Evaluation of Middleware’s Impact on Web-based Distributed Applications

Christos Tselikis\textsuperscript{a} Christos Douligeris\textsuperscript{a} Sarandis Mitropoulos\textsuperscript{a}

\textsuperscript{a}Department of Informatics, University of Piraeus, 80 Karaoli and Dimitriou Str., Piraeus 185 34, Greece, \{ctselik, cdoulig, sarandis\}@unipi.gr

ABSTRACT
In this paper we present a performance analysis regarding the response time of a client server e-banking application implemented by using three different enterprise middleware platforms (namely HTTPServlets, RMI and Web services with JAX-RPC). We conducted performance testing with the purpose to reveal the specific characteristics of each middleware technology and the impact they infer on a Web application’s performance. The distributed object model (Java RMI technology) is shown to provide the best performance. The simulation framework can be further extended as a testing tool able to differentiate between various demand classes and to offer an environment for the study of the distributed systems dynamic system behaviour before their large scale deployment.

Keywords: middleware; client-server interactions; random testing; usage patterns; simulation.

1. INTRODUCTION
There is a great need to develop globally distributed large scale enterprise applications. Customer-to-Business electronic trading and Business-to-Business information exchange in the supply chain are targeted to service endpoints that serve dynamic content. In this context, there is a clear necessity to offer content and services in the most efficient way in terms of responsiveness, interoperability, reusability, maintainability and economics.

A key factor to achieve these objectives are the underlying middleware platforms that are most often based on object technologies, which provide for an open interface to the network services of the Internet.

As new frameworks appear and evolve, like the various J2EE-compliant application servers, the various CORBA implementations or the Microsoft’s .Net platform, it is evident that performance testing should keep in pace with the challenges that new technologies set and that modern needs in performance evaluation should be reliably addressed.

This paper examines the response time and the scalability potential of Web applications running over three J2EE middleware platforms, namely the HTTPServlets in the Tomcat Servlet container \cite{2}, the Sun’s Remote Method Invocation (RMI) system, \cite{3}, and Sun’s Simple Object Access Protocol (SOAP)-based Java Application Program Interface (API) XML-Remote Procedure Call (JAX-RPC) \cite{4}.

A distributed simulation application is built and measurements are conducted. The aggregated results of experimental runs are presented and interpreted. The analysis reveals the effect of certain middleware mechanisms on an e-banking application that is used as a case study.

2. RELATED WORK
In \cite{8} a comprehensive methodology integrated performance modelling parameters and the design of database-centric client-server systems, to estimate the usage of hardware-related resources such as disk, network and CPU times in statements execution. The consumption of software-related resources was bound either to the transactions specifications defined in the specific system examined or to the DBMS-related functionality (query optimization and access plans).

Even though this work took a rather detailed view on the system parameters, there is a need to contemplate on the performance impact of the middleware layers for open communications between remote sites and also to take into account the asymmetry factor, \( f \) that can differentiate workload classes and hence service demands that load the distributed applications under examination.

In \cite{9} S. Chen et al. attempted to tackle the intense need for predicting the behaviour of large-scale and component-based distributed systems, before these become commercially available. They devised a performance cost model which can predict metrics, such as time and cost, for a given application and configuration of the distributed system. This model considers the performance of the underlying network, the response time of the Web servers, and the overhead of the middleware systems.

This work has been partially funded by the University of Piraeus Research Center and by a grant from the PENED Program of the Greek Research and Technology Secretariat.
as response times, that clients experience when they contest for simultaneous database accesses. The system performance was examined as a function of the number of parallel threads that are present in the client workload and also in the container’s concurrency level, thus leaving out other factors which can define distinct demand classes for resources and which might be of major importance in performance prediction.

The performance comparison done by F. Garcia-Sanchez et al. in [10] concerned the sequential vs. the multi-threaded model of execution in implementations based on the stream services offered by the ACE / TAO CORBA A/V framework. These tests proved the increased delay and especially the jitter caused by the multi-threading model when executing real-time applications.

In [11] very light SOAP transactions were used between a wireless PDA and a Unix machine running two different Web services. Direct SOAP messaging was transported very slowly (request completion time took 11 seconds). The server proxy architecture was found faster and so it was proposed as an efficient way to deploy Web services. In our view, it is arguable whether the server-side proxy architecture can improve performance in large-scale systems, since proxies introduce one more layer of delay, especially in traditional client-server interactions.

The above are indications that the role of language, design patterns and application-level architectures is to be further analyzed.

3. DISTRIBUTED MODELS IN THE WEB

Modern distributed systems can be divided in the following two communications models [1]:

A. Synchronous Communications Model

In this model the client blocks waiting for the server’s response. Also the server stays synchronized to the client requests. Since in this paper our objective is to provide the rationale for the choice of the appropriate model and technology amongst various alternatives, we further differentiate the existing platforms on whether they adopt certain generic distributed models to achieve synchronization:

- The Stream-oriented communications model. Processes perform input — output over a bi-directional and synchronized byte-stream. Examples are information exchange over raw sockets with no support of a specific protocol or, as for Web applications, HTTP sessions between browsers and server-side Servlet objects that execute in a Servlet container. Our implementation of this model was built with HTTPServlets running in the Apache Tomcat Servlet container. The servlet-api.jar was included in the Jakarta Apache Tomcat 1.4 Servlet / JSP container, [2].

- The Distributed Object Model. We considered Remote Method Invocation (RMI) as an adequate example of the distributed object model capable to achieve service discovery with distributed RMI registries. Information is passed as arguments and return values between objects that reside on distant machines. Our implementation of the object distributed model used the java.net RMI packages. Common alternative choices for this model are the various available implementations of the OMG CORBA standard. The five RMI over TCP packages of the core Java 2 APIs, [3], were used.

- The Services Oriented Architecture. The “publish, discover and invoke” Web services model is often used in business procurements and transactions trading. We chose to test, as a representative platform, Sun’s XML-based JAX-RPC platform. The jaxrpcapi.jar, as provided in the Sun’s Java Web Services Developer Pack (JWSDP 1.4, SOAP 1.1) toolkit, was used. We integrated this toolkit with Tomcat, [4].

B. Asynchronous Communications Model

Distributed processing with batching capabilities is another possibility. Applications of this model include messaging services offered by Message Oriented Middleware platforms (MOM), or applications such as file sharing and asynchronous communications over peer-to-peer networks.

Our intention here is to investigate how synchronization is achieved by the different middleware platforms available and to highlight their effect on the synchronized e-banking application’s performance. Furthermore, via the use of simulation experiments our goal is to build appropriate usage profiles and empirically test individual middleware features which are inherent in each technological case.

6. SIMULATION PARAMETER SETTING

We categorized the sources of Web traffic based on the number $N$ of the connected users in enterprise workgroups. Table 1 shows the Small Office Home Office (SOHO) usage profile of no more than ten connected members. The next usage profile is that of a SE (Small Enterprise) case with up to 50 connected users in a workgroup. The SME (Small - Medium Enterprise) usage profile scales to traffic generated by 100 employees and for the ME (Medium Enterprises) we assume that
they consist of a C-class network of up to 250 connected machines.

Table 1 Parameter values and ranges of four Internet Workloads

<table>
<thead>
<tr>
<th>Usage Pattern</th>
<th>SOHO</th>
<th>SE</th>
<th>SME1</th>
<th>SME2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency Level N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Arrival Rate $\lambda$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2.25</td>
<td>2.50</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>Message Length L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75kB - 750Kb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetry factor f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1-0.2</td>
<td>0.3-0.5</td>
<td>0.5-0.8</td>
<td>0.9-1</td>
<td></td>
</tr>
</tbody>
</table>

In addition to considering these indicative and widely adopted workgroup sizes, we scaled the concurrent client load by varying the arrival rate $\lambda$ of the transactions requests that reach the server and also the size of messages (submitted queries and returned results) which stress the server’s capability.

The message length $L$ is the user information included as payload data in a packet and affects the server’s throughput or service rate $\mu$.

We controlled the payload size $L$ as the third simulation parameter in between 75kBytes and 150kBytes. Medium size messages utilize efficiently the available capacity of the CPU and the Ethernet link. Heavier messages saturate it to a point where it is impossible to serve any additional transaction. At this load requests are rejected by the server without being committed (roll-back).

Finally, as shown in Table 1, we also incorporated in our parameters the asymmetry factor $f$, an index of the asymmetry between the data volume transferred at the upload the download links during a session of client-server interactions consisting of a number of remote invocations. The asymmetry depends on the content (audio, images, video) the requested page contains. A usual single user uploads simple interrogating traffic (for example, URL requests sent from a browser to a public server) which is remarkably less than the amount of data that he usually downloads. On the contrary, enterprise applications are symmetric in the data volumes exchanged and in the bandwidth consumed in the two communication paths (examples are B2B transactions between active enterprise servers even mass gaming and VoIP applications). Therefore, we reckon $f$ as an application-dependent factor. The use of $f$ attempts to capture in simulation the applications diversity that characterizes today’s Internet.

7. THE SIMULATION APPLICATION

There is a need to capture the dynamic behaviour and performance of live distributed systems. One way to facilitate the behavioural modelling of distributed systems is via Unified Model Language (UML) state and interaction diagrams, [5].

In our study we are interested in experimenting with live distributed systems, before these are deployed in large scale, [6]. In this regard, we extended with transactions capabilities the JavaSim Object Oriented and Discrete Event simulation package, [12]. In effect, we built a distributed simulation application with the client part being a transactions generator (of HTTP, RMI and SOAP-based invocations) running on one machine and the server part, implemented in the three corresponding versions, running on the actual server node we want to test. The two nodes are connected with a Fast Ethernet link in a LAN.

9. EXPERIMENTAL RESULTS

Because of limited space we offer the experimental results of only two of the four test cases, namely these of the SOHO and Medium Enterprise workloads. The values of the application’s response times depicted in following figures are the mean value over three simulation runs.

Test Case 1: The SOHO Usage Pattern

In the case of the SOHO usage pattern with 10 clients, Fig. 2 demonstrates that the Servlets implementation of the test application with persistent connections retains the lowest response times (msecs) in the whole range that factor $f$ takes.

Especially for small message sizes (75kB-250kB), where we expect delays to depend on connection set-up times, Servlets are more responsive than RMI, while for larger messages (250kB-750kB) the two technologies perform similarly. Notably in Servlets we observe a latency reduction at around $f = 0.3$ and then a monotonic increase of the response times. RMI times show an earlier and smaller dip, namely in the $f$ range between [0.06, 0.1] and then start again to increase their latency. We also observe that for $f$ values in range [0.9, 1] RMI times show a small drop, so that for $f=1$ the response
times of Servlets and RMI implementations coincide for the first time in the graph.

Web services with JAX-RPC served relatively slowly the SOHO pattern and gave response times that are ten times that of the Servlets times. Furthermore, in Web services for messages larger than 150kB the CPU was congested due to the XML data overhead and the processing spent for XML parsing. In effect, further requests were rejected and response times exceeded the acceptable values inside LAN boundaries, even for this light load.

It is worth mentioning that, as Fig. 2 shows, the JAX-RPC response times vary in a non-linear fashion with respect to the asymmetry factor $f$. The instability point for the JAX-RPC implementation happened at $f$ around 0.2.

9. AGGREGATED RESULTS INTERPRETATION

Response Time Comparison

To analyze the experimental results and correctly interpret them we averaged the output values on sub-ranges of factor $f$ values. The obtained results are shown in Fig. 4 below.
As Fig. 4 illustrates, the Servlets implementation of the test e-banking application, with two client–server interactions, exhibits the best performance for the SOHO and SE demands. Servlets outperform the RMI response times in these two loads. This outcome applies to the case of low loads as for example is the browsing – intensive usage exercised by simple end-users in the Internet today. As we gradually increase the loading conditions we mimic the usage that small and medium enterprises generate. For this case, although Servlets still perform efficiently enough, compared to RMI, the best performance now belongs to RMI.

Besides, the comparative advantage of the latter increases when the server is loaded with the Small and Medium Enterprise (SME) usage type. In this case RMI performs with roughly one half of the Servlets response times.

Therefore, our results reveal the fact that internet and intranet usage patterns are best served by different component-based middleware technologies that implement different distributed models.

According to Fig. 4, it is clearly obvious that plain JAX-RPC with no binary or other encoding-compressing scheme lacks the required performance efficiency, especially in the synchronous communications model that we studied.

We attribute this to the following reasons:

- The stateless connection management that the JAX-RPC run-time system supports.
- The pool of threads model that causes many contention delays and switches for the management of the shared request queue.
- The SOAP protocol overhead due to SOAP headers and XML encoding (this caused both transmission and processing delays).
- The binding mechanism needed to map Java types to XML elements and vice-versa in order to execute remote server code written with Java classes (parsing delay).
- The overhead induced by the synchronous RPC protocol.

In [7] compression in conjunction with a multicast streaming transport protocol is used with SOAP Web services so that delays are tolerable in real-time distributed simulation applications which generate large volumes of object data with state attributes of specified interest.

Throughput Comparison

The critical bottlenecks, because of concurrency issues, were the server’s processor usage CPUu and the Java Virtual Machine’s sockets buffer memory usage Mu.

We observed that for a given level of simultaneous requests N, there was a threshold value of the arrival rate λs, above which the processor performance degraded with successive transactions failures. According to Java run-time system errors, the requests were rejected due to running out of operating system memory resources, i.e., the overflow of the 1MByte socket buffering space.

With 2.75 incoming transactions per second, the server could successfully process messages with length of up to 750kBytes. This size corresponds to an HTML page with one or two medium-size embedded image files.

Fig. 5 depicts the throughput performance of the three technologies under the four workloads. The results show that the maximum throughput achieved by each technology happens at different values of f in the four workload cases.

One can observe in Fig. 5 that the maximum throughput was achieved for Servlets in the SOHO case and also that as the load increases the Servlets throughput drops monotonically. The f value for maximum rate (3.56Mbps) is 0.3. On the other hand, RMI gave a maximum rate of 2Mbps in the Small Enterprise (50 Clients) case, for f equal to 0.2. It is worth mentioning that in large loads, SME1 and SME2, the maximum RMI transferring capacity was achieved at symmetrical and large message exchanges between client and server (f=0.9 and f=1 respectively).

The throughput achieved with RMI in the SME load is higher than that of Servlets and much higher than that of JAX-RPC.

Therefore, the aggregated results prove that the service look-up mechanism, through the RMI interface BankingIF, has a negative effect on the throughput performance of the middleware. This is even more evident in the case of the JAX-RPC service discovery mechanism with the use of WDSL and the interface BankingIFPort.

However, the stateful character of certain RMI features (such as the connections management, the message handling and the threading mechanisms) improved significantly the application’s response time and the system’s transferring capacity, a fact especially
evident in high loads. In intense loading conditions RMI tends to achieve the best performance.

The JAX-RPC version sustained a constant throughput across all loads, approximately one tenth that of Servlets and approximately five times slower than that of RMI. This behaviour could be beneficial if the throughput value was optimized.

11. CONCLUSIONS

In this paper we presented simulation results when systematically stressing a multi-threaded server that we built over three synchronous component-based middleware platforms. Certain middleware features affect the performance in each technology case.

The distributed object model (Java RMI) proved to be suitable and stable for heavy symmetrical corporate loads (like those generated from Small Medium Enterprises), while Servlets with persistent connections responded more efficiently in lighter loads that match the browsing-intensive SOHO usage or Internet traffic stemming from Small Enterprises activities. Admittedly, the Web service version of the bank server performed poorly, especially when increasing the load.

The purpose of this work was not to judge the one technology against the other, but to offer a performance-testing tool that could validly and efficiently (in terms of time and cost) provide a first-cut performance estimation, helpful for suitable technology choice. In this regard, we intend to enhance the simulation framework we used with performance modelling capabilities in order to handle, as object attributes, information related to performance and distributed systems resources utilization.

13. REFERENCES


[12] Little, M. C., JavaSim Home Page http://javasim.ncl.ac.uk.