An Evaluation of the Performance Impact of Generic APIs on Two Group Communication Systems

Leandro Sales, Henrique Teófilo, Jonathan D'Orleans, Nabor C. Mendonça, Rafael Barbosa, Fernando Trinta

Mestrado em Informática Aplicada, Universidade de Fortaleza
Av. Washington Soares, 1321, CEP 60811-905 Fortaleza – CE

leandro@enovar.com.br, henriquetft@gmail.com, jonathan.dorleans@gmail.com, nabor@unifor.br, trinta@unifor.br, bgrafael@gmail.com

Abstract. *This paper presents an evaluation of the performance impact of two generic group communication APIs, namely Hedera and jGCS, over two well-known group communication systems, namely JGroups and Spread. The evaluation compared the performance of different configurations of the two group communication systems in a four-node cluster, under different message sizes, both in standalone mode and when used as plug-ins for the two generic APIs. The results show that there are significant differences in the overhead imposed by each generic API with respect to the performance of both JGroups and Spread, when used in standalone mode, and that those differences are strongly related to variations in message size and also to the way the generic APIs and their plug-in mechanisms are implemented. Based on those results, this paper discusses some of the circumstances upon which it would be worth implementing group communication using the systems and APIs investigated.*

1. Introduction

Group communication plays an important role in the design of fault-tolerant distributed systems [Coulouris *et al.* 2005]. Classical group communication applications include replication; support for distributed and clustered processing; distributed transactions; resource allocation; load balancing; system management and monitoring; and highly available services [Chockler *et al.* 2001].

However, implementing a fully-fledged *group communication system* (GCS) from scratch can be a daunting (and therefore error-prone) task. To overcome this situation, distributed systems researchers and tool developers have created a number of reusable GCSs, providing a variety of group communication primitives and protocols that can be used as powerful building blocks for the development of reliable distributed applications. Some of the most popular GCSs currently in use are JGroups [Ban 1998], Spread [Amir *et al.* 2000], Appia [Miranda 2001], and NeEM [Pereira *et al.* 2003].

While developers can certainly benefit from such a diversity of GCSs, for example by choosing a solution that best suits their application's needs and constraints, they also face a new challenge: which GCS to use? Choosing an appropriate GCS for a distributed application is an important design decision that can be made difficult by the fact that those systems tend to vary widely in terms of the features they provide, including communication abstractions, quality-of-service guarantees and delivery semantics [Chockler *et al.* 2001]. In addition, once a developer commits to a particular

GCS implementation, the distributed application source code becomes tightly coupled to that system's API. This level of coupling is undesirable in a distributed application for at least two main reasons: (i) it requires changes to the application code every time the chosen GCS's API evolves; and, most importantly, (ii) it may discourage developers from experimenting with new GCSs in future versions of their application.

An interesting solution to decouple a given distributed application from a specific GCS implementation is to rely on generic APIs, such as those offered by Hedera [Hedera 2008], jGCS [Carvalho *et al.* 2006] and Shoal [Shoal 2008]. Each of those systems provides a common programming interface and a plug-in mechanism that allows that common interface to be easily (re)implemented using the services of different existing GCSs. The use of a generic group communication API is also attractive from a performance perspective, as it frees developers to switch to the fastest GCS plug-in available, without the need to change their application code.

Even though there is an extensive body of work on the performance of individual GCSs in the literature (e.g., [Abdellatif *et al.* 2004; Amir *et al.* 2004; Baldoni *et al.* 2000]), some fundamental questions regarding the use generic APIs are yet to be fully explored. For instance, how those generic APIs impact the performance of the different GCSs they encapsulate? How is that impact influenced by factors such as message size, group size, and transport protocol? Apart from the clear software engineering benefits, would there be any performance gain in replacing the services of a given GCS for those provided by a generic API? Clearly, this kind of knowledge would be of great value to distributed application developers, who could decide more confidently about when it is more appropriate (or mandatory) to use a particular GCS, and when it would be worth migrating to a generic API.

In our previous work [Sales *et al.* 2008] we have started to shed some light on some of the above questions by presenting the results of an initial study where we have evaluated the performance impact of two generic group communication APIs, namely Hedera [Hedera 2008] and jGCS [Carvalho *et al.* 2006], over JGroups [Ban 1998]. In this paper, we have extended that study by (i) including a new GCS implementation in the evaluation, namely Spread [Amir *et al.* 2000]; (ii) correlating the results of the two GCSs evaluated in order to investigate whether it would be worth migrating to a faster solution using a generic API; and (iii) providing a more thorough discussion of the merits and limitations of our work. In essence, our new results show that there are significant differences in the overhead imposed by each generic API with respect to the performance of both JGroups and Spread, when used in standalone mode, and that those differences are strongly related to variations in message size and to the way the generic APIs and their plug-in mechanisms are implemented. We also have found that migrating from a moderately slow group communication solution (such as JGroups) to high-performance one (such as Spread) using a lightweight generic API (such as jGCS) may be a viable alternative with clear performance and software engineering gains to the target distributed application.

The rest of the paper is organized as follows. Section 2 gives a quick overview of the two GCSs and the two generic APIs investigated. Sections 3 and 4 describe our evaluation method and results, respectively. Section 5 discusses the implications and limitations of our study. Section 6 covers related work. Finally, section 7 concludes the paper and outlines our future work.

2. Systems and APIs Evaluated

2.1. JGroups

JGroups is an open source reliable group communication toolkit written entirely in Java [Ban 1998]. It offers a high level communication abstraction, called Channel, which works like a group communication socket through which applications can send and receive messages to/from a process group. With this abstraction, the different protocol implementations than can be used by JGroups become totally transparent to application developers, who can reuse their application code across different communication scenarios and network configurations, just by reconfiguring JGroups's underlying protocol stack.

One of the most powerful features of JGroups is that it allows developers to write their own protocol stack, by combining different protocols for message transport (for instance, TCP or UDP over IP Multicast), fragmentation, reliability, failure detection, membership control, etc. This flexibility has made JGroups particularly popular amongst middleware developers, with the system having been used to implement the clustering solution for a number of open source JEE applications servers, including JOnAS [JOnAS 2008] and JBoss [JBoss 2008].

In our study, we used JGroups version 2.6.2, released on February 26, 2008. JGroups is available at http://www.jgroups.org.

2.2. Spread

Spread is another open source group communication toolkit that provides a high performance messaging service that is resilient to faults across local and wide area networks [Amir *et al.* 2000]. It offers a range of reliability, ordering and stability guarantees for message delivery. Spread is aimed at improved scalability and performance, and implements a rich fault model that supports process crashes recoveries and network partitions and merges under the extended and standard virtual synchrony semantics [Amir *et al.* 2000].

Spread adopts a client-server architecture, where the server (called a Spread *daemon*) is responsible for handling all communication amongst group members. Spread can be configured to use a single *daemon* in the network or to use one *daemon* in every computer node running group communication applications. That server-centered communication architecture avoids having heavyweight group communication protocols, like membership management, message ordering and flow control, running on all group nodes.

Although its server component is written in C, Spread provides native APIs for a number of different programming languages, including C++, C#, Java, Perl, Python and Ruby. It also supports cross-platform operation between Unix/Linux and Windows.

In our study, we used Spread version 4.0, released on December 4, 2006 (the latest version available at the time of the study). Spread is available at http://www.spread.org.

2.3. Hedera

Hedera is an open-source Java framework designed to provide a uniform API to different group communication toolkits [Hedera 2008]. Although it can be used in

different application scenarios and network configurations, Hedera was originally targeted at reliable group communication within clustered environments.

Hedera has been used by the Sequoia project [Sequoia 2008] as its group communication layer. Sequoia is a transparent middleware solution offering clustering, load balancing and failover services for replicated databases, originally developed as a continuation of C-JDBC [C-JDBC 2008].

In our study, we used the Hedera plug-ins for JGroups and Spread distributed with version 1.6.3, released on February 8, 2008. Hedera is available at http://hedera.continuent.org.

2.4. jGCS

jGCS is another generic group communication toolkit written in Java, that offers a common API to several existing GCSs [Carvalho *et al.* 2006]. jGCS can be used by distributed applications with different group communication needs, from simple IP Multicast to virtual synchrony or atomic broadcast. The architecture of jGCS relies on the *inversion of control* design pattern [Fowler 2005] to decouple service implementation from service use, thus allowing the same API to be used to access the services of different GCSs. The actual service implementation that is used by jGCS is defined at configuration time.

jGCS has been originally developed within the context of the GORDA project [GORDA 2008], which, like Sequoia, also aims at providing solutions for transparent database replication, but with a particular focus on large-scale systems.

In our study, we used the jGCS plug-ins for jGroups and Spread distributed with version 0.6.1, released on October 29, 2007. jGCS is available at http://jgcs.sourceforge.net/

3. Evaluation Method

Our evaluation was carried out in a clustered environment composed of four computer nodes connected through a dedicated 10/100 Mbps Fast Ethernet switch. Each node had the following configuration: Linux Ubuntu 7.10 (2.6.22.14-generic kernel) operating system; Intel Pentium IV (3.00 GHz) processor; 2 GB (DDR2) RAM; and SUN's Java Virtual Machine version 1.6.0_03 (executed in server-side mode).

This environment was setup to emulate typical clustering scenarios used by JEE application servers, in which process groups are expected to be relatively stable and moderate in size [Lodi *et al.* 2007].

To run our experiments, we used an extended version of the Java application developed by the JGroups team for their performance tests [JGroups 2007]. Our extension consisted of modifying the application source code so that it could also work with Hedera, jGCS and Spread (through its provided Java API).

In each experiment, our test application was executed at each cluster node, with each node being configured to concurrently multicast 1000 messages of equal size to all nodes in the cluster, including its own local node, which means that a total of 4000 messages were delivered at each node during each experiment. This particular group communication pattern was chosen to emulate a highly-available JEE clustered scenario

where each individual application server broadcasts its state changes to all the other servers in the cluster [Lodi *et al.* 2007].

We ran different sets of experiments for JGroups and Spread, both in standalone mode and when used as plug-ins for Hedera and jGCS, respectively, using messages of 1, 10, 100, 1000 and 10000 bytes in size. These numbers were selected so that we could observe the behavior of the two systems under a broad range of message sizes. A similar range of message sizes was also used in a previous evaluation of JGroups in the context of clustered J2EE application servers [Abdellatif *et al.* 2004].

We evaluated two JGroups configurations: one with the transport protocol setup to UDP over IP Multicast, and the other with the transport protocol setup to TCP. In both configurations we used SEQUENCER as the total order protocol, with message fragmentation and bundle disabled. The other configuration parameters were defined according to JGroups' test configuration parameters [JGroups 2007].

In the case of Spread, we evaluated a single configuration since it implements a fixed protocol stack. The configuration used Spread's own total order protocol and had a fully decentralized architecture, with one Spread daemon running in each cluster node.

Finally, we used *message delivery rate* as our performance metric [Jain 1991]. In our study, this metric was computed by calculating the average number of messages delivered per second at each node, at each experiment. To achieve a confidence level of 95% in our results, each experiment was repeated at least 30 times, with extreme outliers being removed using the *boxplots* method [Triola 1997].

4. Results

We first show the impact observed for the two generic APIs over the performance of each one of the two selected GCSs. We then correlate those results with the way messages are transmitted at the transport layer using each generic API, as a way to explain their performance differences.

Because Spread is known to outperforms JGroups by a significant margin [Baldoni 2002], we also compared the performance of the two JGroups configurations, in standalone mode, against the performance of Spread when used as a plug-in for the best performing generic API. This analysis was meant to evaluate whether migrating from a JGroups-based solution to a generic API implemented on top of Spread would bring any notable performance again (i.e., whether Spread's improved performance would compensate for any potential overhead imposed by the generic API).

4.1. API Impact over JGroups

Figures 1 and 2 show the average message delivery rates observed for the two JGroups configurations, respectively, both in standalone mode and as Hedera and jGCS plug-ins, across all five message sizes. In each figure confidence intervals for each mean value are shown as upper and lower limits drawn around the top of their respective bar.

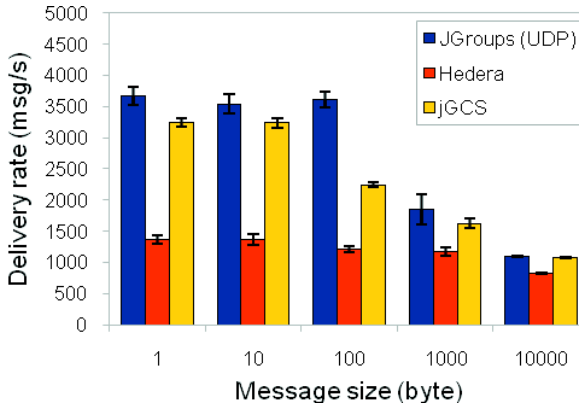

Figure 1. Message delivery rates for JGroups (UDP configuration).

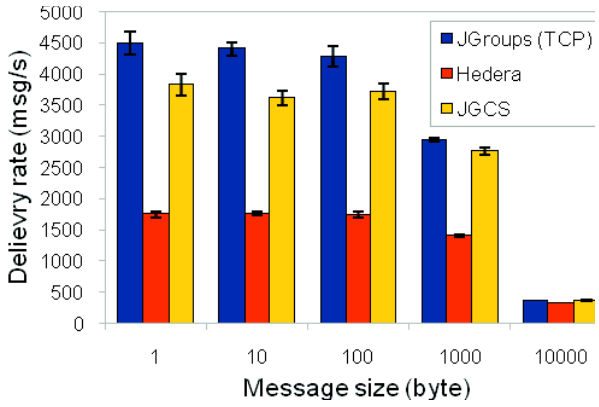

Figure 2. Message delivery rates for JGroups (TCP configuration).

As we can see from Figures 1 and 2, for messages up to 100 bytes in size the impact of jGCS over JGroups, in either configuration, is relatively small, with jGCS delivering about 10-20% less messages on average than JGroups in standalone mode. One notable exception was observed for the UDP configuration with 100 byte messages, where jGCS' delivery rate is about 40% lower than that of JGroups in standalone mode.

Hedera, on the other hand, imposes a huge impact over the performance of the two JGroups configurations for the same range of message sizes, with that generic API delivering about 60-70% less messages than JGroups in standalone mode. Figures 1 and 2 also show that the two generic APIs appear to be generating a constant overhead per message, independently of message size.

For greater messages (in the order of thousands bytes) we observe a steep performance drop for the three systems, with their delivery rate differences falling to less than 30%. Note that the drop is even steeper for the UDP configuration, with the delivery rates of the three systems rapidly falling below the 2000 messages per second mark. We attribute these results to the fact that, for larger messages, the increasing traffic overhead generated by JGroups (due to the increasing message fragmentation happening

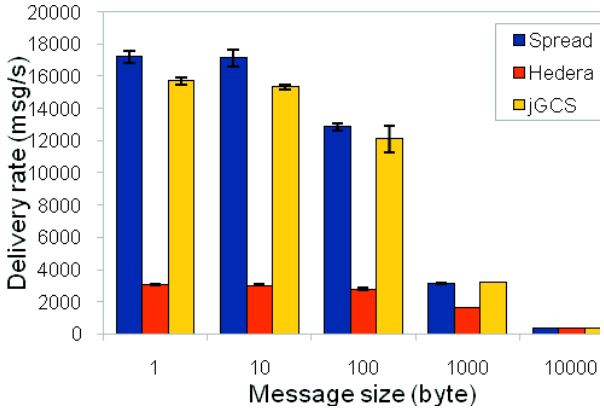

Figure 3. Message delivery rates for Spread.

at the transport and network layers) starts to dominate the processing overhead imposed by the two generic APIs at the application layer.

Note from Figures 1 and 2 that JGroups performs better using TCP than using UDP. This finding has also been observed in previous performance studies of JGroups (see [Abdellatif *et al.* 2004; JGroups 2007]). The explanation for that slight performance gain offered by TCP over UDP lies in the way JGroups implements flow control. With UDP, JGroups implements flow control at the middleware (i.e. application) layer, while with TCP it takes advantage of TCP's native (i.e., faster) flow control implementation at the transport layer.

4.2. API Impact over Spread

Figure 3 shows the average message delivery rates observed for Spread, in standalone mode, and its Hedera and jGCS plug-ins, across the five message sizes. Confidence intervals are shown in the same way as in the previous figures.

As with the two JGroups configurations, we can observe that jGCS's impact over Spread is non substantial for messages up to 100 bytes in size, although this time the performance differences between that generic API and its underlying GCS implementation are even smaller, with the jGCS Spread plug-in delivering about 5-10% less messages than Spread in standalone mode. As before, Hedera offered the worst results by a large margin, with its Spread plug-in delivering about 80-85% less messages than Spread in standalone mode for 1-10 byte messages. On the other hand, Spread was found to be less scalable than JGroups for larger messages, with its performance dropping rapidly (either in standalone mode or as a Hedera or jGCS plug-in) as message sizes reach 1000 bytes. For 10000 byte messages, the performance of Spread drops even further in all three modes, with the impact caused by the two generic APIs being completely dominated by the network overhead. These results corroborate the finding observed with JGroups that the two generic APIs are generating a constant overhead per message. This also means that the impact imposed by the generic APIs may not be dependent on a particular plug-in implementation, and so is likely to apply to other GCSs.

Table 1. Comparison between message sizes for the Generic APIs' plug-ins at the application and transport layers (in bytes).

Application layer	Transport layer			
	Hedera		jGCS	
	JGroups plug-in	Spread plug-in	JGroups plug-in	Spread plug-in
1	878	1147	1	1
10	887	1156	10	10
100	977	1246	100	100
1000	1877	2146	1000	1000
10000	10877	11146	10000	10000

4.3. Transport Layer Overhead

To investigate the possible cause for Hedera's substantial overhead compared to that of jGCS, as described in the previous subsections, we formulated the following hypothesis: For some reason, Hedera could be generating a considerably higher network overhead than jGCS, even for smaller messages, independently of the specific GCS being used as its underlying plug-in. To investigate this hypothesis, we compared the size of every message sent at the application layer with the size of the message that was actually being sent to the other nodes using the Hedera and jGCS plug-ins at the transport layer. The results are shown in Table 1.

By analyzing the numbers in Table 1, it is clear that the two Hedera plug-ins always send messages much larger in size than those sent at the application layer, while no significant transport layer overhead was observed for jGCS with either JGroups or Spread. In fact, Hedera always sends messages with least 878 bytes, independently of the message size at the application layer.

A Further inspection of the Hedera source code revealed that its extra bytes are used as control data, and include, amongst other information, the IDs of all group members to which the message is being transmitted. Given that jGCS also implements a similar set of generic group communication primitives without incurring in any message size overhead, we believe that an important first step towards improving Hedera's performance would be to drastically reduce its communication overhead at the transport layer. In the particular case of Hedera's JGroups plug-in, adding the group members IDs to every message is completely unnecessary, since membership information is already maintained by JGroups as part of the attributes of its *Channel* abstraction [Ban 1998].

4.4. JGroups vs. jGCS/Spread

Figure 4 plots the average message delivery rates observed for the two JGroups configurations, in standalone mode, versus the average message delivery rates observed for the jGCS Spread plug-in, across the five message sizes investigated. From that figure, we can see that the Spread plug-in implemented by jGCS can deliver from 3 to 4 times more messages than the two JGroups configurations in standalone mode, for messages up to 100 bytes in size, and up to 1.7 more messages than the UDP

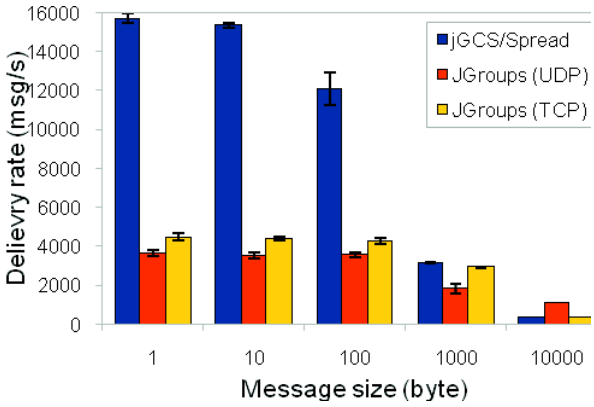

Figure 4. Message delivery rates for JGroups and the jGCS Spread plug-in.

configuration of JGroups for messages of 1000 bytes in size. Note from Figures 1-3 that the same cannot be said about the Spread plug-in implemented by Hedera, whose delivery rates are lower than those observed for both JGroups configurations in either mode. As the message size approaches 10000 bytes, the two systems suffer from severe performance losses, as their implementation differences start to be dominated by the increasing network overhead.

These results show that migrating from a standalone group communication solution to a generic API can be an attractive alternative to improve application performance, as long as: (i) messages are small (in the order of tens of bytes); (ii) the target generic API imposes a low performance overhead (as it is the case with jGCS); and (ii) that API offers another GCS plug-in that is significant faster than the original GCS used by the application (enough to compensate for the performance overhead of the generic API, as it is the case of the jGCS Spread plug-in when compared with the performance of JGroups in standalone mode).

5. Discussion

There are number of factors that should be considered by a distributed application developer when contemplating the possibility of using a generic group communication API. Perhaps the most important one is to consider whether having a loosely coupled communication architecture is a major design concern (for example, if the developer foresees the possibility of switching to or experimenting with new GCSs in the future). This decision is important because generic APIs, such as Hedera or jGCS, as we have shown along the paper, always impose extra levels of indirection between the application and the underlying GCS implementation, thus inevitably resulting in some performance loss.

Another factor that the developer must take into account is message size. In particular, based on the results reported in the previous section, and assuming similar execution and network environments, for messages up to 100 bytes, we can argue that it would be worth replacing an existing GCS (e.g., JGroups) by a lightweight generic API (such as jGCS), from a straight performance perspective, as this would make it easier for the developer to improve the application's performance by switching to a faster GCS implementation (such as Spread). However, for message sizes in the order of thousands

of bytes, the choice between using a particular GCS (such as JGroups or Spread) directly, in stand-alone mode, or indirectly, encapsulated behind a generic API (such as Hedera or jGCS), should be made based entirely on the software needs and constraints of the application at hand. The reason is that, within that message size range, network transmission costs tend to predominate over the performance overhead imposed by the generic API over their underlying GCS implementation, thus reducing the possibility of improving application performance by simply switching to a different GCS plug-in.

Despites the merits of our results, we are aware that our study is still limited in a number of ways. Above all, we have only investigated the behavior of four group communication solutions under a single cluster environment. In this regard, we have deliberately selected two of the most well-known GCSs available, namely JGroups and Spread, and two existing generic APIs which provide plug-ins for those two systems, namely Hedera and jGCS. In addition, we have setup an experimental environment which emulates typical clustered JEE application servers, with similar characteristics to those of earlier studies reported in the literature. All these decisions increase our confidence that our experimental test bed is likely to be representative of the state-of-the-practice in many real-world group communication applications.

Another important limitation of our work is that we have only compared the existing GCS implementations from a performance standpoint. In practice, replacing one GCS for another requires a careful analysis of many other factors, such as fault-tolerance, memory footprint, and the syntactic and semantic differences between their respective APIs. We plan to address those limitations in our future work.

6. Related Work

Being two of the most popular GCSs currently available, JGroups and Spread have already been used as targets for a number of performance evaluation studies (e.g., [Abdellatif *et al.* 2004; Amir *et al.* 2004; Baldoni *et al.* 2000; JGroups 2007]). In [Abdellatif *et al.* 2004] and [JGroups 2007], the authors compare the performance of different JGroups configurations, under a variety of network conditions. A similar study has been described for Spread by Amir *et al.* [2004]. In contrast to those works, the primary aim of our study is not to analyze the performance of different GCSs configurations, in standalone mode, but rather to evaluate the impact that different generic APIs would impose on existing GCSs. In this way, we aim at providing relevant experimental information to help distributed application developers decide on when to use a concrete GCS implementation, such as JGroups or Spread, and when to hide such a system from the application code under a generic API, such as Hedera or jGCS.

Another work comparing the performance of several GCSs written in Java (including an earlier version of JGroups, called JavaGroups), under different usage scenarios and different network conditions, is described by Baldoni *et al.* [2000]. However, that work is not concerned with evaluating the impact of any generic API on any particular GCS.

7. Conclusions and Future Work

This paper presented an evaluation of the performance impact of two generic APIs, namely Hedera and jGCS over two well-known GCSs, namely JGroups and Spread. In essence, our results show that there are significant differences in the overhead imposed by each generic API over the performance of both JGroups and Spread, when used in

standalone mode, with Hedera offering by far the worst results. The study also shows that the performance differences observed across all systems are strongly related to variations in message sizes (for messages sizes in the order of a few thousands of bytes those differences tend to be completely dominated by the systems' increasing network overhead) and also to their implementation characteristics (Hedera's dismal performance is largely due to its significant message size overhead imposed at the transport layer).

We are currently pursuing two main research lines. The first one consists of replicating the same set experiments described here for other GCS implementations (e.g., Appia [Miranda *et al.* 2001] and NeEM [Pereira *et al.* 2003]) and generic APIs (e.g., Shoal [Shoal 2008]), under a wider variety of cluster scenarios. The idea is to investigate whether the results described in this paper can be generalized to those other systems and scenarios. The second research line aims at conducting similar experiments, but within the context of the clustered architecture of an existing JEE applications server, such as JBoss [JBoss 2008] or JOnAS [JOnAS 2008]. The idea, in this case, is to evaluate whether the same performance differences observed in our current test environment will also occur in this new context, where the size of the messages exchanged between group members will vary according to the size of the session states maintained by each replicated JEE server.

8. References

Abdellatif, T., Cecchet, E. and Lachaize, R. (2004), "Evaluation of a Group Communication Middleware for Clustered J2EE Application Servers", In Proc. of the 6th International Symposium on Distributed Objects and Applications (DOA'04), Lecture Notes in Computer Science Vol. 3291, Springer.

Amir, Y., Danilov, C., Miskin-Amir, M., Schultz, J. and Stanton, J. (2004), "The Spread Toolkit: Architecture and Performance", Tech. Rep. CNDS-2004-1, Johns Hopkins University.

Amir, Y., Danilov, C. and Stanton, J. (2000), "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication", In Proc. of the IEEE International Conference on Dependable Systems and Networks (ICDSN'00), IEEE CS Press.

Baldoni, R., Cimmino, S., Marchetti, C. and Termini, A. (2002), "Performance Analysis of Java Group Toolkits: a Case Study", In Proc. of the International Workshop on Scientific Engineering for Distributed Java Applications (FIDJI'02), Lecture Notes in Computer Science Vol. 2604, Springer, pp. 81-90.

Ban, B. (1998), "Design and Implementation of a Reliable Ggroup Communication Toolkit for Java", Cornell University. Available at http://www.jgroups.org/javagroupsnew/docs/ papers/Coots.ps.gz.

Carvalho, N., Pereira, J. and Rodrigues, L. (2006), "Towards a Generic Group Communication Service", In Proc. of the 8th International Symposium on Distributed Objects and Applications (DOA'06), Lecture Notes in Computer Science Vol. 4276, Springer.

Chockler, G. V., Keidar, I. and Vitenberg, R. (2001), "Group Communication Specifications: a Comprehensive Study", ACM Computing Surveys, 33(4):427–469.

C-JDBC (2008), "C-JDBC: Clustered JDBC". Available at http://c-jdbc.objectweb.org.

Coulouris, G., Dollimore, J. and Kindberg, T. (2005), Distributed Systems: Concepts and Design, Addison Wesley, Fourth Edition.

Fowler, M. (2005), "Inversion of Control". Available at http://martinfowler.com/bliki/InversionOfControl.html.

GORDA (2008), "GORDA – Open Replication of Databases". Available at http://gorda.di.uminho.pt/.

Hedera (2008), "Hedera Project". Available at http://hedera.continuent.org.

Jain, R. K. (1991), The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley.

JBoss (2008), "JBoss Application Server". Available at http://labs.jboss.com/jbossas/.

JGroups (2007), "JGroups Performance". Available at http://www.jgroups.org/javagroupsnew/perfnew/Report.html.

JGroups (2008), "JGroups – A Toolkit for Reliable Multicast Communication". Available at http://www.jgroups.org.

JOnAS (2008), "Java Open Application Server". Available at http://jonas.objectweb.org.

Lodi, G., Panzieri, F., Rossi, D. and Turrini, E. (2007), "SLA-Driven Clustering of QoS-Aware Application Servers". IEEE Transactions on Software Engineering, 33(3):186-197.

Miranda, H., Pinto, A. and Rodrigues, L. (2001), "Appia: a Flexible Protocol Kernel Supporting Multiple Coordinated Channels", In Proc. of the 21st International Conference on Distributed Computing Systems (ICDCS'01), IEEE CS Press, pp. 707-710.

Pereira, J., Rodrigues, L., Monteiro, M. J., Oliveira, R. and Kermarrec, A.-M. (2003), "NeEM: Network-friendly Epidemic Multicast", In Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS'03), IEEE CS Press, pp. 15-24.

Sales, L., Mendonça, N. C., Barbosa, R., D'Orleans, J., Trinta, F. and Teófilo, H. (2008), "Um Estudo do Impacto de Desempenho de Dois Sistemas Genéricos de Comunicação em Grupo sobre o JGroups", In Anais do IX Workshop de Sistemas Computacionais de Alto Desempenho (WSCAD-SSC 2008), Campo Grande – MS, Brasil.

Sequoia (2008), "Sequoia Project". Available at http://sequoia.continuent.org.

Shoal (2008), "Shoal – A Dynamic Java Clustering Framework". Available at https://shoal.dev.java.net/.

Spread (2008), "The Spread Toolkit". Available at http://www.spread.org.

Triola, M. F. (1997), Elementary Statistics, Addison Wesley, Seventh Edition.