Computation in Peer-to-Peer Networks

Lichun Ji
Department of Computer Science
University of Saskatchewan, Canada
lij589@mail.usask.ca

ABSTRACT

Peer-to-Peer (P2P) networks are the latest addition to the universe of distributed systems. Emphasizing decentralization and self-organization the P2P networks tend to be more robust and scalable than other forms of distributed systems. Especially in the domain of file-swapping P2P networks seem to outperform other approaches largely due to the anonymity of the participants in the peer-network, low network costs and the inexpensive disk-space. Trying to apply P2P principles in the area of distributed computation was significantly less successful. The much referred SETI@home system and its look-alikes have a far smaller reach and seem to be very limited to single-process multiple data problems.

This paper focuses on the issues and existing approaches for using the idle resources in an enterprise network. Using the results of an ongoing study on the amount and quality of idle resources in a local network it explores the key issues in using peer resources for distributed computation.

General Terms

P2P, Distributed Computation

Keywords

Awareness in P2P, Computation in P2P

1. Motivation

The continuing growth in processing power, memory and bandwidth - which have doubled in the past every 18, 12 and 9 months respectively - has dramatically changed desktop computing. While the desktops of the 90's were resource-sparse devices barely capable of supporting more than one desktop application a time, today's desktops are resource-rich devices that hardly every reach high usage of their resources.

With the pervasive deployment of desktop machines and increasingly powerful handheld devices, an ever-growing pool of resources is emerging. Currently there're over 400 million computers worldwide of which the majority is either idle or underutilized.

This paper investigates the use of Peer-to-Peer computing as a possible approach for utilizing the idle resources within an enterprise for distributed computation.

The paper is organized as follows. Section 2 provides an overview and introduction in P2P networks and is followed by a section reporting the results of a study on the P2P potential. The specific problems of using P2P networks for distributed computing are discussed in section 4. An overview of existing systems to support P2P style computing is given in section 5. The paper continues with an experiment on using awareness as a means to achieve better performance and concludes with a summary and outlook on future research.

2. Peer-to-Peer Systems

Peer-to-Peer (P2P) refers to a class of systems and/or applications that use distributed resources in a decentralized and autonomous manner to achieve a goal e.g. perform a computation. The members of the P2P network (called peers) are always in *total control* of their *local* resources and can therefore choose to impose or change policies regarding their use, which makes this approach different from other distributed computing approaches e.g. the client-server model.

Another important aspect of P2P networks is that the roles of the peers in the network are dynamic and emerge. Rather than having *static* and *predefined* roles for the participants like in the client-server model, P2P networks rely on *emerging* and *dynamic* roles as a result of an ongoing self-organization.

2.1 Self-Organization & Awareness

Self-Organization is a key concept in P2P systems since roles are not predefined and have therefore to emerge. Most often P2P networks achieve self-organization as a result of *self-awareness* of peers. An example for such self-awareness is the Gnutella 0.6 Protocol that allows peers to "volunteer" as ultra-node based on perceived network bandwidth, local resources and user behavior.

In addition to changing their roles in the network based on self-awareness peers can evaluate other peers and adapt their functional dependencies accordingly. By storing the results of evaluations in form of persistent models a peer achieves *peer-awareness* enabling it to avoid unpromising peers in the future.

While self-awareness and peer-awareness are essential to all P2P networks distributed computing also requires a *task-awareness*. Peers need to be able to analyze and express the requirements of tasks as well as the consequences if they accept or reject a task to avoid ripple effects in the networks.

2.2 Classification of P2P

The functionalities and application domains of P2P networks lead to four main categories [28]:

- **File sharing** seems to be the most successful application for P2P networks. The basic idea of file sharing is to use the idle disk space for storage and the available network bandwidth for search and download. Napster, Gnutella, Freenet, FastTrack [2, 3, 4, 9] is just a few of this fastest growing segment of P2P technology.
- **Collaboration systems** allow application-level collaboration between users. This includes real-time exchange of message (Project Jabber [33]), online game/gambling (Zoogi [34]), etc.
- **P2P platform** like JXTA [5] try to support the developers of P2P applications by offering a wide range of libraries and services and like request routing, peer discovery and peer communication.
- **Distributed computing** (in P2P) tries to harvest unused processing cycles of computers in the network and to delegate and migrate tasks. The SETI@Home project [10], which tries to use the idle resources of participating peers for its search for extraterrestrial intelligence, is the most often used example of a successful distributed computing application.

Since this paper's focus is on the use of P2P networks for *distributed computing* the following discussions of issues, approaches and systems will be limited to this single issue ignoring other P2P aspects.

3. Potential of P2P Computing

While it is fairly common knowledge that many of the deployed computers (e.g. desktops, workstations etc) are underutilized it is difficult to obtain exact numbers. Organizations and individuals tend to be reluctant to publish the degree of underutilization of their machines due to fear of negative consequences.

We therefore developed a resource-monitoring program for the popular Windows platforms (using Microsoft DotNet) that records in 10-second intervals the current list of processes and the current usage of memory, processor and network. The resource-monitor which runs as a non-invasive system-tray process writes the data into a text-file on a local drive to avoid remote monitoring which would limit the cooperation of potential participants.

3.1 Setup of the Experiment

This experiment was performed using some of the desktop machines in the lab for mobile and ubiquitous computing (MADMUC [31]). A total of 13 machines were involved consisting of four pool machine that are shared by all researchers, seven machines assigned to individual graduate students and two machines used by faculty members.

The tests were conducted over a period of 7 days (Feb 25th – March 4th) and a period of 2 days (March 26h – March 27th). The results of both tests were very similar in terms of the resource usage. Due to space limitations only the results of the first test will be discussed.

3.2 Results

The used data set is 69 MB large and consists of approximately 1,572,480 data points. Analyzing the large data set indicates:

1. The daily resource usage of available resources is in average below 12% in [Table 1]. Especially the network usage is with 7% relatively low.

Resource	CPU	Memory	Network
Average Usage	10.07%	11.32%	3.61%
Peak Usage	15.14%	12.64%	6.61%

Table 1: Daily Resources Usage

2. The usage of overall resources is medial with little fluctuations. [Figure 3.1]

In the graphs of this section CPU is represented as continuous lines, memory as dotted lines and network is represented as dashed lines.

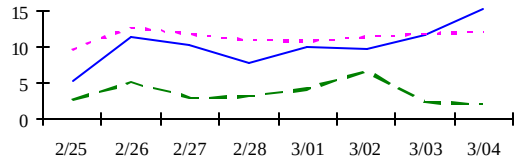

Figure 3.1: Weekly Resource Usage (%)

3. The peak usage of resources of individual computers varies significantly on per day usage [Figure 3.2].

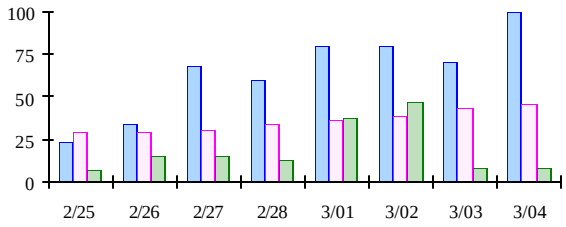

Figure 3.2: Peak Usage (%) for Individual Computer

Columns: 1st CPU, 2nd Memory, 3rd Network

4. The daily resource usage varies from resource to resource [Figure 3.3]. Below is the data for a Thursday (workday) and Saturday (free) is shown.

3.3 Impact on P2P Computation

The above data indicates that there is enough CPU, memory and most importantly network capacity available to support compute-intensive P2P applications without impacting the normal use of the machines.

- Daily resource usage of overall resources is relatively low, allowing the deployment of resource sharing networks. The surprisingly low usage of available network capacity (7% in average) ensures that inter-peer communication will not impact user applications.
- The medial overall resource usage indicates that the environment is not very dynamic and seems to follow predictable patterns of usage. However the test period is small and further tests will be required to confirm this assumption.
- The peaks in resource usages indicate that resources have to be considered volatile, enforcing the use of redundant execution, check-pointing and maybe even transaction management middleware.

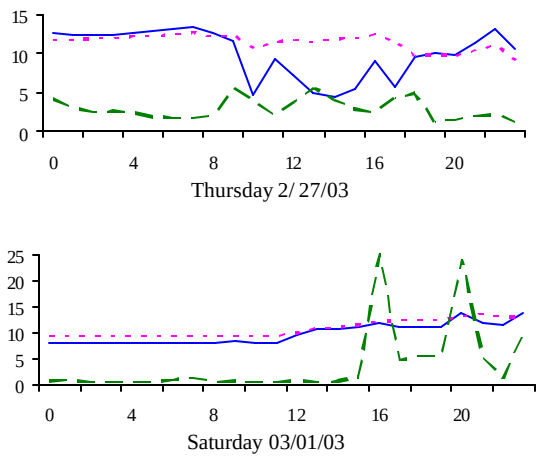

Figure 3.3: Daily Average Usage (%)

4. Computing in P2P Networks

While file-sharing has been a well researched area in P2P networks little work has focused on the issues of distributed computing in P2P networks. In this section the issues using a P2P network for computation are discussed.

4.1 Tasks, Concurrency and Mapping

Dividing a complex computation into smaller computation units (tasks) and *assigning* them to different processors for parallel execution are important issues in *parallel and/or distributed computing* [11]. Determining the tasks and distributing them is pinnacle for obtaining high performance and insure that the number of concurrently executed tasks is maximized so that the complete overall compute time is minimized.

Tasks, here, refer to program-defined units of computation into which the main computation is divided by means of *decomposition*. There're two main types of tasks, *independent* and *dependent* tasks. A set of independent tasks can be executed in any sequence, however most often the tasks are dependent and rely on the data produced by other tasks and thus may need to wait for these tasks to finish execution.

4.1.1 Task Decomposition Techniques

Decomposition is the process of dividing a complex task or computation into smaller parts, some of all of which may potentially be executed in parallel. Decomposition identifies elements that can be executed concurrently and divides a complex task into sub-tasks that can be executed in parallel. The large number of existing decomposition techniques can be classified into categories [11]:

- **Recursive decomposition.** A method divides problems into a set of independent sub-problems and each one of these sub-problems is solved by recursion.
- **Data-decomposition.** A method for partitioning a large data structure on which the computation is performed. This data partitioning is used to induce a partitioning of the computation into sub-tasks.
- **Special purpose decompositions.** These methods apply to specific classed of problems, such as *exploratory decomposition* for problems that search of a large problem space, and *speculative decomposition* for output-oriented problems.
- **Hybrid Decompositions.** Since the decommission techniques are not mutually exclusive they can be combined to achieve better results. A combination of different decomposition techniques is called hybrid decomposition.

The selection of the appropriate decomposition technique is of course highly application specific and will be therefore not be discussed in this paper.

4.1.2 Granularity and Task interaction

The number and size of tasks into which a problem is decomposed determines the *granularity* of the decomposition. From this perspective, a decomposition process can be either *fine-grained* - if the problem can be dived into a large number of small tasks - or *coarse-grained* otherwise. A concept related to granularity is that of the degree of concurrency. The potential for concurrent execution usually increases as the granularity of tasks becomes smaller (finer) and their numbers increase.

The sub-tasks that result from a decomposition often share input, output or intermediate data. The interaction among tasks running on different physical processors limits the ability to obtain unbounded speedup (ratio of serial to parallel execution time) for computation.

While fine-grained decomposition is highly desirable since it allows for a better distribution over the available nodes and therefore a better utilization of computing resources the network bandwidth and latency in P2P networks often enforce a more coarse-grained decomposition.

4.1.3 Mapping

Mapping is the mechanism by which tasks are assigned to processes/processors for execution. A mapping algorithm should seek to maximize the concurrency of tasks by:

- *Distributing independent tasks* onto different processes/processors.
- *Minimizing the total completion* time by ensuring that processes/processors are available to execute the tasks on the critical path as soon as such a task becomes executable
- *Minimizing the interaction* among processes/processors by mapping tasks with a high degree of mutual interaction onto the same process/processor.

It is important to note that these objectives often conflict with each other. For example, minimizing the inter-process interaction will enforce multiple tasks onto a single process resulting in poor balancing of load.

Mapping techniques used in parallel algorithm can be classified into two categories [11]: static and dynamic. *Static mapping techniques* distribute the tasks among processes prior to the execution of the algorithm. Static mapping is used in conjunction with a decomposition based on data partitioning. *Dynamic mapping techniques* distribute the tasks among processes during the execution of the algorithm. Dynamic mapping is necessary in situations where a static mapping may result in a highly imbalanced distribution of work among processes or where the dependency of task is dynamic and unknown at the beginning. Dynamic mapping techniques are usually classified as either centralized or distributed.

4.2 Resource Allocation in P2P

A P2P network consists of autonomous peers that can choose to leave or join the network without notice. Consequently P2P networks are very dynamic and the individual peer resources have to be considered highly volatile. In addition the dynamics of the P2P network make the process of locating resources difficult.

Three basic strategies [1] for resource allocation in P2P networks have emerged:

- **Centralized Service Model.** The peer connects to a centralized directory server, which stores all information regarding location and usage of resources. While resulting in a single point of failure this design has the advantage of simplicity and good performance (e.g. Napster [2]).
- **Flooding Model.** The flooding model avoids a central point of failure by using an unstructured distributing approach. Since no single peer knows about all resources, peers in need for resources flood the network with requests. Using forwarding and range delimiting actions (e.g. TimeToLive, TTL) the reach and network impact can be controlled (e.g. Gnutella 0.4 & 0.6).
- **Routing Model.** The routing model adds structure to the way information about resources is stored using distributed hash tables. Hash tables significantly improve lookup times (e.g. FreeNet [4] and Pastry [14]) at the expense of the autonomy of peers.

4.3 Volatile Resources

In a P2P network, the autonomous peers are free to join and leave the network at any time. Consequently the overall topology of a P2P network is unpredictable as the set of nodes that makes up the network varies over time. Since P2P networks are resource-sharing systems the resources of the system have to be considered volatile. This raises the following problems:

- **Reliability.** Due to the unconstraint joins/departures of peers, the average reliability of any single node is low. Ensuring the reliability of services and resources is a major challenge. The reliability of P2P network can be defined in two dimensions: *vertical reliability* (ensuring that a peers local resources are reliable) and *horizontal reliability* (addressing multi-peer reliability operations).
- **Fault resilience.** Potential changes in the availability of peers require support for automated fault recovery and task resuming. This can be achieved by redundant execution, use of checkpoints and task migration.
- **Performance.** Performance and fault-resilience are two primary aspects when evaluating a system's Quality-of-Service (QoS). In a P2P system, *Quality-of-Service* refers to the capability to ensure certain service parameters.

4.4 Executing Code

Peers in P2P network often have to interact with unknown or unfamiliar peers and need to manage the risk involved with the interactions (transactions) without any presence of trusted third parties or trust authorities. P2P networks have therefore unique requirements when implementing the models for identifying, authenticating and authorizing users and applications across a widely distributed and changing network. Security threats in executing third-party code can be viewed from two perspectives:

- Resource *user –Hostile Host Threat*
  The host can't be trusted to execute the code.
- Resource *provider – Hostile Code Threat*
  The code that should be executed can't be trusted.

To address these problems it necessary is to use encryption, authentication and execution of code in secure environments e.g. sandboxes. In addition information such as feedback about past experiences can help in making decisions. Reputation systems [29] provide another way for building trust through social control without trusted third parties.

5. Case Studies

Utilizing the idle resources is by now means a new idea. Over time many different approaches have been developed. In the following four well established approaches will be discussed namely the now classical *Condor* system which is one of the earliest systems to harvest the unused resources, the *Avaki* system as an example of the currently emerging grid-oriented approaches, the famous SETI@Home network and the JXTA platform.

5.1 Evaluation Criteria

To evaluate the above-mentioned systems the following criteria will be used.

Supported Computing Model

In this paper three different computing models for P2P computing will be considered.

- **Single-Program-Multiple-Data (SPMD)**. Copies of the same program are being executed on different processes/processors with different input. The programs do not coordinate/communicate.
- **Multiple-Program-Multiple-Data (MPMD).** Different programs are being executed on different processes/processors with the same or different input. The programs have little coordination/communication – they are considered loosely coupled.
- **Distributed objects.** The objects of an application are distributed over the peers using an object-oriented middleware [30]. The objects are tightly coupled and have required therefore coordination/communication.

Provided Infrastructure

A P2P network for computation must provide core functionalities for task management, communication and security.

- **Task management.** The system should provide basic managements for task/resource allocation, load balancing, state checking and fault recovery.
- **Communication.** Communication is in a distributed computing is of great importance for task scheduling, task interaction and management. Providing a set of reliable and different (e.g. high and low-level) communication means is therefore essential.
- **Security.** P2P network requires support mechanisms for authentication, authorization, secure and safe execution and peer privacy.

Integrating of existing Code

To minimize adaptation costs it is important to ensure that existing code can be used without the need of major changes.

Supported Platforms

P2P networks tend to be large and heterogeneous therefore scalable approaches for handling dissimilar architectures e.g. operating system are required.

Customization

In order for each job to be executed in a manner that optimizes performance and resource utilization mechanisms for expressing user, task and machine policies are needed.

5.2 Condor

Work on Condor was started in 1988 by the Computer Sciences Department at the University of Wisconsin-Madison with the aim of developing a general-purpose framework that would allow the use of idle CPU cycles for research purposes. Condor is designed to support the execution of independent tasks following the SPMD model. It provides a flexible platform-independent framework for distributing "jobs" (tasks) over a pool of machines (peers) by providing a basic job queuing mechanism, scheduling policies, priority schemas and resource monitoring & management. It is built on the principle of distributing batch jobs around a loosely coupled cluster of computer to enable a high throughput computing (HTC) system.

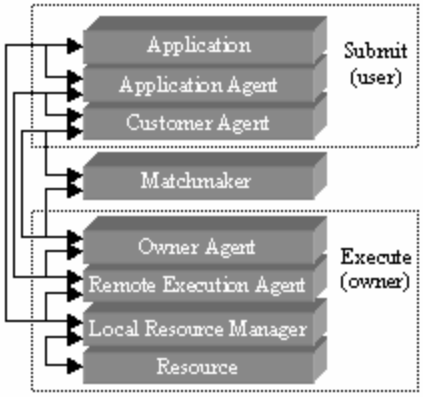

Figure 5.1: The Layers of Condor

Users submit their sets of serial or parallel tasks to Condor in form of jobs. The Condor matchmaker places them into a queue, chooses when and where to run them based on job needs, machine capabilities and usage policies. Condor monitors the progress of jobs and informs the user upon completion of their jobs (Figure 5.1). Condor uses a variety of different concepts to ensure fast and safe execution of jobs.

To protect the host, all jobs are executed in a restrictive sandbox that prevents/intercepts invoking any system calls. Only "remote" system calls are permitted since they will be executed on the host of the job's owner. In addition to this Condor supports strong authentication, encryption, integrity assurance, as well as authorization.

To ensure the fast execution of jobs Condor uses the following techniques:

- **Classified Ads (ClassAd)**

Ads are used for job/machine mapping, ensuring that the requirements of the jobs and the capabilities & policy of the

machine fit. A centralized matchmaker performs this mapping. All machines in a Condor pool advertise their attributes, such as available RAM memory, CPU type and speed, virtual memory size, current load average, the conditions under which will agree to execute a Condor job and the preferred type of job. Likewise, when submitting a job, the user specifies a job ClassAd with requirements and preferences. The job ClassAd includes the type of machine you wish to use.

- **Queuing mechanism with priority settings**

Each user has a Condor queue for all the jobs he submitted. The job priority is a means for users to identify the relative importance of individual jobs within a submitted set of jobs. Condor also uses a user priority ranking to determine the amount of pool resources given to the jobs. The higher the priority of the user the more resources are assigned to her/his jobs.

- **"Flocking" technique**.

Condor supports the linking of independent Condor resource pools. In a linked environment a Condor pool may transfer a job that is submitted to another pool that accepts "foreign" jobs.

- **"Up-Down" algorithm for scheduling.**

The longer a process runs, the lower its priority becomes. This policy is meant to ensure that users avoid long-lived jobs ensuring that the job queues are kept shot.

- **Checkpointing**.

Checkpointing is used to compensate for unexpected failures of the host and or job. Condor requires that each job is capable of saving its state in certain time intervals in form of an image and offers a library to implement this functionality. A checkpoint image contains the process's data and stack segments, as well as information about open files, pending signals, and CPU states. When the job is restarted, the state contained in the checkpoint file is restored using at startup routines in the checkpoint library. The process resumes the computation at the point where the checkpoint was generated.

Condor harvests the otherwise wasted CPU power of desktops, workstations, servers and clusters. According to the usage statistics [32] Condor delivers on average 650 additional CPU days (the sum of all CPU run over one day) to the researchers at the University of Wisconsin-Madison.

Condor is designed primarily for SPMD but supports also MPDM. Condor offers a variety of security features and has been enriched with a large number of inter-job communication libraries e.g. MPI.

Task management is centralized and ensures that jobs are executed in an efficient and secure based on the specified requirements of provider and consumer. However a consumer has little control over the location and manner in which its job is executed.

5.3 Avaki

Andrew Grimshaw at the University of Virginia initiated the Avaki project [19] in 1993, and re-launched it as Avaki Corporation in 2001. Avaki is a grid middleware that enables sharing of data, applications, and computing resources targeting the enterprise-wide computing area.

The Avaki grid is can contain desktops, workstations, servers and clusters. Each machine in the grid is autonomous and consequently the system management is distributed. Avaki is able to interoperate with queuing systems, load management systems, and/or scheduling systems. Like Condor queues are used to manage the resources and control the access.

Avaki is composed of three services layers (Figure5.2):

- The Grid Protocol layer, which provides protocol adapters, security, and naming and binding;
- The System Management Services layer, which provides capability for implementing and managing distributed solution;
- The Application Services layer, which provides high-level services;

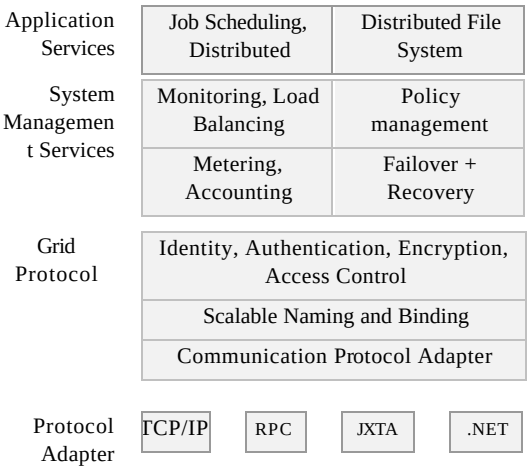

Figure 5.2: The Layers of Avaki

Each resource made available to the Avaki grid has a *unique logical identifier*. Avaki manages grid resources and application as follows:

- **Access controls.** A user or application may or may not have access to specific service or host computer.
- **Matching.** Though matching of application requirement and host characteristics.
- **Prioritizing.** Through prioritization based on polices and load conditions.

To ensure the safe and secure execution of the code Avaki uses the following approaches:

- **Checkpointing**

Avaki uses checkpointing to minimize the loss of information in the event of a host or network failure. Hosts, jobs and queues automatically back up their current states.

- **Redundancy**

Avaki networks are designed to scale allowing for the use of redundancy as an additional means for coping failures. Avaki migrates running applications to another host based on predefined deployment policies and resource requirements.

- **Authentication**

The Avaki authentication reduces the need for other software-base security control, substantially reducing the overhead of sharing resources. Avaki's authentication is based on the resource identify and uses the Public Key Infrastructure (PKI) technique. It allows local administrator control the access to their resources. It also includes user access authorization and resource access authorization.

The Avaki grid provides high-end computing power for scientific problems and is currently being evaluated at various research labs. It supports SPMD, MPMD and offers a range of inter-job communication options. As shown in figure 5.2 task management, secure and safe execution, load management is all integral parts of the Avaki grid. Avaki also supports all required security functions including authentication of provider and consumer, authorization and access. But by relying on job queues the consumer has again little control over the way in which and the location where its jobs are executed.

5.4 SETI@Home

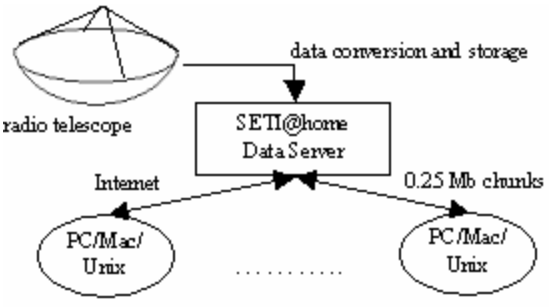

Figure 5.3: SETI@home Architecture

The idea for SETI@Home, as the project was called, came in 1996 from computer scientist David Gedye, along with Craig Kasnoff and astronomer Woody Sullivan. *SETI* (the Search for Extraterrestrial Intelligence) is a collection of research projects aimed at discovering alien civilizations using radio telescopes. Since the analysis of the extensive radio telescope data (about 35 GB per day) requires significant computing resources a P2P approach for distributed computing was chosen. SETI@Home engages internet users around the world in the effort of the distributed signal analysis.

As shown in Figure5.3, a central SETI@Home server divides the data into chunks (work-unit) designed for an average desktop computer. Participating peers contact the server and download a chunk of data. After downloading the peer starts processing the data in its idle time e.g. when the screen-saver is active. The result of the analysis is sent back to the central server and a new cycle of requesting-data, processing data and reporting results begins.

The tasks in SETI@Home are independent and can be executed without the need of any connection. Network connectivity is only needed for receiving data and sending results. The peer data - including the number of work units completed, time of last connection, and team membership – are reported on web-sites allowing users to compete for the biggest CPU contributions. SETI@Home uses a check-pointing mechanism to recovery from faults by saving all 10 minutes the dataset and the progress in analyzing it to the hard drive. SETI@Home also injects "test signals" intentionally into the system to confirm that the hardware and software is working properly. The "suspicious" responses to work unit or the lack of reported results will be recorded and used in evaluating the level of trust assigned to the peer.

The major contribution of SETI@home is to demonstrate how to apply distributed computing challenges in a P2P network. SETI@Home has managed to attract several hundred thousand active participants, which hope to be the "one" to discover ET.

SETI@Home is limited to SPMD problems and offers therefore no communication support (except for requesting data and sending results). Hiding the details of the communication protocols, requiring that users install SETI@Home software prior to joining the network and avoiding any code migration provides basic security. SETI@Home uses redundancy and tests data to ensure correct processing and flags suspicious peers.

Due to the large number of freely available computing resources no efforts for optimizing the execution of tasks are necessary. SETI@Home has only a single consumer and is in total control of where the data chunks are being processed.

5.5 JXTA

Project JXTA was started at Sun Microsystems in 2001. It is an open-source project (www.jxta.org) and was initiated by Bill Joy to standardize a set of protocols for building P2P applications. JXTA aims at providing a general framework that is independent of software and hardware platform.

JXTA defines currently six protocols:

- Endpoint Routing (ERP),
- Rendezvous Protocol (RVP),
- Peer Revolver Protocol (PRP),
- Peer Discovery Protocol (PDP),
- Peer Information Protocol (PIP) &

- Pipe Binding Protocol (PBP).

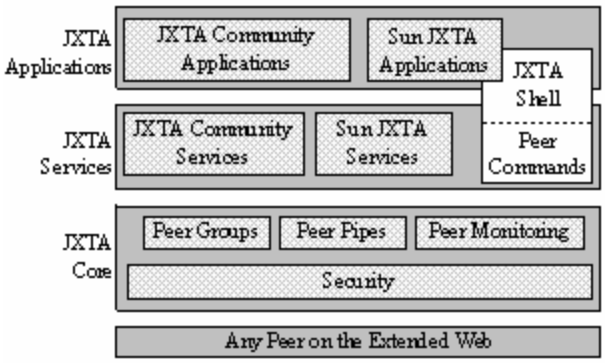

Figure 5.4: JXTA Architecture

Currently Java and C implementation of the JXTA protocols are available and a .Net version of JXTA is in development.

JXTA has several powerful platform-independent features that make it widely adopted by current P2P application designers, e.g. the Anthill [26] project.

- **Unique IDs for entities and advertisements.**

Each entity (peer, peer group, pipes, advertisement, etc) is assigned and identified by a unique ID. Similar to Condor, all resources in JXTA network are represented by *advertisements* but the ads in JXTA are XML formatted making them platform independent and extendable. Peers cache, publish and exchange ads to discover and find available resources. The advertisement mechanism makes all available network resources visible to peer and peer group architecture for work group generation.

- **Concept of Peer groups**.

Peers in the JXTA network are linked to at least one *peer group*, which is a dynamic set of peers that share interests and have agreed upon a common set of policies and services. Each Peer Group is a virtual network space consisting of a subset of all devices accessible via an overlay network. The JXTA *overlay network* is a middleware messaging system designed to allow for end-to-end connectivity between devices across sub-networks.

- **Transparent Communication via Pipes**

JXTA uses asynchronous communications channels, called *pipes*, for sending and receiving messages. Pipes offer two modes of communication: point-to-point pipe and propagate pipe and allow for a simple and transparent form of communication.

- **Rendezvous peers.**

JXTA provides a revolver services based on Rendezvous peers. *Rendezvous peers* are well-known peers that have agreed to cache a large number of advertisements for exchanging and trading information.

- **Peer-monitoring.**

Peer-Monitoring is a core mechanism of JXTA. It enables control of the behavior and activity of peers in a peer group and can be used to implement task management function for fault detection and recovery.

- **Entry-level trust model.**

Project JXTA provides an entry-level trust model, Poblano [16], which permits peers to be either own certificate authorities or socially accumulated inter-peer communication. In addition, the secure communication in JXTA is based on the Transport Layer Security (TLS).

JXTA provides a general-purpose P2P network programming and computing infrastructure and consequently it supports basic security and communication features. However due to its platform nature much of the task management, and the support for different computing models is left to the developer of the application.

5.6 Summary

The above-mentioned systems showed great diversity of approaches. While all support SPMD and some support MPMD only JXTA is capable of supporting distributed objects.

But while the computing model can be seen as a less important design issue of the distributed computing application, the mapping of tasks to processes/processors is essential. With the exception of JXTA (it has no mapping implemented) the above-mentioned systems rely on *static* and *centralized* mapping approaches.

As mentioned earlier self-organization due to awareness is an essential component of P2P networks. Once the peer is aware of itself (self-awareness), the network environment (peer-awareness) and the tasks (task-awareness) it can determine the “best” peers for performing a task. Awareness can therefore be used for a *decentralized* and *dynamic* mapping of the tasks allowing for more autonomy and flexibility of consumer and provider. Using such a decentralized and dynamic approach would allow for a better distribution of (mapping) load and enable more flexible, localized and reactive mapping. Using past experiences as well as usage negotiation (between provider and consumer) would also be possible enabling a better mapping.

6. Awareness & P2P Computing

To evaluate the idea of using awareness as a means for enabling a decentralized and dynamic mapping of task a simulation was used. The simulation consists of clients (consumers) and servants (providers) and is aimed at studying the impact awareness has on the throughput when using independent tasks.

6.1 Self-Awareness Experiment

The simulation uses the recursive calculation of Fibonacci numbers as an example workload. In the simulation clients can request the calculation of a Fibonacci number from one of three servants with different performance features (fast, medium and slow). The clients can either use a randomly selected servant or the “best”

currently available one. The two clients are implemented as threads on the same machine and have a well-known table of the entire servant group, including the servant network position and service binding but differ in terms of their servant selection algorithm:

- The normal consumer/client selects the servant randomly.
- The 'Smart/Aware' consumer selects the best performance (shortest elapse time of task completing) servant basing on the historical average task completion time of each servant. The average is calculated by the following formula:

$$A_i = \alpha * C_i + (1 - \alpha) * A_{i-1} \ (0 \le \alpha \le 1) \qquad (1)$$

Where

C: the completion time of one service.

A: the average completion time of the period of time.

i: time number of one service call, i.e. first, second, …

α: Dependency of completion time of last call and previous calls. The higher α is, the more emphasis is given to the completion time of the last call when determining the servant performance.

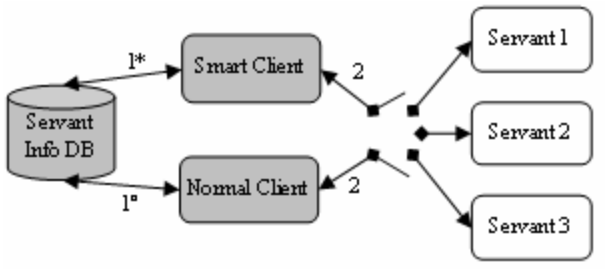

Figure6.1 Simulation architecture

1* - smart selection: chose the fastest servant base on the history of task completion time;

1? - random selection: chose the servant randomly;

6.2 Results

The results are based on data collected from one-hour tests and a α value of 0.8 (for the formula 1). Figure 6.2 and 6.3 show the average throughput and service completion time respectively. As expected the 'smart/aware' approach is significantly better with almost twofold throughput and halved completion time after a short downturn at the beginning where the initial data are collecting

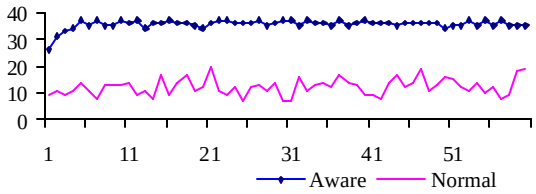

Figure 6.2: Average Execution Throughputs

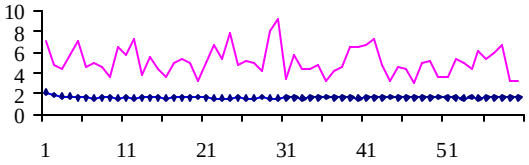

Figure 6.3: Average Completion Time (Minutes)

7. Summary and Future Work

P2P networks are a recent addition to the already large number of distributed system models. Based on decentralization and self-organization P2P networks appear to be more scalable and robust than other approaches which makes them ideal candidates for harvesting the ever-growing pool of idle computing resources.

The focus of this paper is on the issues and existing approaches for using the idle resources in an enterprise network for distributed computing. The paper reports about an initial study on the availability of idle computing resources, points out the key challenges in using P2P networks for distributed computing and provides an overview of existing systems and approaches.

The idea of dynamic and decentralized mapping is identified as a key issue of using P2P network for distributed computing and evaluated using a simulation.

Future work will focus on using more refined awareness concepts for mapping sets of dependent tasks. Staring with the use of larger and more feature-rich simulations the aim is to develop techniques that can be integrated in existing P2P middleware.

8. REFERENCES

[1] Dejan S.Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, HP Laboratories Palo Alto. Peer-to-Peer Computing. HPL-2002-57, March 8th, 2002

[2] Napster, http://www.napster.com/

[3] Clip2, http://www.clip2.com. The Gnutella Protocol Specification v0.4, Document Revision 1.2

[4] I.Clarke, O.Sandberg, B.Wiley, T. W. Tong. Freenet: A distributed anonymous information storage and retrieval system.

[5] Project JXTA, http://www.jxta.org

[6] Ian Foster. The Grid: A New Infrastructure for 21st Century Science. February 2002. http://www.aip.org/pt/vol-55/iss-2/p42.html

[7] Lance Olson. NET P2P: Writing Peer-to-Peer Networked Apps with the Microsoft .NET Framework. Microsoft MSDN magazine

[8] jxta.org. Project JXTA: An Open, Innovative Collaboration. http://www.jxta.org/project/www/docs/ OpenInnovative.pdf

[9] FastTrack, http://www.fasttrack.net

[10] SETI@home, http://setiathome.ssl.berkeley.edu

[11] Anath Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to Parallel Computing. Second Edition

[12] G.A.Geist, J.A.Kohl, P.M.Papadopoulos. PVM and MPI: a Comparison of Features. May 30, 1996

[13] M. Beck, J.J. Dongarra, G.E. Fagg. G. Al Geist, etc. HARNESS: A Next Generation Distributed Virtual Machine. June 24, 1998

[14] Antony Rowstron, Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

[15] Condor Project, http://www.cs.wisc.edu/condor/

[16] Bernard Traversat, Mohamed Abdelaziz, Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul and Bill Yeage, Sun Microsystems, Inc. Project JXTA Virtual Network. October 28, 2002

[17] Krishna Kant and Ravi Iyer, Enterprise Architecture Lab, Intel Corporation, OR. A Performance Model for Peer to Peer File Sharing Services. November 16, 2001

[18] Joseph A. Kaplan, Michael L. Nelson, NASA Langley Research Center. A Comparison of Queueing, Cluster and Distributed Computing System. June 1994.

[19] Avaki, http://www.avaki.com

[20] Condor Team, University of Wisconsin-Madison. Condor Version 6.4.7 Manual.

[21] Patrick Wagstrom. An Overview of Condor. February 19, 2002

[22] Wilmer Caripe, George Cybenko, Katsuhiro Moizumi, and Robert Gray. Dartmouth College. Network Awareness and Mobile Agent Systems.

[23] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky. SETI@home: Massively Distributed Computing for SETI

[24] Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch, Ilya Sharapov. Framework for Peer-to-Peer Distributed Computing in a Heterogenous, Decentralized Environment. Sun Microsystems, Inc.

[25] David Molnar. The SETI@home Problem. ACM Crossroads Student Magazine. Fall 2000.

[26] The Anthill Project. Department of Computer Science, University of Bologna. http://www.cs.unibo.it/projects/anthill

[27] Murali Krishna Ramanathan, Vana Kalogeraki, Jim Pruyne. Finding Software Technology Laboratory, HP Laboratories Palo Alto. Good Peers in Peer-to-Peer Networks. HPL-2001-271. October 23, 2001

[28] John D. Musa, Anthony Iannino and Kazuhira Okumoto. “Software Reliability: Measurement, Prediction, Application”. ISBN 0-07-044093-X

[29] R. A. Malaga. Web-based reputation management systems: Problems and suggested solutions. Electronic Commerce Research, 1(4), 2001.

[30] Chen Wang, Yong Meng Teo. Support parallel computing on a distributed object architecture. The Journal of System and Software 565(2001)261-278.

[31] http://bistrica.usask.ca/madmuc

[32] CondorView Pool Statistics, http://pumori.cs.wisc.edu/condor-view-applet/

[33] Jabber Organization website http://www.jabber.org/

[34] Zoogi website, http://www.zoogi.com