BOINC on JXTA - suggestions for improvements

mgr inż. Marcin Cieślak

June 26, 2007

This is a partial translation of work named "BOINC – increase in efficiency". It was presented at Technical University of Wroclaw, Poland. It was never used for commercial benefits. All sources that the paper base on were described. All rights are reserved. E-mail: marcincies@yahoo.co.uk

This paper covers parts that are sufficient for understanding the presented solutions. It lacks in technical details which the author can make available on demand. The first part covers improvements designed for BOINC. The second part is a project of an architecture that has functionality similar to BOINC and it is based on Sun's JXTA.

Table of Contents

Abstract

Distributed computing systems allow acquiring resources that lie outside the capabilities of even the largest supercomputers. Indisputable example is prosperity of SETI@Home project which gathered at its peak ca. 850,000 users. Its success is partially related to well designed architecture – BOINC. Many projects work on this architecture, using all together the computing power of ca. 400 TeraFLOPs. This is more than the most hi-tech supercomputer. But even BOINC architecture struggles with scalability concerns. The creators may not have predicted the degree of projects' popularity. The purpose of this work is to find new solutions for volunteer computing. This paper contains overview of existing systems which elements might be useful for this goal Next it describes BOINC parts along with propositions of changes which can benefit its effectiveness. In the second part has been shown how the popular JXTA platform can be utilized to completely redesign the existing structure. A complete project of new architecture has been introduced which has the same functionality as BOINC, but it overcomes its limitations and the resources are utilized better. This work also involves partial implementation and points out the directions of future development.

Part I – Existing solutions

1. Overview of selected systems for distributed computing and file sharing

The goal of this chapter is show the solutions used in other systems that influenced the suggested improvements. Two systems for file sharing and three for distributed computing are presented.

- Napster – served as a P2P system for sharing MP3 files [18]. It was created in the year 1999 by Shawn Fanning in only few months. The idea was both simple and effective. A user that downloaded the application shared a directory with his MP3 files and was registered on Central Indexing Server. When a user was looking for a file, he contacted the server which in turn queried registered computers. If one of them had the needed file, it directly contacted the seeking computer and started the transfer. After the finish, it disconnected. Even though the server was used for indexing, the real file sharing was performed without its participation, so we call Napster a P2P system. Despite of small range of duties (seeking, not sharing data), the server became a bottleneck and limited scalability of the system.

  The possibilities and simplicity of use have started the big popularity of this program, which drawn the attention of producers of original albums with music. The activity of Napster was suspended for piracy reasons. Nevertheless it started a new era in resource sharing with the help of Internet.

- Kazaa – serves as a system for sharing files [22], created in the year 2001. Its potential is not limited to music files only: users can share any kind of data. Unlike Napster, Kazaa uses seek algorithm called FastTrack, which is called a 2nd generation P2P algorithm. It introduces Supernodes, fast computers with big computational power and broadband connection to the net. A simple node connecting to the net registers at the Central Server and receives a list of active Supernodes. If it fulfils the requirements, it may be chosen as a Supernode itself, without notifying the user. When looking for a file (figure 1.1), a node contacts not the Central Server, but one of the Supernodes which in turn forwards the query to other Supernodes and simple nodes (a single Supernode can serve 60 to 150 simple nodes). The query is forwarded till it reaches the limit of 7 hops. If one of the queried nodes has the needed resource, it replies the first node and data transfer begins. If the user receives replies from many nodes, it downloads parts from many locations and merges them into one file. Kazaa uses also MD5 hash function to check the correctness of the downloaded file.

Decentralized architecture allows higher effectiveness and scalability than in Napster. Users of Kazaa get both the answer (the Central Server is not responsible for it) and data (downloading from many users) much faster than it was in its ancestors. There are some systems with the same functionality and similar architecture , for example eMule, Gnutella. Kazaa, same as Napster, met with protests from music distributors. Nevertheless, it exists to the present day, same as its clones.

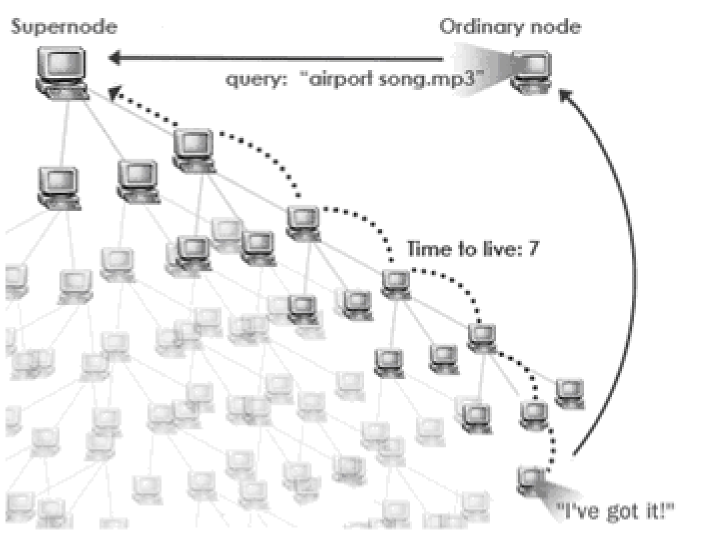


Figure 1.1: Seeking files in Kazaa [22]

For the purpose of searching for resources or services many algorithms were created, more or less centralized, for example Chord, CAN, Tapestry, Pastry [11].

- Distributed.net – non-profit organization founded in the year 1997, which mission is solving world problems requiring long calculations. It accomplishes it by distributing work to users' computers. One of the current problems is breaking RC5 (Rivest Cipher) with 72-bit key.

  Distributed.net is one of the first systems using volunteers' computers to achieve a common goal which is processing large amounts of data.

- Great Internet Mersenne Prime Search (GIMPS) – project which goal is to find Mersenne prime numbers. This task requires enormous amounts of simple calculations. Users download and install small application that test numbers using only little resources. We can call GIMPS a system for volunteer computing. Single test takes about one month. The results are periodically saved on a single server.

  GIMPS gathered massive computational power: 2000 GigaFLOPs per week. This corresponds to power of 400 supercomputers Cray T932 type. This project has successful results: 10 Mersenne prime numbers have been found, including the biggest currently known $2^{32582657} - 1$.

- Avaki – serves as a system for managing distributed data processing. It was created in the year 1994 and was formerly known as Legion. The goal of the system is to enable perceiving as a single virtual machine many computers localized all over the world. This system architecture is based on platforms like JXTA and .NET, allowing cooperation of heterogeneous systems. Every computer has a unique address and set of interfaces it provides (Avaki strongly bases on remote method invocation). Because of different OS and file systems installed on participating computers, Avaki builds over them its own layer, hiding the complexity of the lower layers. As shown on figure 1.2, the architecture has many layers. It makes management and usage simple. On the other hand it decreases the effectiveness comparing with the direct communication between computers. This system indicated high increase in effectiveness for tasks that can be divided into smaller parts.

Avaki Coproration is developing this system, now named Avaki Compute Grid. The company put on the market also other products: Avaki Data Grid, used for managing data located on remote computers, and Avaki Comprehensive Grid, a connection of two first ones.

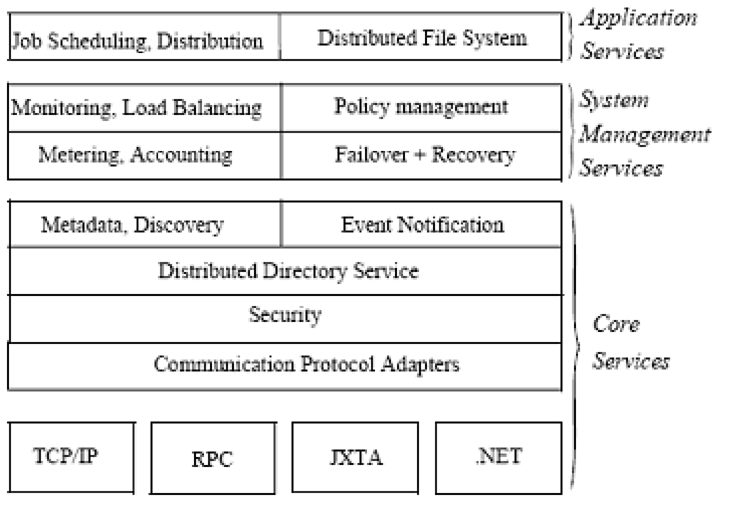


Figure 1.2: Layers in Avaki [11]

2. Description of BOINC

(The explanation of BOINC follows. Based on [2,3,4,6,8,12,19])

2.4 Identification of problems and limitations

Basing on the analysis of the existing solutions and available publications, problems concerning BOINC architecture can be identified. They were already pointed out in the previous chapters, here summary follows:

- Server overload – important issue, which despite of the recent (2005) efforts has not been completely solved [6]. The cause lies not in the server efficiency, but in the organization of communication. Although SETI@Home is a great success, the creators of BOINC made an architecture which is not completely scalable. It is based on a central task server, which controls the work of all project participants. Sometimes their number is about few hundred thousands. Obviously the server becomes a bottleneck, decreasing the effectiveness. It should be change in the design of such large system.
- Redundant communication – it is related with the previous issue. The same data is downloaded from a server by many clients. It concerns both, short and frequent connections with the task server and file downloading (which size is sometimes up to few gigabytes) from the data server. Situations when the result is erroneous make it worse – the resources used to compute it are lost and another client must repeat the calculations. Information flow is incorrectly planned. The authors mention using P2P connections between the clients while omitting the servers.
- Improper task distribution – the mechanism of task distribution on the both server and client sides is imperfect. Algorithms often base on partial information, not sufficient to make an optimal work schedule. It leads to some negative effects: insufficient work, project overload, missing deadlines. The creators of BOINC consider some algorithm improvements, for example simulation of CPU scheduling on the server, but this would only increase the tension on already overloaded servers and trigger even more communication.
- Resources limitations – BOINC projects are mainly academic, thus missing larger funds. They might not be able to afford additional servers or other elements. That is why the solutions that increase the efficiency must take into the account current projects' hardware resources and their immutability. This adds to limitations of upgrades that can be suggested

for BOINC.

Despite of the fact that the problems are few, they strongly influence effectiveness of the system.

This chapter concludes the short description of BOINC. Next one focuses on possible improvements of its elements, without drastic changes in the structure. The changes suggested by the author mainly only tune the system rather than rebuilding it.

Part II – Suggested system improvements

3. Improvements of the existing BOINC architecture

There is range of modification that can be brought about in order to reduce or remove troublesome factors mentioned in the previous part. There are also many improvements that could influence the efficiency. BOINC developers are constantly working on the variations of current solutions. In their articles [19] they present development paths and ides which they want to bring to life. In this chapter special kind of solutions is suggested. They increase the effectiveness of BOINC and can be used without heavy modifications in its structure. They were inspired by tips and statistics given by BOINC development team and by elements of other systems, that could be useful here. The implementation of these solutions shouldn't be too difficult and the results can be very interesting. Constant experiments on algorithms could lead the existing architecture to perfection, but its foundation has limitations that no improvements can remove. This is the same situation as in the large mainframes and computer networks (including overwhelming possibilities of volunteer computing) that took their place. The limitations in the foundation is the reason why this paper focuses on complete rebuilding of BOINC architecture (chapter 5 and later). This is also the reason why suggestions of improvements in this part are presented only in theory, without real implementation and testing. Nevertheless, the solutions are logical and all reasons for elaborating, known benefits and disadvantages are summarised. In the following chapters more drastic changes are presented.

3.1 Sending results

The processing of a whole result and then receiving the answer from the validator that the outcome is not correct means cancelling whole work. It can be quite expensive especially for long results.

In some projects it is possible to find a partial outcome. If we consider SETI@Home: maybe a part of signal data can be processed and closed – never used again. Sending data in bigger batches instead of splitting them into small parts has its explanation in minimization of network traffic. It doesn't mean though that a workunit is cannot be divided. Similarly, multiple processing of the same data can lead to partial outcomes after each iteration. Such outcomes could be reported and validated earlier than only after processing the whole workunit. This can lead to earlier detection of errors. The erroneous result will not be processed in vain but cancelled. Such mechanism will increase efficiency in projects where workunits have long processing time but it will be virtually harmful for quickly processed workunits. In the latter case flood of messages with partial results could be encountered. It should never be allowed, especially that the final outcome can be found quickly enough.

Partial outcomes are useful not only because of early halt of an erroneous result, but also for other improvements described in this paper.

3.2 CPU scheduler

- Partial outcomes – the idea of partial outcomes has been described in the paragraph 3.1. Although partial outcomes are mainly a trigger to complete abandoning an erroneous result, they can also influence the planning process. The mentioned feedback information, concerning outcomes reached by other results, can be useful not only to decrease the server load (the outcome is analysed on a client, not the server), but also for synchronization of computations within one workunit. Progress information from other clients could be a hint to slow down or to hurry execution of the current result. Taking the partial outcomes into account will allow reaching partial outcomes in the same time. The scheme of using partial outcomes can be as follows:

```
if the result R has reached the partial outcome № X
    send it to the task server and wait for a response
if there are no partial outcomes № X from other clients
    decrease the R priority
else if there are no partial outcomes № Y (that Y > X)
    set the R priority to normal
else increase the R priority proportionally to relation of clients that reached the Y outcome
to the total number of clients that compute current workunit; also proportionally to the
difference: Y – X
```

The result priority can be used between 6th and 7th step of the current CPU scheduling algorithm. It should not be more important than CPU scheduling period, because process switching is expensive. The terms of receiving the partial outcomes should be treated "softly" i.e. they should never bring harm and in some cases they should help. On the other hand they can be considered more important than the project debt, because the resource share should be equalized only in long period of time [19]. Clients that compute the same workunit sometimes greatly differ in speed. A slow client will also be "hurried" by others that received the next partial outcome faster. Nevertheless, current mechanism of reducing project "overworking" should be sufficient in the most cases. In case one of the results is shown too much attention, there will be simply no more results downloads for this single project. Unfortunately, such a mechanism could show being sufficient, if the result processing takes a lot of time and the partial outcomes are reached very often. In that case this single result will always try to be up-to-date and will dominate the others. This should be avoided from the very beginning. If by are reason this situation exists, additional safety means should be used. A critical limit for project debt can be set. If it will be crossed, the debt will have "right of way" (become more important) over the result priority calculated from the partial outcomes.

- Running the CPU scheduler at checkpoint set up. This simple modification can have big impact on the results that have long periods between checkpoints. If there are no other checkpoint modifications, this single one could surely limit the loses caused by yielding a result that set up a checkpoint long time ago. Current algorithms for CPU scheduling and enforcement allow such situation. If CPU scheduler would be run at the moment of result's checkpoint set up and it would find out that exactly this result should be yielded, then CPU time loss would be minimal.

  When introducing this modification, a limitation on the frequency of CPU scheduler runs should be added. If multi-processor client processes workunits from different projects, and checkpoints are set up very often, then the scheduler can be run every few seconds. There should be minimum time between consequent scheduler runs.

- Suggesting work unity. The creators of BOINC on their web pages encourage participating in many projects [1,19]. Obviously, this is beneficial for the projects, because each of them will receive a part of client's resources. Unfortunately, it has negative impact on the overall architecture effectiveness. If a client belongs to only one project, there are no debt or yielding problems. Especially the latter is expensive. A better state is if every client is participating in one project only. On the other hand this would be harmful for less popular projects and not interesting for the users and would. That's why this modification should be used together with "client trading", which is described in the next chapters.

3.3 Schedule enforcement

- Requesting checkpoints. This modification changes not only the schedule enforcement mechanism, but also project applications. Currently, workunits usually contain some kind of application, often common for the whole project, that processes input files specific for the workunit. This application should periodically checkpoint i.e. save its state, so in case of interruption its work doesn't have to be started all over again. Presently checkpointing is done at a suitable moment, picked by the application, for example after processing a part of data. But there is a class of calculations in which a checkpoint can be set up anytime or in consequent, short periods of time (same as reaching partial outcomes). For such projects a mechanism for requesting a checkpoint is worth considering.

  The application listens for a signal from the managing application. After receiving the signal, it checkpoints as soon as possible. In BOINC there are projects that will not use this mechanism (current projects that would not add the processing of the signal to their applications; projects in which checkpointing is not always possible) and those that use it.

  Benefits from adding a checkpoint request is easily noticeable: a yielded result would checkpoint immediately, so no work is lost. There is no need for additional resources to repeat the calculations. Algorithm of schedule enforcement that uses this mechanism is described at the end of this chapter.

- Predicting checkpoints. As already has been mentioned, the results checkpoint at the moment suitable for them. It can happen at some point in calculations, after processing a part of data, or never (when results are short). For the first and partially second case (when we can predict how much time is will pass till the next checkpoint) a mechanism for checkpoint prediction can be elaborated. This solution can turn useful for the projects where checkpoint request is not possible (described in the previous paragraph).

  To predict the moment of checkpointing, some assumptions must be made. We can assume that the moment of checkpointing depends on processing time of the result. If the result sets up a checkpoint after 160 seconds of processing we can consider the checkpointing period of 160 seconds. If it checkpoints after 160 seconds for the first time and then after 300 seconds, we can take the average:

  $((300+160)-160)/2 = 150$

  and get the checkpointing period equal to 150 second. This method is quite primitive and will only work for very linear calculations. To make it more complex the implemented mechanism for result's progress reporting can be used. The results should show their progress in computations as exactly as possible and report it to the managing application. This application can try to find dependency between this value and the moment of checkpointing. This method can be good for the results that checkpoint after processing a fixed part of data.

  The best option is to report the estimated time till checkpointing by the results themselves. It can be reported together with information on progress. This method is very useful but it cannot be used in results in which estimation of checkpointing time is impossible. It also requires some modifications in the present projects.

  If the time of checkpointing is known, it can be utilized in the first step of existing schedule enforcement algorithm. A time limit can be set for the result that should be yielded. If the result checkpoints in time, there will be resources lost. The limit may be also set on the cost of yielding. Yielding means loosing some CPU time. If the loss would be too great, the schedule enforcement for this CPU should be deferred.

Deferring the schedule enforcement creates the risk of schedule obsoleteness. If deferring of the schedule takes too much time the scheduler should be run again. Additionally this modification is bound with described earlier running the scheduler after setting up a checkpoint. If it is known that the scheduler will be run after setting up a checkpoint by a result, there is no need to actively wait for it. There should be no schedule enforcement for this CPU and the whole schedule should be cancelled. The enforcement would be executed after creation of a new schedule i.e. after setting up a checkpoint by the result. In this case the exactness of estimation is important. If the result "promises" a checkpoint soon, but it does not set it up, it could be running forever taking resource belonging to other projects.

Taking into consideration improvements described above, the algorithm of schedule enforcement can be as follows (meaning of X, Y, T is the same as in the BOINC documentation):

1. If one the results in X missed its deadline, send a signal to checkpoint to all results from Y. If there is a result which accepted the signal and correctly checkpointed within the time limit, yield it and run a result from X.
2. If the mechanism for running the CPU scheduler at the moment of checkpointing is implemented, and there are no results from Y that checkpointed during the last, very short period of time (needed to prepare the schedule and execute step 1), cancel the enforcement (STOP).
3. If there is a result in Y that will probably checkpoint in the nearest time (fixed time limit), defer yielding till this checkpoints.
4. If there are still results in X, yield a result from Y which has the shortest wall clock time from the last checkpoint and run result from X.
5. If there is a result in Y which checkpointed earlier than T, yield it and run a result from X.

3.4 Result download

The improvements presented here are closely related to changes of strategy of sending work by the task server. They mainly develop the idea of simulating result execution and checking how much will it take. The need of such simulation is only mentioned in BOINC documentation [19], but there are no tips for elaborating it. This kind of simulation parameters should take into consideration:

- present state of running and suspended results: estimated time of finishing, time of checkpoint set up (and along with the changes suggested earlier, also estimated time of estimated time of checkpoint set up), deadlines, project resource share.
- calculation capabilities of a client: theoretic capabilities estimated by benchmarks, enriched by statistics of real time available for BOINC. These statistics are created on-the-fly by the existing architecture. They are saved ad time fractions: available for a client, available for a project, available for a result for real.
- time needed to download a file from the data server
- parameters of downloaded result: estimated finish time, deadline, project resource share

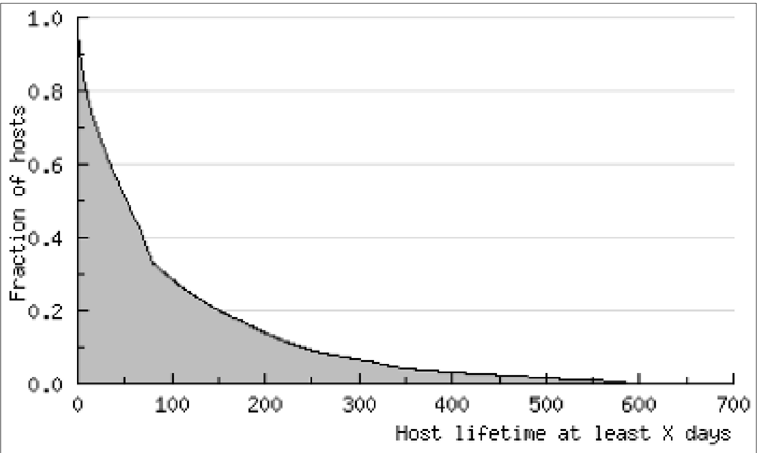


Figure 3.1: Host lifetime [5]

Properly constructed simulation would be capable to forecast the CPU load at a particular moment. Such simulation utilizing statistics and information of current results could estimate when a result will finish its execution (if to add previous improvements, then also time when a checkpoint will be set up) and how CPU scheduler and enforcement will behave. The simulation could forecast for few scheduling cycles forward, of course with decreasing accuracy. The simulation could not foresee influence of changing environment – other applications that decrease BOINC efficiency and events initiated by the user (turning off the client software, project suspension and so on). Nevertheless if the applications would work long enough to make the statistics accurately reflect computer capabilities in long period of time, and this application would work in stable conditions (same or similar CPU and memory load), then the simulation results could be satisfying enough to make them a foundation for further improvements.

Unfortunately there is a risk that a client won't participate in a project long enough to gather meaningful information about him. The statistics shown in picture 3.1 confirm that. Many clients are active only for a short period of time after which they completely detach from a project.

It is worth noticing that the idea of making a simulation on the server, as suggested by BOINC developers, is very brave. Even now a big problem is server overload. If to give them some additional responsibilities, there could not service all the clients. Instead making a simulation on the server, it is possible to try it on a client. Below there are useful modifications of simulation:

- List of workunits download. This modification is quite simple, but can bring good effects. Client sends a work request to the task server. The contents of such request may not differ from the presently used in BOINC architecture. There should be at least one more parameter, which will allow the server to filter out some part of workunits. The parameter is estimated time for running the next result. A client should make a simulation and check when it is possible to run the downloaded result. This information is very useful for the server, which in the current form bases on present time (it assumes that the result will be run immediately).

  After receiving RPC request, the task server answers with a description of a few results with different parameters. It also takes into account that every one of them can be accepted by the client and cannot be distributed to others (information for the transition deamon). After receiving this description, the client as quick as possible makes the simulation for each result and chooses the optimal one (or many). Optimal means the lowest risk of missing deadlines by other results and debt compensation. Then the clients sends his choice to the server and continues his work.

  This solution increases the safety of keeping the deadlines and protects from taking too much work by a client. On the other hand the net traffic is increased – communication with the server is executed twice. There should be also a mechanism to block the results which have been sent to a client for consideration. This modification at the client requires a few simulation, which should be made very quickly.

- Checking client capabilities – In this modification a client makes the simulation himself, trying to find how much date and in what time he is capable to process for a given project. Taking into consideration the average size of a workunit for the given project, a client should be able to make the simulation on workunits with similar size. After that he should be able what kind of a workunit he is able to process in an hour, day or a week (there should be some fixed time periods. Only after finding this information he should send it to the server, which will filter out those available workunits, which client can't take. Then it spawn a result and send it to the client.

This solution should give effects similar to the previous one, but without the price of increasing the network traffic. On the other hand a very good simulation model is needed in order to accurately predict client capabilities. Besides using CPU for simulation means fewer resources for data processing.

- Storing parameters of clients processing a workunit. This modification is tightly connected with partial outcomes. If server had information when and what clients received results of a given workunit, it could distribute the remaining results more effectively. If from a given workunit some results were spawned and send recently, another result should be sent to a similar client. If some time have already passed (rare case for popular projects), another result should be assigned to a faster client which is able to catch up with the previous ones.

  At this point it is worth noticing that implementing improvements that were quite generally described by the BOINC developer would positively influence the system work.

3.5 Giving credits and computation redundancy

It is difficult to modify the ideas of giving credits and making redundant computations, because they are simply good and even necessary in voluntary processing systems. To the present day in BOINC credits were given for real work made for a project. It is certainly good for the system but not very fair for users. A project participant who has an old computer with little resources won't receive many credits even if he makes it available for the project for a long time and in a large degree. A user that has brand new computer and shares just a little bit of its computational power will be awarder more. Awarding credits could be independent of client's hardware configuration, just like in eMule. In that system a user fills in or system itself detects the computer capabilities. Then the user declares the part that he wants to give for the system. Allowing an user to declare the resources could lead to cheating but it has to be made available. If a user has for example internet connection with limited band then the application might detect it incorrectly. A deamon that monitors the network traffic and makes computer capabilities statistics could be used. After few days basic computer capabilities would be known and the longer the demon worked the more accurate they would be. Still, the user could feel unsafe if some software scans his system. He should have full knowledge what is happening in his computer. Nevertheless if to find the border limits of system resources and user preferences for the degree of their usage, credits can be awarder without the penalty for less technically advanced hardware. The credits should be given depending on the dedication to the project and not for the wealth of a user (or at least in a smaller degree). It would surely give more chances to less wealthy users to take higher places in the ranks and would be an additional boost to participation. Even the amount of processed data should be a secondary factor, still taken into account because the charts would be quickly filled with equal places for computers completely dedicated to a project.

The same goes to repeating the calculations on few computers. It increases their correctness which is difficult to obtain with other methods. That is why the suggested improvements mainly concern the effectiveness of redundant calculations and not the idea itself. Also the work of deamons responsible for the redundant calculations mechanism is so deeply built-in into the system that without changing the architecture it is difficult to suggest any improvements in this area.

3.6 Main, task and data servers

As has already been mentioned those servers are often bottlenecks of the whole architecture. Some improvements were introduced allowing wider use of single servers but finally BOINC developers were obligated to make multiple servers available. This is a new ability (introduced in the year 2005) and still hardly optimized. For now the solution is that a client either chooses himself the server from which the data will be downloaded or he is bound with one by the main server. Of course the data on data servers doesn't have to be replicated (sometimes it is) and a workunit is responsible to choose from which server the files should be downloaded. The task server also has ability to tell a client to use another server. Unfortunately the documentation was not updated correctly so it is not obvious when the server exactly does it. To summarize, many task and data server decreases the load on each of them and increases the system reliability. However there is a range of improvements that can be used to use the available resources better:

- Monitoring the load of RPC requests. Quite simple modification. When the task server has too many requests it can redirect a client to an another server. The client however could veto such redirection, since the suggested target server could be unavailable or have too many requests itself.
- Dividing clients by server connection speed. This proposition means adding to a client some kind of decision making algorithm to which server send requests. It would be a rather bad situation id a server located for example in America would service request from Europe and the European one would take care of clients from America. In the existing architecture it is possible. Checking client's and servers' IP, geographical localization and choosing a server closest to a client could be a solution. However more accurate one would be to ping the existing task servers and check connection speed to each one of them. Additionally the choice should not be dependent on the answer speed or the geographical location but also on potential server load – a situation in which on one server 90% of users is registered, and on the other one only 10 % should not happen. It this case redirecting a client from one server to another would be rather long-term.

  This improvement would allow gathering on the server clients that have quick connection to it.
- Sharing servers by projects. This idea develops the previous one. If on one server put databases from different projects, and make all servers common, then the dividing clients by the connection speed is even more effective. If in one project there would be servers in America and Europe, and in another project in Africa and Asia, then after making them common each of those projects would have servers that cover all those continents. The speed of servicing a client would increase but the connection number would remain the same.
- Trading clients. If to create a function that for each client assigns some meaningful value, a mechanism for exchanging clients between projects could be elaborated. This value could be for example computational power of a client which is monitored by the existing architecture elements anyway. In exchange for a client with some value a project would receive another client (or clients) with the same value. The benefits of such solution are:
  - better binging of clients to the closest servers. A projects would give away the clients located far away from its servers and would take closer ones.
  - better support for homogenous redundancy. If one project requires a specific hardware platform or simply a homogenous redundancy, while having many different clients, trading with another project would lead to clusterization and equalization and so to satisfaction of project requirements.

- less result yielding. This benefit is connected with suggesting work unity, as described with CPU scheduler improvements.
- choosing clients that could be pylons (this term is described below), If some projects has many clients far away from its server and between them there is no machine that could be fulfils the requirements of promotion to a pylon, then it could check some other projects, seeking the right client and possibly make an exchange.

There are many benefits from client trading, but there are many aspects to consider before even trying to implement this mechanism. There is also the question of user liberty. If he had chosen 3 projects to participate in, he shouldn't be made to process data from only one of them. Such user could be simply asked if he allows such mechanism of trading between projects to be active. It is also possible to send him the credits received by other clients for work which he should be doing. Finally, this information could be completely hidden from a user.

Utilization of this modification requires controlling the clients which have been exchanged and sending information concerning processed results. Perfectly it would omit the server and all the needed data would be send directly (P2P).

- Anonymous server ("pylon"). A typical feature of BOINC is the variety of clients. The statistics in [5] show that well. On picture 3.2 – 3.4 and in table 3.1 a summary of BOINC clients' capabilities is gathered. The statistics of floating number operations show that although most of the clients have low computational capabilities, but there is a group which shares more resources. Picture 3.4 shows that although there are many users with high speed connection but also a large group has very limited network bandwidth. So always between clients machines that are more dedicated to a project can be found. Using the statistics gathered on the client machines can be found that are almost always on, have client software active and have high speed connection. Additional positive factor would be high distance from other project servers. Table 3.1 shows how the geographic localization is various.

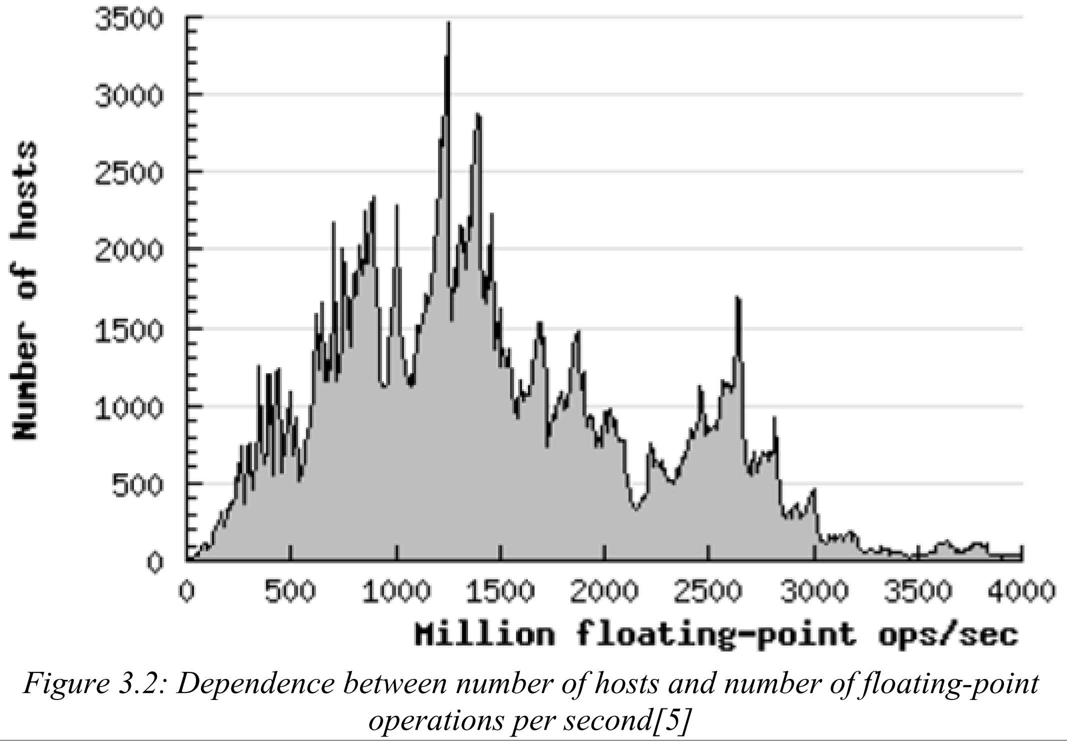


Figure 3.2: Dependence between number of hosts and number of floating-point operations per second[5]

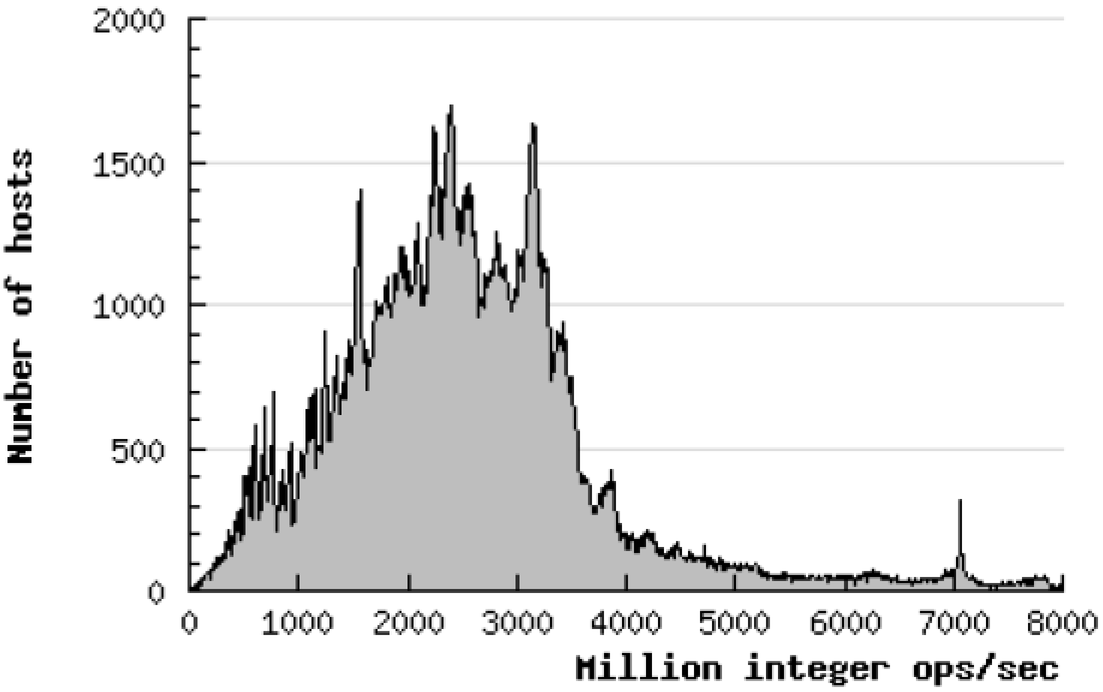


Figure 3.3: Dependence between number of hosts and number of integer operations per second[5]

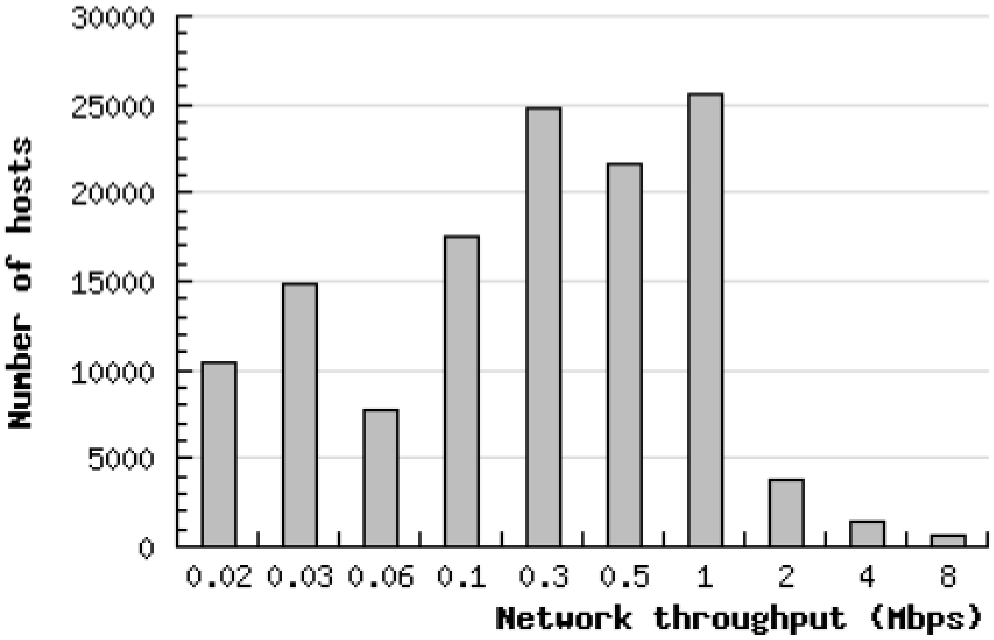
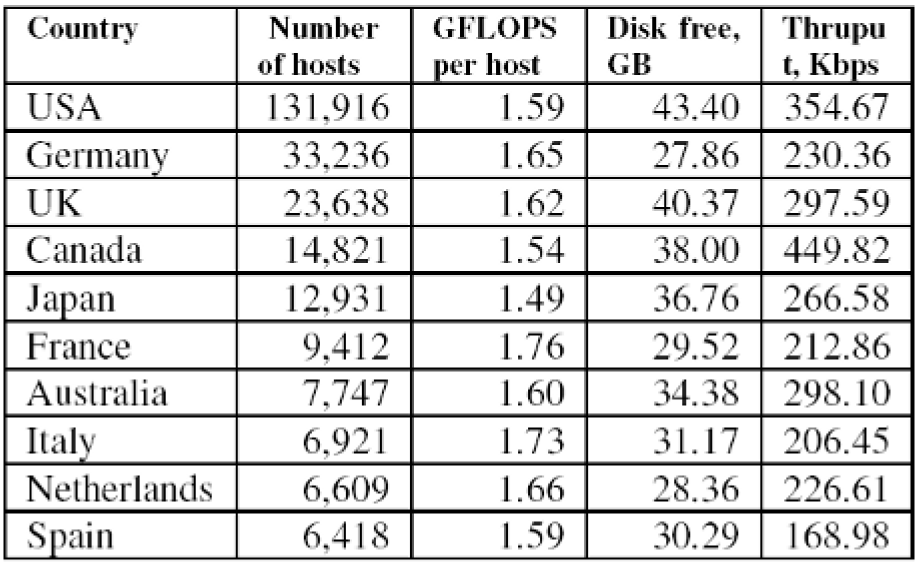


Figure 3.4: Dependence between number of hosts and network throughput [5]



Country	Number of hosts	GFLOPS per host	Disk free, GB	Thruput, Kbps
USA	131,916	1.59	43.40	354.67
Germany	33,236	1.65	27.86	230.36
UK	23,638	1.62	40.37	297.59
Canada	14,821	1.54	38.00	449.82
Japan	12,931	1.49	36.76	266.58
France	9,412	1.76	29.52	212.86
Australia	7,747	1.60	34.38	298.10
Italy	6,921	1.73	31.17	206.45
Netherlands	6,609	1.66	28.36	226.61
Spain	6,418	1.59	30.29	168.98

Table 3.1: Clients in different countires [5]

Chosen computers could become anonymous task or data servers. In this paper the name "pylon" has been used. The main task server could analyze the available statistics and make a decision of redirection a part of the work to such pylon. A part of database would be copied to pylon. The clients that are located in the nearby of the pylon would be informed of existence of such server. There could be many pylons depending on load on the standard servers or on the world coverage. They could be established or turned off dynamically. The name "anonymous server" comes from the fact that the user of the computer that has been chosen for such a pylon could even not know about it. This would be an additional protection from unfair clients. A user knowing that there is a pylon on his computer could try to falsify data in the database and make the connecting clients process data delivered by such user. This user could learn it by for example monitoring network traffic. It can be avoided by for example running periodical check of downloaded workunits (a random client checks if data sent by pylon is the same as date from the standard server).

The existence of pylons would decrease the severity of effects of turning off the standard server (clients still are able to download data from pylons), on the other hand it would sometimes lead to staling of many workunits. This situation could emerge when the pylon is turned off and returns to work after long period of time or not at all. It is a right of a user and therefore the choice of client who would be promoted to a pylon should be based on verified statistics. Additionally the standard task server should periodically check the availability of pylons under its management.

Of course to create a pylon a proper database is needed. Such base should be included in the client software so no external tools (for example MySQL) would be needed.

It is easy to notice that the improvements stated above are practical only for the task server. It is because the data server is dependent on the task server. Input files are split between the data servers rather than not replicated. From which server will the client download the data it is dependent on a workunit. Therefore monitoring of requests or making a pylon has no application here. It is worth considering though to change the orientation for data server in modifications concerning dividing clients by connection speed or sharing servers. There are more requests for task servers but it is the data servers that upload big files and the connection speed is also very important here. If to focus on data server then querying, checking speed and geographical clusterization would concern them, not task servers. A client after choosing a data server would send this information along with RPC requests to the task server. The task server would filter out those workunits which have input files from other data servers.

Improving the main server is rather meaningless. A client connects to it very rarely, so its main function is to gather statistics and create charts. Of course it is necessary to collect data about clients but this doesn't affect directly the system effectiveness.

4. JXTA platform

4.1 Introduction

The worth of direct connections between similar machines is well known in the technology for many years. In the year 1991 application named Napster was very famous. Its purpose was to share music. It was quickly brought down by law issues. Nevertheless the potential of such solutions has been noticed. Later there were other applications not only to share files but also for communication, games and widely understood cooperation.

This chapter contains short description of popular JXTA platform, used to create a new architecture for volunteer computing. It is the main part of this work.

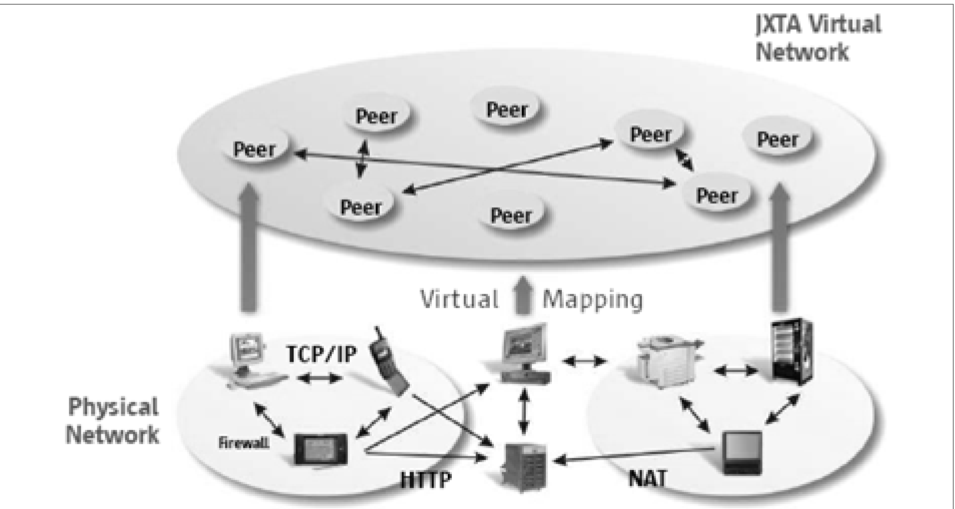


Figure 4.1: JXTA maps available nets into one logical virtual net [16]

P2P networks cannot be called an evolution. They are simply other type of systems. They have their specific usages where client-server architecture wouldn't be efficient. However P2P is not always the best solution, especially that completely distributed algorithms are often unreliable and difficult to implement. Nevertheless networks of this kind are the future in the rapidly developing world. Servers that serve millions of users can be a expensive bottleneck in every system. If to eliminate them from the communication path between two computers, then the following is obtained:

- system work speed up – some parts of the chain are removed
- better scalability
- expenses reduction
- ability to dynamically connect to the system
- better privacy
- bigger independence

Many systems with central management have reached their limits and sooner or later they will have to use some kind of distribution. One of such systems is BOINC [4].

The benefits of P2P have led to creation of many applications that use this mechanism. However they have divided the world to many subnets, unavailable for each other. If someone uses eMule, then usually doesn't use Kazaa because they are software that have similar functionality. Although they are similar they don't understand the other's protocols and cannot use the other's resources. Users are divided between those that like eMule and thos who prefer Kazaa or other kind of application (torrent, DC++ and others). Of course a user can work with many programs of this kind, besides there are hybrid solutions which understand many solutions. Nevertheless this situation is unnatural, since the file resources the same and should be shared in universal way for all applications.

The situated stated above leads to one conclusion: there should be a common communication platform for all P2P applications. Such need has been also noticed in Sun Microsystems. Its founder and main scientist dr Bill Joy has initiated project JXTA which was released in 2001. JXTA is "a set of simple, open P2P protocols which allow every device in the network to communicate, cooperate and share resources. JXTA nodes create a virtual, dynamic net on top of the existing nets, hiding their complexity" (picture 4.1 and [4]). JXTA platform is therefore an additional layer which uses available networks. The set of services and protocols offered by this platform allows easy development of P2P applications. JXTA is portable between hardware platforms and operating systems. It uses technologies like HTTP, TCP/IP and XML. There are implementations in Java, C, C++ and C#. There is about 80 projects that use JXTA and this number grows every year. One of these projects is platform for distributed computing developed also by Sun.

The easiness of using JXTA and constant development by Sun may someday make this platform a standard in this kind of communication. One of the arguments is lack of dangerous competition. .NET platform is also very popular but is neither free nor open. That's why the future of JXTA is promising.

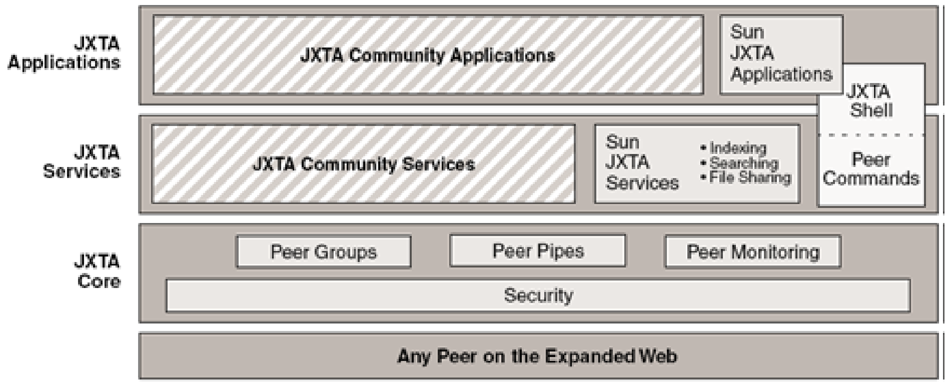


Figure 4.2: JXTA layers[14]

4.2 Functional elements

JXTA platform consists of tightly connected elements. These are:

Advertisement – a XML document. Different elements in JXTA network have their own Advertisements, for example nodes, groups, services. The Advertisements have strict structure. They can be published and kept in node's cache or send to rendezvous node. Advertisements are used to find the structure of a network and available resources. They are supported by Discovery Service. For more specifying queries Resolver Service is used.

Peer – is represents a single computer in JXTA network. A peer has its own Advertisement that contains its ID and list of services made available by this peer. For finding Peers Information Service is used.

Service – the functionality available on a Peer. Every application that uses JXTA most likely will define its own Services. Information about service is split into few advertisements: information of the service existence, of its interface and implementation.

Rendezvous Peer – it is a peer on which advertisements are published (more correctly: information where such advertisements can be found). A peer that looks for some functionality "meets" another peer (to be more exact: its advertisement) on a rendezvous peer.

Proxy Peer – a peer used for getting data from peers behind a firewall. If someone from outside would like to contact such peer, the firewall could not let him through. That is why it leaves information on proxy peer which is contacted periodically by the peer behind the barrier.

Peer Group – a set of peers. A peer can be in many groups. In each of them it has its unique ID (usually it's the same ID in many groups). Most of the communication (finding and publishing advertisements, querying) is made on the group level. A group has its own advertisement that contains its own ID and list of services which are made available by this group. A service is available in a group if at least one peer is able to do it. Connecting to a group may require peer authentication. A group also gives an implementation of standard services to its members, but it is usually the same implementation as in Net Peer Group.

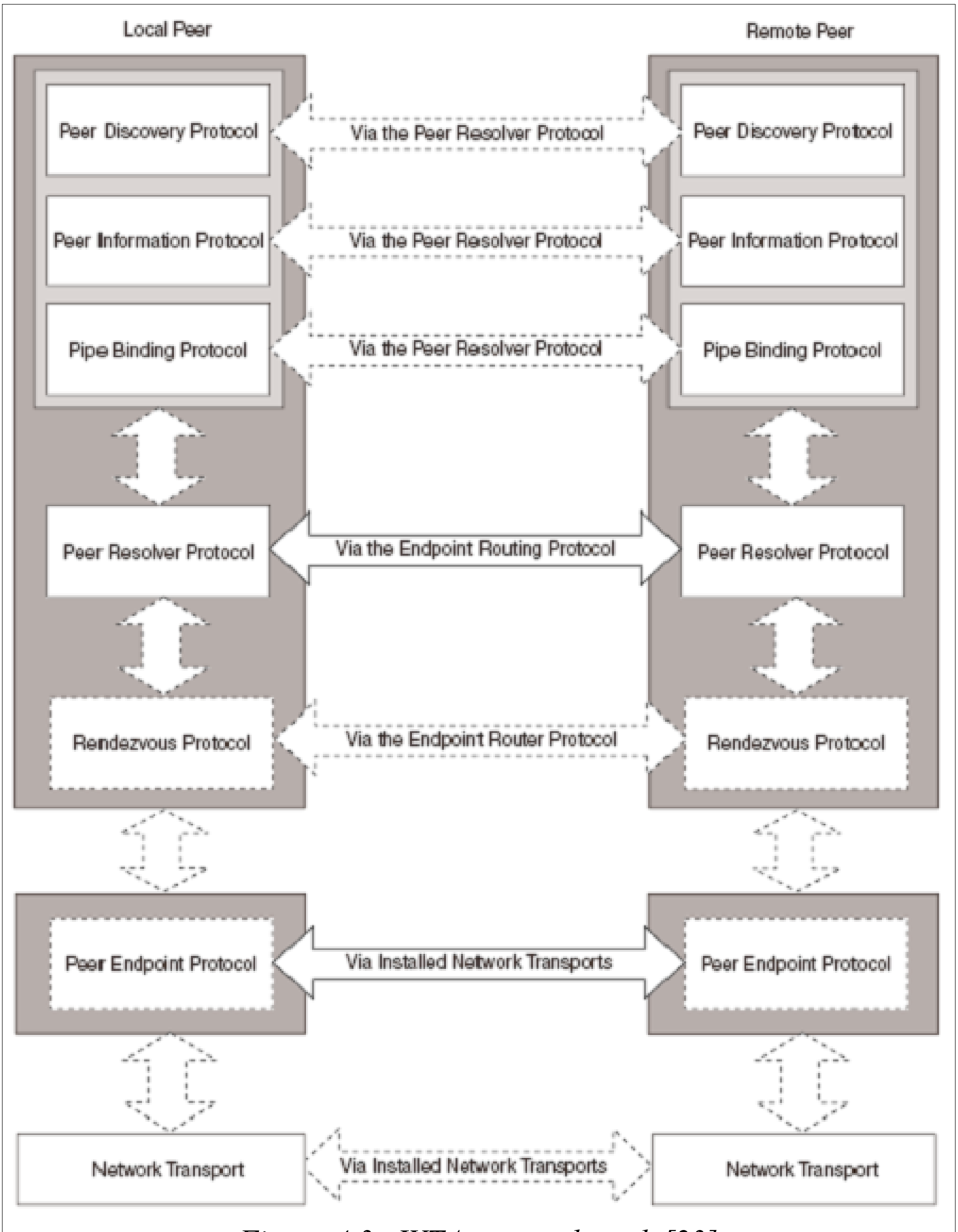

Local Peer
Remote Peer
Peer Discovery Protocol
Via the Peer Resolver Protocol
Peer Discovery Protocol
Peer Information Protocol
Via the Peer Resolver Protocol
Peer Information Protocol
Pipe Binding Protocol
Via the Peer Resolver Protocol
Pipe Binding Protocol
Peer Resolver Protocol
Via the Endpoint Routing Protocol
Peer Resolver Protocol
Rendezvous Protocol
Via the Endpoint Router Protocol
Rendezvous Protocol
Peer Endpoint Protocol
Via Installed Network Transports
Peer Endpoint Protocol
Network Transport
Via Installed Network Transports
Network Transport

Figure 4.3: JXTA protocol stack [23]

Net Peer Group – the group to which all peers belong. It contains the standard implementation of services. Most applications require creation of private groups, though.

JXTA platform is built on several layers as shown on picture 4.2. This platform has also its own stack of protocols. A programmer can use each of them depending what he needs but the protocols standing higher in the hierarchy are usually more convenient. The functionality of the protocols fulfils the promise given by JXTA developers: communication between peers is simple and effective. Messages sent by the protocols are written in XML. The stack consists of:

- Peer Discovery Protocol – for publishing resources and finding advertisements.
- Pipe Binding Protocol – for establishing communication path between two peers
- Peer Information Protocol - for sending queries to other peers and receiving answers
- Peer Endpoint Protocol – for finding network paths between two peers
- Rendezvous Protocol – for message propagation in network

Every protocol has been wrapped in a service. This means that instead building messages required by the protocol himself a programmer can use a convenient set of classes and methods which make most of JXTA functionality available to him. The service names correspond to protocol names. There is also the standard implementation of services available, but its utilization is not obligatory. A programmer can implement his own set of services. They must have the same functionality and use standard protocols. Most groups use the standard implementation, though.

4.3 Usage description

For better understanding JXTA platform it is worth studying a standard scenario in which its functionality is used. Lets assume that there is an application for sharing files which is based on JXTA. There are also peers that use this application.

1. **Connection to JXTA network**

On a computer the application is installed. During the first run it communicates with the user who sets the preferences for peer name, protocols and ports used for network communication and if he would like to become rendezvous and/or proxy peer. There are also few standard steps after which the computer is recognizable in the JXTA network and belongs to Net Peer Group.

2. **Finding peers with file sharing service**

The peer has file sharing service advertisement created by the application. It looks for other peers that have the same service. It may do it on the Net Peer Group or connect to some other group which advertisement is also delivered by the application. In order to find other peers it uses rendezvous service to connect to rendezvous peers. Then it uses Discovery Service to find desired peers.

3. **P2P communication**

After finding peers that use the same application the peer contacts them to find the list of their resources. It can be done by advertisements, Resolver Protocol or even lower level communication. In case of using Resolver Protocol the peer registers on the group level a Query Handler. It will process the queries and answers from other peers. Then it sends a query to other peers which have also registered Query Handlers. They respond with lists of available files. The peer chooses some files and with Binding Protocol creates a connection with the chosen peer. Then the file transfer commences.

4. **End of peer activity**

After exchanging file the peer unregisteres the Handlers and switches off. Its advertisement is still maintained on rendezvous peers for some time but other peers won't be able to contact it.

4.4 Remarks

JXTA platform is a step towards simple and effective communication between peers. It does have a good set of tools that allow using resources available in the network.

However while working with this platform many mistakes can be noticed. The documentation for many important elements hardly exists. The tools for XML processing, used mainly with advertisements, are difficult to utilize. Despite of supposed popularity of JXTA there are still no professional products based on it. Implementation of JXTA protocols still has many undocumented elements.

The next chapter describes an architecture based on JXTA. Implementation of communication in JXTA network seemed easy since the available tools have convenient methods. Unfortunately during the experiments it showed up how far from perfection is JXTA. Many initial solutions had to be changed. Despite of the promises from Sun and big hopes, JXTA is a platform which won't become a P2P standard to fast.

5. Pinion architecture

5.1 Introduction

BOINC architecture, despite of being a big success and having hundreds of thousands participants, has its imperfections and limitations. The imperfections are mainly connected with typical software development stages and experimenting with new solutions. The limitations come from mistakes in design. Creators of BOINC may have not expected such popularity of their project. Regardless of this fact the architecture is evolving mostly with patches rather than by complete rebuild. Developers from Berkeley institute must keep in mind that the current solutions someday will meet their limits. There won't be anything else to do but to redesign it thoroughly.

The specialists from Sun are already making a new architecture for volunteer computing based on JXTA [21]. This is a chance for BOINC to completely reorganize its structure and prepare itself for the future. Even BOINC developers mentioned that porting to this platform [1] is an option, but for now nothing more is known. It is possible though that in the upcoming years not only BOINC, but many other well known applications will move to JXTA.

The author of this paper created an architecture for distributed computing that uses JXTA. The project is closely related with the existing BOINC architecture. Most of its elements have been kept and they have similar functionality. Nevertheless, the method of fulfilling their functions, the organization and used tools change radically so they comply with JXTA standard. Moreover the project enables creation of hybrid architecture. There are possible modifications of the implemented system that allow the existing projects to continue their work without much additional work. This allows a smooth transition from one architecture to another.

The goal of porting BOINC to JXTA is not only to show a model how such processing can look like. It isn't only a presentation of JXTA elements either. The most important goal of the new system is to increase the effectiveness and remove most limitations of BOINC. The best test for verifying accomplishment of these theories would be executing the same job by BOINC and the new system. Although the architecture design is quite complete and explains most technical issues, the implementation is partial and shows its basic functioning without any optimization, checking all critical situations or implementation of additional elements (for example mechanism for awarding credits). These are the reasons why the test described in 7th chapter focus on the work of the new architecture and the real increase in effectiveness can be deducted from the used solutions. In this paper both used methods and their predicted impact on BOINC are described.

The author named the new system "Pinion" to distinguish it from the existing BOINC architecture. The reader must remember though that Pinion is really a porting of BOINC into the new platform while keeping its functionality. It should not be considered a completely different system, but rather a modification aiming at effectiveness increase and opening the way for further improvements.

The vocabulary used in the descriptions in the main part conforms to the terms used both in BOINC and JXTA. There word "client" used in BOINC has been translated into "peer" which has closer meaning to P2P networks.

First the present implementation of the architecture is described with additional notes. After learning its structure and principles of work conclusions for effectiveness increase are made. Then future plans for development are presented.

5.2 Outline of architecture elements

The designed architecture is a true porting of BOINC to JXTA. Below the architecture description “from bottom to top” is presented. By analyzing the elements the design of architecture is depicted.

Workfile – a standard physical file. Unlike in BOINC it can be downloaded also from other peers. For the downloading purposes it is divided into parts. Each part is downloaded separately, possibly from different peers. In addition to standard file parameters it is described by:

- calculated MD5 hash value
- status if it is a persistent file (used by many workunits)
- XML description

Result – it the same as in BOINC, but except the standard parameters it has:

- status if the peer that processes it should do the validation
- value – it corresponds to number of integer and floating number operations needed to process the result, similarly to the parameters used in BOINC. It can be taken “as is” for example as sum of those numbers. Another way of setting this value is giving all workunits the same values or proportional to their size
- XML description

Peer (in code: “PinPeer” as Pinion Peer) – it corresponds to client in BOINC and peer in JXTA. A peer is a computer with client software installed (it’s described in the next chapters). It realizes the functionality of the client: result downloading, processing, outcome reporting. Additionally it downloads files from other peers in the group, shares its own files, exchanges group parameters and can become the Super Peer.

It has been designed as a standard JXTA peer which means:

- it has its unique ID (PeerID)
- it has its advertisements (PeerAdvertisement)
- it can connect to a group (PeerGroup)
- it can become a Rendezvous Peer

A peer also has these elements common for all projects:

- hardware resources – limited by hardware configuration of the computer and user preferences
- modules that do the processing (client software) – a single application that manages all projects
- user preferences for the way of using the resources (implemented, not used) – limitations set for the degree of usage of resources like CPU or hard drive space.
- inner settings concerning work of the system – parameters of some inner mechanisms, for example maximal time for keeping an advertisement or socket timeout.

A peer has these elements specific for each project:

- parameters:
    - activity time in the project (runningTime) – the time since joining the project till now

    - sum of validated results values (valueSum)
    - fitness – it sets the usability of a peer in the project. It is a relation of the sum of the validated values to the client application running time.
    - membership in a work group and in a project group
    - status if the peer has been removed from the group (isRemoved). Removing from a group does not mean immediate leaving of the peer. It rather finishes its results, validates them and finds a new group first.
- results that are being downloaded, waiting to run, running or finished
- input files and their advertisements (workGroupFileadvertisement and projectGroupFileadvertisement). These files are obviously related to the owned results, but the peer manages them as a single resource – all of them are made available to other peers.
- responsibilities depending the Super Peer status – if a peer became the Super Peer, new functionality is made available to him

Work group (Workgroup) – it is a group both in JXTA meaning (peerGroup) and a set of peers that realize some functionality. The work group has these attributes:

- it was realized by the mechanism available in JXTA. It is recognizable as a PeerGroup, it has its own PeerGroupAdvertisement and is checked by the same authentication rules (in the current implementation connecting is controlled by the client application and not by the standard authentication mechanism).
- it is responsible for processing the whole workunit. Currently in BOINC a client would only download a result and after processing send its outcome to the server, where validation and assimilation commence. Work group is responsible for downloading a whole workunit and divide it to results for its members (peers). It is also responsible for processing, validation and sending whole outcome (calculated from all processed results) to the server.
- it has parameters which values must be known for other peers in the Group. They are stored in XML. Presently the parameters are:
    - maximum number of members in the group (maxGroupCount)
    - number of members in the group (groupCount) – more exactly: number of members without those, which status is "removed"
    - Group ID (PeerGroupID)
    - timestamp – it is not the creation time of the parameters, but a relative value, incremented each time the parameters were changed. For example if a peer takes a new Result, it increments the timestamp, rewrites updated group parameters and propagates them to other work group members. They read that the timestamp is higher then the one they own, so they accept the new parameters.
    - Super Peer ID (SuperPeerID) – it is the ID of one of the group members
    - list of group members IDs along with some of their parameters: fitness, list of results and possibly "removed" status.
    - Group fitness (groupFitness) – it is the average from fitness values of the group members. It reflects the usability of the group for the project. Variance of values that make this average should be as small as possible. If the group members have similar fitness, then the group fitness correctly depicts computational capabilities of the group. Basing on it, the group can be assigned a workunit that is properly

chosen for its capabilities.

- it always has one and only one Super Peer
- it is self-organizing – in the working group there should be peers that are as similar to each other as possible. The rules that regulate the group composition lead to this goal. The inspiration for this approach was self-organizing Kohonen net.

Super Peer – it is a normal peer but it also has some additional abilities and responsibilities. There is exactly one in each work group. Any peer can become the Super Peer by election. If there is only one peer in the group (if the group is newly created, for example), it becomes the Super Peer. A Super Peer has these attributes:

- it is a rendezvous peer both in the work group and the project group.
- during normal work of the system it is virtually the only peer that contacts the task and data servers – it vastly reduces the load on these servers. The exceptions for this monopoly of contacting the servers are:
    - if the downloaded file has its hash value other than it appears on the data server – the file is downloaded again, this time from the server.
    - reporting the validated outcome also omits the Super Peer (designed, not implemented)
- it downloads workunits for the whole group and assigns results on peers' requests – it "represents" the group and is responsible for the work inside the group. Obviously, it becomes the Transition Deamon.
- it should be the most stable of all group members. Super Peer shouldn't change often, because its election takes additional time and resources.
- in the current implementation Super Peer becomes the peer which fitness is closest to the average fitness of the group. It ensures long membership in the group – it won't be removed as the peer that differs the most from the other members. In the future version parameters like availability time or speed of connection to the servers (similarly to Kazaa or pylon that was suggested earlier) should be taken into account.
- each group member can become the Super Peer anytime. Having the current group parameters, each peer has all information it needs to become the Super Peer. It is also a requirement for the group parameters – they should be sufficient for a peer to become the Super Peer.

project group (projectGoup) – it is the group that all peers participating in the project belong to. All work groups inherit from it (they have the same implementation of Services). We can say that project group "contains" all work groups. This group is also a standard peer group in the JXTA meaning. On its level there are:

- communication of Super Peers with task and data servers
- persistent files sharing between Super Peers
- looking for the existing work groups

The project group advertisement is delivered to the peer at the moment of joining a project. The rendezvous peers inside this group are only Super Peers, servers and maybe some additional peers chosen by the project managers.

The project group doesn't do any important operations. It gives the peers common communication channel on the group level. For normal peers it is not very useful, but it is very important for the net structuring.

Net Group (NetPeerGroup) – the group that all peers belong to, not only in the Pinion architecture. Some communication is done on this group level.

Task and data servers – in the current implementation they are peers that work on the project group level. They don't belong to any work group. They are implemented as simple Query Handlers on the Discovery Service levels.

Deamons managing calculation redundancy – in BOINC they were usually situated on the task servers or any other computers. In Pinion the Transition Deamon works on the Super Peer, validator and assimilator on peers and file deleter is currently missing (the files stay at the server till the manual deletion).

5.3 Lifecycle description

For better understanding the Pinion architecture, apart from studying its elements, knowledge of their functionality is needed. The best way to do that is following the life cycle of a peer, starting from joining the first project.

1. **Running the client application**

   A user downloads the client application, installs on his computer and runs it. The program connects to JXTA network, registers as a peer and gets a unique ID. The modules responsible for operations of the whole peer are turned on. They stay inactive since the peer is not connected to any project.

2. **Joining a project**

In the final version the scenario is following – the client application:

- connects to the main server of the chosen project and registers
- creates the structure of folders required by the project
- receives list of task and data server IDs
- receives project group advertisement and joins it
- resets the parameters related with the project
- actualizes the XML file with the list of projects it belongs to

In the current implementation joining projects requires the existence of the main server and some interaction with the user, which are not needed to show the real work of the architecture. That is why used in the test are already past these steps, although they are capable of joining a new project (methods are implemented).

If the client application is switched off and later switched on, it will read all the data related to the project from the XML file. It contains description of owned Results, their processing time and state (finished, should be validated etc.) and other. This file is renewed with each important change and enables peer to be operational after a restart.

3. **Joining a work group**

The peer contacts Rendezvous peers which operate on the project group level. After connection it looks for work group advertisements. Lets assume that this is the firs peer in the project and no work groups exist. In the peer the timeout for query (sent on the Discovery Service levers) answers is reached. The peers assumes that there are no work groups or it is unable to contact them. It creates a new work group and publishes its advertisement. It creates standard work group parameters, elects itself for the Super Peer and starts working.

In the final version a minimal group count should be taken into consideration – the peer shouldn't download workunits if there is not enough peers to process it. This is related with the idea of "freelancers" which is described in the suggested improvements for Pinion.

Lets assume now that some other peer joined the project. It finds the work group advertisement created by the first peer. If the groups should be one peer only, then the other peer creates its own work group. If not then it will be accepted into the group. Since it is a completely new peer in the Project, it is impossible to find its usability (fitness) and therefore the similarity to other group members. This is the reason for which new peers in the project are accepted without any restrictions. The peer gets the current group parameters from the other peers and updates them by adding its own membership.

4. Downloading work

After joining the work group the peer asks for a Result. The principle for work download is that a peer should have at least that many Results how many are scheduled, but no less than one.

In the group there are no downloaded workunits. If the peer is the Super Peer, then it checks itself that it is unable to spawn a Result and has to connect to the task server. If the peer is not the Super Peer, the query will be send to the Super Peer which will act as if it needed the work itself. Then it will send the work to the peer that sent the query (Super Peer mediates between the peer and the task server). The query is send on the Resolver Service level. The task server checks group fitness and available workunits deadlines. If some of them can be processed by this group in time, it shifts it from the queue of available workunits to the queue of sent workunits. If the server receives the acknowledgement of workunit receiving, then it deletes the workunit. If it won't receive such acknowledgement, the next request for a workunit is fulfilled by the workunit from the queue of sent workunits (of course it is suitable for the querying group).

Now that there are some workunits available, the Super Peer checks them and if the number of spawned Results is less then required for quorum, it creates a new Result (just like the Transition Daemon) and sends it to the querying peer.

Independently of the fact that the querying peer was the Super Peer or a normal peer, it the Super Peer that updates the group parameters and publishes them inside the work group.

It is also worth noticing that practically with every communication between Super Peer and normal peers the time of this event is saved. It is because of unreliability of message transport in JXTA. Common peers also use every chance they get to update their information on the Super Peer availability.

5. Downloading files

The peer checks the availability of the files that are required by the workunit. If they are not available in the group (in our situation they won't be for sure: the workunit was just downloaded from the server) it contacts the data server and downloads the files.

The peer that received a Result informs its File Download Module of his needs. The Module checks the hard drive for existing file. For those which are missing it creates separate threads that download file parts. Then it looks for advertisements with file lists that come from the other peers. Each advertisement is parsed and if there are available parts of the needed files that aren't downloaded or being downloaded, it runs the file downloading thread. There is a limit for the number of simultaneously downloaded parts (so downloading files is not the main function of a peer) and for the number of parts that came from the same advertisement (so the parts are not downloaded from only one peer). Such thread establishes a connection with Input Pipe (InputPipe) which is included in the advertisement and sends a request for a specific file part (one connection – one part). Then the data transfer commences through the socket (JXTASocket), After receiving the part the check is run to verify that the size of the downloaded part is correct. If this was the last part of the file, all the parts are merged and the whole file is moved to the directory with ready files. Then the value of MD5 function is generated and the data server is contacted to compare the hash value of the file that is stored on the server. If the hash value is not the same the file is deleted and the File Download Module gets another request for the same file with annotation that it should be downloaded not from the peers, but from the data server. Current implementation also allows checking the file only by comparing the supposed and real size. It is also possible to turn off the checking completely.

After downloading a file part advertisement with updated file list is published. Other peers now know that this peer downloaded some parts which they might need, so they connect and start the download. There is a limitation on the frequency of publishing such advertisements (buffering).

Checking the hash value and possibly downloading the file from the data server are rare occasions when a common peer contacts the servers. It protects from the Super Peer cheating. It is the Super Peer that distributes the Results and it could prepare the messages in such way so the peers would process the files delivered by the Super Peer itself. The data server is assumed safe and that it is impossible to illegally switch the file there. So in case of inconsistency, a peer will contact the data server.

6. Running the Results

When the last file needed by the Result has been downloaded, the Result moves to runnable state. File Download Module notifies the Scheduler of the need for rescheduling. The Scheduler has been implemented similarly to the one in BOINC. It doesn't support suspending of Results and projects, though. The Scheduler considers all runnable Results and creates a schedule. After finishing, the work Download Module is run again to check if the number of the Results available for the peer is enough. If not, it downloads additional Results. After creation the schedule is enforced. It is also implemented similarly as in BOINC.

7. Result validation

Together with a new Result the peer gets the information if it's responsible for the validation. Such responsibility is placed on the peer that downloaded the last Result from the workunit. It increases chances that it will be the peer that will finish the Result as the last one. Is the peer does not have to validate, then after finishing the Result all it has to do is to schedule other Results. However if it is the one that should do the validation, it checks the number of peers that process this workunit. If there are no computational redundancy (only one Result), it displays the outcome. If there is more than one result, the peer sends a query on the Resolver Service lever to other peers that process the same workunit, asking them for the outcomes. The peer that receives the query and has the outcome, sends it. A peer that still doesn't have the outcome sends estimated time to finish. If a peer doesn't answer at all (it is turned off?), for its time to finish a standard value is taken. The validation peer chooses the longest time to finish and waits. Then it repeats the query. When finally it gathers all outcomes, a comparison is made. At this point the validating function given by the project should be used, but in the current implementation the function is simply a test for equality. In the final implementation in case of disagreement the Super Peer should be notified and it should generate a new Result and assign it to some peer. In the current implementation the validation result is simply displayed for the user. After a successful a report to the task server should be send (not implemented; it is a rare case when common peer connects to the server). After reporting an information of finished validation is send. After receiving it, the peers delete not persistent files related to this workunit. The validating peer updates the parameters and sends to the others.

8. Changing the group

After processing few workunits the statistics concerning the peer becomes more correct. First of all it is possible to check its fitness and decide if it should belong to this group. Leaving the group can be initiated by the peer itself (periodical seek of more suitable groups). If a new peer gains the membership, it can also trigger removal of another peer. If the group is full and the new member is more suitable for the group that one of the current members, it joins the group and propagates its membership. The Super Peer detects that the group is overcrowded and chooses a peer that is the least suitable for the group. Then is sends the peer the information of removal. The peer accepts it, but doesn't leave the group immediately. It doesn't download any more Results and it is not counted as a group member, though. The state of overcrowding is kept till the removed peer finishes all its results and they are validated. At this moment it leaves the group and seeks another one, more suitable. It downloads group advertisements and checks their parameters. The new group should have its average fitness similar to the peer's. If in the found group the difference between the peer's fitness and the average group fitness is bigger that for any other member of this group, this group is not taken into consideration (the peer would be a worse member then any of the current ones). Also if the difference is bigger then some fixed threshold (20%), the group is also not considered. If the peer found a proper group, it joins it and propagates its membership.

Periodical, self-initiated search for a new group leads to constant movement of members till the moment when the groups will become stable. Being stable means that a group doesn't change its members, because they fit well in it. Such mechanism of setting network topology has been inspired by Kohonen nets [13,17]. The aim is to make all groups stable, so the members change very rarely.

9. Leaving the project

This step finishes the life cycle of a peer in the project. It leaves the work group and the project group. This step has not been implemented yet, but is not of the highest importance to show the work of the Pinion architecture.

5.4 Overall schema of the client application

A part of elements that create Pinion is typical for JXTA. Most important ones, used in this project, has been described earlier. More exact descriptions can be found in many available books [7,15,9]. The description of the client application presented in this and the following chapters is lacking the details, but it helps to understand the work principles. The technical details can be found in the source code.

The system has a modular design. One of the goals was to make the modules as separated from each other as possible. Other modules can operate even in case of one of them fails. Most of the modules are implemented as separate threads, but their cooperation is indispensable for correct work.

Splitting the code to modules is rather logical than physical. A single module may consist of many classes and use other modules' resources (as few as possible). The modules in Pinion are:

<u>File Downloader Module</u> – responsible for:

- getting file requests from the other modules
- dividing requests into parts

- searching for advertisements with lists of available files
- contacting data servers
- establishing connections to other peers and creating bridges for data transfer
- downloading file parts
- merging parts into whole
- notifying other modules of new available files
- partially also for checking the Super Peer activity

File Publisher Module – responsible for:

- getting file requests from the other peers
- establishing connections to other peers and creating bridges for data transfer
- creating advertisements with file lists and publishing in the net

Runner Module – responsible for:

- keeping information on projects
- queuing results (CPU scheduler)
- enforcement of the schedule
- running and stopping results
- control over the results – sending messages to the user, waiting for results to finish

Group Management Module – it is not a thread, more a set of tools used by other modules. It is responsible for:

- creating new groups
- searching for existing groups
- joining a group
- checking and publishing group parameters
- calling Super Peer elections
- Super Peer functionality
- answering some queries concerning the group

Work Downloader Module – responsible for:

- queuing work download
- downloading work
- creating results
- validation results

Rendezvous Module – responsible for keeping the connection with Rendezvous peers.

Classes that are not separate modules, but are also very important for the architecture:

PinPeer – it is the central object, holds references to all modules. It is responsible for:

- connection to JXTA network
- running the modules
- joining a project (implemented, not used)
- some useful functions, for example for XML parsing

Project – represents a project the peer joined. It is responsible for:

- keeping the project parameters
- keeping the results
- joining groups at the moment of running the client application
- switching on and off the Super Peer mode

Result – represents a result and the corresponding system process. It is responsible for:

- parsing result description written in XML and on its basis building an object that is ready to run
- notifying the File Downloader Module of the need for files are required to run the result
- managing the result interruption or finishing (parameter updates, storing in XML, etc.)

Workfile – it reflects a file. It has some additional parameters

Internal settings – it holds the settings that are unavailable to the user, which influence the system work. They are for example socket timeout or maximum number of connection to server attempts.

User settings – it keeps the settings chosen by the user, which influence the system work.

Decoy – it is a test application that is used in the Workunits. It is configurable in many ways. It can be set how much work it should do, with what breaks, how much operational memory and hard drive space should it use and if it should do some I/O.

There are two additional objects that work separately from each other and the main core of the program. These are the task server (SchedulerServer) and data server (DataServer).

5.5 Benefits of Pinion architecture

After describing the Pinion architecture, the influence of the used solutions on effectiveness can be outlined. Most of the mechanisms has been redesigned so they try to fix the problems BOINC was coping with. Some of the benefits are not direct and obvious, but in a wider view they increase the value of the solution.

In this chapter only the benefits of implemented parts of the architecture are presented. Planned modifications have also great impact on the system. They were described in the next chapter. With them, the Pinion project is a complete solution for part of the limitations of BOINC.

- Porting to JXTA platform. It is not an obvious advantage for the BOINC effectiveness. Quite the contrary, BOIINC uses lower levels protocols and hence the overhead is smaller. The solutions were upgraded during the last few years so they reached some degree of perfection. Architecture was tested, the errors were detected and partially corrected. Porting to JXTA brings all the tuning back to the start. Inevitably, there will be some problems that the new architecture will have to cope with. Changing some mechanisms leads to necessity of testing them again. Besides, even JXTA itself is not a perfect product and its stack of protocols adds to the overhead.

Why port BOINC to this platform? Because it is the future of P2P networks and opens a vast range of new possibilities. If Sun will continue to promote this platform as it does now, then JXTA will become more and more popular. Sun is also building an architecture for distributed computing that uses JXTA. If everything goes smoothly, then in few years large part of all computers will be globally identified in the JXTA network. They will provide all kinds of services: file sharing, communicators, remote access to resources, participation in many different projects. The computers will not use different protocols that cannot be understood by each other. There will be no need for hybrid applications for example for downloading files from eMule, Kazaa and torrent. All this will operate on one level. It is worth noticing though that after the implementation and tests, it appeared that this platform is unfinished in quite big degree and still needs a lot of work till it can be called a standard.

The development of BOINC has proven the client-server structure limitations. Purchasing further hardware is always possible, but sometimes it is difficult for academic project for financial reasons. Widening the infrastructure leaves some solutions ineffective anyway. So complete reorganization of the structure, direct communication between the clients is possible and needed. JXTA delivers convenient mechanisms for P2P communication. They just have to be adapted to the needs of distributed computing. Even BOINC team was interested in JXTA platform [20,1]. Pinion architecture is an attempt to do what in the upcoming years will be done by the volunteer computing fans anyway. Despite of the skeleton implementation, the project presents the ease of using JXTA protocols and getting similar functionality that BOINC written in C++ has. The are new tools available (like automatic passing of the firewalls), which can be used for BOINC development. Messages from this system become readable for other JXTA-based system, what enables cooperation. Besides, JXTA is developed by large, professional company and we can hope that one day this platform will be finished in details.

Even if porting BOINC to JXTA does not seem to give short-term boost in effectiveness, in the future it is profitable.

- <u>Decreasing the task server load</u> – overload of MySQL database and net connection was a serious problem for the designers of projects that are based on BOINC. Developers tried to solve them by distributing the load on many servers (first in Folding@Home, then as a standard in BOINC projects) and by moving the database into RAM. Despite of these solutions the servers still have their limitations which will be reached in the future growth of the projects.

  Designed for Pinion, the system of work groups and Super Peers (partially based on Kazaa and FastTrack protocol) in great degree limits the number of connections to the task server. Current rules of result assigning made every client, that processed the same Workunit, connect to the data server and download the same data. Also later, during the reporting every result had to connect to the server again. Additionally, the validator usually worked on the same server.

  After redesigning, the whole workunit is downloaded by a group only once. In the group there are result spawning, processing and validation. Later, only one connection to the server is established to report the outcome. The idea of “pylons” (also in the form of a Super Peer), as described in the Pinion improvements, additionally decreases the load.

  Pinion architecture allows using the existing task servers.

- <u>Decreasing the data server load</u> – this issue is similar to the task server case, only this time the load is even bigger. Sometimes large files are downloaded from the server. Simultaneous transfer of large amounts of data effectively makes the server a bottleneck. The BOINC programmers have suggested P2P networks for file sharing.

In the Pinion architecture, downloading files required by a workunit is done by one peer only. Then the files are shared inside the group. They are divided and shared by parts like in eMule, what enables simultaneous download from many peers and increases the reliability of the mechanism. Moreover, the files that are common for many workunits can be shared in the project group.

Data server load is decreased significantly. It is worth noticing though that the server must support the queries for files' hash values – they are needed to validate downloads. The final number of requests stays the same, but amount of transferred data is vastly limited.

- Network self-organization – Pinion platform has a good solution for controlling the composition of groups. The goal is to create groups with peers of similar capabilities. This allows choosing proper workunits for a group. It limits the danger of missing deadlines. Also it enables synchronization inside a group and prevents leaving unfinished work for later, when slower peers become available. If the requirement is homogenous hardware platform of the members, clients needed for workunit processing will be gathered in one place. Otherwise they would be chosen one by one to satisfy the requirement.
- Shifting more work to peers – many current responsibilities of servers were moved to groups and peers.

6. Pinion architecture improvements

The improvements presented in this chapter add to the picture of the new architecture. In the previous chapters the basic implemented structure was presented. The real boost in the effectiveness can be observed after using the improvements described below.

6.1 CPU scheduler

The Pinion architecture in the current version has the CPU scheduler very similar to the one in BOINC architecture. The improvementsm that can be used here, were previously presented: running the CPU scheduler at the moment of checkpointing and utilizing partial outcomes. Especially the latter one can be used effectively. Partial outcomes are ditributed in one group only, so there is no danger of flooding the server with messages. Partial outcomes also improves the strategy of results distribution. A workunit is assigned to a group and inside it the result spawning, processing and validation are done. The task server sees the group as the whole, so the synchronization is important only inside the group. The server should see the group the same way as it sees a single client now. Processing of results should finish roughly in the same time. Using the peers' fitness to move them between groups helps to achieve this goal. Sharing information with the help of partial outcomes adds to the group "consciousness" and allows better synchronization, not mentioning other benefits of this mechanism.

6.2 Schedule enforcement

Further development of Pinion architecture in this aspect should be accomplished in few stages:

- Implementing the algorithm available in the current architecture. Currently Pinion uses a simple algorithm to find the needed amount of work, as described earlier. Even implementing the algorithm that uses shortfalls, stopping and suspending project is beneficial for the architecture. The description of this mechanism is available on the BOINC web site.
- Running the Work Downloader Module as a separate thread. This modification is also

taken from the existing BOINC architecture. In BOINC the work download is not a separate thread, but it at least it runs independently of the CPU scheduler work. If to implement the present BOINC algorithm for work download, Work Downloader Module also should be made independent.

- Omitting the Super Peer in assigning a result from a downloaded workunit. Currently, it's the Super Peer that assigns a result to a requesting peer. It is obvious in the situation when a completely new Workunit is downloaded, because only the Super Peer contacts the task server. But later, while spawning new results required for quorum, there is no need to involve the Super Peer. Any peer could create the result and broadcast this fact to the group. The only problem is the synchronization. The Super Peer processes the request one by one, but in the group it could be done simultaneously and two peers could spawn a result from the same workunit. There is a risk of doing unnecessary work. If the quorum is N and, after simultaneous spawn of the results, the number of peers that process the workunit is N+1, then exactly one should resign.

- Downloading workunits in forward. This modification is described in the chapter about the task server and the idea of using the Super Peer as a pylon. Here it is also useful. If the Super Peer ensures that there is always few different workunits available, then the peers could assign the results to themselves with lower risk they would choose the same workunit. This modification also requires changing the message with group parameters – currently a workunit taken by the group must be assigned to at least one peer. With the modification, it could be "free" and waiting for its first peer, so it should be described in group parameters separately. Spare workunits would also allow the improvement described below.

- Performing a simulation on the client. This paragraph evolves the idea of making simulations of downloaded results processing. It was already described in the previous chapters. That idea if far more useful to the Pinion architecture than to BOINC. The reason is locality of workunit assignment. If there are some free workunits in the group, and any of them can be taken by a peer, then the peer can make a simulation without any guessing or downloading the list of some available workunits (as was in the improvements suggestions for BOINC). The peer can make a simulation for each of the free, available workunits and choose the perfect one for it. Of course if the Super Peer is not omitted in the work distribution process, then the simulations can be done on the Super Peer, not the peer. This is a better situation anyway. Summarizing, only in the Pinion architecture making simulations is effective.

6.3 Outcome reporting

- Full implementation of validator and assimilator. Current functionality of these daemons is partial. Future versions of the system should be able to use a validating function provided by a project, to report correct outcomes to the server and to notify transition daemon on the Super Peer of the validation outcome. Reporting outcomes to the server should be done by the peer, not the Super Peer (so the assimilator can operate on the same peer as the validator). The reason for such solution is the possibility of the peers being cheated by the Super Peer. The Super Peer could get an outcome from a peer but report something completely different to the server. A protection from this kind of actions is the mechanism described below (random second validation). Besides, it is not a rule that only the Super Peer can contact the servers. Most important is that it will be done only by one group member for one report, so no redundant calls are made.
- Keeping other peers' outcomes. Small modification that does not require much effort to implement. If a validating peer receives information about deferring the validation, instead

of deleting the outcomes, it should keep them. When the validation attempt is repeated, then it only queries those peer which did not answer the last time.

- Utilizing partial outcomes. The presented earlier mechanism here has better application then in BOINC (similarly to CPU scheduler simulations). It is because of the locality of the messages: they omit the server. The partial outcomes exchange also will not increase the network traffic too much, and it will bring all benefits of this mechanism. Of course, it will be limited by the same rules that were described in the improvements suggestions for the BOINC architecture, for example the workunits should be long enough. In Pinion the utilization of partial outcomes will allow quick reactions for erroneous outcomes and peer synchronization.

  One might consider also a "lazy" version. Instead of getting partial results with a query on the Discovery Service level, the message can be send together (piggy-backed) with the group parameters. It will limit the network traffic, but there is a risk that the group parameters will be update too rarely. The groups belong to one project and are designed in such way, that the peers belonging to the same group process the same workunit in similar time. For example, there are 5 members in a group, 3 of them process the workunit A and the remaining 2 workunit B. Each peer has already downloaded the required files and started processing that will take long time (a month for example). In this situation the group parameters might be not exchanged for long period of time. Partial outcomes attached to the group parameters may reach the peers after finishing the whole workunit, what makes the useless. On the other hand, broadcasting the group parameters at the moment of getting a partial outcome is pointless and its better to use Resolver Service to do it.

- Choosing the slowest peer as the validating one. In this version of the system, the peer that is responsible for validation is the last one that received a result from particular workunit. It might happen though, that it is very quick and will process the result as the first one. It will start frequently repeating the queries for the outcomes from other peers. A better algorithm for choosing the validation peer is worth elaborating. For example if Super Peer sends the last result in a workunit, it asks other peers to report their progress (for example for how long the result was processed). Then, taking into account the peers' fitness, it chooses the one that will probably finish the processing last. Then it sends him message that this peer should become the validating peer for this workunit. The election does not have to be held by the Super Peer, but for example the peer that took the last result. If the choice is correct, the validating peer will send a message asking for the other peers' outcomes at the moment when all of them are already available, so there would be no message repetitions.

- Delegating other peer to validation. This improvement, same as the previous one, concerns choosing the peer that would finish the processing as the last one. In this case the solution is missing any speculations. The peer that is chosen for the validation at the moment of completing the result, sends a message asking for other peers' outcome. If one or more peers respond that they are still not ready and send the estimated time for completion, the validating peer chooses the longest one and delegates the responsibility for validation to the chosen peer. There is a high possibility that it will be the last one which gets the outcome and the request for other outcomes will be broadcasted only once more. This solution can give even better results when combined with the previous one.

- Repeated validation. The suggested improvements not only increase efficiency, but unfortunately also bring some risks. During the validation one of the peers can try to fake its outcome, so it is never the same as the other ones. The peers will not come to agreement and all the work will be in vain. Furthermore, the user of such peer can speculate for the algorithm of validation assigning and try to be always the validating peer. If he is successful, the whole work of the group is in vain. The situation is similar to the one in current BOINC architecture – users tried to fake the outcomes to get the credits or simply for malice. There must be some additional protection. If the "vicious" peer sends the outcome without validation, then this operation should be detected – the peers wait for the validation for some time (for example till the moment close to the deadline), then another peer does the validation. If it discovers some incorrect operations, for example that the outcome has already been reported, then it marks the peer that was responsible for it as untrustworthy. This can be interpreted in various ways – such peer can be never chosen for validation again or even be removed from the project. The punishment should be chosen with care because it might appear that the faulty behavior comes from communication problems, what happens more frequently on JXTA than in the typical solutions.

Another possibility of cheating is to call the validation, get the outcomes from other peers and then report some other outcome that is not the result of validation. Even if this does not bring any benefits to the cheating peer, these actions can be done only to slow down the project. The protection might be as follows. The validating peer along with the query sends its own outcome. Then every peer, after sending its outcome, checks if it should verify the validation. Peers don't do that very often, so it does not influence normal processing much. If the peer should check the validation, it sends to other peers processing the workunit (except the previous validation peer, so it does not detect the verification) a query for outcomes. When they answer, having all the outcomes, the peer does the validation once again. Then it compares the outcome with the one that was reported by the previous validating peer. It they do not match, it marks that peer as untrustworthy and stores the correct outcome on the server.

Unfortunately this way of protecting the outcomes enables another form of cheating. A peer could pretend that it checks the validating peer (which works correctly), then mark it as untrustworthy and send incorrect outcome to the server. This thinking is paranoid, but the situation might happen. This way of cheating should not be detected by another algorithm but rather with statistics. There should be an internal function that observes the peer behavior. If it will be similar to some illegal operations, it should be reported to the main server, and the peer be marked as untrustworthy. Just like the firewall which uses the statistics, not a single fact, detects sniffing in a local network.

6.4 Awarding credits. Calculation redundancy

As it was noticed in the BOINC improvements, credit awarding and calculation redundancy have to stay integral parts of the system and the changes will rather boost their accomplishment, rather that completely changing them.

The first step in the awarding system is implementing it in the Pinion. For the needs of this project this system was not necessary, but for the real system it is inevitable. Currently the credits are awarded for correct processing of a result and it can stay that way. The only modification will be that the points can be awarded to a group for the whole workunit. Then they will be distributed among the peers that actually processed the workunit. In other approach the credits could be awarded directly for the fitness – this parameter reflects the results of peer's work for the project. The ranking could be based not on the number of credits, but on the fitness. In BOINC there wis an important limitation that allows to award credits only for the results processed in time. This means that sum of the processed workunits' values (sumValue in the Project class) would increase not when the result is finished but when the correct validation is accomplished.

The barrier for slow computer owners who share a lot of resources is worth reminding. In that case the time spent for a project should be taken into account first and only then a factor such as fitness.

If it comes to calculation redundancy, then whole Pinion architecture seriously modifies the rules of this mechanism. Except for the changes of transition daemon, validator and the suggestion to relocate the assimilator to the validating peer, the operation of file deleter should also be altered. In the current version files that should stay on the hard drive are marked as “persistent”. They are never deleted. The other ones are deleted after getting the outcome. A better solution would be a special order to delete the persistent files. When the task server gets the outcome of Workunit validation which uses a file as the last one, it should append to next RPC requests the information that this file is no longer needed. This information should be broadcasted in groups. Peers that have the file should delete it. There should also be some limitations for sending such order. The task server does not have a list of groups that received the order so it should keep sending it to all groups for some period of time. The question is: for how long it should do it so all groups get the information and not overload the answers for RPC requests with useless entries.

6.5 The main, task and data servers

When creating the Pinion architecture, one of the goals was to leave the work of the servers unchanged, so hybrid architecture could be created. It is possible since the BOINC servers work with open formats based on XML. The current version of Pinion uses its own servers, except for the main server, but with little modification the presently used BOINC server could be utilized. This is a big advantage in future development of Pinion. The Pinion and BOINC users could operate simultaneously for some time, while their differences would be unrecognizable to the servers and to each other. On a single computer two client applications could be run and their work compared. Unfortunately, this kind of testing might not bring correct results – two systems operating in parallel decrease the efficiency of each other. Below the suggested modifications of Pinion servers are presented:

- Main server – its functionality can be slightly changed. Same as before, it should allow registration and keep the user statistics. The latter should be modified depending on the strategy of awarding credits. The main server should also provide the project group advertisement, task servers’ (possibly along with pylons), data servers’ and initial rendezvous peers’ addresses. The initial rendezvous peers' addresses are needed in case a peer looking for groups couldn’t find any. This means that in some proximity (in the meaning of hop count) no one belongs to Pinion and a peer with known address should be contacted. This situation can take place if there are only few work group advertisements and it is difficult to choose a suitable group for the peer. Another improvement would be selecting by the peer the closes rendezvous peer. It can be done by analyzing the IP or response time. This modification will increase the chance of finding groups in the nearby.

- Task server – in Pinion the task server is similar to a Super Peer and its structure is very simple. It is not difficult to adapt the existing server for the needs of Pinion. The code should be slightly changed, so a workunit that should be processed by few clients would be assigned only once – to a whole group.

- Data server – same as the task server, this server has been conveniently implemented just like the File Downloader Module in Pinion. Still, after small modifications, the existing servers can be used. They can be server taken from BOINC or simple FTP servers. It seems that the possibility of downloading parts of files from the data server is also a good idea, since this increases the reliability of such operation. It is worth to leave the current form of the Pinion data server, with additional modifications, as described in 3.6.

Most suggested modifications for BOINC architecture would fit also for Pinion. Additional changes should concern the strategy of workunit distribution. Their final form should depend on the range of improvements that were chosen from those which were suggested in this paper.

The idea of pylon is even more important now. Since the whole Pinion architecture is more distributed than BOINC then the idea of pylon should be seriously considered. In the next paragraphs a proposition of using a Super Peer as a small pylon is described.

6.6 File exchanging

File exchanging is more effective in Pinion than it was in BOINC, but some modifications could be added:

- Using other tools than JXTASocket. As [10] shows, the usage of JXTASocket class is not effective. It has too big overhead related to large amount of data that needs to be send before the real transfer begins. There are other solutions: standard Java socket or even FTP. Until the JXTASocket gains on effectiveness some other reliable approach to file sending should be used.
- Verifying checksums of file parts. Since some input files for workunit have up to few gigabytes, then downloading such file and by the end learning that the checksum is incorrect is a rather big loss for computers with slow connection to the internet. It is a good idea to get together few parts of the downloaded file and check their checksum. If an error be detected, the loss will be less serious. There can be also a system that downloads only the corrupted parts. Of course, if the file we are downloading is not the right file, we could be a victim of attempt of illegal usage of our resources.

  A disadvantage of such solution is the need of gathering few adjacent parts. With the current way of downloading parts it is not always possible right away. The parts can be downloaded in any order. There is no point in keeping checksums of all combinations of parts or in asking a peer to calculate such checksum for the chosen group of file parts. It can be used though, if the proper group of parts will be downloaded.
- Using a faster hash function. The MD5 hash is widely used in files authentication, but generating the value takes relatively much CPU time, which is valuable for Pinion. The main thing is to check if the downloaded is the same one as on the data server. A strong hash function would be useful for long workunits. If they are short, to a cheating user there is no point in making a file that has the same hash value, since the original file will be quickly processed and deleted. Besides the generated fake file would have low value for the user – the cheating is useful if the peers process important but fake data, not just any data. So any fairly strong hash function is sufficient for Pinion. Even if it lowers the security the hashing is not that important to sacrifice the high degree of effectiveness.
- Downloading few parts from a single query. Small, but important part to implement. Currently the input pipe from the File Publisher Module advertisement is used to establish many connections, one for each file part. It should be done differently: sending a whole list of needed file parts with only one request. Of course, this require adding some data that would indicate end of one part and start of another.

It is worth noticing that the improvements stated above fit perfectly the data server and it should be considered in future architecture development.

6.7 Super Peer

Despite of the fact that any group member can become the Super Peer, its proper choice and the range of responsibilities is very important for the system. The implemented election method is only partial and must be extended by additional elements. They are:

- Client application running time – a Super Peer should available as much as possible to prevent frequent elections. The perfect solution is a dedicated computer (always on), but not every group can have one. The statistics concerning time available for processing are already gathered by BOINC. There is also a very good idea for omitting the long periods of inactivity. If the computer was always on and once the user went for holidays for a month, his statistics should not be drastically lowered by this fact.
- Speed of connection to server – since a Super Peer is the only peer that contacts the servers, it must have a fast connection to the task server and even faster to the data server. If one of peers in a group has a faster connection than the others, it should count in the elections.

Moreover the idea of using a pylon, same as some other ideas, has a different meaning in Pinion. It is much more useful here. The peer that gathers few workunits (mini-pylon) can be the Super Peer itself. Workunit buffering allows less connections to the task server and higher group independency. Even when the Super Peer is off, the remaining peers would not be idle since each of them has full information on the workunits (this information should be a part of group parameters). This reliability distinguishes this idea from the idea of a separate pylon (see: improvements for BOINC). In that idea turning off a pylon brings the risk of missing the stored workunits deadlines. If the task server has crashed or is overloaded, a group could operate separately for some time, cumulating the outcomes. The group would send them all when the server is back to life. Also gathering the list of workunits descriptions allow simulation of results processing time (see 3.4). It could increase the effectiveness of result distribution.

The idea of taking the mini-pylon responsibility by a Super Peer requires some additional control. Super Peer can produce workunits itself and distribute them among the group members, making them work for the Super Peer. That is why the workunit batch, which is stored on a Super Peer, should be periodically compared to the ones stored on the task server.

6.8 Group members

The composition of a group changes. Below some improvements concerning groups and their members are presented:

- Group parameters – as already was mentioned, they have to be as small as possible, but sufficient for peer cooperation and synchronization and that any peer can become the Super Peer anytime. First of all there should be a function that creates one set of group parameters for sending and one for the peer itself. Such internal, not distributed entries are for example result running time or the information of its completion. These are not needed for the remaining group members. On the other hand, the parameters concerning the peer itself do not have to be stored in a file – they can always be recreated from the current state of Modules and other objects.
- Introducing buffering of group parameters changes – same as it is was done in the File Downloader Module. If a peer finishes a result, downloads a new one and is doing the validation, it should not inform about every of these facts separately. It should gather information about all these events and send it with one message. It is important to tune the time of deferring, so the parameters known by the group do not differ too much from those stored by the peer.
- Integrity of group parameters that come from different sources – If a peer receives group parameters with the same timestamp that have the parameters stored by the peer, it checks if all entries are the same. If not, it should combine the received parameters and the parameters stored internally. Currently it is implemented this way and it should be kept if a decision is

made to change the parameters' structure. Additionally, the parsing function should check what kind of differences are in the parameters and add only the necessary information.

- Tuning the member count – although the maximum member count can be set for each group, it should be fine-tuned. The suggested number is 2*N-1 members, where N is the average number of results in a workunit. In the most stable group some peers will be faster. If the redundancy is 3 and there are 5 peers, then 3 of them are processing workunit A and remaining two are processing workunit B. If one of these 3 finishes first, then it has a chance to catch up with the other two that process workunit B. The number of members should be correctly picked so if one peer is much faster than the others, it will not be processing next workunits alone but it will rather catch up with the other peers. The number of peers equal to the redundancy will paralyze a young group if one of the members will stay inactive for a long time. Such option can be considered only in a stable group, where it is known that even if a peer stays inactive, in the long period of time it will do the required amount of work. Too many members in a group will lead to member separation into subgroups. Its better to split it into two or more normal groups.

- Groups in the same local network – groups in Pinion architecture are logical groups, not geographical groups. It is worth considering the advantages and disadvantages of groups located in the same local area network. A basic criterion of accepting a member into a group is the difference between its fitness and the average group fitness. Computers that belong to the same local network can be very similar and create good groups (computer room in a company or a university). They can also be very different (academic network or in apartments). Nevertheless, there are some reasons to take such possibility into account. Peers in the same network:

- have very fast communication – no peer for forwarding messages are needed, there is no need for hop count limits or to check the latency. Everything is in LAN so all the data is delivered instantaneously.

- no need for proxy peers – the solution used in JXTA that allows communication with peers behind a firewall is a good one, but it increases the network latency. If the peers are on the same side of a firewall, there is be no need for such elements.

- new solutions for file sharing – in one network some specific solutions for file exchanging can be used. If a peer asks for files for the new result and it is possible to predict which peers will also need these files, it is possible to synchronize their requests and send the data to the broadcast address. If the net is based on a hub, all peers interested in the files will receive the data during only one transfer. It is especially useful for stable groups – the moment of finishing results is similar for all members.

Despite of the advantages stated above it's the group integrity (similar fitness) that should be the main criteria for choosing members. When a peer changes the group is should ping the available groups and check the closest ones, ideally in the same network. The querying should not be done during the initial parameters check (NewMembergroupParams), since gathering the parameters could take some time. The ping result will not be correct. Its better to use the standard "ping" command or to use a similar method in the JXTA Information Service.

- Changing the group – except the implemented triggers for changing the group there should be some additional ones:

- after few first results – the statistics become accurate and the peer capabilities can be checked

- if peer's fitness differs from the average for more than some border value (20%). It means that this peer should not be a member for sure and it should look for a more suitable group.

- "freelancers" – in Pinion there should be peers that are always available and help the groups that have insufficient members (less than the workunit redundancy). If the group is missing some members it may be unable to process a workunit. Idle waiting for additional peers means blocking valuable resources or piling unfinished workunits. That is why there should be some peers that help the groups to get the minimum member count. They might be provided by project creators.

An idea of cooperation of groups that are missing members also can be considered. There also could be a pseudo group made of peers that wait for other peers to connect. The peers would operate as if they belonged to a group, but for other peers they would seem free and waiting for connections.

7. Summary

Owing to an original idea and attractiveness, BOINC has reached great popularity – bigger than the designers expected. It was also a source of problems. BOINC has some limitations in design, which make it not completely scalable and decrease its effectiveness. They were presented in architecture description (chapter 2) and the most important were gathered in chapter 2.4.

This paper focuses on analysis of current solutions and suggestions for their improvement. Part of this work concerns analysis of the most important elements in BOINC. The author suggested various improvements and modifications that lead to solution of the problems (chapter 3). If they were introduced into the system, they would increase the effectiveness of processing and utilization of the available resources. They concern both algorithms and the system structure. They are not that complicated so the work needed for their implementation would not lower their value.

There are many systems for distributed computing – bigger or smaller. In each of these systems there are original ideas and interesting elements. There is no perfect system, though. BOINC has original approach to processing, which allowed it to gather resources with potential of supercomputers. Still, it has limitations that will not be passed by further algorithm tuning, as it is tried to be solved today. The experience from other programs should be used. For example, file sharing systems such as eMule or Kazaa - they have effective mechanisms for downloading and uploading even large files.

Most of this work is related to the step that BOINC developers might be thinking of: a complete redesigning BOINC in order to dismiss the limitations and increase the effectiveness. The author created a unique project of system (chapter 5 and later) that has the functionality of BOINC and is compatible in a high degree. The solutions are partially taken from the existing system, choosing the most useful elements. The architecture is very scalable and prepared for growth. It allows better utilization of resources and relieves the network from the traffic in some important locations. By increasing the degree of computations distribution and the independence of clients, it lowers the servers load. It solves problems of BOINC, what was the goal of this project.

This architecture is based on JXTA platform (chapter 4), which would soon become the P2P network standard. The new architecture uses standard JXTA protocols what enables future communication with other applications. JXTA is fully portable between hardware platforms and so is the new architecture.

The implementation of the suggested architecture is partial. Nevertheless, it allows observation of the used mechanism and proves their proper design. There are also many suggestions for modifications, which in a great degree increase the effectiveness of processing.

References

[1] ACM Queue, *A Conversation with David Anderson*, ACM Queue vol. 3, no. 6, July/August 2005 http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=313&page=1

[2] Anderson D. P., *BOINC: A System for Public-Resource Computing and Storage,* Space Sciences Laboratory, University of California, http://boinc.berkeley.edu/grid_paper_04.pdf

[3] Anderson D. P., *Public Computing: Reconnecting People to Science*, Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madrit, Spain, 17-19 November 2003, http://boinc.berkeley.edu/boinc2.pdf

[4] Anderson D. P., Cobb J., Korpela E., Lebofsky M., Werthimer D., *SETI@home: An Experiment in Public-Resource Computing,* Communications of the ACM, W: ACM, 1515 Broadway 17th Floor, New York 10036 USA, 10.2002, Vol. 45 No. 11, 56-61

[5] Anderson D. P., Fedak G., *The Computational and Storage Potential of Volunteer Computing,* Space Sciences Laboratory, U.C. Berkeley, INRIA, http://boinc.berkeley.edu/boinc_papers/internet/paper.pdf

[6] Anderson D. P., Korpela E., Walton R., *High-Performance Task Distribution for Volunteer Computing,* First IEEE International Conference on e-Science and Grid Technologies, 5-8 October 2005, Melbourne

[7] Brookshier D., Govoni D., Krishnan N., Soto J. C., *JXTA: Java™ P2P Programming*, W: Sams Publishing, ISBN: 0-672-32366-4, 22 March 2002

[8] de Zutter W., *BOINCstats: szczegółowe statystyki użytkowników, komputerów, zespołów i krajów w BOINC i wszystkich stowarzyszonych projektach*, 2007, http://pl.boincstats.com

[9] Gradecki J. D., Gradecki J., *Mastering JXTA: Building Java Peer-to-Peer Applications*, W: John Wiley & Sons, New York, NY, USA, ISBN: 0471250848, 2002 r.

[10] Jan M., *Performance Evaluation of JXTA Communication Layer*, PARIS Research Group, November 2004

[11] Milojicic D. S., Kalogeraki V., Lukose R., Nagaraja K., Pruyne J., Richard B., Rollins S., Xu Z., *Peer-to-Peer Computing*, HP Laboratories Palo Alto, HPL-2002-57, 8 March 2002

[12] Ramone B., *Lista projektów działających pod BOINC,* 2006, http://www.boinc-polska.prv.pl/

[13] Scholarpedia, *Kohonen Network*, http://www.scholarpedia.org/wiki/index.php?title=Kohonen_Network

[14] Sun Microsystems, *Project JXTA: An Open, Innovative Collaboration,* Sun Microsystems, Inc.,4150 Network Circle, Santa Clara, CA 95054, U.S.A., April 2001, http://www.jxta.org/

[15] Sun Microsystems, *JXTA v2.3.x: Java™ Programmer's Guide*, Sun Microsystems, 7 April 2005

[16] Sun Microsystems, *JXTA Technology: Creating Connected Communities*, Sun Microsystems, Inc.,4150 Network Circle, Santa Clara, CA 95054, U.S.A., January 2004, http://www.jxta.org/

[17] Taner M. T., *Kohonens self organizing networks with „conscience"*, Rock Solid Images, November 1997, http://zkiem.imir.agh.edu.pl/dydakt/neur/Kohonen.pdf

[18] Tyson H., *How the Old Napster Worked*, http://computer.howstuffworks.com/napster1.htm

[19] University of California, *Berkeley Open Infrastructure for Network Computing,* 2007, http://boinc.berkeley.edu/

[20] Vance A., *Sun and UC Berkeley are about to BOINC*, The Register, 17 grudzień 2003, http://www.theregister.co.uk/2003/12/17/sun_and_uc_berkeley

[21] Verbeke J., Nadgir N., Ruetsch G., Sharapov I., *Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment*, Sun Microsystems, Inc., Palo Alto, CA 94303, http://www.jxta.org/project/www/docs/mdejxta-paper.pdf

[22] Watson S., *How Kazaa Works*, http://computer.howstuffworks.com/kazaa.htm

[23] Wilson B., *Jxta*, W: New Riders Publishing; 1st edition, ISBN-10: 0735712344, ISBN-13: 978-0735712348, 15 June 2002