts and properties. That's why we implemented the above-mentioned triplet. With SAW, there is currently no deployment descriptor files or whatsoever. Deploying a Session binz only means copying its compiled class, either a `.class` Java file or a `.so` shared library in C/C++, in the appropriate subdirectory of the root directory of the SAW server.

Our paradigmatic example is a Web application for managing a conference. It has been implemented for the IEEE-RIVF conference [5]. It covers every aspect of a conference or congress: chairs and program committee management, conference tracks, conference dates, associated tutorials and workshops, authors administration,

papers submission, papers evaluation workflow, registrations and payments, mailing, and so on. We will refer to this application as IEEE-RIVF. As a Web application for conference management, it is rather complete. It also implements a light CMS, a Content Management System, which allows the administrators to modify online the static content of the Web site pages.

The IEEE-RIVF application in Java is 35,000 lines of Java, 3,000 lines of Javascript and 1,200 lines of CSS. These 35,000 lines of Java include the numerous functions provided by the Web application. The core of SAW is less than 6,000 lines.

2 Overall Description of SAW

SAW is a standalone server. It can be installed alone if it has access to the appropriate ports of the computer server and if the computer is freely connected to the Internet. Otherwise, it can be installed behind an Apache server that acts as a proxy for SAW, as explained below. The second solution seems to be the appropriate one because a single application must not monopolize the ports of the server.

2.1 Session Binz

When the server receives a request, it asks a Session binz object to answer the request after the parameters of the request have been decoded. If the Session binz object does not exist in the server workspace, its compiled class is dynamically loaded from the file system, from a class file (`.class`) in Java and from a shared library (`.so`) in C/C++. Before using the Session binz object, the server checks if its class has been modified in the file system. If so, the Session binz object is renewed. That is why Session binz object are intended to be stateless.

Programming a stateful Session binz object is possible but only if the programmer is sure that it will not be renewed. And one must be aware of concurrent access to Session binz internal data because the same object may be used for answering concurrent HTTP requests. Thus, the synchronization of methods is often employed in this case. Synchronization is native in Java and implemented with POSIX semaphores in C/C++. Moreover, the internal state of a Session binz object is never saved. Thus, it will be lost if the server terminates its execution. If data have to be saved at the termination of the server, the Web application programmers must use Entity binz objects.

2.2 Session Binz Containers

Containers are one of the most important concepts in Web Application Servers. They provide an execution context with many internal services dedicated to the Session binz. In SAW, containers, called Transaction objects, manage Session binz objects. The Transaction container is responsible for handling the connection with support of SSL and for decoding the HTTP request header, GET parameters and POST content. It provides all kinds of HTTP response headers. The Transaction objects provide

standard HTTP response such as Page Not Found, Internal Error, Entity Request Too Large and so on. It is responsible for zipping the response stream if the service client claims to support gzip encoding in its HTTP request fields. There is no need to provide databases access because the application data are handled by Entity binz as described below. Transaction objects also provide the support of session variables in the sense of LAMP architectures.

Thus, programming a Session binz object is rather simple. First, the Session binz object queries the Transaction object to get the HTTP request parameters. Then the Session binz object does its business work and computes an output. A HTTP response header method provided by the Transaction object is called depending on the response content computed by the Session binz object. Then, the Session binz object writes its output and terminates its execution.

If Session binz objects need to maintain session data, they can, as it is classical when implementing applications on LAMP architecture, rely on session variables in the sense of PHP session variables. Session variables are associated to a user and are stored and retrieved by the Transaction container.

As every server, SAW is able to deliver HTML static pages, image files, CSS files, Javascript files and every data file that has a mime type. All these static data are cached in memory if they are not too big. If cached, they are renewed when the original file is modified in the file system. When possible, in the case of HTML, CSS and Javascript, the contents are compacted by eliminating comments and useless space and end-of-line characters.

Specialized Session binz objects are provided for delivering such data. There exist specialized Session binz for HTML, CSS, Javascript and a more general Session binz for all other types of static data, essentially binary data such as images. Moreover, specialized Session binz are used to collect several files, several CSS files or several Javascript files, into one delivery avoiding the well-known problem of requiring several HTTP requests to get all the files. For instance, in IEEE-RIVF, a classical Session binz use only two CSS files. The first is the main CSS that is global to the web site and loaded with all the pages (it is loaded only once by the browser). The second is a page specific CSS file. The main CSS is a collection of several CSS files that is compacted, cached and delivered as one file.

The work of these Session binz is simply to check if the cache must be renewed, to renew it if necessary and to send the appropriate HTTP header for the response with the data contained in the cache. The cache contains the compacted version of the data and a zipped version of it. Thus, if the client accepts the gzip encoding, SAW is able to deliver the gzipped content. The approach is the same in Java and C++. As we will see, caching the static data in memory is made possible because the server is dedicated to one Web application.

2.3 Entity Binz

The internal state of a Web application can be handled as usual by using database storage, MySql or another, since all the libraries from Java or C/C++ are available to the programmer. However, SAW proposes to use a special class of data objects called Entity binz objects. As usual, Entity binz objects are persistent data. They allow

shared access that can be synchronized or not depending on the data intended lifecycle. They have primary keys and may participate in relationships with other Entity binz objects. They can be grouped in collections named Catalog binz.

An Entity binz object is intended to contain serializable data. Serialization is native in Java and must be implemented by the programmer in C/C++. The Entity binz is loaded from a disk file at the start of the server and saved to a disk file at the termination of the server. Because we can always be afraid of a power failure or a crash of the software, Entity binz objects are backed up at regular interval. This interval is set to 15 minutes for IEEE-RIVF. Session binz objects can of course access Entity binz objects to build their responses to HTTP requests. This access may require synchronization because Entity binz are shared inside the Web application.

We have heavily used hash table based indexing so that Entity binz objects can be indexed and retrieved using keys as we do in databases [6]. Using hash tables for indexation is known to be a memory waste but it is very efficient because the size of our intended Web applications and current computers memory size allow it. For instance, users in IEEE-RIVF are indexed by last name, by email and by their internal numeric id. The SAW server in many circumstances also uses internally this type of indexing. For instance Session binz objects are also indexed this way with the key being the HTTP request path and the value being the Session binz object loaded and created from the disk `.class` object in Java and the `.so` shared library in C/C++.

2.4 Elements and Compilation

To produce its output, the Session binz object, at the time of its creation, can build a SAW element similar to Document Object Model (DOM) objects, that means a tree representation of XML documents in memory. This SAW element may represent a HTML page or any HTML element being part of a page. It may contain non HTML elements that are business dedicated. For instance, the title of the output page may be produced by a special element programmed to produce a title from the content of a given Entity binz object. These elements can produce their output if required by the Session binz. However, this is not very efficient. That is why we introduced the compilation of elements.

SAW elements can be compiled into SAW compilations that are no more than arrays of atomic compilations. Atomic compilations are either raw array of bytes that can be instantly output or a special SAW compilation where an output function must be called to produce a computed output. For instance, if the programmer has a rather simple page where the only special element is the page title as described above, the result of the compilation will be an array of atomic compilations containing the constant array of the bytes before the title occurs, a special SAW compilation corresponding to the output of the title and, finally, the constant array of bytes after the title occurs. Using the SAW elements greatly simplifies the programming of the output page and the SAW compilation process greatly improves the efficiency since arrays of already encoded bytes are ready in the application memory to be output by the Session binz object in a single write operation.

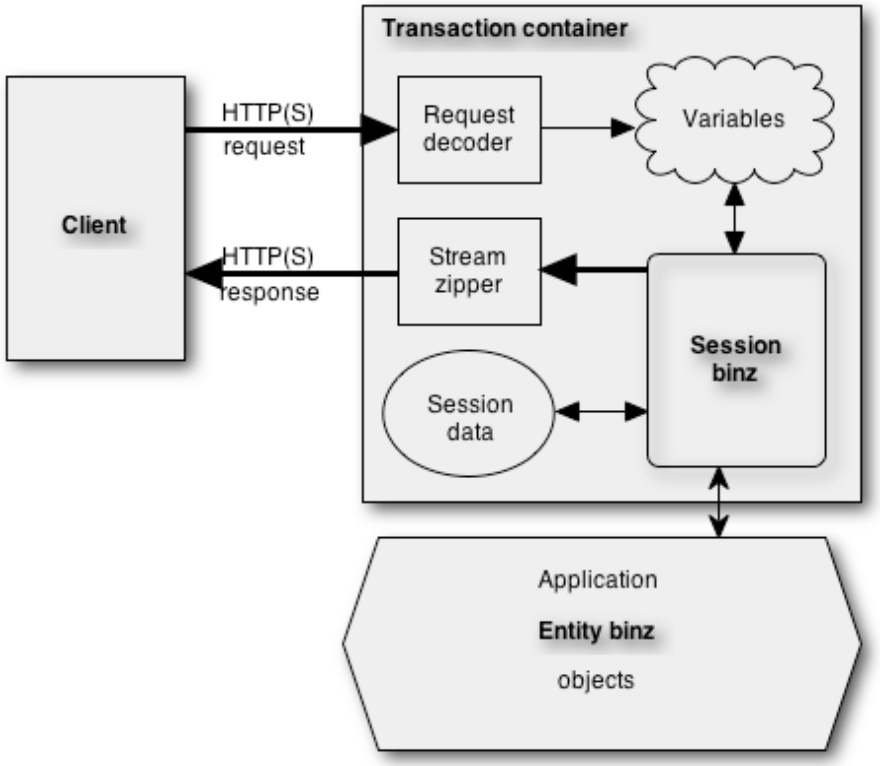
2.5 Web Applications

More technical details will be given in the next sections but we have now a good portrait of the SAW server and a Web application implemented with it. Stateless Session binz objects are dynamically loaded and used to build responses to HTTP requests. They have access to stateful Entity binz objects that contain the Web application data. Entity binz objects are global to the application.

Thus, a Web application written using SAW can be seen as a Java or C/C++ program with its internal global data being contained in the Entity binz objects. It has entry points represented by the Session binz objects that can be dynamically updated without restarting the server. This is an important point when debugging the Web application. It allows to modify some services implementations without restarting the server. Of course, it is less efficient than if everything were hard coded.

Session binz are executed in a Transaction container that provides the most important services to the Session binz object so that the Session binz program is mainly the implementation of a business method.

When stopped and later restarted, the Web application recovers its state represented by the Entity binz objects of the application. One can use all the functionalities of the language, either Java or C/C++, to program the Web application.



3 Technical presentation of SAW

A Session binz object is responsible for answering a HTTP or HTTPS request. A Transaction object, which acts as a container for it, runs the Session binz handle method. The container preserves the Session binz from all the technical requirements of a HTTP(S) transaction. Thus, programming a Session binz is mainly programming the business method only. The procedure for handling a request is as follows:

- A client sends a HTTP or HTTPS request on the appropriate port. A threaded Transaction object receives the HTTP(S) request and decodes it to recover GET and POST transaction variables, the session id cookie, and useful client properties as needed by the application. In order to improve efficiency, not all HTTP headers properties are decoded; only properties required by the Web application are decoded.
- From the session id cookie, the Transaction object recovers a session object (this is not a binz) containing the session variables similar to PHP session variables. The Transaction object is also responsible for handling the session time out. The session object contains user information if the user is logged, its IP address to avoid a session capture from another computer, and any information the Web application requires. The session object is fully customizable by the Web application programmers.
- If the client accepts gzip encoding, the Transaction object initializes a gzip streamer to be able to zip the HTTP response output. Note that the Session binz object will never be aware of that, it simply outputs data that are zipped by the Transaction container object if the client agrees with that.
- According to the request path, the Transaction object selects a Session binz. The correspondence between request paths and Session binz objects is done via a hash table where the keys are the relative request path strings and the values are the Session binz objects that are created or renewed if necessary.
- The handle method (the business method) of the Session binz is executed with the Transaction object as a parameter so that the Session binz object has access to its container Transaction object, its data and methods. Data of the Container object are mainly the session object (with its session variables) and the GET and POST variables of the request. Important methods of the Transaction object are methods to output a HTTP response header, to output characters (strings) in the appropriate encoding or to output already encoded bytes arrays.

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

- If necessary, the Session binz 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 its rights in the context of the Web application.

- Then the method looks at the transaction variable (GET and POST variables) to validate them. If they are not sound, the Session binz handle method can ask the Transaction object to issue a bad request answer (suspecting a fraud tentative for instance) and then terminates.
- Now, the method does its business, preparing its response. To do that, the Session binz object has access to the Web application Entity binz objects that contain the Web application data. As all Session binz objects have *a priori* access to all Entity binz objects, some kind of synchronization must be required depending on the application. It also depends on the language. In C/C++, the use of a non-coherent object may result in a serious error requiring signal handling meanwhile in Java, it will raise an exception that can be easily caught. When the Java exception is caught, the container will output an internal error response. With C/C++, operations are heavier and are not handled by the container but we get the same output result.
- Finally, the Session binz handle method produces its output. First, it asks the container Transaction object to issue the appropriate HTTP response header. Then the Session binz object writes its output using the bytes or characters write methods provided by the Transaction object. And it terminates its execution.

Of course, the four 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 normal output. That is why all checks have to be done before the output begins.

Entity binz objects contain the application data. They can be accessed by Session binz. Entity binz are loaded from the file system at the start of the server and they are saved to the file system at the termination of the server. That means that Entity binz must be serializable. Serialization is native in Java. It must be programmed in C/C++. In C/C++, Entity binz are saved in text files with prefixed notation. Entity binz access must be synchronized because they can be accessed by several Session binz running concurrently in the server for answering parallel HTTP requests. However, it depends on the Entity binz. For instance, in the IEEE-RIVF Web application, the description of the scientific tracks is held by an Entity binz. Once the server is in operation, the scientific tracks will not evolve because papers are submitted to tracks and any change in the scientific tracks would be too complex to handle. Thus, the tracks Entity binz will never be changed. In this case, synchronization of access is not necessary: there will be only read operation.

Collections of Entity binz may exist, for instance the collection of the users of the Web site. SAW provides indexing primitives. Index keys can be integers or strings. 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.

SAW provides elements that are similar to the DOM elements in the memory representation of XML documents. Thus, when programming a Session binz object, the programmer can write a response page this way:

```
Element myPage
  = new Html(new Head( … ),
             new Body("bodyStyle",
                      new H1("h1Style",
                             new CData("Page Title")),
                      … ) ) ;
```

There exists a base class `Element` able to implement all XML elements with styles, attributes and inner elements. Other elements, mainly XHTML elements, are derived from it.

The Web application programmer can implement customized and dynamic elements. For instance the page title may be contained in an updatable Entity binz. Therefore, it must be dynamically produced. The programmer implements a class `WindowTitle` that is a dynamic element and use it like that:

```
… new Head(new Title(new WindowTitle()), …), …
```

Elements can be compiled. The result is an array of atomic compilations. In an element, such as `myPage` above, some parts of the output are known: all parts except dynamic elements are invariable. Therefore, atomic compilations can be arrays of bytes ready to be output with a single write operation to the output stream. They correspond to the known parts of the element. Other atomic compilations correspond to the dynamic elements that are application dependant. Output of a compiled element means alternatively outputting array of bytes and calling dynamic compilations. That's the most efficient way we found for compiling pages.

SAW also proposes special elements related to the Transaction container. Transaction GET and PUT variables values can be produced as output strings using the dynamic element `TransactionVariable(String variableName)`. The Session binz can also modify the Transaction variables before output.

Finally, the `Hole(String holeName)` element allows the programmer to specify any compiled element in place of the hole before the output occurs.

4 Analysis of SAW

In our approach, requests are answered using Session binz objects kept in memory and renewed if necessary. Application data are stored in memory in Entity binz objects and finally 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 can be, this would not have been realistic. We would have to implement some kind of Entity binz hibernation technology [10].

Let us give the numbers concerning the IEEE-RIVF application. The total size of data when serialized in Java does not exceed 5 MB for a conference with more than 750 users registered on the web site, more than 140 submitted papers, 120 reviewers and at least 3 reviews per paper. The server execution memory in Java is 20 MB in

the average with some peaks at 150 MB, very soon and always recovered by the Java Garbage Collector. This is perfectly acceptable with a modern computer having several Giga bytes of physical RAM memory. In C/C++, memory and CPU requirements are lesser but at the cost of complexity in programming.

According to [4], we do not run a thread each time the server receives a request because thread initialization is an expensive operation. Instead, we manage a pool of threads which size can be customized. Each thread is a loop waiting for a HTTP or HTTPS request arriving in a synchronized queue. The thread gets the request and processes it as explained in previous sections using Transaction containers and Session binz objects. Then the thread reinitializes its internal data and returns to its waiting state. Therefore, threads are created at the start of the server and no thread is created when the server is running. This is interesting because thread creation is an expensive process collecting many resources.

The only case where a thread is created occurs in the C/C++ server when a serious error occurs: the faulting thread is cancelled, its resources are de-allocated and finally a new thread is created to replace it. However, this must not occur in normal operations. In Java, an error results in an exception that is caught inside the thread and the thread does not need to be renewed.

Native serialization is one of the benefits of Java. However, if the Entity binz are corrupted because of a bug or if we want to implement a modification of the Entity binz, then we have to write a Java program to do that. This type of program can be very complex, juggling with data types. One very good point in Java is that we were able to use the reflection tools of the `java.lang.reflect` package to build generic load and save operations for Entity binz objects. That means that if a class inherits from the class `EntityBinz`, then it automatically inherits the two methods `saveToFile()` and `loadFromFile()` provided by SAW

With the C/C++ implementation, objects are saved (serialized) as text with a prefixed notation. Each Entity binz object is a particular case with specific implementations for serialization. Modifying the C/C++ serialized Entity binz objects serialized text is easier using a few PERL or `awk` commands. Moreover, the size of the C/C++ serialized form is largely smaller than the Java serialized form.

As we have reproduced well-known servers schemes, we also have reproduced well-known servers vulnerabilities. For instance, a DoS (Denial of Service) attack is always possible if one knows the Web application implemented on the SAW server. The SAW server implements no protection against these attacks. Such an attack is able to exhaust, for instance, the available session objects table because we cannot release the session objects before they expire. But, as pointed out by Apache security team [8], this is a problem inherent to servers: if you want to serve, you have to accept clients and, if then intent to block you, they will.

No need to say that Java programming is much easier than C/C++ programming. In C/C++, any runtime error must be avoided meanwhile we can rely on exceptions catching in Java. In C/C++, the memory must be carefully managed meanwhile we can rely on the Garbage Collector in Java. Java “varargs” are much easier to handle than C/C++ “varargs”. Java provides many collections classes. However, for internal purpose, we have rewritten most of them, based on the original code available on Internet, to get better performances in our specific context. We used exactly the same

data structures, implemented the same way, in C/C++. The classical Java or C/C++ collections are still available to the Web application programmers.

When a bug occurs in the Java implementation, it is an exception that is caught and displayed in the logs with a print of the execution stack. Finding the bug in such circumstances is easy. It is an interesting property of Java. When a bug occurs in the C/C++ implementation, we have to run the debugger and try to reproduce the bug. It is a very difficult task because the bugs of this type of application may not be easy to reproduce and also because debugging a C/C++ application with many threads is a known challenge.

Another important point is portability. The "compile once – run everywhere" of Java is an important pro for Java. Meanwhile the C/C++ code has to be ported on the different operating systems, this can be a very difficult task despite of the fact that SAW code is modular. The main drawback of Java is lack of performances despite of the real progresses done by the JVM. The Java implementation is slow and uses much more memory than the C/C++ implementation.

Finally, in the Java server, we were obliged to forget "good" Java programming practices, as we teach to students, in the Transaction object that has to decode the requests. First, we used characters streams and string operations but it was really too slow, especially when decoding `multipart/form-data` POST contents that appear for instance on a paper upload page of the IEEE RIVF application. We have rewritten all the request decoding using bytes streams, ordinary Java array of bytes and C-like programming to get admissible performances. The result is that the code for request decoding is roughly the same in Java and C/C++.

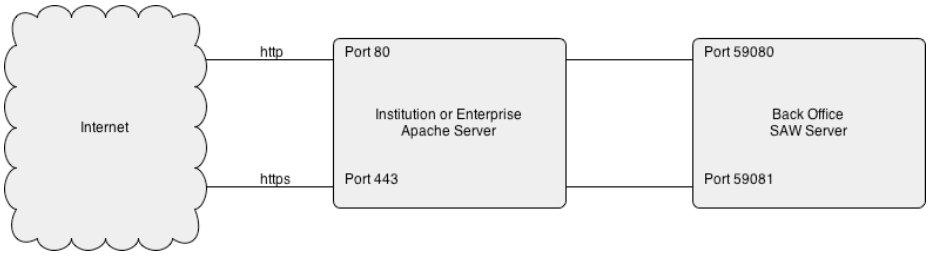
We have done some performance measurements by calling static pages 10,000 times. The client sends the HTPP requests and reads the raw HTTP answer without any parsing of the HTTP response in order to not interfere with the execution time. We got the following results showing a clear advance of the C++ server:

	User time	**System time**	**Total time**
Java server	0.07 s	0,68 s	12.118 s
Apache server	0.04 s	0.60 s	6.208 s
C++ server	0.04 s	0.48 s	1.543 s

This is only a simple test done on a local computer under MacOSX. A more significant measurement has been done with the IEEE-RIVF Web application. Previous implementation was a LAMP implementation running on a 128 cores and 16 GB SUN server with Apache. Current running implementation is a Java SAW server running on the same SUN server and proxied by the same Apache server. For all pages, the Java SAW server is twice as fast as the LAMP implementation.

If a computer is available and freely connected to the Internet, then one can install the SAW server and run it on this computer. However, in most organizations, for understandable security reasons, only a few computers and a few services have a direct access to the Internet. Then, the solution is to use the Apache server of the

organization as a proxy server. SAW can then be run on another computer using any computer ports.



Such a deployment requires convincing the system engineer to configure the Apache server to work as a proxy. Then, the SAW server web master can easily work on its back office computer. One drawback is that the SAW server logs only show the Apache proxy as a client. To obtain the true clients, one has to parse the Apache server logs.

5 Conclusions and Future Works

SAW allows the building of a Web application or a Web site using the power of languages such as Java or C/C++. The application can be thought as an ordinary application with many entry points, the Session binz objects, and data as Entity binz objects that are persistent from one run of the server to the next.

We did not talk about Web services or CRUD services because we did not implement them. It was not necessary for our conference management application. It is a planned task to provide a standardized Web services implementation. Implementing Web services means listening on a port with dedicated Transaction containers able to decode and encode SOAP or RPC service calls using appropriate XML libraries or packages. This can be easily done, although it requires a lot of work, by using the freely available XML libraries of both languages.

Now, we decided to focus on the C/C++ implementation of SAW because it is faster and requires less memory. This implementation currently runs under MacOSX. Programming is, of course, more difficult but safe programming techniques can be used. The main point is about memory management to avoid serious errors. To avoid memory leakage or memory bugs, we always use objects with constructors and destructors. We do not use graph or circular data structures. We use only tree-based structures for simplicity. And, of course, we always initialize pointers.

Another important future work will be a BSD licensing and packaging of the software and documentation of the core of SAW, as it is now, that could be used by other developers around the world.

References

1. S. A. Walberg. “Tuning LAMP systems’. IBM DeveloperWorks, March 2007. http://www.ibm.com/developerworks/linux/library/l-tune-lamp-1/
2. http://java-source.net/open-source/web-servers
3. B. Tindale. “Hash Tables in Java”. The Linux Gazette, n° 57, September 2000. http://tldp.org/LDP/LG/issue57/tindale.html
4. K. Zorgdrager. “Tips and tricks for better Java performance”. IBM DeveloperWorks, May 2001. http://www.ibm.com/developerworks/ibm/library/it-kelby_tips/
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. Available at: http://www.theserverside.com/news/1369777/Free-Book-Mastering-Enterprise-JavaBeans-Third-Edition
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.