Jalapeno – Decentralized Grid Computing using Peer-to-Peer Technology

Niklas Therning labe@ce.chalmers.se
niklas@therning.org

Department of Computer Science and Engineering
Computer Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
+46 31 7728441

ABSTRACT
This paper presents the Jalapeno grid computing system. Jalapeno is implemented in Java and uses peer-to-peer technology provided by Project JXTA. The Jalapeno system consists of manager, worker and task submitter hosts. The task submitter submits a collection of tasks, a task bundle, to be processed by the system to a randomly chosen manager. The manager splits the bundle into a set of new, smaller bundles which are forwarded to equally many, randomly chosen, managers which repeat the process. Each manager has a small number of connected workers. During task bundle splitting the manager may, depending on its current load, reserve a number of tasks for its workers. Workers return the results to their managers which forward them to the task submitter.

The system is self configuring: hosts volunteering their computing power will at first become workers only but will eventually become managers if they can not connect to another manager within a certain time.

The major contributions of this project are: an implicit hierarchy of hosts which changes randomly over time and requires no effort to maintain, a framework for applications solving embarrassingly parallel type of problems which automatically partitions the problem into smaller sub-problems and ease of use through the use of Sun's Java Web Start technology.

Two applications have been developed for the system to evaluate its performance: an RC5 key cracking application and a 3d ray-tracing application. The entire system is available for download at http://jalapeno.therning.org.

Categories and Subject Descriptors
C.0 [General]: system architectures

General Terms

Keywords
GRID, P2P, distributed computing.

1. INTRODUCTION

Computers, whether they are office workstations or home PCs, are becoming more and more powerful but are still mainly used for tasks such as word processing and Internet browsing, leaving their CPUs idle most of the time. Connecting these mostly idle computers in a network creates a grid computing system which makes it possible to utilize the otherwise wasted CPU cycles to solve problems in a distributed manner. Grid computing systems are especially suitable for solving extremely large problems which can be split into a set of independent tasks. This class of problems is very coarse-grained, requiring almost no communication between tasks. Problems of this kind are also referred to as embarrassingly parallel problems.

Throughout the literature a number of different names have been used for the different roles the hosts in a grid computing system may have. In this paper hosts volunteering their idle CPU cycles will be referred to as workers while hosts submitting tasks to the system will be referred to as task submitters. Finally, hosts receiving tasks from task submitters and distributing tasks to idle workers will be referred to as managers.

The goal of this project was to develop a decentralized grid computing system based on peer-to-peer technology. The term peer-to-peer (or P2P) refers to networks not having a fixed number of clients and servers, but a number of peer hosts that function as both clients and servers to the other hosts on the network.

A small application running on every host of the system to evaluate its performance: an RC5 key cracking application and a 3d ray-tracing application. The entire system is available for download at http://jalapeno.therning.org.

Heterogeneity. The network will most likely consist of machines of different platforms and architectures.

Scalability. The system is scalable from the size of a small LAN up to the size of the Internet.

Fault-tolerance. Hosts are expected to be joining and leaving the system at any time thus the system has to be able to deal with host failures.
Security. Tasks are executed in a sand-box environment at the host protecting the host’s private data and resources.

Anonymity. The task submitter does not need to have login access to individual hosts.

1.1 Project JXTA

The reference implementation developed by Project JXTA [1] provides Jalapeno with peer-to-peer communication abilities. Project JXTA, pronounced juxtapose or juxtapa, defines a set of XML-based protocols allowing any network connected device to communicate in a peer-to-peer manner. JXTA hosts create a virtual network where any host may communicate directly with any other host regardless of firewalls and NAT devices.

The basic building blocks of a JXTA network are peers and peer groups. Peers are the individual hosts comprising the network and are grouped into peer groups.

Peers use pipes, which may be of unicast, bi-directional or broadcast type, to communicate with each other. By having many pairs in a peer group listening on the same pipe redundancy may be achieved.

The JXTA protocols define how peers discover other peers or network resources, how network resources are advertised and how messages, used by peers to communicate, are routed. When a peer joins a peer group it automatically seeks a “rendezvous” peer for that group or dynamically becomes one if none is found. A rendezvous peer is like any other peer but also forwards discovery requests to help other peers discover resources.

For a peer behind a NAT device to be reachable by peers outside the private LAN it has to register with a “relay” peer. The relay peer temporarily stores messages destined for the peer which, periodically, connects to the relay peer and downloads any new messages.

The project JXTA web site [1] provides Java (J2SE and J2ME) implementations of the protocols as well as C, Perl and PocketPC implementations. The Jalapeno system uses the J2SE implementation.

1.2 Paper Organization

The remainder of this paper is organized as follows. Section 2 outlines the architecture of the Jalapeno system. In Section 3 the performance results of running a number of different distributed applications on the system is presented. Section 4 discusses related work while Section 5 gives directions for future work. Finally Section 6 presents the most significant contributions.

2. THE JALAPENO GRID COMPUTING SYSTEM

The Jalapeno system consists of the following JXTA peers: manager peers, worker peers and task submitter peers. Each host may have one or more of these roles. Figure 1 shows the structure of a Jalapeno network. To achieve high scalability the system consists of many managers, each managing a small set of workers (<100). Each manager forms a peer group which its workers have to join. Within this worker peer group workers may communicate directly with other workers or the manager.

Initially every host starts a worker peer only which starts to search for available manager peers. When a manager is found the worker tries to connect to it. Managers may only have a limited number of connected workers. Accepted workers will join the worker group created by the manager and start executing tasks.

If a worker is unable to connect to a manager within a certain time it will start a new manager peer on the local host and connect to it. The first host will automatically become a manager after some time and start to accept worker peers. New groups of workers will appear spontaneously as new hosts join the network. When a manager becomes unavailable its workers will either find other managers to connect to or become managers themselves and start accepting workers.

2.1 Task Submission

To solve a problem the task submitter submits a task bundle, a collection of tasks, to a randomly chosen manager as in Figure 2. The manager splits the received bundle into a number of smaller bundles. Managers keep a limited set of bundles from which tasks are extracted and handed to the connected workers. If the set of current bundles is not full the manager will reserve one bundle during the splitting process to be executed by its workers. The rest of the bundles will be forwarded to a number of other, randomly chosen, managers which will repeat the process. Bundles which can not be executed nor forwarded are returned to the task submitter. All bundles submitted to the system include a unique id which the managers use to identify previously received bundles. Already processed bundles will be rejected to prevent loops during the forwarding process.

When a worker finishes a task it will return the result to its manager which will forward the result to the task submitter.

Each host runs a monitor peer which is used to provide network and system status information. The Jalapeno host application will automatically start a monitor instance on startup. If a manager has been started the local monitor will query the manager periodically for its current status and send that information to all other monitors in the system.

All Java byte-code required to solve a particular problem is packaged in JAR-files and uploaded by the task submitter to the first manager as part of the task submission process. Managers forwarding task bundles will also forward the JAR-files if necessary to other managers. Workers download the JAR-files from their manager when needed. All JAR-files are cached on each host speeding up future task and task bundle distribution. Task and task bundle object instances implement the Task and TaskBundle interfaces and are distributed as serialized streams of bytes.

2.2 Framework for Embarrassingly Parallel Problems

The developed framework for embarrassingly parallel type of problems features automatic task bundle splitting and task distribution among workers, resubmission of bundles which can not be handled by any manager and load balancing through work stealing scheduling. Whenever a worker in a system using a work stealing scheduling algorithm runs out of tasks it will steal tasks from other workers. Work stealing is implemented by letting the manager assign already assigned tasks to idle workers when there are no more unassigned tasks available. By having faster workers stealing tasks from slower workers load balancing is achieved. Fault tolerance is a consequence of the work stealing process because a dysfunctional worker is nothing but an infinitely slow worker. Work stealing has been used also in other grid computing systems.

One restriction is that the total number of entities (e.g. keys in a brute-force encryption key search problem) to process can not be infinite. The developer describes the solution space (its dimensionality and each dimension’s limits) and the size in that space of each work unit given to workers. E. g. a brute-force search for the correct key to decrypt an encrypted message would have a one-dimensional space with the total range of keys as limits.

The framework automatically creates new bundles to be submitted and keeps track of finished tasks and bundles. The first bundle
to be submitted encapsulates the entire solution space. Upon splitting, the space encapsulated by the bundle is split into a number of sub-spaces. The sub-spaces are used to create equally many new bundles which are forwarded to other managers. If a manager accepts tasks a small number of tasks, equal to the number of workers, each encapsulating a sub-space will be reserved by that manager.

Bundles failing forwarding will be returned to the task submitter and the framework which will try to merge as many as possible of the sub-spaces of the returned bundles and then, after some time, submit a new bundle containing the merged spaces. The task submission process implemented by the framework is illustrated in Figure 3.

The framework implements a work stealing type of scheduling algorithm as described above to distribute tasks among a manager’s workers. It maintains a list of those tasks which have been distributed to workers and are currently being executed. When no more new tasks are available for distribution to idle workers tasks in this list will be used instead.

2.3 Java Web Start

The Jalapeno host application is available as a Java Web Start application at http://jalapeno.therning.org. Java Web Start, developed by Sun, provides a platform-independent, secure, and robust deployment technology. By making the application available on a standard web server developers are able to distribute their work to end-users. Using any web browser, end-users can launch the application. Java Web Start ensures they always have the most recent version of the application.

In a sense Java Web Start is similar to the Java applet technology. The most significant difference is that Java Web Start allows the deployment of real Java applications. By utilizing the Java Web Start technology the Jalapeno system achieves the same level of availability and ease of installation as the applet based systems presented in Section 2 without suffering from the restrictions put on such a system.

3. EXPERIMENTAL RESULTS

To determine the potential performance of the Jalapeno system a number of initial experiments were conducted. These will later be followed by extended studies. A homogeneous collection of 8 workstations were used to form the group of workers. In the first experiment presented in Section 3.1 the workstations were all equipped with Intel Pentium 4 1.8 GHz processors and 512 MB of RAM running the FreeBSD 4.8 operating system and Sun’s J2SE 1.4.1 version of Java. In the second experiment presented in Section 3.2 the workstations were all equipped with Sun UltraSPARC 440 MHz processors and 256 MB of RAM running the SunOS 5.9 operating system and Sun’s J2SE 1.4.2 version of Java. Except for the Jalapeno host application no other user applications were running on the workstations.

A dedicated manager (i.e. not running a worker) was used accepting as many as 32 workers in total. To show the heterogeneity of the Jalapeno system a Sun UltraSPARC machine running SunOS 5.8 and Sun’s J2SE 1.4.2 was chosen as manager while an Intel Pentium III 700 MHz machine running Linux and Sun’s J2SE 1.4.2 was chosen as task submitter.

3.1 Running an RC5 Key Cracking Application

To determine the scalability of the system running an embarrassingly parallel problem an RC5 key cracking application was used. The tasks distributed to the workers each contained a sub-space of 54,000,000 keys from the total key space which would take 1-2 minutes for a worker to finish. The first key to start the search was chosen to make the correct key end up in the middle of the 32nd sub-space, i.e. a single worker would need to search 31.5 sub-spaces before finding the correct key.

The time needed to find the correct key when using \( n \) workers was measured and compared with the time \( T_{seq} \) needed by the sequential version of the application. Figure 4 shows the measured speedup \( (T_{seq}/T_n) \) plotted against the ideal situation. Table 1 lists the measured running times.

The scalability, as seen in Figure 4, is very close to the ideal
situation. The obtained speedup was as expected since the problem solved by the RC5 application is very coarse-grained; it is highly computationally intensive while having low bandwidth requirements.

The scalability would probably have been slightly better if the task submitter had been on the same LAN as the manager. In the used setup the task submitter needed approximately one minute to discover the manager and connect to it to submit the initial bundle. Having the two peers on the same LAN would have significantly lowered the discovery time decreasing its contribution to the measured speedup. Also, if the number of keys to search before finding the correct one would have been larger the experiments would have needed more time to finish giving even more precise speedup figures.

3.2 Running a Ray-Tracing Application

A ray-tracing application has also been developed for the system. In this experiment a 3d scene was rendered resulting in a 640 by 480 pixel image. The tasks distributed to the workers each rendered a sub-image of 50 by 50 pixels. The time was measured from the startup of the application until the last sub-image had been finished. Figure 5 shows the measured speedup ($T_n / T_{eq}$) plotted against the ideal situation. Table 2 lists the measured running times.

Table 2: The running times in seconds, ($1$ worker) and $T_n$ ($n = 2, 4, 8$), needed to finish the rendering and the resulting speedup figures.

<table>
<thead>
<tr>
<th>Workers</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2133.0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1195.7</td>
<td>1.78</td>
</tr>
<tr>
<td>4</td>
<td>625.7</td>
<td>3.41</td>
</tr>
<tr>
<td>8</td>
<td>379.6</td>
<td>5.62</td>
</tr>
</tbody>
</table>

The scalability, as seen in Figure 5, deviates more from the ideal than in the previous experiment. In this experiment the computation to communication ratio is much lower compared to the RC5 application. The amount of data in the results is a lot larger (~ 10 kB compared to a single boolean value) while the time to finish a task is lower (~ 20 s compared to ~ 2 min). Having more complex scenes and larger sub-images would increase the speedup figures.
Figure 3: The task submission process implemented by the framework. Initially, the entire space of 1000 entities is submitted as shown in step a. The system returns four sub-spaces which could not be processed. In step b the remaining sub-spaces have been merged into two sub-spaces which are submitted. Two sub-spaces are returned by the system. Finally, in step c, the last sub-space is submitted and processed by the system. All sub-spaces have now been completed.

4. RELATED WORK

The Cilk [3] runtime system is important, even though it predates Java, since it introduced the work stealing scheduling algorithm which also provides a degree of fault tolerance.

Charlotte [4] was one of the first Java based systems. The Charlotte system used Java applets running in Java-capable web browsers to execute parallel tasks. SuperWeb [5], Javelin [6], Bayanihan [7] and Popcorn [8] are all other examples of grid computing systems based on Java applets running in Java-capable web browsers.

Because of their use of Java applets these systems are essentially ubiquitous: the only requirement on the participants is a Java-capable web browser. No software installation is necessary and participating is as easy as visiting a web site. However, being applet based has its drawbacks. Applets run in a very restrictive runtime environment: they may only open outbound TCP connections to the web server hosting the applet. This limitation severely limits the scalability of the system since all communication between workers must be routed through the web server. This one server may become the single point of failure for the entire system.

Javelin++ [9] and Javelin 2.0 [10] extend the work originated in Javelin by moving from Java applets to Java applications. The system architecture of Javelin++ and Javelin 2.0 was similar to that of Javelin but included a hierarchy of managers instead of a single manager (the web server) to improve scalability. Some of the people behind the Javelin projects started the CX [11] project, which seems to be in active development. In CX the hierarchy of managers is strictly maintained in a so-called sibling-connected height-balanced fat tree which provides a very robust network topology tolerating a large number of host failures.

JNGI [12, 13] is a more recent system based on peer-to-peer technology. JNGI uses the previously described reference implementation of the JXTA protocols developed by Project JXTA [1, 14] to communicate in a peer-to-peer fashion. JNGI and Jalapeno share some similarities. The managers of Jalapeno are almost identical to the task dispatchers of JNGI but do not provide the redundancy provided by the JNGI task dispatcher groups. In the early designs of the Jalapeno system managers were grouped into manager groups identical to the task dispatcher groups of JNGI and exchanged results periodically. However, this idea was abandoned because such a mechanism would not be suitable for all types of applications. E.g. in a ray-tracing application the partial results exchanged between managers could be very large in size requiring a substantial amount of communication bandwidth.

The most fundamental difference between the two systems is the task submission process. In Jalapeno a collection of tasks is submitted. The tasks are then spread throughout the system without requiring any action from the task submitter. In JNGI the task sub-
mutter first has to request access to a worker group and then send the collection of tasks to a task dispatcher in the worker group. JNGI does not provide the ability to distribute a large number of tasks to many worker groups at once.

Jalapeno’s framework for embarrassingly parallel type of problems provides automatic solution space partitioning. Developers need only to define the dimensionality of the solution space and how large sub-spaces finally processed by workers should be. JNGI is restricted to solving embarrassingly parallel type of problems but still does not provide developers with this kind of facility.

SETI@home [15] and distributed.net [16] are perhaps the two most famous existing projects which utilize idle Internet-connected computers to solve problems too hard for any desktop computer to solve on its own. A number of other similar projects have followed. All of these projects have a centralized server architecture (though not necessarily a single server), are limited to a single very coarse-grained problem and rely on the good-will of participants volunteering their computers’ idle cycles.

5. FUTURE WORK

There are a number of possible enhancements that would improve the current system:

Load balancing There is currently no way of adapting the size of tasks to suit individual workers’ current performance and bandwidth. The work units distributed to workers by the framework for embarrassingly parallel problems are equal in size. The size is specified by the developer. Also, during task submission, managers “closer” (e.g. higher bandwidth) to the task submitter should be preferred over managers farther away.

Security One aspect of security is the protection of proprietary application code. It is often easy to retrieve the source code from Java class files using Java byte-code decompilers. In [7] a number of techniques for protecting private Java classes are presented.

Another aspect of security is the verification of results preventing sabotage and cheating. A sophisticated sabotage prevention technique which tries to minimize the number of extra tasks processed while still providing a high degree of sabotage-tolerance is described in [17].

Manager redundancy JNGI’s manager (task dispatcher) redundancy was considered in the early designs of the Jalapeno system but was later abandoned. If implemented there must be a mechanism for developers to specify if a particular application benefits from it. Managers must also decide not to exchange results if the amount of data is too large.

Message passing The Jalapeno system has been designed to allow direct message passing between workers connected to a manager but message passing has not yet been implemented in the prototype system. By adding message passing Jalapeno could be suitable for a larger number of, less coarse-grained, problems.

6. CONCLUSIONS

The Jalapeno system has a number of unique features not seen before in grid systems. The most important contributions are:

Automatic solution space partitioning. The automatic task bundle splitting, implemented by the framework for embarrassingly parallel problems, frees the developer from the complicated task of partitioning the solution space, distributing the sub-spaces and keeping track of finished sub-spaces.

Implicit hierarchy. To achieve higher scalability, robustness and fault-tolerance many of the other projects presented in Chapter 4 use some kind of hierarchy of managers. However, maintaining the hierarchy is, in most cases, complicated to implement. Jalapeno has an implicit hierarchy of managers which only exists during task submission and changes randomly over time with every task submission. Nothing is required by the system to maintain this hierarchy.

Ease of use by using Java Web Start. By utilizing the Java Web Start technology participating in the network is as easy as visiting a web site. This technology has never been used for this purpose before (at least not to the authors’ knowledge). The Jalapeno host application is of course available as an ordinary application as well.

Furthermore the requirements presented in Section 4 are fulfilled by the system:

Heterogeneity. Heterogeneity is a side-effect of the choice of implementing the system using pure Java which is designed to be platform independent.

Scalability. The system scales well for a small number of hosts. The scalability when using a large number of hosts on a global scale is yet to be investigated.

Fault-tolerance. The system provides fault-tolerance through its self-configurability. Workers leaving the system will be replaced by new workers. The framework for embarrassingly parallel problems ensures that tasks will be reassigned to other workers when a worker abruptly leaves the system without finishing its current task. Managers abruptly leaving the system will in time be replaced by new ones.

Security. Security is another side-effect of the choice of implementing the system using Java. The same security mechanisms used to deny Java applets access to the local host’s file system or network interfaces is used in Jalapeno to protect against malicious tasks. This support for a strong security model is built into the core Java classes.

Anonymity. JXTA provides the required anonymity. Peers may join the system regardless of their identity and may submit tasks even if they do not have user accounts on the machines running the worker peers.

7. REFERENCES


