Juxta-Cat: A JXTA-based platform for distributed computing

Joan Esteve Riasol  
Computing Laboratory  
Informatics Faculty of Barcelona  
Campus Nord, C5  
C/Jordi Girona 1-3  
08034 Barcelona, Spain  
jesteve@fib.upc.edu

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

ABSTRACT

In this paper we present a JXTA-based platform, called Juxta-CAT, which is an effort to use the JXTA architecture to build a job execution-sharing distributed environment. The Juxta-CAT platform has been deployed in a large-scale, distributed and heterogeneous P2P network, based on open JXTA protocols. The goal of our platform is to create a shared Grid where client peers can submit their tasks in the form of java programs stored on signed jar files and are remotely solved on the nodes of the platform. Thus, the main goal of our platform is giving direct access to resources and sharing of the computing resources of nodes, in contrast to other well-known P2P systems that only share hard disk contents.

The architecture of our platform is made up of two types of peers: common client peers and broker peers. The former can create and submit their requests using a GUI-based application while the later are the administrators of the Grid, which are in charge of efficiently assigning client requests to the Grid nodes and notify the results to the owner’s requests. To assure an efficient use of resources, brokers use an allocation algorithm, which can be viewed as a price-based economic model, to determine the best candidate node to process each new received petition. The implementation and design of peers, groups, job and presence discovery, pipe-based messaging, etc. are developed using the latest updated JXTA libraries (currently release 2.3.7) and JDK 1.5 version.

Several types of applications arising from different fields such as scientific calculations, simulations, data mining, etc. can be solved in Juxta-CAT. To create a suitable demo scenario and test the proposed platform, we have joined and used the PlanetLab platform (http://www.planet-lab.org/). Juxta-CAT Project and its official web site have been hosted in Java.NET community at https://juxtacat.dev.java.net.

1. INTRODUCTION AND MOTIVATION

The rapid development of Internet and other new technologies has yielded to new paradigms of distributed computing, among others the Computational Grids and P2P systems. Computational Grids were introduced by Foster and other researchers in late '90 [7, 8, 6]. The main focus of this new distributed paradigm was that of providing new computational frameworks by which geographically distributed resources are logically unified as a computational unit. As pointed out by Foster et al. Computational Grid is "... a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically depending on their availability, capability, performance, cost, and users’ QoS requirements." Grid computing motivated the development of large scale applications that benefit from the large computing capacity offered by the Grid. Thus, several projects such as NetSolve [4], MetaNeos Project (Metacomputing environments for optimization), and applications for Stochastic Programming [9], Optimization Problems [15], to name a few, used grid computing.

On the other hand, P2P systems [13, 11, 2] appeared as the new paradigm after client-server and web-based computing. P2P systems are quite known due to popular P2P systems such as Napster, Gnutella, FreeNet and others. One of the main characteristics of such systems is file sharing among peers. Due to this, and in order to stress the difference between computational grids and P2P systems, Foster et al. remarked that "... the sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokerage strategies emerging in industry, science, and engineering".

It should be noted however that since the appearance of first grid systems and P2P systems, the grid computing and P2P computing has evolved so that both paradigms have many characteristics in common, among which we distinguish the distributed computing. It is precisely this characteristic that we explore in this work, that is, the use of resource sharing for problem solving in dynamic environ-
ment. Grid systems has shown to be very useful for real
world applications, indeed, several types of grids has ap-
ppeared such as Compute Grids, Data Grids, Science Grids,
Access Grids, Bio Grids, Sensor Grids, Cluster Grids, Cam-
pus Grids, Tera Grids, and Commodity Grids, among oth-
ers. The success of computational grids could be explained
by the development of several middleware, such as Globus,
MPI-Grid2 and Condor-G that facilitate the development
of grid-based applications. On the P2P side, the improve-
ment of P2P protocols is enabling the development of P2P
applications others than the well-known file-sharing applica-
tions. 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. This is precisely the motivation of this work. We
propose a P2P platform, called Juxta-CAT, developed using
JXTA architecture.

The goal of the Juxta-CAT platform is to create a shared
Grid where client peers can submit their tasks in the form
of java programs stored on signed jar files and are remotely
solved on the nodes of the platform. Thus the Juxta-CAT
is a real peer-to-peer environment that represents a workspace
structured as a grid, where peer clients can share their re-
sources of CPU, memory and hard disk for solving their
problems.

The architecture of our platform is made up of two types
of peers: common client peers and broker peers. The former
can create and send their requests using a GUI-based ap-
plication while the later are the administrators of the Grid,
which are in charge of efficiently assigning client requests to
the Grid nodes and notify the results to the owner’s requests.
One key issue in such systems is the flexible and efficient use
of resources [5, 3, 14, 12] in order to benefit from the com-
puting capacity as efficiently and economically possible. To
this end, the brokers in our platform use an allocation algo-
rithm, which can be viewed as a simple price-based economic
model, to determine the best candidate peers to process each
new received petition. The implementation and design of
peers, groups, job and presence discovery, pipe-based mes-
saging, etc. are developed using the latest updated JXTA
libraries (currently release 2.3.7) and JDK 1.5 version. Our
proposal extends and improves several existing features of
the JXTA, especially those related to the management of
presence mechanism and the pipe service system.

Several types of applications arising from different fields
such as scientific calculations, simulations, data mining, etc.
can be solved using Juxta-CAT. To create a suitable demo
scenario and test the proposed platform, we have joined and
used the PlanetLab platform (http://www.planet-lab.org/).
In this stage of the work we have considered some simple ap-
plications (e.g. computing PI number with high precision),
namely applications that are composed of many independ-
ent tasks that can be simultaneously submitted and solved
in the grid. Using this type of applications we were able
to test the robustness of the Juxta-CAT and measure its
performance as regards the speed-up of computation. Our
experimental results show the feasibility of the Juxta-CAT
for developing large-scale applications as we have deployed
the grid applications in a real P2P network based on Plan-
etLab nodes.

Juxta-CAT Project and its official web site have been
hosted in Java.NET at https://juxtacat.dev.java.net and has
also been published in the JXTA Developer Spotlight of

The paper is organized as follows. We give in Sect. 2 some
preliminaries on JXTA protocols that we have used in the
development of Juxta-CAT. The architecture of the Juxta-
CAT platform is given in Sect. 3. In Sect. 4 we give some
of the implementations issues, the improvements on JXTA
protocols as well as the allocation algorithm used by bro-
kers. The evaluation of the Juxta-CAT platform and some
computational results are given in Sect. 5. We conclude in
Sect. 6 with some remarks and point out further work.

2. OVERVIEW ON BASIC JXTA PROTO-
COLO

JXTA technology 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 man-
ner (see e.g. [2, 10]). 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 be-

hind fire-walls and NATs or when they are on different net-
work transports.

The current specification of this architecture determines
the management of the clients (peers) of the developed net-
works.

One important advantage of JXTA is its platform inde-
pendence as regards:

- **Programming language**: Implementations of JXTA pro-
tococols are found in Java, C, C++ and other languages.
- **Transport protocol**: It is not required to use an specific
  net structure. P2P applications can be developed over
  TCP/IP, HTTP, Bluetooth and other transports.
- **Security**: Developers are free to manage the security
  issues of the developed platform.

The updated core libraries for each implementation are
found at the official JXTA page at http://www.jxta.org. For
the purposes of the development of Juxta-CAT platform, we
needed to only work with the J2SE version of JXTA.

The JXTA protocols have been structured as a working
stack, as can be seen in Fig. 1. It should be noticed that
JXTA protocols are independent among them. A definition
of a JXTA peer does not require implementing the set of all
protocols in order to participate in the network. For in-
stance, a node may need not statistical information about
the other members of the peer group, therefore it is not neces-
sary for that node to implement the Protocol of Information
or a node could have previously stored in its memory the set
of peers or services to work with, and hence it does not need
the additional protocol of resource discovery.
3. JUXTA-CAT ARCHITECTURE

In this section we describe the basic aspects of the Juxta-CAT platform. Before starting the design of the platform we carefully identified the possible roles that each peer participant was going to play in the Juxta-Cat platform. The importance of a clear specification of the peer roles was crucial to the architecture of the platform. Indeed, each role represents a “subsystem” of the platform requiring a different treatment during design and implementation stages. From now on, we use the term peer to denote the machine or node on the Juxta-Cat platform. As we show next, peers can execute either a client or a broker and hence we speak of client peers and broker peers.

We will also speak of request, petition, job or task indistinctly to denote a task submitted to the grid. To be precise, the request will contain all the information that a grid node, which receives the request, needs for solving a task.

3.1 Client peers

Client peers are the end users of the Juxta-CAT. Client peers are obtained by downloading and installing the application from the official page of Juxta-CAT. Once the machine is “converted” into a client peer, on the one hand, the user will connect to the peer-to-peer network and submit execution requests to their peergroup partners. One the other hand, client peers will also be able to process received requests sent to it by other nodes through the brokering and notify them the result of the request, once it is completed (see Fig. 2 for an UML diagram representation.)

Therefore, a client peer plays two possible roles as regards the resolution requests:

(a) submitting job requests to the grid and receiving the results.

(b) receiving job requests submitted to the grid by other client peers and notifying the result to the corresponding owners of the requests.

Note that, while submitting job requests to the grid, a client peer can also decide whether it includes itself as a possible candidate peer to receive the proper submitted requests or not.

3.2 Broker peers

Broker peers are the governors of the request allocation in Juxta-Cat (see Fig. 3 for an UML diagram representation.)

Brokers act like bots of the network: they are connected to the to P2P platform and are in charge of receiving and allocating the requests sent by clients of the peergroup. Whenever a broker receives a request, it explores the state of the rest of nodes currently connected to the network, examining their working and connection statistics. Then, it uses this historical/statistical data to select, according to a simple price-based economic model, the best candidate peer for processing that request.

Once the best node, for a given request, is selected, the broker sends to that node a message with the information needed for the processing the request. If there were some error in sending the message, or if there were no nodes available to receive the requests (a request not necessarily is processable by all nodes) then the broker adds the request to
the queue of pending requests. Thus, each broker maintains and manages its own pending request queue, which is updated according to the state of the network and the arrival on new requests (see Fig. 4).

Figure 4: UML diagram of the requests queue maintained by the Broker.

4. IMPLEMENTATION ISSUES

The design pattern used for Juxta-CAT is an integration of the Model-View-Controller and Observer patterns. We have thus three independent modules or layers: View, Model and Control incorporating the observer pattern to communicate the results between different layers (from Control to Model and from Model to View). We give this organization of Juxta-CAT in layers in Fig. 5 (see also Fig. 6).

Figure 5: JXTA architecture (MVC+Observer).

Further, as can be seen from Fig. 6, we decided to implement two types of client peers: GUI client peers and CMD client peers. The former (implemented using Java Swing) allows the peer to have both functions (a) and (b) (see Subsect. 3.1), that is, submitting and receiving requests, while the later can only receive requests. In this way, any user can contribute his/her resources to the platform even when he/she is not interested in submitting requests to the grid.

During the development of Juxta-CAT, it has been necessary to improve the original JXTA libraries regarding the presence mechanism and pipe service system. JXTA is an open specification that only offers a standard mechanism, based on concrete but adaptable protocols, to build very basic peer-to-peer applications. In a certain sense, these libraries are just the starting point for the definition of a basic skeleton of the final application.

4.1 Improvements to JXTA protocols

In the case of the Juxta-CAT platform, it is very important to maintain and efficiently manage the updated publication of the presence information and statistics as well as its publication to the nodes. The existing mechanisms offered by JXTA did not match our requirements. Therefore, it was necessary to change part of the implementation of the Discovery Service and Pipe Service of JXTA to adapt them to the requirements of Juxta-CAT. We deal with these changes in next paragraphs.

Improvement of the JXTA Discovery Service. Originally, JXTA maintains the peergroup information by the notification of presence that each node publishes in the cache of other nodes. But this procedure is not automatic, therefore, if we do not worry to refresh this information to the Discovery Service of Juxta-CAT, we will never be able to guarantee that the collected data are updated.

Therefore, Juxta-CAT will have its own process, a Java Thread, that periodically updates the information contained in his local cache and trying to diminish the data traffic of the network. In this way, the discovery service in Juxta-CAT follows essentially these steps:

- Periodically publish the presence advertisement in its own local cache using the method publish() of the DiscoveryService.
- Publish remotely this advertisement in the cache of the rendezvous node of the peergroup (remotePublish() method).
• Send a discovery query to the peergroup asking for other presence advertisements.
• Copy responses to local cache and delete those that have expired.

(See also Fig. 7 for an UML diagram representation.)

Figure 7: UML diagram of the Juxta-CAT Discovery Service manager.

Improvement of Pipe Service. We explain next how is improved the Pipe Service system that performs communication between nodes of a JXTA network. This is certainly one of the most important aspects of the Juxta-CAT. The main observation here is that JXTA Pipe Service doesn’t check whether a message has been successfully delivered to its destination. In case of failure, JXTA doesn’t attempt a new delivery of the message. Clearly, there is no timeout control as regards message delivery.

In order to send messages or data between peers, JXTA uses a pipe mechanism. 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. Different programming languages such as C++ or Java have specific libraries with the necessary system calls for managing pipes with opening, reading, writing and closing operations of the connections.

Next we briefly describe how we solved the above mentioned problems of the JXTA Pipe System resulting thus in an improved Pipe Service (see Fig. 8).

• Presence through Pipe Advertisement: a Juxta-CAT node declares, when its launched, an advertisement with the information relative to the user: role (client peer vs broker peer) within the grid, IP address, node name, and the unique JXTA identifier. This Advertisement is distributed by the network through Discovery Service, notifying to the rest of users of the grid the address of the pipe.

• Inbox messages control (see Fig. 9 for an UML diagram representation). Each node creates one pipe for receiving messages. This inbox pipe is managed by an independent thread that notifies to the superior layers of the system all the messages that are sent to the node.

Figure 8: UML diagram of the package net.juxtacat.modules.pipe.*.

Figure 9: UML diagram of the Juxta-CAT Inbox manager.

• Outbox messages control (see Fig. 10 for an UML diagram representation). In order to assure that the messages are successfully delivered to their addresses, we have developed a managing system based on queues. Each queue belongs to a possible destination node of our messages. When we want to send a message, we add it to its respective queue. Periodically, the manager sends simultaneously, according to the order of messages in the sequence, all the messages that have been kept in each queue. If a connection establishment fails, the message returns to the first position of the destination queue. A queue without pending messages to be sent remains blocked, and it does not consume resources nor memory of the Java Virtual machine.

• Timeout Window. A timeout window is used while trying to establish a connection. Each attempt to connect with a remote pipe through its advertisement has
a timeout limit. We have defined a time interval, in a way that any failed connection retries the delivery with an increased timeout. We give in Table 1 the values for the timeout window used in Juxta-CAT.

Table 1: Timeout Window for sending and receiving.

<table>
<thead>
<tr>
<th>Tries</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>&gt; 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker’s Sending Window (millisec)</td>
<td>1750</td>
<td>2500</td>
<td>4000</td>
<td>6000</td>
<td>12000</td>
<td>18000</td>
</tr>
<tr>
<td>Client Sending Window (millisec)</td>
<td>2000</td>
<td>3500</td>
<td>6000</td>
<td>12000</td>
<td>18000</td>
<td>30000</td>
</tr>
</tbody>
</table>

During the testing phase of Juxta-CAT we observed that the number of lost messages ranges in 0% - 1% of the total number of sent messages, which requires sending them again.

4.2 The allocation of jobs to resources

Allocation of jobs to resources is a key issue in order to assure a high performance of applications running on the Juxta-CAT. In this section we show the algorithm implemented by brokers to allocate requests to the resources (see Fig. 11 for an UML diagram representation.)

The algorithm can be seen as a simple price-based model and uses historical information maintained by Juxta-Cat brokers. Therefore, it is fundamental for any broker to maintain updated information about the state of the network and statistics on the performance of the nodes of the grid.

When a broker receives an execution request, it will consult its own historical information and use it (according to the price-model) to determine the price of the allocation to different nodes and finally will decide which is the “cheapest” candidate to execute this request. This policy of task allocation is common for all brokers in Juxta-CAT.

The allocation algorithm works as follows. Based on the historical information on the nodes, the broker uses a set of criteria to compute a score for each candidate node. These criteria are quantitative and can be independent among them. Altogether, these criteria must contribute to bring knowledge to the broker about the “economic saving” that would report any candidate node for the given task resolution. The score for any node is computed according to the following criteria:

1. Total amount of resolved jobs: the larger is this number the better is the node and vice-versa.
2. Number of enqueued jobs waiting for execution: the larger is this number the worse is the node and vice-versa.
3. Number of enqueued JXTA messages waiting to be sent: the larger is this number the worse is the node and vice-versa.
4. Average number of resolved versus failed jobs: the larger is the number of successfully resolved jobs and smaller the number of uncompleted jobs, the better is the node and vice-versa.
5. Average number of successfully sent messages to the P2P network: the larger is this number the better is the node and vice-versa.

We use a simple scoring system. The candidate which receives the best score in a criterion will receive a fixed value of 10 points. The second best node has the second better score, 8 points. The rest of scores are 6, 5, 4, 3, 2 and 1 respectively. Furthermore, the scores of the candidates are weighted according to the user’s priority, that is, the user of the application can indicate to the Juxta-CAT which is the most important criterion, the second most important criterion and so on.

Thus, by letting $N$ the number of candidate nodes for the task resolution, $K$ the total number of criteria, $w_i$ the weight or priority of criterion $i$ and $S(i, j)$ the score of the $i$th candidate under criterion $j$, then the total score $S_i$ of the $i$th candidate node is:

$$S_i = \sum_{j=1}^{K} w_i \cdot S(i, j).$$
The best candidate is then the one of maximum score, that is, candidate node \( i_{\text{max}} \) such that

\[
S_{\text{max}} = \max_{1 \leq i \leq N} S_i.
\]

It should be noted that it is very important to keep the historical information updated in the cache of each broker. To achieve this, any node periodically communicates to the brokers its recent statistics. The statistics generated and sent by broker peers are:

- **JXTA statistics** regarding: sent messages, lost messages, current message queue size, average time in message delivery, visible nodes, connections and disconnections to the network.
- **Brokering statistics** regarding: successfully assigned requests, pending requests in the queue, lost requests, average time in allocating tasks to candidate nodes. Also, the historical information of most recent assignments is maintained.

On the other hand, the statistics generated and sent by client peers are:

- **JXTA statistics** regarding: sent messages, lost messages, current message queue size, average time in message delivery, visible nodes, connections and disconnections to the network.
- **Execution statistics** regarding: number of requests accepted, successfully completed tasks, uncompleted tasks, number of pending tasks.
- **Statistics on sent requests** regarding: number of requests sent to the grid, number of requests waiting for the broker response, number of requests waiting for the response of the remote node, number of requests successfully completed, lost or cancelled requests and requests under execution.

In order to publish its statistics, each node generates at a certain interval rate an XML document storing the statistical information indicated above. This document is sent then to the P2P network as an advertisements through the JXTA Discovery Service. This service publishes the advertisement in the local cache of the node and in the cache of the Rendezvous node of the peer group. After that, it is the Rendezvous node who propagates this information to the rest of caches of the grid nodes.

Although this allocation algorithm is simple, it uses relevant information as regards the performance of the network. This model is flexible as regards the number of criteria to be used as well as the weights to be given to these criteria.

### 5. Evaluation of the Juxta-CAT Platform

We have completed a first evaluation of the proposed platform. At this stage of the work, the objective of the evaluation was twofold:

- First, to see the feasibility of the Juxta-CAT as a distributed computing environments regarding scalability, robustness and consistency of the network. In other terms we wanted to evaluate that the network allows broker peers and client peers to join and perform their responsibilities and also end users to submit and solve their tasks using the platform.
- Secondly, to measure the performance of the Juxta-CAT as regards the efficiency of solving problems. We are concerned here mainly with the speedup obtained in solving problems decomposable into independent tasks submitted to the grid. But, also we wanted to measure the time needed by the Juxta-CAT to allocate the tasks to the grid nodes.

We describe next the scenario used for Juxta-CAT testing. It is important to notice that the evaluation process is carried out in a real grid of nodes (see Subsect. 5.2) consisting in a set of geographically distributed real machines.

#### 5.1 Evaluation Scenario

We have chosen a simple application scenario yet having rather reasonable computational cost. This is the computation of \( \Pi \) number using approximation series, more precisely the Gregory’s series:

\[
\Pi = 4 \sum_{i=0}^{\infty} \frac{1}{2i+1}.
\]

This problem is approximated in a set of geographically distributed real machines. We run a simple java program on a local machine (P4 2.4 Ghz 512 Mb RAM) that solves the problem and measured the time for different input sizes (values of \( N \)) given in Table 2.

<table>
<thead>
<tr>
<th>( N )</th>
<th>Result</th>
<th>Computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.0418396189294032</td>
<td>trivial</td>
</tr>
<tr>
<td>( 10^3 )</td>
<td>3.140592653397941</td>
<td>trivial</td>
</tr>
<tr>
<td>( 10^6 )</td>
<td>3.1415916535897744</td>
<td>trivial</td>
</tr>
<tr>
<td>( 10^9 )</td>
<td>3.1415926525880504</td>
<td>2 minutes 9 secs</td>
</tr>
<tr>
<td>2 \cdot ( 10^9 )</td>
<td>3.1415926530880768</td>
<td>4 minutes 16 secs</td>
</tr>
<tr>
<td>3 \cdot ( 10^9 )</td>
<td>3.1415926532549256</td>
<td>6 minutes 36 secs</td>
</tr>
</tbody>
</table>

This problem is efficiently parallelized by splitting\(^1\) the whole series into as many parts as nodes of the grid to be used, sending each part to the nodes and sum up the partial results. Thus in this scenario, we generate as many tasks as number of nodes to be used in computation, submit these tasks to the grid and notify the final result. Thus, we had to basically implement the following classes in Java:

- **Sum.java**: receives in input parameters \( i_{\text{init}} \) and \( i_{\text{final}} \) and computes the sum of fractions comprised between \( i_{\text{init}} \) and \( i_{\text{final}} \). This class will be run by client peers of the grid.
- **Collector.java**: receives in input a text file each line of which is the partial computed by a peer and computes the final result.

\(^1\)One could use a more efficient version known as the classical partial sum problem.
These classes are then packed in jar files. Thus, for instance, an end user can submit the following four requests to compute the approximate value of \( \Pi \) for \( N = 10^9 \) terms:

```java
java -cp samples.jar juxtacat.samples.pi.Pi -params 0 49999999
java -cp samples.jar juxtacat.samples.pi.Pi -params 500000000 99999999
java -cp samples.jar juxtacat.samples.pi.Pi -params 1000000000 1499999999
java -cp samples.jar juxtacat.samples.pi.Pi -params 1500000000 1999999999
```

Note that the efficiency of approximating \( \Pi \) will certainly depend on the efficient allocation of tasks to the nodes of the grid by the broker peers.

For any instance of the problem, the total execution time (from submitting the task to the grid till outputting the final result) is composed of the time needed to allocate tasks to grid nodes, communication time and the proper computation time of the nodes. Thus, in conducting the experiment we measured the speedup of the computing the approximate value of \( \Pi \) in grid as the number of grid nodes used increases and also the time needed by the grid to allocate the tasks as the number of task increases. Speedup is defined in the usual way:

\[
\text{speedup} = \frac{\text{resolution time in the grid}}{\text{nbNodes} \cdot \text{resolution time in the grid using nbNodes}}
\]

where \( T_{seq} \) is the sequential resolution time and \( T_{grid} \) the resolution time in the grid using \( \text{nbNodes} \).

### 5.2 Platform settings

In order to deploy Juxta-CAT in a real grid, we joined the PlanetLab platform [1]. The sample set of PlanetLab’s machines prepared for this analysis is about 20 nodes distributed around the European continent. The following nodes\(^5\) were used to install\(^6\) the CMD Juxta-Cat version (client without GUI).

<table>
<thead>
<tr>
<th>Host</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>planet1.manchester.ac.uk</td>
<td>University of Manchester</td>
</tr>
<tr>
<td>lsirextpc01.epfl.ch</td>
<td>École Fédérale de Lausanne</td>
</tr>
<tr>
<td>planetlab1.polito.it</td>
<td>Politecnico di Torino</td>
</tr>
<tr>
<td>planetlab1.info.ucr.ac.be</td>
<td>University of Louvain</td>
</tr>
<tr>
<td>planetlab2.upc.es</td>
<td>Universitat Politècnica de Catalunya</td>
</tr>
<tr>
<td>planetlab1.sics.se</td>
<td>Swedish Institute of Computer Sci.</td>
</tr>
<tr>
<td>planetlab1.ifi.uio.no</td>
<td>University of Oslo</td>
</tr>
<tr>
<td>planetlab3.upc.es</td>
<td>Universitat Politècnica de Catalunya</td>
</tr>
<tr>
<td>planetlab1.ls.fi.upm.es</td>
<td>Universidad Politécnica de Madrid</td>
</tr>
<tr>
<td>planetlab1.hii.fi</td>
<td>Technology Institute of Helsinki</td>
</tr>
<tr>
<td>planetlab-1.cs.uchicago.edu</td>
<td>University of Chicago</td>
</tr>
<tr>
<td>planetlab1.ru.is</td>
<td>University of Reykjavik</td>
</tr>
<tr>
<td>planetlab2.sics.se</td>
<td>Swedish Institute of Computer Sci.</td>
</tr>
<tr>
<td>planetlab1.mini.pw.edu.pl</td>
<td>Telekomunikacja Polska Warsaw</td>
</tr>
<tr>
<td>planetlab1.cs.uu.se</td>
<td>University of Uppsala</td>
</tr>
<tr>
<td>planetlab-02.ipv6.lip6.fr</td>
<td>Laboratoire d’Informatique de Paris</td>
</tr>
</tbody>
</table>

We have also used a small cluster\(^5\) of 6 machines: nozomi.lsi.upc.edu, which manages the processes and has access\(^2\) to Internet (Celeron 2.5 Ghz, 1 Gb RAM, 2 HD IDE RAID1) and 5 equal nodes (AMD64X2 4.4 Ghz, 4 Gb RAM DDR ECC, 2 HD IDE RAID1) making an independent Gigabit network.

### 5.3 Computational results

Computational results were obtained for the execution scenarios given in Table 4 the experimental setting.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( p )</th>
<th>Number of petitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^7 )</td>
<td>5</td>
<td>5 tests</td>
</tr>
<tr>
<td>( 2 \cdot 10^9 )</td>
<td>5</td>
<td>5 tests</td>
</tr>
<tr>
<td>( 3 \cdot 10^9 )</td>
<td>5</td>
<td>5 tests</td>
</tr>
</tbody>
</table>

For each combination (input size, number of machines) results\(^6\) are averaged over 5 executions; the standard deviation is also presented. We present\(^7\) in Fig. 12 and Fig. 13 the resolution time and speedup for \( N = 10^9 \) and in Fig. 14 and Fig. 15, for \( N = 2 \cdot 10^9 \).

In these figures we can observe:

- Using grid nodes yields a clear reduction in resolution times. However, increasing the number of grid nodes is “justified” as far as the complexity of the problem needs more processors. In our case, using more than 16 grid nodes does not help in reducing the resolution time. In fact, from Fig. 12 we see that for \( N = 10^9 \) this “critical” number of processors is around 8 and increases up to 12 when the input size is doubled (see Fig. 14).

- A similar observation holds for speedup. As the number of grid nodes participating in the resolution of the problem increases, the speedup decreases due the increment in the communication times and the time needed by the grid to allocate tasks to grid nodes. Again, by considering a certain number of grid nodes the speedup increases as the input size of the problem increases. Thus, for \( N = 10^9 \) and 16 grid nodes the speedup is around 0.35 (see Fig. 13) and when the input size is doubled the speedup is around 0.45 (see Fig. 15) and increases up to 0.6 for \( N = 3 \cdot 10^9 \).

We also measured the time needed by the grid system to allocate the requests, that is, the brokerking time. We give in Fig. 16 the slowest and averaged brokerking time for different numbers of hosts. As can bee seen from this figure, there is a reasonable increase (which seem to be proportional to the increase in the number of grid nodes) in the average brokerking time.

---

\(^2\)In general, just one request is submitted.

\(^3\)PlanetLab node satisfy the following minimum requirements: CPU of 2.0 Mhz, 1024 Mb RAM.

\(^4\)We remark that we have access to PlanetLab nodes only through SSH connections on text-mode.

\(^5\)At the Department of Languages and Informatics Systems, Polytechnic University of Catalonia, Spain

\(^6\)See also the documentation section related to the JuxtaCAT at https://juxtacat.dev.java.net

\(^7\)We omit the figures for \( N = 3 \cdot 10^9 \).
Figure 12: Resolution time for input size $N = 10^9$.

Figure 14: Resolution time for input size $N = 2 \cdot 10^9$.

Figure 13: Speedup for input size $N = 10^9$ and different numbers of grid nodes.

Figure 15: Speedup for input size $N = 2 \cdot 10^9$ and different numbers of grid nodes.
6. CONCLUSIONS AND FURTHER WORK

In this work we have presented the Juxta-CAT platform developed using JXTA protocols and Java language. Juxta-CAT offers a P2P system for distributed computing. Juxta-CAT can be used from users to submit their tasks in the form of Java programs stored on signed jar files and benefit from the large computing power offered by this distributed platform. Users can join Juxta-CAT either as a simple contributor with their machine(s) or as a client peer. The development of our platform required certain improvement/extensions to the original JXTA protocols, especially as regards the Presence Service and Pipe Service protocols. The architecture of Juxta-CAT relays on two types of peers: broker peers that are in charge of managing the allocation of tasks to grid nodes and client peers.

The Juxta-CAT environment has been deployed in a large-scale, distributed and heterogeneous P2P network that we obtained by joining the PlanetLab platform. We have used some simple scenarios to validate and evaluate the proposed platform. The experimental study show the feasibility of our approach and its usefulness in using the Juxta-CAT to speedup task resolution. Moreover, the Juxta-CAT’s architecture does not suppose any obstacle for the “gridification” process of task resolution.

We plan to continue the work on Juxta-CAT. On the one hand, we would like to add new functionalities and improvements. Some of these would comprise:

- Allow the resolutions of problems written in languages other than Java; currently, we can only send statements of problems written in Java language.
- Develop a remote launching system for Juxta-CAT’s brokers and clients. Due to security restrictions on PlanetLab nodes, we have to launch each peer by connecting first to the remote machine using SSH.
- Provide new and more powerful economic-based models for allocation of tasks to nodes.

On the other hand, we plan to use the current version of Juxta-CAT to develop more realistic applications. Thus we are envisaging the development of an application to process large log-files of daily activity of students at a Virtual University.

Juxta-CAT is included in one of the best showcases of the Java developer’s community, where new interested members who would like to participate can be registered.

Finally, there are other issues related to Juxta-CAT such as security issues not mentioned in this paper. The reader is referred to https://juxtacat.dev.java.net for further details.

Acknowledgements

We would like to thank many participants of the Java.net and Jxta.org for their help. In particular, we are grateful to Daniel Brookshier for his support during the development of Juxta-CAT project.

7. REFERENCES