# A practical example of convergence of P2P and grid computing: an evaluation of JXTA's communication performance on grid networking infrastructures

Gabriel Antoniu, Mathieu Jan, David Noblet

# A practical example of convergence of P2P and grid computing: an evaluation of JXTA's communication performance on grid networking infrastructures

Gabriel Antoniu
INRIA - Rennes Research Centre
Campus de Beaulieu, 35042 Rennes cedex, France

Mathieu Jan*
CEA, LIST, LaSTRE
CEA Saclay, 91191 Gif-sur-Yvette, France

David Noblet
California Institute of Technology
1200 E California Blvd, MC 256-80, Pasadena, CA 91125, USA

## Abstract

*As the size of today's grid computing platforms increases, the need for self-organization and dynamic reconfiguration becomes more and more important. In this context, the convergence of grid computing and peer-to-peer (P2P) computing seems natural. However, grid infrastructures are generally available as a hierarchical federation of SAN-based clusters interconnected by high-bandwidth WANs. In contrast, P2P systems usually run on the Internet, on top of random, generally flat network topologies. This difference may lead to the legitimate question of how adequate are the P2P communication mechanisms on hierarchical grid infrastructures. Answering this question is important, since it is essential to efficiently exploit the particular features of grid networking topologies in order to meet the constraints of scientific applications. This paper evaluates the communication performance of the JXTA P2P platform over high-performance SANs and WANs, for both J2SE and C bindings. We discuss these results, then we propose and evaluate several techniques able to improve the JXTA's performance on such grid networking infrastructures.*

## 1 Introduction

Nowadays, scientific applications require more and more resources, such as processors, storage devices, network links, etc. Grid computing provides an answer to this growing demand by aggregating processing and storage resources made available by various institutions. As their sizes grow, grids express an increasing need for flexible distributed mechanisms allowing them to be efficiently managed. Such properties are exhibited by peer-to-peer (P2P) systems, which have proven their ability to efficiently handle millions of interconnected resources in a decentralized way [7]. Moreover, these systems support a high degree of resource volatility. The idea of using P2P approaches for grid resource management has therefore emerged quite naturally [10, 18].

To our knowledge, the very few practical attempts of convergence between P2P and grid computing have taken two different paths. One approach consists in implementing P2P services on top of software building blocks based on current grid technology (e.g., by using grid services as a communication layer [19]). Conversely, P2P libraries can be used on physical grid infrastructures as an underlying layer for higher-level grid services [1]. This is a way to leverage scalable P2P mechanisms for resource discovery, resource replication and fault tolerance. In this paper, we focus on this second approach.

Grid applications often have important performance constraints. In most cases, grids are built as cluster federations. System-Area Networks (SANs), such as Giga Ethernet or Myrinet (which typically provide Gb/s bandwidth and a few microseconds latency), are used for connecting nodes inside a given high-performance cluster; whereas Wide-Area Networks (WANs), provide a typical bandwidth of 1-10 Gb/s, but a higher latency (typically of the order of 10-20 ms), are used to connect the clusters. Consequently, sending a small message between two nodes within the same SAN may be 1,000 times less expensive then doing the same operation across a WAN. Such a discrepancy cannot be neglected, since the efficient use of the network characteristics is a crucial issue in the context of performance-constrained scientific applications.

In contrast, P2P applications generally do not have important performance constraints, as they usually target the edges of the Internet (with low-bandwidth and high-latency links, such as Digital Subscriber Line (DSL) connections). In such a context, the latency between arbitrary pairs of nodes does not exhibit a high variation. Therefore, most published papers on P2P systems generally model the communication cost as a distance based on the number of logical hops between the communicating entities, without taking into account the underlying physical topology. When

* This author's work has mainly been done at INRIA - Rennes Research Centre.

running P2P protocols on grid infrastructures, this factor clearly has to be considered in order to efficiently use the capacities of the available networks to provide the performance required by the applications. Therefore, using P2P libraries on grid infrastructures as building blocks for grid services is a challenging problem, since this is clearly an unusual deployment scenario for P2P systems. Consequently, it is important and legitimate to ask: are P2P communication mechanisms adequate for a usage in such a context? Is it possible to adapt P2P communication systems in order to benefit from the high potential offered by these high-performance networks, in order to meet the needs of scientific grid applications?

Most of the few attempts to realize the P2P-grid convergence have been based on the JXTA [24] open-source project (see the related work below). In its 2.0 version, JXTA consists of a specification of six language- and platform-independent, XML-based protocols that provide basic services common to most P2P applications, such as peer group organization, resource discovery, and inter-peer communication. This paper discusses the appropriateness of using the JXTA P2P platform for high-performance computing applications, running on grid infrastructures. This case is challenging for JXTA, as it evaluates to what extent its communication layers are able to leverage high-performance (i.e. Gigabit/s) networks. For the purpose of our evaluation, we use the well-known bidirectional bandwidth benchmark, widely used to evaluate networking protocols. This paper focuses on the evaluation of JXTA-J2SE and JXTA-C[1] over high-performance SANs and WANs. It also proposes and discusses several solutions to improve the raw performance observed.

The remainder of the paper is organized as follows. Section 2 introduces the related work: we discuss some JXTA-based attempts for using P2P mechanisms to build grid services and we mention some past performance evaluations of JXTA. Section 3 provides an overview of the communication layers of both JXTA-J2SE and JXTA-C. Section 4 describes the experimental setup we used for SAN and WAN benchmarks. Sections 5 and 6 present the benchmark results of JXTA over these three types of networks. In Section 7, we discuss the measurements from a global perspective through a comparison with other middleware typically used for building grid applications. We also provide some hints on how to efficiently use JXTA's communication layers. Finally, Section 8 concludes the paper and discusses some possible future directions.

## 2 Related Work

Several projects have focused on the use of JXTA as a substrate for grid services. The Cog Kit JXTA Project [25] and the JXTA-Grid [26] project are two examples. However, none of these projects are being actively developed and none has released any prototypes. The Service-oriented Peer-to-Peer Architecture [1] (SP2A) project aims at using P2P routing algorithms for publishing and discovering grid services. SP2A is based on two specifications: the Open Grid Service Infrastructure (OGSI) and JXTA. None of the projects above has published performance evaluations so far. Finally, JUXMEM [2] proposes to use JXTA in order to build a grid data-sharing service. All projects mentioned above share the idea of using JXTA as a low-level interaction substrate over a grid infrastructure. Such an approach brings forth the importance of JXTA's communications performance, which may be critical for scientific grid applications.

In this paper, we focus on the communication layers of the main two bindings of JXTA: JXTA-J2SE and JXTA-C. The performance of the widely-used *pipe* communication layer of JXTA-J2SE has been the subject of many studies [11, 12, 15, 16, 13, 17] and has served as reference for comparisons with other P2P systems [5, 14, 20]. However, these studies are primarily based on JXTA 1.0 [11, 12] or even older [15, 16]. The most recent evaluations of both main JXTA bindings are [3] and [4]. The main focus of [3] is a performance evaluation of JXTA-J2SE in a LAN environment using Fast Ethernet. It also provides the first evaluation of JXTA-C and gives hints on how to use both bindings of JXTA in order to get good performance on this kind of networks. In [4], the same benchmark code is used but in a very different context: performance evaluations are performed on grid infrastructures consisting of SAN-based clusters interconnected by high-bandwidth WANs. To the best of our knowledge, [4] is the first attempt to discuss the appropriateness of using the JXTA P2P platform for high-performance computing on grid infrastructures.

This paper presents an extended and updated version of the results preliminarily presented in [4], which are synthesized and discussed from a larger perspective. We include some new performance figures for SAN and WAN benchmarks, using a more recent version of JXTA-C (2.1.1), which exhibits significant improvements compared to previous performance measurements published in [4]. In the context of the convergence of P2P and grid computing, these updated results can help the user to draw a clear picture of the potential of JXTA-C on high-performance networks available on grid infrastructures.

## 3 Overview of JXTA Communications Layers

JXTA provides three basic transport mechanisms for inter-peer communication, each providing a different level of abstraction.

At the lowest level, information is exchanged between peers in discrete units known as *JXTA messages*. JXTA specifies two possible wire representations for a JXTA message: binary, where a transport protocol such as TCP is available; and XML, in case the underlying transport protocol is not capable of transmitting binary data. In either case, a JXTA message is comprised of a series of named and typed *message elements* [27], any number of which may be

[1] The only two bindings compliant to JXTA's specifications version 2.0

required by the transport protocol or added by the application as the message payload. These message elements can be of any type, including, for example, an XML document.

**The bottom layer: the endpoint service** The endpoint service is JXTA's point-to-point communication layer. It provides an abstraction for the available underlying transport protocols (called *endpoints*) which can be used to exchange data between one peer and another. Currently, supported transport protocols common to both implementations of JXTA are TCP and HTTP. However, regardless of the underlying transport protocol, all communications at the endpoint level, are asynchronous, unidirectional and unreliable.

In general, the endpoint service should not be utilized directly by applications, but rather indirectly through the use of one of the upper communication layers, such as the pipe service or JXTA sockets. Therefore, the aim of benchmarking the endpoint service is primarily to gather performance data on the endpoint service for the purpose of explaining the performance measured for these upper layers.

**The core communication layer: the pipe service** The pipe service supplements the endpoint service by incorporating the abstraction of virtual communication channels (or *pipes*). The aim of the pipe service is to provide the illusion of a virtual endpoint that is independent of any single peer location and network topology, as stipulated by JXTA specifications. Like peers, each pipe also has an identifier unique to the JXTA virtual network; this is known as the Pipe ID and is used by the pipe service to bind peers to *pipe-ends*. Before a message is transferred between peers, each end of the pipe is resolved to an endpoint address, through the use of JXTA's pipe binding protocol, and the endpoint service is used to handle the actual details of transferring messages between peers (the resolution is only done once for each pipe and is subsequently checked from time to time).

Like endpoint communications, pipe communications are also asynchronous and unreliable. However, the pipe service offers two modes of communication: point-to-point (*unicast pipes*) and propagate mode (*propagate pipes*). In propagate pipes, a single peer can simultaneously send data to many other peers. In this study we focus on unicast pipes because of their general-purpose nature and because they serve as a basis for the implementation of the higher-level JXTA sockets.

**Enabling sockets over P2P: JXTA Sockets** The JXTA sockets introduce yet another layer of abstraction on top of the pipes and provide an interface similar to that of the more familiar BSD socket API. Compared to the JXTA pipes, JXTA sockets add reliability and bi-directionality to JXTA communications. Additionally, JXTA sockets transparently handle the packaging and un-packaging of application-specific data into and out of JXTA messages, presenting a data-stream type of interface to each of the communicating peers. However, it should be noted that this layer is not part of the core specifications of JXTA and has not been implemented in JXTA-C so far. It was introduced in JXTA-J2SE 2.0, with reliability support added in 2.1.

The data-stream interface also introduces another interesting parameter which can be used to tune JXTA sockets. Indeed, it is possible to configure the size of the output buffer of a JXTA socket. This value has an impact on the way the socket packages the data it receives for transmission into a series of separate JXTA messages that can be sent using the pipe service. This is significant because the JXTA socket creates a new JXTA message every time the buffer becomes full or the buffer is explicitly flushed by the application. In all versions of JXTA, the default buffer size is 16 KB.

## 4 Description of the Experimental Setup

For all reported measurements we use a *bidirectional bandwidth* benchmark (between two peers), based on five subsequent time measurements of an exchange of 100 consecutive message-acknowledgment pairs sampled at the application level. We chose this test as it is a well-established metric for benchmarking networking protocols, and because of its ability to yield information about important performance characteristics such as bandwidth and latency. Moreover, such information is necessary in order to evaluate JXTA's adequacy for performance-constrained grid applications. Both bindings of JXTA were configured to use TCP as the underlying transport protocol.

When benchmarks are performed using JXTA-J2SE, the Sun Microsystems Java Virtual Machine (JVM) 1.4.2 is used and executed with `-server -Xms256M -Xmx256M` options. Note that when the J2SE binding of JXTA is benchmarked, an additional warm-up phase based on 1000 consecutive message-acknowledgment pairs is performed, to make sure that the Just-In-Time (JIT) compiler is not disturbing the measurements. The JXTA-C benchmarks are compiled using `gcc` 4.0 for the SAN and WAN benchmarks. In both cases, the `O2` level of optimization is used. Finally, note that all source codes required to perform the benchmarking of each communication layer of JXTA-J2SE and JXTA-C have been made available via the web sites of the JDF [8] and JXTA-C [28] projects.

**SAN benchmarks.** The networks used for the SAN benchmarks are Giga Ethernet and Myrinet-2000 (GM driver, version 2.0.11). When the network layer is Myrinet, nodes consist of machines using 2.4 GHz Intel Pentium IV processors, outfitted with 1 GB of RAM each, and running a 2.4 version Linux kernel. For Giga Ethernet, nodes consist of machines using dual 2.2 GHz AMD Opteron processors, also outfitted with 1 GB of RAM each, and running a 2.6 version Linux kernel. Benchmarks were executed using versions 2.2.1 and 2.3.2 of the J2SE binding of JXTA. In this paper, we mainly report figures for the 2.3.2 version. For the C binding, version 2.1.1 was used. In all cases, direct communication between peers has been enabled, since

| Version of JXTA | JXTA-J2SE 2.3.2 | |
|---|---|---|
| Network | Myrinet | Giga Ethernet |
| Endpoint service | 624 $\mu$s | 294 $\mu$s |
| Unicast pipe | 1.7 ms | 711 $\mu$s |
| JXTA socket | 2.4 ms | 977 $\mu$s |

**Table 1. Latency results for JXTA-J2SE 2.3.2 on SAN.**

this feature is usually available among the nodes of a SAN-based cluster within currently deployed grids.

**WAN benchmarks.** The platform used for the WAN benchmarks is the Grid'5000 French national grid platform [6]. Tests were performed between two of the Grid'5000 clusters located in Rennes and Toulouse. On each side, nodes consist of machines using dual 2.2 GHz AMD Opteron processors, outfitted with 1 GB of RAM each, and running a 2.6 version Linux kernel. The two sites are interconnected through a 1 Gb/s link, with an average measured latency of 11.2 ms. We used the same versions of JXTA as for the SAN experiments. In this case too, we configured JXTA peers to enable direct exchanges.

# 5 Performance Evaluation of JXTA over System-Area Networks

This section analyzes the performance of JXTA's communications layers on SANs. Note that for Myrinet, the *Ethernet emulation* mode of GM 2.0.11 is used and configured with jumbo frames. This mode allows Myrinet to carry any packet traffic and protocols that can be transported by Ethernet, including TCP/IP. Although this capability is bought at the cost of losing the main advantage of a Myrinet network (the OS-bypass mode), it allows the same socket-based benchmarks to be run unmodified. On this configuration, the bandwidth and latency of plain sockets is around 155 MB/s and 60 $\mu$s respectively, whereas on Giga Ethernet it is around 115 MB/s for the bandwidth and 45 $\mu$s for the latency. These values are used as a reference performance bound.

## 5.1 JXTA-J2SE: Analysis

**JXTA-J2SE endpoint service.** Figure 1 shows the bandwidth of the JXTA 2.3.2 endpoint layer, which does not saturate the potential bandwidth of the networks. This is explained by a new implementation of the endpoint layer that shipped with JXTA 2.3. The profiling of JXTA has pointed out that this drop of performance since JXTA 2.2.1 is due to the mechanism used for limiting the size of messages sent by the endpoint layer. We therefore removed this limit to enable our measurements. Table 1 shows that the latency of the JXTA 2.3.2 endpoint service over Giga Ethernet reaches a value under 300 $\mu$s. Moreover, it goes down even further to 268 $\mu$s and 229 $\mu$s when using the SUN 1.5

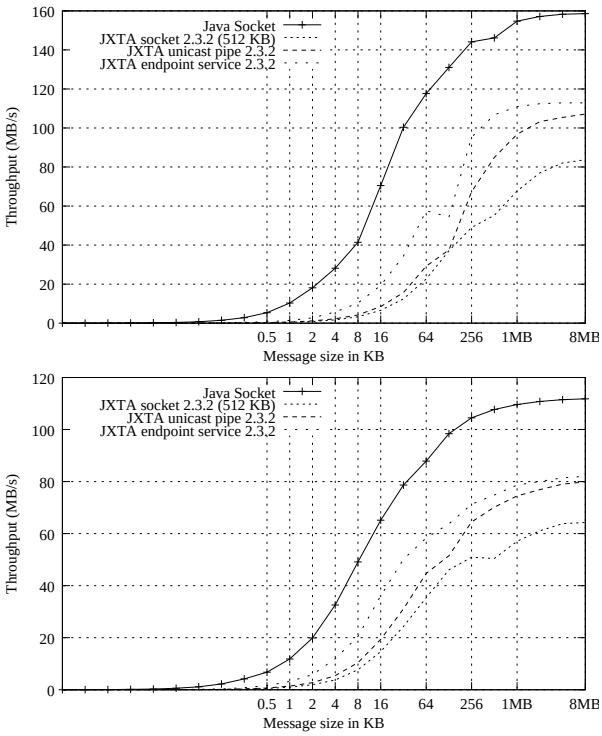


**Figure 1. Bandwidth of each layer of JXTA-J2SE 2.3.2 as compared to Java sockets over a Myrinet network (top) and a Giga Ethernet network (bottom).**

and IBM 1.4.1 JVMs, respectively. Note that, the difference between Myrinet and Giga Ethernet results is due to the hardware employed, as the Ethernet emulation mode is used for Myrinet.

**JXTA-J2SE unicast pipe.** In addition, Figure 1 illustrates the bandwidth available with the JXTA-J2SE unicast pipes. Overall, the small performance degradation as compared to the endpoint layer is explained by the composition of a pipe message: the presence of an XML message element requiring a costly parsing prevents this layer from reaching the performance of the endpoint layer. Moreover, as shown on Table 1, this extra parsing required for each pipe message also affects latency results: compared to the endpoint layer, latencies increase by more than 400 $\mu$s. However, unicast pipes are still able to achieve latencies in the sub-millisecond range, at least on Giga Ethernet.

**JXTA-J2SE sockets.** As opposed to the lower layers, JXTA sockets are far from reaching the performance of plain Java sockets. In their default configuration (e.g. with an output buffer size of 16 KB), JXTA sockets 2.2.1, for instance, attain a peak bandwidth of 12 MB/s over a Myrinet network. We were able to significantly improve the bandwidth and achieve 92 MB/s by increasing the size of the

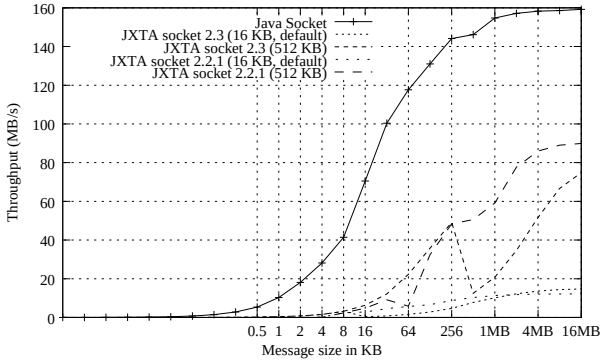


**Figure 2. Bandwidth of JXTA-J2SE sockets 2.2.1 and 2.3.2 over a Myrinet network for two different output buffer sizes, compared to Java sockets.**

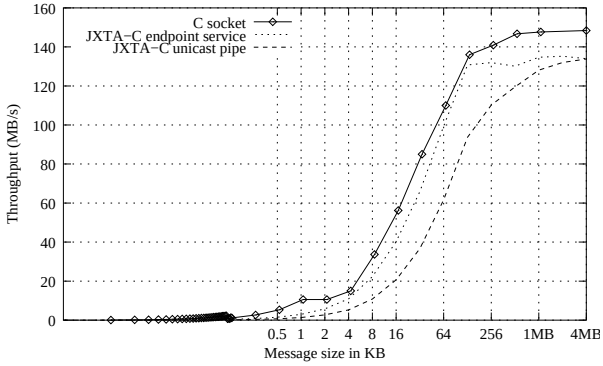


**Figure 3. Bandwidth of each layer of JXTA-C 2.1.1 as compared to C sockets over a Myrinet network.**

output buffer to 512 KB, as shown on Figure 2. Similar results were obtained over Giga Ethernet (for the sake of clarity, they are not represented on the Figure). Figure 2 also clearly shows the performance degradation between JXTA sockets 2.2.1 and 2.3.2. The irregular shape of JXTA sockets 2.2.1 curves has not been explained so far. Again, we suspect the first drop is due to an inefficient thread scheduling policy. The next drop, when the message size is about the size of the output buffer, seems to be due to bugs discovered in the reliability layer since JXTA-J2SE 2.3.2. To the best of our knowledge, some progress has been made on these issues even if it can still be observed in JXTA 2.3.3. Table 1 highlights the progress being made by JXTA Sockets 2.3.2 as regards latency: on Giga Ethernet it is able to reach a latency under one millisecond.

**Discussion.** In conclusion, JXTA-J2SE 2.3.2 communication layers do not saturate saturate SANs (as opposed to previous versions [4]. In contrast, latency results have largely improved since JXTA 2.2.1, but without reaching reasonably good performance for SANs. Finally, this evaluation has also highlighted that, in their default configuration, JXTA sockets achieve a very poor bandwidth. However, this result can significantly be improved by increasing the output buffer size. This requires the JXTA socket programmer to explicitly set this parameter in the user code. Based on these results, we can conclude that JXTA-J2SE can be adapted in order to benefit from the potential offered by SANs, at least as bandwidth is concerned.

## 5.2 JXTA-C: Analysis

Figures 3 and 4 show the bandwidth measurements of all the communications layers of JXTA-C 2.1.1 over SANs. Note that, as in the previous section, C sockets are used as an upper reference bound. The peak bandwidth values we measured for the endpoint service over Myrinet and Giga Ethernet are 135 MB/s and 103 MB/s respectively. The upper layer (unicast pipe) reaches bandwidths of 133 MB/s and 96 MB/s over Myrinet and Giga Ethernet, respectively. On Myrinet, this is an increase by 30 MB/s compared to results published in [4]. This highlights the improvements that have been made since JXTA-C 2.1. Based on these encouraging results, we have optimized JXTA-C 2.2 communication layers by implementing a message element caching mechanism. Message elements including source and destination information of messages are no longer included in each message sent by the endpoint service. Each peer is indeed aware of this information through the setting-up of the communication. In addition, we have removed an unneeded message element as direct exchanges between peers are possible (so called direct connection optimization). These aforementioned optimizations have a little impact on the peak bandwith of JXTA-C 2.2 on Giga Ethernet: both layers reach 100 MB/s, a small increase compared to JXTA-C 2.1.1. However and as expected, our optimizations have a major impact on the latency results, as shown on Table 2. On a Giga Ethernet network, the latency of JXTA-C 2.2 reaches 84 $\mu$s (endpoint service) and 90 $\mu$s (unicast pipe service). Compared to JXTA-C 2.1.1, this is a decrease by more than 200 $\mu$s for the latency of a unicast pipe and of 53 $\mu$s for the endpoint service. At the endpoint layer, the improvement is mainly explained by the caching mechanism of message elements. For unicast pipes, further improvements are explained by the so called direct connection optimization. However, the latencies are still higher than the latency of plain sockets over Giga Ethernet (39 $\mu$s).

| **JXTA-C** | 2.1.1 | | 2.2 |
|---|---|---|---|
| **Network** | Myrinet | GEther. | GEther. |
| **Endpoint service** | 310 $\mu$s | 137 $\mu$ | 84 $\mu$s |
| **Unicast pipe** | 690 ms | 298 $\mu$s | 90 $\mu$s |

**Table 2. Latency results for JXTA-C 2.1 and 2.1.1 on SAN (Giga Ethernet noted GEther.).**

Based on this evaluation, we can conclude that, in

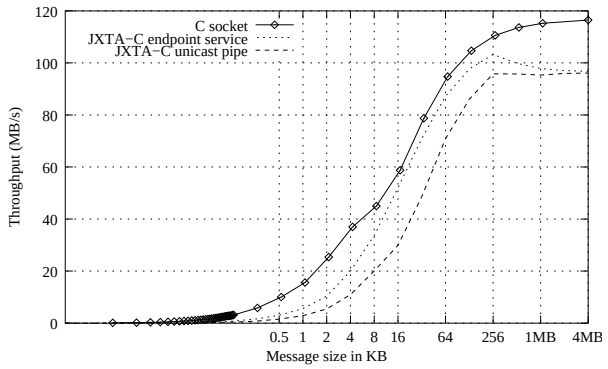


**Figure 4. Bandwidth of each layer of JXTA-C 2.1.1 as compared to C sockets over a Giga Ethernet network.**

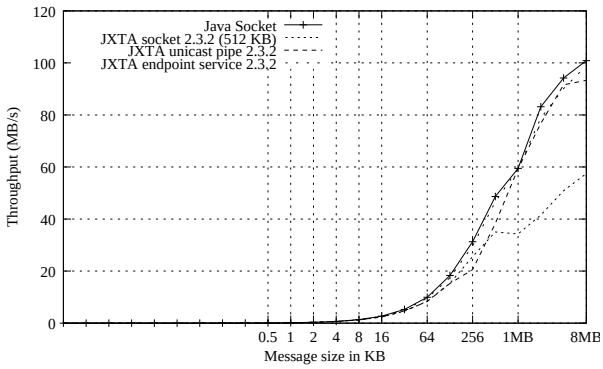


**Figure 5. Bandwidth of each layer for JXTA-J2SE 2.3.2 compared to Java sockets over a high-bandwidth WAN.**

their current implementation, the communication layers of JXTA-C are nearly able to saturate SANs, by reaching bandwidths values above 1 Gb/s. In addition, let us note that on the latency side, major improvements have made throughout JXTA-C releases, especially if we include our optimizations. To summarize, a path towards an efficient use of SAN capacities have been opened, even if work remain on the latency side.

## 6 Performance Evaluation of JXTA over Wide-Area Networks

This section analyzes the performance of JXTA's communications layers on WANs. Note that we had to tune the network settings of the nodes used for this benchmark. Our default maximum TCP buffer size initially set to 131072 bytes was limiting the bandwidth to only 7 MB/s. Based on the $bandwidth * delay$ law, we computed a theoretical maximum size of 1507328 bytes and increased this value by an arbitrary factor of 1.2. Therefore, we set the maximum TCP buffer sizes on each node to 1959526 bytes; `ttcp` configured with this value measured a raw TCP bandwidth of 107 MB/s, a reasonable level of performance.

**JXTA-J2SE's performance.** As for Giga Ethernet SAN benchmarks, Figure 5 shows that the endpoint layer and unicast pipes of JXTA-J2SE are able to perform similarly to plain sockets over a high-bandwidth WAN of 1 Gb/s. This level of performance was reached by modifying JXTA-J2SE's code in order to properly set the TCP buffer sizes. Using the default setting, a bandwidth of only 1 MB/s was reached for JXTA 2.3.2.

**JXTA-C's performances.** Figure 6 shows that the peak bandwidth for both communication layers of JXTA-C 2.1.1 over WANs are 94 MB/s, for a message size of 4 MB. As for the the SAN benchmarks, these higher results compared to results published in [4] are explained by the

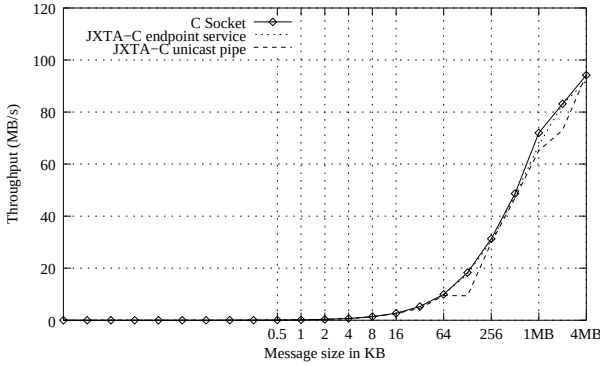


**Figure 6. Bandwidth of each layer for JXTA-C 2.1.1 compared to C sockets over a high-bandwidth WAN.**

improvements made since JXTA-C 2.1. However, this level of performance was only reached by modifying JXTA-C's code in order to properly set the TCP buffer sizes.

**Discussion.** Based on this evaluation, we can conclude that JXTA's communication layers, when used on high-bandwidth WANs, are able to reach the same bandwidths as for SAN benchmarks. JXTA-J2SE is able to efficiently use the bandwidth available on the links used for interconnecting sites of a grid. However, JXTA sockets are unable to successfully exploit the bandwidth available on WANs, even if JXTA sockets 2.3.2 achieve some improvement compared to previous versions [4].

## 7 Discussion

JXTA aims at providing generic blocks for building P2P services or applications. Such services or applications may have various requirements with respect to the performance

of inter-peer communications, but also with respect to the desired guarantees. It is, therefore, necessary to pick the appropriate communication layer according to the application requirements, and to properly configure it in order to efficiently use JXTA.

The fastest layer of JXTA is clearly the endpoint service. However, direct use of the endpoint service is not recommended, as communications are unreliable and only suitable for static point-to-point interactions. Moreover, this layer may be subject to short-term modifications, which may result in large amounts of work when upgrading to newer versions of JXTA. On the other hand, this layer provides the developer with full control of the logical topology and therefore allows one to implement alternative routing schemes. Therefore, a direct use of this layer is reserved to JXTA experts willing to develop highly specific P2P systems.

As regards the upper layers, using JXTA sockets or JXTA pipes is not an easy choice, at least on SANs and WANs. More specifically, the overhead introduced by the JXTA sockets on LANs compared to the underlying unicast pipes is low, if we take into consideration the features offered by this layer: reliable, bidirectional communications and the availability of a data-stream mode. Note however that this overhead is low only when JXTA sockets are configured to use larger output buffer size. In contrast, on SANs and WANs, the performance degradation of JXTA sockets compared to unicast pipe is high, therefore choosing unicast pipes seems more appropriate.

The good performance of JXTA for Fast Ethernet LANs [3] and high-speed WANs, at least in terms of bandwidth capability, makes JXTA a particularly good candidate for many wide-area Internet applications dealing with large data transfers over such networks. We can therefore say that JXTA-based collaborative platform such as JXCube [23] or projects supporting distributed computing on large data sets such as P3 [17] or JNGI [21], to name a few, have made a reasonable choice by using JXTA.

Different considerations need to be taken into account when using JXTA as a means of achieving a form of convergence of P2P and grid middleware. In this context, it is an important challenge to allow JXTA-based applications targeting grid infrastructures to transparently exploit these high performance networks. We have shown that, if is correctly tuned, the JXTA platform can deliver adequate performance (e.g., over 1 Gbit/s bandwidth and low latency for JXTA-C at least). Furthermore, we explain how this performance can be improved thanks to specialized external libraries. The overall conclusion is that JXTA may provide adequate communication performance that are required by grid computing applications.

However, these evaluations have revealed some weaknesses of JXTA's communication layers: 1) with JXTA-J2SE, the bandwidths of all layers have degraded since JXTA 2.3, hindering JXTA from saturating SAN links; 2) the communication layers of JXTA were not optimized for direct connections between peers therefore limiting the bandwidth but more importantly latency results on SANs. Therefore, we have successfully optimized JXTA-C 2.2 in order to be able to fully exploit SANs capacities.

By way of comparison with other middlewares, let us cite a few latency results results for various platforms typically used for building grid applications. We use [9] as a reference paper for the latency results of J2SE-based middlewares. However, given that the hardware setup used in this study is not identical to ours, we can only sketch rough trends. In [9], three types of platforms are benchmarked: Object Request Brokers (ORB), component-oriented platforms and web services. We can see that at least the 294 $\mu$s latency of JXTA 2.3.2's endpoint service outperforms middlewares in all three categories (517 $\mu$s for Java RMI over IIOP, 633 $\mu$s for OpenORB 1.4.0 and 2070 $\mu$s for ProActive 2.0, to name one in each category). Nevertheless, some ORB middlewares and component-oriented platforms, like ORBacus (115 $\mu$s) or Java RMI over JRMP (123 $\mu$s) and Fractal RMI (151 $\mu$s) respectively, achieve better results. Note however that, for a fair comparison, component-oriented platforms and web services should be compared to the higher communication layers of JXTA, such as unicast pipes and JXTA sockets. In that case, JXTA-J2SE layers do not range in the top any longer, even if they still outperform some platforms based on web services (4742 $\mu$ for Apache Axis 1.1). As regards JXTA-C, according to results available with the CORBA benchmark project [22], the endpoint and pipe layers can achieve better results than OmniORB 3.0 (173 $\mu$s) or ORBacus 4.0 (338 $\mu$s). However, they are still a little bit behind the performances of OmniORB 4.0 (52 $\mu$s). Overall, we can however conclude JXTA's communication layers performs reasonably well given the provided functionalities.

## 8 Conclusion

In the context of the current efforts for building grid services on top of P2P libraries, an important question is: to what extent is it reasonable to rely on P2P mechanisms to support the dynamic character of the grid? Are P2P techniques only useful for resource discovery? Or is there a way to take one step further, and efficiently exploit P2P data communication mechanisms? The question of the adequacy of P2P communication mechanisms for performance-constrained usages is therefore important in this context.

In this paper, we focus on benchmarking a key aspect of one widespread P2P open-source library: the performance of JXTA communication layers. We provide a detailed analysis and discussion of the performance of these layers for the most advanced bindings of JXTA (J2SE and C) over networks typically used for building grid infrastructures, such as SANs and WANs. We show that the JXTA platform can deliver adequate performance on grid infrastructures (e.g., over 1 Gbit/s bandwidth and low latency, at least for JXTA-C) if it is correctly tuned and optimized. Moreover, we explain how this performance can be further improved thanks to specialized external libraries. We also give some hints to designers of JXTA-based applications or services on how to

efficiently use each layer. This should allow developers to build higher-level services based on building blocks whose costs are known and optimized, which should lead to reasonable implementation choices.

However, these evaluations have revealed some weaknesses of JXTA in both SAN and WAN areas. JXTA-J2SE peak bandwidth values has degraded since JXTA 2.3, hindering JXTA from saturating SAN links. Moreover, the communication layers of JXTA are not optimized for direct connections available on SANs. Therefore, we have improved JXTA-C in order to be able to fully exploit capabilities of SANs. More precisely, we have optimized message composition emits by JXTA-C when direct communications are available as well as use a caching mechanism for some message elements. In addition, we have optimized the message path throughout the JXTA-C stack as well as implemented a zero-copy mode. In further preliminary experiments on a Myrinet 10-G network we could reach a bandwith of 850 MB/s for both communication layers of JXTA-C 2.3, while plain sockets reach 925 MB/s.

Even if this paper explored the impact of multiple factors, this research is not an exhaustive evaluation of all aspects that may impact on JXTA's communication performance. In particular, the work presented in this paper could be extended by using different virtual network topologies, involving more complex communication schemes.

## References


[1] M. Amoretti, G. Conte, M. Reggiani, and F. Zanichelli. Service Discovery in a Grid-based Peer-to-Peer Architecture. In *Int. Workshop on e-Business and Model Based IT Systems Design*, Saint Petersburg, Russia, Apr. 2004.

[2] G. Antoniu, L. Bougé, and M. Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. *Scalable Computing: Practice and Experience*, 6(3):45–55, September 2005.

[3] G. Antoniu, P. Hatcher, M. Jan, and D. A. Noblet. Performance evaluation of jxta communication layers. In *Proc. Workshop on Global and Peer-to-Peer Computing (GP2PC 2005)*, pages 251–258, Cardiff, UK, May 2005. Held in conjunction with the 5th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID 2005).

[4] G. Antoniu, M. Jan, and D. A. Noblet. Enabling the p2p jxta platform for high-performance networking grid infrastructures. In *Proc. of the 1st Intl. Conf. on High Performance Computing and Communications (HPCC '05)*, number 3726 in Lect. Notes in Comp. Science, pages 429–439, Sorrento, Italy, September 2005. Springer-Verlag.

[5] S. Baehni, P. T. Eugster, and R. Guerraoui. OS Support for P2P Programming: a Case for TPS. In *22nd Int. Conf. on Distributed Computing Systems (ICDCS '02)*, pages 355–362, Vienna, Austria, July 2002. IEEE Computer Society.

[6] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quétier, O. Richard, E.-G. Talbi, and T. Irena. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, Nov. 2006.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. of the 2003 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[8] C. Cheng, E. Pouyoul, and M. Jan. JXTA Distributed Framework. `http://jdf.jxta.org/`, 2003.

[9] C. Demarey, G. Harbonnier, R. Rouvoy, and P. Merle. Benchmarking the round-trip latency of various java-based middleware platforms. *Studia Informatica Universalis Regular Issue*, 4(1):7–24, May 2005.

[10] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence on Peer-to-Peer and Grid Computing. In *2nd Int. Workshop on Peer-to-Peer Systems (IPTPS '03)*, number 2735 in Lect. Notes in Comp. Science, Berkeley, CA, Feb. 2003. Springer-Verlag.

[11] E. Halepovic and R. Deters. The Cost of Using JXTA. In *3rd Int. Conf. on Peer-to-Peer Computing (P2P '03)*, pages 160–167, Linköping, Sweden, Sept. 2003. IEEE Computer Society.

[12] E. Halepovic and R. Deters. JXTA Performance Study. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM '03)*, pages 149–154, Victoria, B.C., Canada, Aug. 2003. IEEE Computer Society.

[13] E. Halepovic and R. Deters. JXTA Messaging: Analysis of Feature-Performance Tradeoffs. Submitted for publication, 2005.

[14] M. Junginger and Y. Lee. The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks. In *2nd Int. Conference on Peer-to-Peer Computing (P2P '02)*, pages 49–56, Linköping, Sweden, Sept. 2002. IEEE Computer Society.

[15] J.-M. Seigneur. Jxta Pipes Performance. `http://bench.jxta.org/papers/jmjxtapipesperformance.pdf`, 2002.

[16] J.-M. Seigneur, G. Biegel, and C. D. Jensen. P2P with JXTA-Java pipes. In *2nd Int. Conf. on Principles and Practice of Programming in Java (PPPJ '03)*, pages 207–212, Kilkenny City, Ireland, 2003. Computer Science Press, Inc.

[17] K. Shudo, Y. Tanaka, and S. Sekiguchi. P3: Personal Power Plant. GGF10: Open Grid Service Architecture - Peer-to-Peer Research Group (OGSA-P2P RG), Mar. 2004.

[18] D. Talia and P. Trunfio. Toward a Synergy Between P2P and Grids. *IEEE Internet Computing*, 7(4):94–96, 2003.

[19] D. Talia and P. Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. In *Euro-Par 2004: Parallel Processing*, number 3149 in Lect. Notes in Comp. Science, pages 1022–1031, Pisa, Italy, Aug. 2004. Springer-Verlag.

[20] P. Tran, J. Gosper, and A. Yu. JXTA and TIBCO Rendezvous - An Architectural and Performance Comparison. `http://www.smartspaces.csiro.au/docs/PhongGosperYu2003.pdf`, 2003.

[21] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In *3rd Int. Workshop on Grid Computing*, Lect. Notes in Comp. Science, pages 1–12, Baltimore, MD, Nov. 2002. Springer-Verlag.

[22] Open CORBA Benchmarking. `http://nenya.ms.mff.cuni.cz/~bench/`.

[23] JXCube - Jxta eXtreme Cube project. `http://jxcube.jxta.org/`.

[24] The JXTA project. `http://www.jxta.org/`, 2001.

[25] Cog Kit JXTA project. `http://www-unix.globus.org/cog/projects/jxta/`.

[26] JXTA-Grid project. `http://jxta-grid.jxta.org/`.

[27] JXTA specification project. `http://spec.jxta.org/`.

[28] Project JXTA-C. `http://jxta-c.jxta.org/`.