Parameter Tuning of JXTA-based P2P Platforms

Fatos Xhafa
Dept. of Languages and Informatics Systems
Technical University of Catalonia
Campus Nord, C/Jordi Girona 1-3
08034 Barcelona, Spain
EMail: fatos@lsi.upc.edu

Zorion Arrizabalaga
Dept. of Languages and Informatics Systems
Technical University of Catalonia
Campus Nord, C/Jordi Girona 1-3
08034 Barcelona, Spain
EMail: zarrizabalaga@lsi.upc.edu

Jordi Missé Bertran
Dept. of Languages and Informatics Systems
Technical University of Catalonia
Campus Nord, C/Jordi Girona 1-3
08034 Barcelona, Spain
EMail: jmisse@lsi.upc.edu

Leonard Barolli
Dept. of Information and Communication Eng.
Fukuoka Institute of Technology
3-30-1 Wajiro-higashi, Higashi-ku
Fukuoka 811-0295, Japan
EMail: barolli@fit.ac.jp

Abstract

P2P technologies have appeared as disruptive technologies that have spawned large scale collaborative applications involving thousands or even millions of users world-wide. Undoubtedly, most successful applications of P2P paradigm are information sharing between individuals and groups of users, which enable sharing and searching for digital content information, and exchanging via P2P communication such information. However, P2P applications are still difficult to program and to efficiently deploy in real network infrastructures and usually have network efficiency problems. A part from intrinsic difficulties of P2P algorithmics, there are difficulties in efficiently tuning P2P applications in real environments. Indeed, P2P applications usually involve many parameters, whose values cannot be arbitrarily fixed but require a careful tuning to achieve expected performance. In this work, we show by example of a P2P JXTA-based platform, how to tune the parameters of the platform using existing software tools (JConsole and JHat) and enhance its performance.

1. Introduction and motivation

P2P systems [9], [8], [5] appeared as the new paradigm after client-server and web-based computing. P2P systems are quite popular systems to share huge amount of data such as Napster, Gnutella, FreeNet and others. One of the main characteristics of such systems is file sharing among peers. However, the improvement of P2P protocols is enabling the development of P2P applications others than the well-known file-sharing applications. However, there is still few work to bring P2P system to real word applications, mainly due to the lack of robust P2P platforms that would allow the deployment of large P2P systems. The JXTA platform [4], [6], [7] is making possible the development of P2P real-world applications.

P2P applications are still difficult to program and to efficiently deploy in real network infrastructures and usually have network efficiency problems [3]. A part from intrinsic difficulties of P2P algorithmics, there are difficulties in efficiently tuning P2P applications in real environments. Indeed, P2P applications usually involve many parameters, whose values cannot be arbitrarily fixed but require a careful tuning to achieve expected performance. This is precisely the motivation of this work; we address the issue of how to tune the parameters of P2P platforms using existing software tools, such as JConsoloe and JHat t enhance their performance. We exemplify the approach through a P2P JXTA-based platform, namely, JXTA-Overly platform [10], [11]. Although most current research effort are devoted to tuning network applications, testbed and configuration issues of deployment of P2P systems are attracting the interest of researchers [1], [2].

The rest of the paper is organized as follows. We give an overview of the JXTA library in Section 2. The design of the JXTA-Overlay is given in Section 3. In Section 4 we give a description of most important parameters of JXTA-Overlay P2P platform and their tuning. Finally, we conclude in Section 5 with some remarks and indicate directions for future work.

2. JXTA networks

JXTA library is a set of open protocols proposed by Sun Microsystems that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner (see e.g. [4], [7]). JXTA peers create a virtual network in which any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs. We describe next the main entities of JXTA networks.

Peer: Any interconnected node is called peer. Peers work independently and asynchronously with other peers. Peers publish one or more interfaces that are used by other peers to establish peer-to-peer connections.

PeerGroup: A PeerGroup is a collection of peers that are joined to provide a secure shared environment for the participating peers. PeerGroups can decide their own policy of peer membership. Note that peers can simultaneously belong to more than one PeerGroup. As in the case of peers, PeerGroups offer their services such as discovery services, pipe services, peer join services, monitoring services etc.

Pipes: A pipe is a virtual communication channel established between two processes. A computer connected to the network can open, at transport level, as many pipes as its operating system permits. These channels act as data links between the two communication points. Through pipes can be sent any types of objects such as strings, binary code, java objects, etc. JXTA offers both unidirectional, not secure pipes, and bidirectional secure pipes.

Messages: Messages are objects used for communicating and interchanging data. A message is essentially an ordered sequence of tags/values, which could also include binary code (appropriately encoded).

Advertisements: The JXTA resources/services are represented using advertisements. An advertisement is a meta-data structured information (XML document), which are published with a certain lifetime that specifies its availability. JXTA offers the following types of advertisements: *Peer Advertisement*, *Peer Group Advertisement*, *Pipe Advertisement*, *Module Class Advertisement*, *Module Spec Advertisement*, *Module Impl Advertisement*, *Rendezvous Advertisement* and *Peer Info Advertisement*.

Further, peers can be classified in *Limited Edge peer* (peers that can send and receive messages but cannot send advertisements neither store advertisements in its cache); *Complete Edge peer* (peers that can send/receive messages, store advertisements but cannot manage resource discovery advertisements); *Rendezvous peer* (besides functionalities mentioned above, Rendezvous peers can manage resource discovery advertisements. Each group must have at least a rendezvous; when a peer joins a group of peers, it automatically looks for a rendezvous in the group. Moreover, rendezvous keep a list of other rendezvous and their respective peers); and, *Relay peer* (this type of peer serves as an intermediate nodes. Relay peer keep routing information to other peers and sends messages to other peers that can't access directly to another peer.)

3. The JXTA-Overlay

The objective of the overlay is to provide a set of basic primitives for file sharing, the discovery and allocations of resources, instant messaging, etc. This set of primitives is intended to be as complete as possible regarding the functionalities needed in most P2P applications. An important characteristics of the primitives is their independence from the applications that will be using them. Moreover, the primitives allow to keep the intrinsic decentralized nature of P2P systems.

3.1. The primitives

The set of primitives includes functionalities that allow peer discovery, peer's resources discovery, resource allocation, file/data sharing, discovery and transmission, instant communication and peer group functionalities, among others.

The primitives are organized in interfaces according to an affinity criterion. In what follows we give the main functionalities of these interfaces; here we use indistinctly the terms peer and resource.

Authentication: This interface includes typical methods for authentication of the final user/application that will be using the resources managed by the overlay. It should be noted that another authentication could be established at the application level, which would be independent of the overlay. In this interface we have, among others, the methods `connect` (which verifies the authentication by calling the `verifyAuthentification` method of JXTA, connects to a broker, flushes the local cache and fires an event); `configure` (which configures the local cache; it is called just once at the beginning); `login` and `disconnect`.

Resource discovery and information: This interface includes functionalities related to the discovery of peers managed by the overlay. The implementation of these functionalities is later done by using JXTA discovery services. It should be noted that, as part of resource discovery, the overlay includes functionalities to discover a resource of certain desired characteristics. Thus we have, among others, the methods `discoveryEvent`, `getPeerName` and `getPeerID`.

Management of executable tasks: An important place in the primitives is given to functionalities related to the management of executable tasks in distributed applications. These functionalities are intended to give service to users/applications on top of the overlay that submit executable tasks and receive results in turn. Thus we have, among others, the following methods:

- `executableTaskRequestRandom`: a new executable task is requested to be executed in another peer selected at random.
- `executableTaskRequestSelectedPeer`: a new executable task is requested to be executed in a (group of) selected peer(s).
- `executableTaskRequestDataEvaluator`: evaluates the execution of a task according to task data/characteristics.
- `executableTaskRequestEconomicEvaluator`: evaluates the execution of a task according to a given economic model based on task data/characteristics.
- `executableTaskDel`: deletes a task from list of pending tasks.
- `executableTaskAccepted`: indicates acceptance of a task execution in a specified resource.
- `executableTaskDeny`: indicates deny of a request for a task execution
- `executableTaskFinished`: advertises that the execution of task has successfully been finished.
- `executableTaskCanceledRequest`: requests cancellation of task execution.
- `executableTaskCanceledByDestination`: indicates that the executable task has been cancelled at destination resource.
- `executableTaskCanceledBySender`: indicates that the executable task has been cancelled by task's sender.
- `addExecutableType`: adds a new type of executable tasks that a resource supports. Similarly there is the `delExecutableType` method.
- `addExecutableTypes`: adds a list of executable types that a specified group of resources can support. Similarly there is the `delExecutableTypes` method.
- `getExecutableTaskLocalInfo`: returns the local information about an executable task.
- `getExecutableTaskRemoteInfo`: returns the information about an executable task at a remote peer (including the prediction for the completion time of the task).
- `getTasks`: returns all pending tasks of a specified type (e.g. executable task, file transfer, etc.).

File sharing, discovery and transmission: This includes sharing, discovery and transmission of files, which are basic functionalities of the overlay. The objective of these functionalities goes beyond the sharing in P2P systems since file transmission is necessary for submitting tasks to resources. Thus we have, among others, the following methods: `addSharedFile`, `addSharedDirectory`, `delSharedFile`, `getSharedFiles`, `sendFilePeer`, `sendFileAccepted`, `sendFileDeny`, `sendFileGroup`, `findFile`, `fileRequestRandom`, `fileRequestSelectedPeer`, `fileRequestEconomicEvaluator`, `cancelTransfer`, `localInfoTransfer`.

Instant communication: This interface includes methods for instant communication between peers and include `sendMsgPeer` (for sending a message to a specified peer) and `sendMsgGroup` (for sending a message to a group of peers).

Peer's statistics: Statistic information of resources is relevant for applications that will be built on top of the overlay. Thus we have, among others, the following methods:

- `getPeerStatistics`: returns statistic information on a specified resource.
- `getGroupStatistics`: returns statistic information on a group of resources.
- `getBrokerStatistics`: returns statistic information on a specified broker resource.
- `getBrokerStatistics`: returns statistic information on the broker of a specified group of resources.
- `getClientStatistics`: returns statistic information on a specified edge peer.
- `getClientStatistics`: returns statistic information on a specified group of edge peers.

There are also some primitives related to the peer's local cache that we have omitted here.

3.2. Peers in JXTA-Overlay

The set of the primitives is implemented by taking advantage that JXTA allows different types of peers. In the overlay we have defined *client peers* and *broker peers*. The former are *complete edge peer* while the later act as rendezvous and relays. Any PeerGroup has at least a broker to which client peers get connected and send their requests (e.g. resource allocation petitions).

Broker peers: The mission of a broker node is to manage and control all requests received from client peers. Also, broker manages the events produced by such requests and propagates them to superior levels of overlay. Among broker's functionalities we distinguish: event management, controlling the resources connected to the broker, maintaining the organization of resources in groups, finding the best resource for file sharing, finding the best resource for executing submitted tasks, maintaining updated statistic information, etc.

Client peers: Client peers can be of two types: *CMDClient*, which is instantiated by peer that do not interact with final user and *GUIClient* that offers a graphical interface for the final user.

4. Parameter Tuning

In this section we first present the most relevant parameters in JXTA-based P2P platforms and then show how to use existing software, such as JConsole and Jhat, for their tuning.

4.1. JXTA-Overlay parameters

The parameters in JXTA-Overlay can be divided into two groups: parameters related to Broker peers and parameters related to Client peers. The parameters are given in Table 1.

As can be seen from Table 1, Broker peer parameters include discovery time, publishing time (for resource discovery and rooms), advertisement time for broker's statistics and expiration time for broker's statistics. It should be noted that high values of these parameters could lead to an inconsistent state of the network; for instance, a high value of advertisement time for broker's statistics related to peer statistics could yield to obsolete information in case a peer is disconnected from the network. On the other hand, small values of the parameters could cause congestion of Broker peers due to very frequent updates.

Similarly, in case of Client peer the parameters include time parameters related to peer's advertisements, publishing, statistics, expiration as well as parameters related to pipes. Again, small values of these parameters could cause saturation of Client peers due to very frequent generation of advertisements and high traffic in the network due to the publishing and propagation of the peer's information.

These time parameters could be actually called *polling*-like parameters since they directly relate to the time intervals for checking the state of the network, performing actions or sending state information to the network.

Therefore, finding parameter values adjusted to the P2P networks is crucial to achieve expected performance. In particular, the values of time parameters should allow the P2P platform to successfully process all events and threads (*executors*) generated in the platform. Inappropriate values of the parameters could cause the accumulation of many events and executors that the platform might not be able to process and hence causing collapse of the platform.

4.2. Tuning parameters of JXTA-Overlay

JConsole[1] and JHat (Java Heap Analysis Tool)[2] are standard Java tools for analyzing performance of Java-based applications. JConsole offers many functionalities for detecting the lack of memory, deadlocks as well as monitoring various threads of an application. JHat can be used in conjunction with JConsole, through its option to create a dump file (created using MBeans) that JHat can analyze the performance of the application's heap.

The performance of the JXTA-Overlay is done by analyzing Broker peers and Client peers performance (in terms of their parameters). The experimental study started with a default setting of parameters (see Tables 2 and 3) and was done in several iterating steps with the aim of finding most appropriate values of the parameters. In order to realistically measure the performance of the platform, peers shared a considerable number of files, initiated several task executions, some peers initiated file search etc. Thus, the analysis is conducted under a realistic workload of the P2P platform.

Broker peer's performance analysis. We analyzed Broker's parameters when client peers were trying to connect to the network. Using the default values of Broker's parameters, we observed that the CPU usage, heap and number of threads reach levels that can not be functional (see snapshot in Fig. 1).

The rather large number of executors could be explained because executors are created faster than Broker peer can process them (`flush_cache` was already applied to clear expired executors). Therefore we conducted several iterations and changed the values of parameters slowly. The final values of parameters (see Tables 2 and 3) resulted in stable values in number of threads as well in CPU and Heap usage (see snapshot in Fig. 2).

1. http://java.sun.com/javase/6/docs/technotes/tools/share/jconsole.html
2. http://java.sun.com/javase/6/docs/technotes/tools/share/jhat.html

Table 1. Broker and Client peer parameters.

Broker peer's parameters	Client peer's parameters		
broker.discovery.time	client.ping.time	client.criteriums.adv.lifetime	discovery.room.publish.time
broker.discovery.publish.time	client.discovery.time	client.criteriums.adv.expiration.time	discovery.room.time
broker.stats.adv.time	client.discovery.publish.time	client.stats.adv.time	pipe.adv.room.time
broker.stats.adv.lifetime	client.info.adv.time	client.stats.adv.lifetime	pipe.adv.broker.time
broker.stats.adv.expiration.time	client.info.adv.lifetime	client.stats.adv.expiration.time	pipe.adv.client.time
broker.discovery.room.publish.time	client.info.adv.expiration.time client.criteriums.adv.time	client.stats.update.time client.stats.max.times	pipe.adv.lifetime pipe.adv.expiration.time

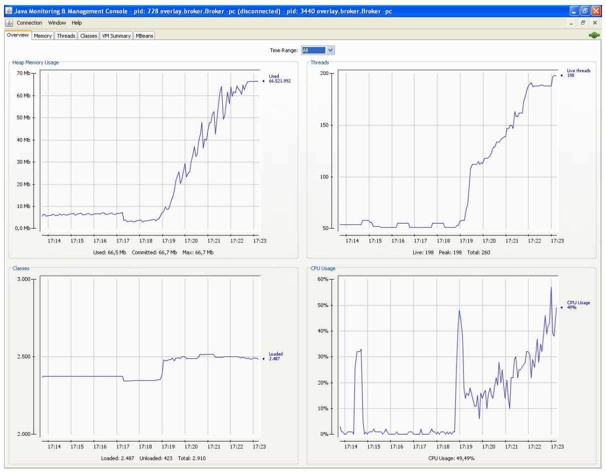
Java Monitoring & Management Console - pid: 728 overlay.broker.Broker -pc (disconnected) - pid: 3440 overlay.broker.Broker -pc
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range:
Heap Memory Usage
Threads
Classes
CPU Usage
Used: 66,5 Mb Committed: 66,7 Mb Max: 66,7 Mb
Live: 198 Peak: 198 Total: 260
Loaded: 2.487 Unloaded: 423 Total: 2.910
CPU Usage: 49,49%

Figure 1. CPU usage, Heap and number of threads of Broker peer (initial configuration).

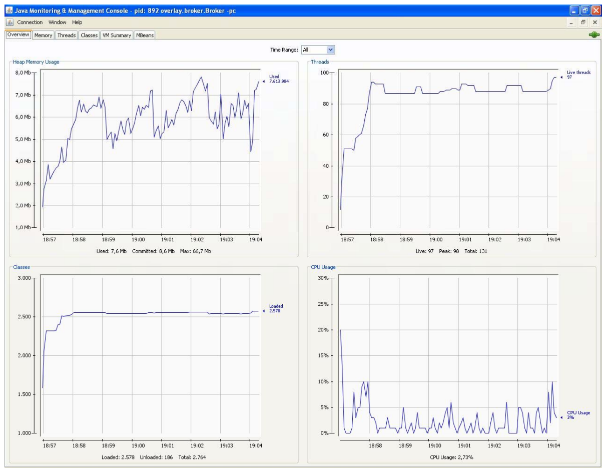
Java Monitoring & Management Console - pid: 892 overlay.broker.Broker -pc
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range: All
Heap Memory Usage
Threads
Classes
CPU Usage
Used: 7,6 Mb Committed: 8,6 Mb Max: 66,7 Mb
Live: 97 Peak: 98 Total: 131
Loaded: 2.578 Unloaded: 186 Total: 2.764
CPU Usage: 2,73%

Figure 2. CPU usage, Heap and number of threads of Broker peer (final configuration).

Client peer's performance analysis. We analyzed in a similar way the performance of client peer performance. Initially we used the default configuration for client peers and monitored the CPU and Heap usage as well as number of threads. As can be seen from Fig. 3, the Heap usage and number of threads are high.

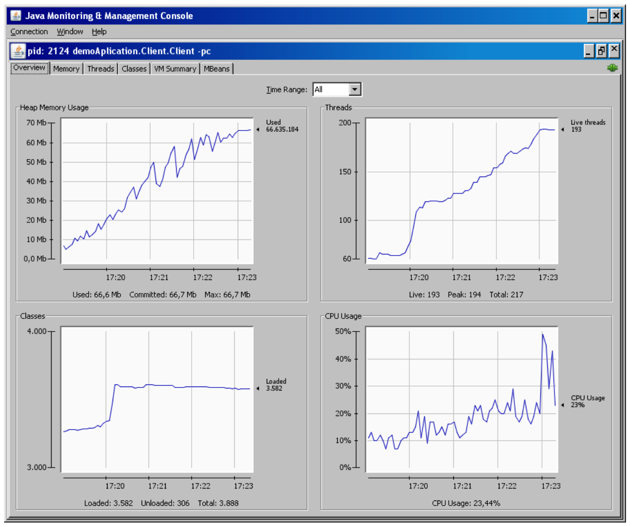
Java Monitoring & Management Console
Connection
Window
Help
pid: 2124 demoAplication.Client.Client -pc
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range: All
Heap Memory Usage
Used 66.635.184
Threads
Live threads 193
Classes
Loaded 3.582
CPU Usage
CPU Usage 23%
Used: 66,6 Mb Committed: 66,7 Mb Max: 66,7 Mb
Live: 193 Peak: 194 Total: 217
Loaded: 3.582 Unloaded: 306 Total: 3.888
CPU Usage: 23,44%

Figure 3. CPU usage, Heap and number of threads for Client peer (initial configuration).

Again, we conducted several iterations and changed the values of parameters slowly. The final values of parameters (see Table 2 and Table 3) resulted in stable values in number of threads as well in Heap usage (see snapshot in Fig. 4).

5. Conclusions

In this work we have addressed the issue of tuning P2P applications in order to enhance their performance. P2P networks involve a large number of parameters and their performance largely depends on appropriate values of the parameters. We have exemplified the approach of parameter tuning of P2P platforms

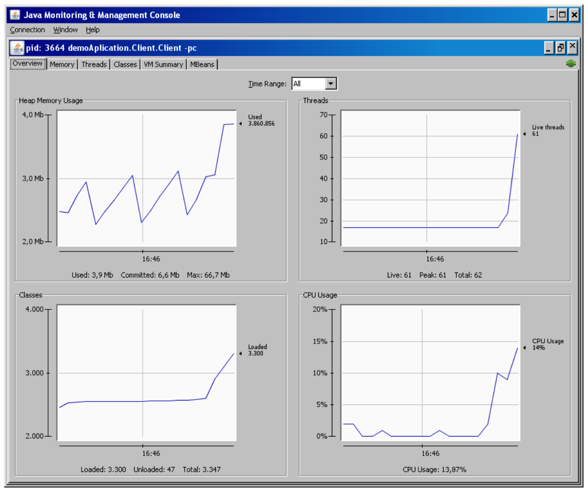
Java Monitoring & Management Console
Connection Window Help
pid: 3664 demoAplication.Client.Client -pc
Overview Memory Threads Classes VM Summary MBeans
Time Range: All
Heap Memory Usage
4,0 Mb
3,0 Mb
2,0 Mb
Used 3.860.856
16:46
Used: 3,9 Mb Committed: 6,6 Mb Max: 66,7 Mb
Threads
70
60
50
40
30
20
10
Live threads 61
16:46
Live: 61 Peak: 61 Total: 62
Classes
4.000
3.000
2.000
Loaded 3.300
16:46
Loaded: 3.300 Unloaded: 47 Total: 3.347
CPU Usage
20%
15%
10%
5%
0%
CPU Usage 14%
16:46
CPU Usage: 13,87%

Figure 4. CPU usage, Heap and number of threads of Client peer (final configuration).

Table 2. Parameter values for Broker peer: default and new configuration.

New Broker configuration	Default Broker configuration
# UNITS	# UNITS
# In minutes:	# In minutes:
# - client.ping.time	# - client.ping.time
# - client.stats.update.time	# - client.stats.update.time
# - client.stats.adv.lifetime	# - client.stats.adv.lifetime
# - client.stats.adv.expiration.time	# - client.stats.adv.expiration.time
# - executable.small.time	# - executable.small.time
# - broker.stats.adv.lifetime	# - broker.stats.adv.lifetime
# - broker.stats.adv.expiration.time	# - broker.stats.adv.expiration.time
# In seconds: the rest	# In seconds: the rest
broker.discovery.time=18	broker.discovery.time=5
broker.discovery.publish.time=10	broker.discovery.publish.time=5
broker.stats.adv.time=20	broker.stats.adv.time=30
broker.stats.adv.lifetime=5	broker.stats.adv.lifetime=5
broker.stats.adv.expiration.time=5	broker.stats.adv.expiration.time=5
discovery.room.publish.time=50	discovery.room.publish.time=5
discovery.room.time=50	discovery.room.time=5
pipe.adv.room.time=50	pipe.adv.room.time=5
pipe.adv.broker.time=10	pipe.adv.broker.time=5
pipe.adv.client.time=10	pipe.adv.client.time=5
pipe.adv.lifetime=25	pipe.adv.lifetime=10
pipe.adv.expiration.time=50	pipe.adv.expiration.time=50

through the tuning of parameters of JXTA-Overlay using existing software tools such as JConsole and Jhat. The proposed approach is useful for tuning also other important parameters such as size of PeerGroup, number of tasks and shared files in client peers in order to evaluate the scalability of the P2P platform.

Table 3. Parameter values for Client peer: default and new configuration.

New Client peer configuration	Default Client peer configuration
# In minutes:	# In minutes:
# - client.ping.time	# - client.ping.time
# - client.stats.update.time	# - client.stats.update.time
# - client.stats.adv.lifetime	# - client.stats.adv.lifetime
# - client.stats.adv.expiration.time	# - client.stats.adv.expiration.time
# - executable.small.time	# - executable.small.time
# - broker.stats.adv.lifetime	# - broker.stats.adv.lifetime
# - broker.stats.adv.expiration.time	# - broker.stats.adv.expiration.time
# In seconds: the rest	# In seconds: the rest
client.ping.time=3	client.ping.time=3
client.discovery.time=10	client.discovery.time=5
client.discovery.publish.time=15	client.discovery.publish.time=5
client.info.adv.time=7	client.info.adv.time=5
client.info.adv.lifetime=12	client.info.adv.lifetime=10
client.info.adv.expiration.time=60	client.info.adv.expiration.time=60
client.criteriums.adv.time=15	client.criteriums.adv.time=15
client.criteriums.adv.lifetime=15	client.criteriums.adv.lifetime=15
client.criteriums.adv.expiration.time=65	client.criteriums.adv.expiration.time=45
client.stats.adv.time=20	client.stats.adv.time=30
client.stats.adv.lifetime=5	client.stats.adv.lifetime=5
client.stats.adv.expiration.time=5	client.stats.adv.expiration.time=5
client.stats.update.time=5	client.stats.update.time=5
client.stats.max.times=24	client.stats.max.times=24
discovery.room.publish.time=50	discovery.room.publish.time=5
discovery.room.time=50	discovery.room.time=5
pipe.adv.room.time=50	pipe.adv.room.time=5
pipe.adv.broker.time=10	pipe.adv.broker.time=5
pipe.adv.client.time=10	pipe.adv.client.time=5
pipe.adv.lifetime=25	pipe.adv.lifetime=10
pipe.adv.expiration.time=50	pipe.adv.expiration.time=50

References

[1] D. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Experience with an evolving overlay network testbed. *ACM SIGCOMM Computer Comm. Review*, **33**(3):13–19, 2003.

[2] B. Butnaru, F. Dragan, G. Gardarin, I. Manolescu, B. Nguyen, R. Pop, N. Preda and L. Yeh. P2PTester: a tool for measuring P2P platform performance. Proc. of Int'l Conf. on Data Engineering, 1501–1502, IEEE 2007.

[3] H. Bal, H. Casanova, J. Dongarra, and S. Matsuoka. Application-Level Tools. In Foster and Kesselman, eds, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 24, 463–489. Morgan 2003.

[4] D. Brookshier, D. Govoni, N. Krishnan, and J.C Soto. *JXTA: Java P2P Programming*. Sams Publishing, 2002.

[5] J. Crowcroft, T. Moreton, I. Pratt, and A. Twigg. Peer-to-Peer Technologies. In Foster and Kesselman, eds, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 29, 593–622. Morgan 2003.

[6] S. Li. *Early Adopter JXTA*. Wrox Press Information Inc., 2003.

[7] S. Oaks, B. Traversat, and L. Gong. *JXTA in a Nutshell*. O'Reilly, 2003.

[8] A. Oram, ed. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 1st edition, 2001.

[9] C. Shikey. What is P2P ... and what isn't. O'Reilly Network, November 2000.

[10] F. Xhafa, L. Barolli, T. Daradoumis, R. Fernandez, S. Caballé. Extension and evaluation of JXTA protocols for supporting reliable P2P distributed computing. *International Journal of Web Information Systems*, **4**(1), 121-135, 2008, Emerald Pub.

[11] F. Xhafa, L. Barolli, T. Daradoumis, R. Fernandez, S. Caballé. Jxta-Overlay: An Interface for Efficient Peer Selection in P2P JXTA-based Systems. *International Journal on Computer Standards & Interfaces*, Elsevier, 2008. In Press