Porting *PHYLIP* phylogenetic package on the Desktop GRID platform *XtremWeb-CH*

Nabil ABDENNADHER[a], Regis BOESCH[a]
[a]*University of Applied Sciences, Western Switzerland*

Abstract. This paper describes the parallelization (gridification) of the phylogenetic package *PHYLIP* on a desktop GRID platform termed *XtremWeb-CH*.
PHYLIP is a package of programs for inferring phylogenies (evolutionary trees). It is the most widely-distributed phylogeny package. *PHYLIP* has been used to build the largest number of published trees. It's known that some modules of *PHYLIP* are CPU time consuming; their sequential version can not be applied to a large number of sequences.
XtremWeb-CH (XWCH) is a software system that makes it easier for scientists and industrials to deploy and execute their parallel and distributed applications on a public-resource computing infrastructure. Universities, research centres and private companies can create their own *XWCH* platform while anonymous PC owners can participate to these platforms. They can specify how and when their resources could be used. The objective of *XWCH* is to develop a real High Performance Peer-To-Peer platform with a distributed scheduling and communication system. The main idea is to build a completely symmetric model where nodes can be providers and consumers at the same time.
In this paper we describe the porting, deployment, and execution of some *PHYLIP* modules on the *XWCH* platform. The parallelized version of *PHYLIP* is used to generate evolutionary tree related to HIV viruses.

Keywords. Phylogeny, *PHYLIP*, Grid, *XtremWeb-CH*.

Introduction

It is commonly accepted that contemporary genes, genomes, and organisms evolved from ancestors under the influence of natural selection. Consequently, the knowledge of the evolutionary tree behind their origin is crucial for understanding these entities. Knowledge about the relationships within gene families plays an important role in understanding, for example, the origins of biochemical pathways, regulatory mechanisms in cells as well as the development of complex systems. For example, knowing relationships between viruses is central for understanding their ways of infection and pathogenicity.

In a medical context, the generation of a life tree for a family of microbes is particularly useful to trace the changes accumulated in their genomes. These changes are due, inter-alia, to the "reaction" of viral strains to medical treatments.

In this context, computer applications dealing with the reconstruction of evolutionary relationships of organisms, genes, or gene families have become basic

tools in many fields of research [1, 2, 3, 4]. These applications "reconstruct" the pattern of events that have led to "the distribution and diversity of life". These relationships are extracted from comparing Desoxyribo Nucleic Acid (DNA) sequences of species. An evolutionary tree, termed life tree, is then built to show relationship among species. This tree shows the chronological succession of new species (and/or new characters) appearances. The majority of reconstruction methods of evolutionary trees optimize a predefined objective function. Thus, a given tree can easily be evaluated. The "optimal" tree is the one which is supposed to be the most "realistic" one.

The problem of finding an optimal evolutionary tree has been shown to be NP-complete for a quite number of reconstruction methods. In order to reduce the computational burden and to limit the vast number of trees to be examined, heuristics have been suggested: stepwise insertion with local and global optimizations [5], the Quartet Puzzling algorithm [6], star decomposition [7], etc. Recently, Bayesian approaches [8], genetic algorithms [9], and simulated annealing [10] have entered the field. However, approximate and heuristic methods do not solve the problem since their complexity remains polynomial with an order greater than 5: $O(n^m)$ with $m > 5$. Parallelization of these methods could be useful in order to reduce the response time of these applications.

The most widely-distributed phylogeny packages are *PHYLIP* [11] and *PAUP* [12]. These packages have been used to build the largest number of published trees. This paper deals with the parallelization of a sub-set of modules implemented by the *PHYLIP* package. It particularly describes the parallelization of the heuristic reconstruction method Fitch (proposed as a module in the *PHYLIP* tool).

The targeted machine is a network of computers equipped with the *XtremWeb-CH* (www.xtremwebch.net) middleware. The *XtremWeb-CH (XWCH)* project aims to build an effective Peer-To-Peer (P2P) System for CPU time consuming applications. Initially, *XWCH* is an upgraded version of a Global Computing environment called *XtremWeb (XW)* [13]. Major improvements have been brought in order to obtain a reliable and efficient system. The software's architecture was completely re-designed. The communication routines based initially on Remote Procedure Calls (Java RMI) were replaced by socket communications. New modules were added in order to enrich the system by new functionalities.

A typical *XWCH* platform is composed of one coordinator and several workers (remote resources). The coordinator is a three-tier layer allowing "connection" between the users and the workers.

XWCH supports three new features which, from our knowledge, do not exist in similar "prototypes": support of communicating tasks, direct communication between workers and granularity and load balancing management. These features are described in [25, 26] and will not be detailed in this paper.

This document is organized in 5 sections. After the introductory section, section 1 presents the sub-set of the *PHYLIP* modules that was ported on *XWCH*. Section 2 describes the different components of the *XWCH* middleware. Section 3 presents the gridification of *PHYLIP* on *XWCH*. Section 4 presents some experiments carried out in order to evaluate the proposed gridification. Finally, section 5 gives some perspectives of this research.

1. PHYLIP

PHYLIP (the PHYLogeny Inference Package) is a package of programs for inferring phylogenies (evolutionary trees). Developed during 1980s, *PHYLIP* is one of the most widely-distributed phylogeny packages. It has been used to build the largest number of published trees. *PHYLIP* has over 15,000 registered users. The package is available free over the Internet, and written to work on as many different kinds of computer systems as possible. The binary and source code (in C) are distributed. In particular, already-compiled executables are available for Windows, MacOS and Linux systems.

Methods that are available in the package include parsimony, distance matrix, and likelihood methods, including bootstrapping and consensus trees. Data types that can be handled include molecular sequences, gene frequencies, restriction sites and fragments, distance matrices, and discrete characters.

Five modules were ported on *XWCH*: *Seqboot, Dnadist, Fitch-Margoliash, Neighbor-Joining* and *Consensus*. Input data of these modules are nucleotide sequence data (DNA and RNA) coded with an alphabet of the four nucleotides *Adenine, Guanine, Cytosine,* and *Thymine*. Each nucleotide is denoted by its first letters: *A, G, C* and *T*. Every nucleotide sequence belonging to the input data is a leaf node of the evolutionary tree to be constructed.

The evolutionary tree is composed of several branches. Each branch is composed of sub-branches and/or leaf nodes (sequences). Two sequences belonging to the same branch are supposed to have the same ancestors. To construct the tree, the application defines a “distance” between all pairs of sequences. Evolutionary tree is then gradually built by sticking to the same branch, the pairs of sequences having the smallest distance between them. Even if the concept is simple, the algorithm is a CPU time consuming. This complexity is due to two factors:

1. Methods used to group sequences into branches are complex. As an example, the *Fitch* program, one of the most used methods, takes two hours to execute on a Pentium 4 (3 GHz) with 120 sequences.
2. The application constructs not only one tree from the origin data set, but a set of trees generated from a large number of bootstrapped data sets (somewhere between 100 and 1000 is usually adequate). These data are randomly generated from origin data. The final (or consensus) tree is obtained by retaining groups that occur as often as possible. If a group occurs in more than a given fraction of all the input trees, it will definitely appear in the consensus tree.

Seqboot is a general bootstrapping and data set translation tool. It is intended to generate multiple data sets that are re-sampled versions of the input data set. It involves creating a new data set by sampling *N* characters randomly with replacement, so that the resulting data set has the same size as the original, but some characters have been left out and others are duplicated.

Dnadist uses sequences to compute a distance matrix. It generates a table of similarity between the sequences. The distance, for each pair of sequences, estimates the total branch length between the two sequences, it represents the divergence time between those two sequences.

Fitch-Margoliash (Fitch) and *Neighbor-Joining (NJ)*: These two programs generate the evolutionary tree for a given data set. *Fitch* method is a time consuming method. Its sequential version can not be applied to a large number of sequences.

Consensus: This program constructs the consensus tree from the collection of intermediate trees generated from bootstrapped data sets.

The application, as developed, has two parameters (fed by the user): the set of nucleotide sequences from species under investigation and the number of replications. The higher is the replication, the finest is the result.

2. XtremWeb-CH

The majority of Global Computing (GC) projects adopted a centralized structure based on a Master/Slave Architecture: BOINC [14], Entropia [15], United Devices [16], Parabon [17], XtremWeb [13], etc. A natural extension of the GC consists on distributing the "decisional degree" of the master in order to avoid any form of centralization. Thus, architectures such as Clients/Servers and Master/Slaves would be withdrawn. This concept, known as Peer-To-Peer, was successfully used to share and exchange files between computers connected to Internet and broadcast micro-news among internet users. The most known projects are BitTorrent [18], eDonkey [19], Kazaa [20], Gnutella [21], Freenet [22] and FeedTree [23].

XtremWeb-CH (XWCH) is composed of four modules: coordinator, worker, warehouse and broker. The coordinator module is the main component of *XWCH*. It is considered as the master of the *XWCH* system; it has the responsibility of managing communication between the clients (users) and the workers (resource providers).

The worker module is installed on each provider node. It manages execution of tasks and the transfer of data from/to the worker. Workers are considered as the slaves of the *XWCH* system.

A broker module is a “compiler” which transforms the user request (application submission) into a set of tasks compliant to the “format” recognized by *XWCH*. Every family of applications has its own broker. The *XWCH* broker module can be compared to the Globus broker which is responsible of transforming a high level RSL (Request Specification Language) request into a low level RSL request [24].

2.1. The coordinator

It is a three-tier architecture which adds a middle tier between client and workers. The coordinator accepts execution requests coming from clients, assigns the tasks to the workers according to a scheduling policy and the availability of data, transfers binary codes to workers (if necessary), supervises task execution on workers, detects worker crash/disconnection and re-launches tasks on any other available worker. The coordinator is composed of three services: the workers’ manager, the tasks’ manager and the scheduler.

2.1.1. The Workers’ Manager

The workers’ manager maintains a list of connected workers. It receives four types of *common requests/signals* from the workers: *Register Request (RR), Work Request (WR), Life Signal (LS)* and *Work Result Signal (WRS)*. The *Register Request* allows a worker to subscribe nearby the coordinator. When the Workers’ Manager receives a *Work Request*, it searches for the most appropriate task [25] to be assigned to the

concerned worker. During the execution of the task, workers send *Life Signals* to the coordinator to inform about their status. When a worker finishes its execution, it sends a *Work Result Signal* to inform the coordinator about the location of the data it has produced.

2.1.2. The Tasks' Manager

A parallel and distributed application is composed of a set of communicating tasks whose structure is described in [25] and [26]. A task is considered to be "ready" for execution if its input data are available. It is in "blocked" status if its input data are not yet available. Two lists are maintained by the Tasks' Manager: *blocked tasks* and *ready tasks*. When receiving a *Work Result Signal*, the Tasks' Manager checks whether the new available data correspond to input data awaited by one or several blocked tasks; it updates the lists of blocked and ready tasks accordingly.

2.1.3. The scheduler

A Work Request transmits, as input parameter, the performance that can be delivered by the concerned worker. When receiving this request, the coordinator launches a scheduler module which selects the "most appropriate" ready task to be allocated to that worker. The concept of "most appropriate" is detailed in [26].

2.2. The workers

The worker module includes two components: the activity monitor and the execution thread. The activity monitor controls whether some computations are taking place in the hosting machine regarding parameters such as CPU idle time. The execution thread extracts the assigned task, starts computation and waits for the task to complete.

2.3. The warehouses

XWCH supports direct communication between workers executing two communicating tasks. Direct communication can only take place when the workers can "see" each other. Otherwise (one of the two workers is protected by a firewall or by a NAT address), this kind of communication is impossible. In this case, it is necessary to pass by an intermediary: *XWCH* coordinator for example. However, to avoid overloading the coordinator, one possible solution consists of installing "warehouse" nodes which acts as an intermediary. These nodes are used by workers to download input data needed to execute their allocated task and/or upload output data produced by the task. A warehouse node acts as a repository or file server. It must be reachable by all workers contributing to the execution of a given application.
The protocol is the following:

- The list of available warehouses is received by a worker when it registers nearby a coordinator (*Register Request*)
- When a worker finishes the execution of a task it uploads its result in a one of the known warehouses (selected randomly). Thus, the result is stored in the worker and in the warehouse,

- The worker sends a *work result* signal to the coordinator with the two locations (IP address and path) of the result produced by the given task,
- When a worker sends a *Work Request* to execute a new task, it receives as a reply, the binary code of the allocated task and the two locations of its input data.

3. *PHYLIP* Gridification

The "gridification" is the process of parallelizing and/or porting a High Performance application on a Grid platform. The gridification should take into account several constraints linked to the targeted Grid platform: volatility and heterogeneity of nodes, limited bandwidth of the network, etc.

This section describes the gridification of five modules of *PHYLIP*: *seqboot, dnadist, Fitch, NJ* and *consensus* on the *XWCH* middleware. Communications between tasks are based on file transfers.

As stated in section 1, the application, as developed, has two parameters (fed by the user):

1. set of nucleotide sequences from species (or viruses) under investigation. In the reminder of this paper, the number of sequences is noted by *s*.
2. Number of replications (*r*): used to produce multiple data sets from original DNA sequences by bootstrap re-sampling. The higher is this number, the finest is the result.

The structure of the obtained parallel/distributed application is shown in Figure 1.

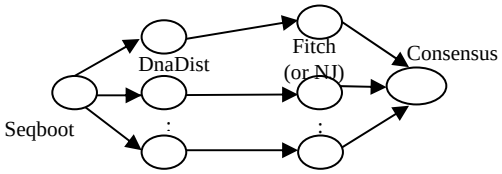


Figure 1. Data flow graph of the modules *SeqBoot, DnaDist, Fitch/NJ* and *Consensus*

The *Seqboot* task generates a multiple data sets. Each of these data is used by a *DnaDist* task to generate one distance matrix. This matrix is then used by a *Fitch* (or *NJ*) task to generate an intermediate evolutionary tree. Finally, the consensus task constructs the evolutionary tree from the intermediate trees. As explained in section 2, the *Fitch* module is time consuming ($O(n^5)$). This is not the case of modules *Seqboot, DnaDist, NJ* and *Consensus* modules.

In order to apply the *Fitch* module to a large number of sequences, a parallel version of this package was designed and ported on *XWCH*. The data flow graph of the parallel implementation of the *Fitch* module is given in Figure 2. Each *Fitch* node in Figure 1 is thus replaced by the graph of Figure 2.

The evolutionary tree is a non-root tree represented by two sets of nodes:

External (or leaf) nodes (square nodes in Figure 2): They represent the sequences under investigation. An external node is always linked to one internal node. When the evolutionary tree is completely constructed, the number of external nodes is equal to *s*.

Internal nodes (circle nodes in Figure 3) are virtuals, they don't represent sequences. Each internal node is linked to exactly three other nodes (internal or

external). When the evolutionary tree is completely constructed, the number of internal nodes is equal to *s-2*.

The evolutionary tree is generated progressively. The *Fitch* algorithm starts by creating a tree with one internal and three external nodes. In each step, the method inserts one sequence (external node) in every possible branch of the already constructed tree, and evaluates an objective function (*Test_Branch* tasks in Figure 2). The selected branch is the one that minimizes a pre-defined criterion *F* (*Best_Topology* tasks in Figure 2). In addition to the external node inserted in each step, an internal node is also created and inserted in the same step. This process is repeated until the insertion of all the sequences. The last step contains 2s – 5 "*Test_Branch*" tasks.

Thus, the number of "*Test_Branch*" tasks for one parallel *Fitch* is $O(s^2)$, *s* being the number of sequences. Since there are a maximum of *r* (*r* = number of replications) *Fitch* tasks, the maximum number of *Test_Branch* tasks is $O(r*s^2)$. The maximum number of parallel *Test_Branch* tasks that could be executed at the same time is equal to: *r*(2s-5)*. The execution time of a "*Test_Branch*" task increases with the size of the evolutionary tree.

4. Experiments

This section presents some performance analysis regarding the gridification of the package *PHYLIP*. Our results demonstrate the performance of the system and highlight promising areas for further research. The objective of these experiments is to validate our approach. They are not carried out to prove that the system delivers a maximum power for a given execution: the project's challenge is to extract, at low cost, a reasonable computing power from a widely distributed platform rather than extracting the maximum power from a local supercomputer or a dedicated GRID platform.

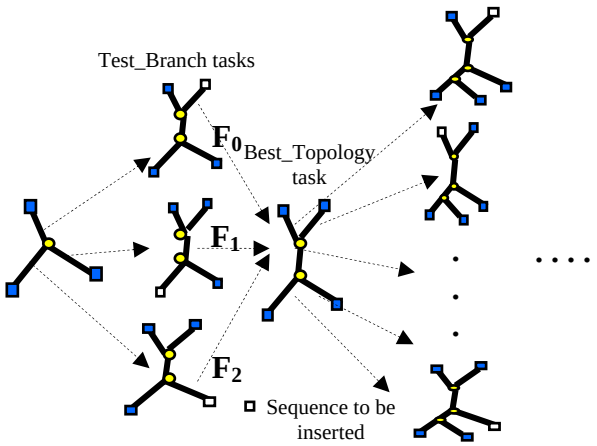


Figure 2. Data flow graph of a parallel *Fitch* task

The parallelized version of *PHYLIP* is used to generate evolutionary tree related to HIV sequences. The application is used by the virology laboratory of Geneva Hospital.

In this context, one needs to keep in mind that the number of sequences *s* can vary from 100 to 300 while the number of replications *r* varies from 100 to 1000.

A specific broker (web service) was developed in order to allow a dynamic configuration of the application regarding the current state (number and performance of the workers) of the platform: number of "*Fitch*" tasks and number of trees generated by each "*Fitch*" task, etc.

The experiments detailed in this section do not implement the parallel version of *Fitch* (Figure 2). They corresponds to the application represented if Figure 1. Executions were carried out on a platform with one coordinator (Linux OS), 250 heterogeneous windows workers ranging from Pentium I to Pentium IV, and 2 warehouse nodes. The workers are geographically located in two different places (Engineering Schools of Geneva and Yverdon). During execution, the 250 workers are used by students; they are often switched off or disconnected.

In order to evaluate the performance of the load balancing strategy implemented by *XWCH*, two versions of *PHYLIP* were deployed on the platform: the first version (*Version 1* in Figure 3) is composed of *r Fitch* tasks. Each task processes one tree. In the second version (*Version 2* in Figure 3), the number of *Fitch* tasks and the number of trees generated by each *Fitch* task are processed depending of the state of the platform (number and performance of workers).

Execution times consumed by the two versions are shown in Figure 3. The difference of execution times in Figure 3 is due to the synchronization between the coordinator and workers.

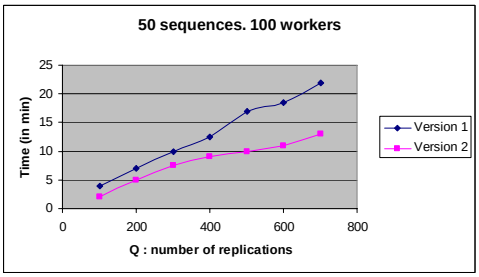
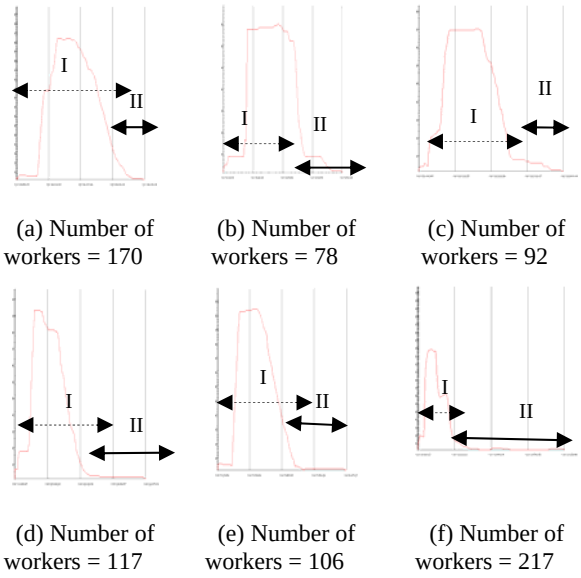


Figure 3. Execution times of *PHYLIP*

Figure 4 illustrates the total number of parallel tasks during the execution of the application. Since the "*Fitchs*" are the most time consuming tasks, this study focuses on the number of these tasks.

Steps I correspond to the execution of the *Fitch* tasks which finish, in general, at the same time. However, some *Fitch* tasks finish their execution later (step II in Figure 4). This is due to at least to one of the following factors:

1. The workers disappear during the execution,
2. As it is implemented today, workers' performance is only represented by the CPU power (CPU frequency). This model is not realistic; the system should take into account other criteria such as main memory, processes, applications and services installed locally on the workers, etc.



(a) Number of workers = 170 (b) Number of workers = 78 (c) Number of workers = 92

(d) Number of workers = 117 (e) Number of workers = 106 (f) Number of workers = 217

Figure 4. X-coordinates: Time, Y-coordinates: Total number of parallel executing *Fitch* tasks.

6. Conclusion

This paper presents the gridification of a sub-set of modules of the phylogeny package PHYLIP on the Large Scale Distributed platform XtremWeb-CH (XWCH). XWCH is a GC environment used for the execution of high performance applications on a highly heterogeneous distributed environment. This middleware can support direct communications between workers, without passing by the coordinator. A scheduling policy is proposed in order to minimize synchronization between coordinator and workers and optimize load balancing of workers.

The porting of *PHYLIP* on *XWCH* has demonstrated the feasibility of our solution. The next step consists of adapting the granularity of the parallel version of Fitch. Two parameters should be fixed according to the state of the targeted platform:

1. Number of parallel Test_Branch tasks executed during the insertion of one sequence.
2. Merging of several Test_Branch and Best_Topology tasks into one task according to the number of sequences.

The current version of *XWCH* allows the decentralization of communications between workers. The next step consists of designing a distributed scheduler. This scheduler shall avoid allocating communicating tasks to workers that can not reach each other and/or not belonging to the same “domain” (Local Area Network for example). This approach offers a strong basis for the development of distributed and dynamic scheduler and could confirm and reinforce the tendency detailed in section 2.

7. References

[1] http://biowulf.nih.gov/apps/tree-puzzle-doc.html
[2] http://www.tree-puzzle.de/
[3] http://www.dkfz.de/tbi/tree-puzzle/
[4] Heiko A. Schmidt, Phylogenetic Trees from Large Datasets, 'Ph.D.' in Computer Science, Düsseldorf, Germany, 2003.
[5] Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol., 17, 368–376.
[6] Strimmer, K. and von Haeseler, A. Quartet puzzling: A quartet maximum–likelihood method for reconstructing tree topologies. Mol. Biol. Evol., 13, 964–969, 1996.
[7] Adachi, J. and Hasegawa, M. MOLPHY Version 2.3 – Programs for Molecular Phylogenetics Based on Maximum Likelihood, vol. 28 of Computer Science Monographs. Institute of Statistical Mathematics, Minato-ku, Tokyo, 1996.
[8] Huelsenbeck, J. P. and Ronquist, F. MRBAYES: Bayesian inference of phylogenetic trees. Bioinformatics, 17, 754–755, 2001.
[9] P. O. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. Mol. Biol. Evol., 15, 277–283. 1998.
[10] Salter, L. A. and Pearl, D. K. Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. Syst. Biol., 50, 7–17, 2001.
[11] http://www.phylip.com/12. http://paup.csit.fsu.edu/
[12] http://paup.csit.fsu.edu/
[13] Gilles Fedak et al. XtremWeb : A Generic Global Computing System. CCGRID2001, workshop on Global Computing on Personal Devices. Brisbane, Australia, May 2001. http://xtremweb.net
[14] http://boinc.berkeley.edu/
[15] http://www.entropia.com/
[16] http://www.ud.com/home.htm
[17] Parabon Computation, Inc: The Frontier Application. Programming Interface, Version 1.5.2. 2004.
[18] http://www.bittorrent.com/
[19] http://www.edonkey2000.com/
[20] http://www.kazaa.com/us/index.htm
[21] KAN G., Peer-to-Peer: harnessing the power of disruptive technologies, Chapter Gnutella, O'Reilly, Mars 2001.
[22] Ian Clarke. A Distributed Decentralised Information Storage and Retrieval System. Division of Informatics. Univ. of Edinburgh. 1999. http://freenet.sourceforge.net/
[23] http://feedtree.net/
[24] http://www.globus.org/
[25] N. Abdennadher, R. Boesch, A Large Scale Distributed System for High Performance needs. HP-ASIA 2005, Biejing, China, December 2005.
[26] N. Abdennadher, R. Boesch, A Scheduling algorithm for High Performance Peer-To-Peer Platform. CoreGrid workshop, EuroPar 06, Dresden, Germany, 2006