Evaluation of Current Java Technologies for Telecom Backend Platform Design

Bruno Van Den Bossche

Supervisor(s): Bart Dhoedt, Filip De Turck

Abstract— **The Java programming language and more specifically the J2EE platform has evolved towards the most important software framework for designing and implementing business logic on a telecom backend platform. However, the real time aspects of J2EE based telecom applications are often questioned. In 2004, a new Java-based application server technology, JAIN SLEE, has been standardized which seems a promising candidate for the development of (soft) real time telecom applications. A functional comparison of J2EE and JAIN SLEE and the concepts they are built on is presented.**

Java, and JAIN SLEE in particular, has been subject to a performance evaluation study evaluating the real time aspects of both the application server and the Virtual Machine. The evaluation procedure and obtained evaluation results are detailed. Moreover, the influence of the Java Virtual Machine tuning parameters for the Garbage Collector has been investigated and will be reported upon.

Keywords— **Telecom Applications, Java Garbage Collection, Performance Evaluation**

I. Introduction

THE Java language is highly structured, strongly typed and object oriented. It is not compiled to machine specific instructions but a *byte code*, which is executed on a virtual machine, available for all sorts of platforms. This makes Java highly portable although at a certain performance cost. Another feature with important implications is the use of a garbage collector. Java includes automatic memory management (garbage collection) as a part of the Java runtime. This means that many very common errors made by developers related to memory management cannot occur. However, since garbage collection is part of the Java runtime, it is not under control of the application developer.

As telecom applications typically have very specific requirements regarding high throughput (e.g. the number of Voice over IP (VoIP) calls a softswitch can process per second) and low latency (e.g. the setup of a call should be very fast) Java might not seem to be the best possible solution for designing telecom backend applications or Service Enabling Platforms (SEP). To meet the requirements of the Telecom industry the JAIN [1] (Java APIs for Integrated Networks) initiative is providing us with an extensive set of standardized APIs for network related applications. JAIN builds on the portability of Java and standardizes the signaling layer of the communication networks in the Java language. The importance of the appropriate tuning of the Java Virtual Machine will be addressed in this paper.

As handling of VoIP calls is an important example of current telecom applications we evaluated the available Java technologies by studying the performance of Session Initiation Protocol (SIP) Proxy applications. The Session Initiation Protocol is typically used for the setup and teardown of VoIP calls.

B. Van Den Bossche is with the Department of Information Technology, Ghent University (UGent), Gent, Belgium. E-mail: Bruno.VanDenBossche@UGent.be .

II. JAIN SLEE

The JAIN SLEE specification, defined in Java Specification Request 22 and 240, provides us with an application server tailored for telecom. There is a certain similarity between J2EE and JAIN SLEE but there are some important differences as well. We will first give an overview of the architecture of the SLEE (Service Logic Execution Environment) followed by a comparison with the J2EE architecture.

A. Architecture

The basic architecture of a JAIN SLEE application server is similar to the J2EE architecture. Applications in a JAIN SLEE are hosted in a container and consist of one or more Service Building Blocks (SBB). Such an SBB can be best compared with an Enterprise Java Bean (EJB) in a J2EE server. SBBs are usually organized in a hierarchy with one root SBB.

Internally almost all communication in the SLEE happens through events. The SLEE uses the publish-subscribe model for event distribution. The idea behind the event based communication between SBBs is that every SBB performs its own task and then hands the result off to the next SBB in line. One could compare this to an assembly line in a factory.

The communication with the outside world happens through the so called Resource Adaptors (RA). This RA then accepts incoming messages, parses them and turns them into events understandable by the SLEE. Through the use of RAs the JAIN SLEE can be protocol agnostic. This means that any protocol can be supported by adding an appropriate RA.

B. Relation to J2EE

As already mentioned JAIN SLEE and J2EE have a lot in common. Both are container based designs and the SBBs of JAIN SLEE are the equivalent components of the EJBs in J2EE. Both technologies also exploit the component based architecture by allowing application designers to transparently use home made and third party components in order to build applications. The use of off-the-shelf components is made very easy.

Nevertheless, there are some significant differences as well. JAIN SLEE is strongly optimized for asynchronous event-driven logic. J2EE on the other hand, was primarily designed for synchronous calls. It does have some support for asynchronous logic through the use of the Java Message Service (JMS) and Message Driven Beans. The use of JMS would require a lot of extra work when developing and when deploying if the same level of abstraction would be required.

III. Session Initiation Protocol

The Session Initiation Protocol (SIP) is defined in RFC3261 and describes an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. SIP is used in most VoIP systems.

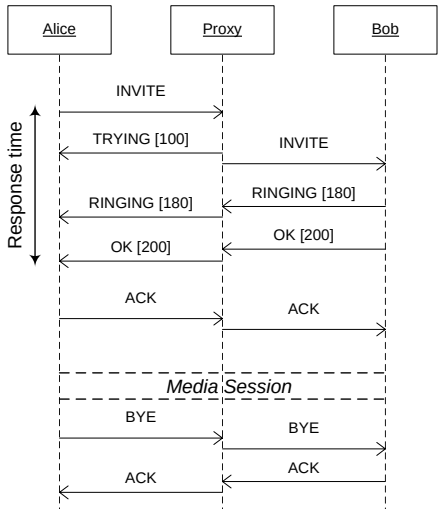

Fig. 1. SIPStone proxy 200 test

The scenario used for the testing and benchmarks is the Proxy 200 test shown in figure 1. This is one of the benchmarks defined in the SIPstone [2] benchmark specification. We are interested in the time it takes to set up a call. This is the time it takes from the initial INVITE of Alice till she receives an OK from Bob, indicating the call is set up. After this Alice will send and acknowledgement to Bob saying she received the OK and the media session (e.g. voice or video conference) can start. As soon as this media session has ended a BYE is sent and acknowledged to end the call. Note that the BYE may be sent by both parties. When benchmarking no media session was initiated and the call was terminated immediately after it was established.

IV. Performance Tuning: JVM tuning

As important as the actual hardware platform the application is run on, are the used Java Virtual Machine and the used tuning parameters. Of great importance is the Garbage Collection. Garbage collection may lead to the whole virtual machine being paused. With requirements of response times being within 25ms a pause of the virtual machine of multiple milliseconds may be dramatic. Appropriate tuning of the virtual machine can improve the perceived latency and improve the garbagecollection results. The most important choice is the actual garbagecollector. In this case the combination of a parallel collector and a concurrent collector proves to be the most succesful. A detailed description of available garbage collection options can be found in [3].

V. Results

The graph presented in Figure 2 shows the obtained results of the OpenCloud JAIN SLEE implementation Rhino using the default (faded) and optimized (strong) JVM options. One of the first things to notice is the almost perfectly linear correlation between the system load and number of calls per second, which shows the scalability of the technology. The cpu-load for the optimized JVM-options is also higher than for the default options.

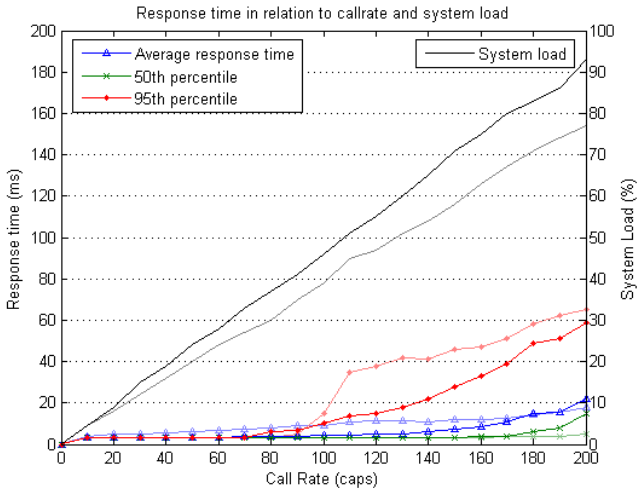

Fig. 2. Performance results of JAIN SLEE on a dual processor architecture with default (faded) and optimized (strong) garbage collection options.

If we analyze at the average response times and the 50th percentile theres only a small increase as the number of calls increases. At 80 caps the average response time is approximately 4ms and the 50th percentile lies below 3ms. The 50th percentile remains at 3ms till 150 caps, at that point the average response time is approximately 8ms. The 95th percentile however starts increasing at around 80 caps, till that point it also was below 3ms. Notice the system load has yet to reach 50%. We assume that the explanation of these results is related to the garbage collection. The system load presented in the graph is the average obtained during the test run. In reality the system load does vary and small peaks do occur. The messages arriving simultaneously with these peaks are most likely those to suffer the small delays surfacing when looking at the 95th percentile.

VI. Conclusions

The results show that JAIN SLEE is a scalable technology and that the requirements for the telecom applications, in this case a SIP Proxy, are met. The average response time for 95% of the calls is below 50ms and the average response time for 50% of the calls is below 25ms up to over 160 calls per second. Without the tuning of the Java Virtual Machine these requirements are met until 100 calls per second.

References

[1] John de Keijzer, Douglas Tait, and Rob Goedman, "JAIN: A New Approach to Services in Communication Networks," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 94–99, January 2000.

[2] Henning Schulzrinne, Sankaran Narayanan, Jonathan Lennox, and Michael Doyle, "SIPstone - Benchmarking SIP Server Performance," April 2002, http://www.sipstone.org/.

[3] Sun Microsystems, "Tuning Garbage Collection with the 5.0 Java™ Virtual Machine," 2003, http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html.