An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications

Rajkumar Buyya, Jonathan Giddy, and David Abramson

School of Computer Science and Software Engineering and CRC for Enterprise Distributed Systems Technology, Monash University, Caulfield Campus, Melbourne, AUSTRALIA.
Email:*{rajkumar, jon, davida}@csse.monash.edu.au*

Key words: Grid Computing, Computational Economy, Resource Trading, Nimrod/G Resource Broker, Scheduling, and Parameter Sweep Applications.

Abstract: Computational Grids are becoming attractive and promising platforms for solving large-scale (problem solving) applications of multi-institutional interest. However, the management of resources and scheduling computations in the Grid environment is a complex undertaking as they are (geographically) distributed, heterogeneous in nature, owned by different individuals or organisations with their own policies, different access and cost models, and have dynamically varying loads and availability. This introduces a number of challenging issues such as site autonomy, heterogeneous substrate, policy extensibility, resource allocation or co-allocation, online control, scalability, transparency, and "economy of computations". Some of these issues are being addressed by system-level Grid middleware toolkits such as Globus.

Our work in general focuses on economy/market driven resource management architecture for the Grid; and in particular on resource brokering and scheduling through a user-level middleware system called Nimrod/G and economy of computations through a system-level middleware infrastructure called GRACE (GRid Architecture for Computational Economy). Nimrod/G supports modeling of a large-scale parameter study simulations (parameter sweep applications) through a simple declarative language or GUI and their seamless execution on global computational Grids. It uses GRACE services for identifying and negotiating low cost access to computational resources. The Nimrod/G adaptive scheduling algorithms help in minimising the time and/or the cost of computations for user defined constraints. These algorithms are evaluated in different scenarios for their effectiveness for scheduling parameter sweep applications in Grid environments such as GRACE and core middleware (Globus, Legion, and/or Condor) enabled federated Grids.

1. INTRODUCTION

The growing popularity of the Internet/Web and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we do computing and use computers. The interest in coupling geographically distributed resources is also growing (leading to what are popularly called *Computational Grids* [11]) for solving large-scale problems. The management of resources in a Grid environment becomes complex, as they are (geographically) distributed, heterogeneous in nature, owned by different individuals or organisations each having their own resource management policies, different access-and-cost models, and subjected to dynamically changing load-and-availability conditions. This introduces a number of challenging issues including site autonomy, heterogeneous substrate, policy extensibility, resource allocation or co-allocation, online control [10], and "economy of computations" [4] that Grid resource management systems need to address. Some or all of these issues are being addressed by a number of (on going) Grid computing projects world-wide [2][7] including Globus [8], Legion [13], Information Power Grid [21], NetSolve [14], Ninf [20], AppLes [15], Nimrod/G [1] [4], DISCWorld [16], and JaWS [22]. Although computational economy is one of the key issues in Grid computing, it is rarely taken into consideration in the design of most of these systems.

We strongly feel that for the ultimate success of Computational Grids as a production-oriented commercial platform for solving problems, they need to support market/economy-based mechanisms in resource management. In [4], we present a number of arguments for the need of computational economy. It primarily offers a mechanism for encouraging resource owners to contribute their resource(s) for the construction of a Grid and compensate them based on the resource usage or value of work done. This concern is also being expressed in Scientific American journal [6]: "So far not even the most ambitious metacomputing prototypes have tackled accounting: determining a fair price for idle processor cycles. It all depends on the risk, on the speed of the machine, on the cost of communication, on the importance of the problem—on a million variables, none of them well understood. If only for that reason, metacomputing will probably arrive with a whimper, not a bang". In a Grid environment, a set of resources can dynamically team up (on demand) to solve a given problem and have their own mechanism for sharing earnings/profits among themselves. This type of mutually agreed teaming up is quite useful for developing computational economy for executing parallel application tasks that have high degree of message communications for sharing partial results. This is a subject of our future investigation.

The remaining sections are organised as follows. In section 2, we focus the use of economy-based model in resource selection through trading services. In section 3, we discuss adaptive algorithms for scheduling parameter sweep applications on the Grid and in section 4, we present their evaluation for various scenarios.

2. GRACE-ENABLED NIMROD/G

A number of Grid computing systems [2] are being developed, implemented, and deployed mainly based on three architectural models [5]: hierarchical, abstract owner, and market. In our earlier work [4], we proposed market/economy model based architecture for Grid resource management. One of the possible (implementation) architectures for this market/economy model is shown in Figure 1. The architecture varies depending on the method/protocol used (by trade manager) in determining or negotiating the resource access cost. The key components of economy-driven resource management system include,

- User Applications (sequential, parametric, parallel, or collaborative apps)
- The Grid Resource Broker (a.k.a., Super/Global/Meta Scheduler)
- Grid Middleware
- The Domain Resource Manager (Local Scheduler or Queuing system)

We briefly discuss some of these components and further details can be found in our earlier works [1][4][5].

The *resource broker* acts as a mediator between the user (application) and Grid resources using middleware services. It is responsible for the management of the whole experiment on the Grid. This includes resource discovery, resource selection and trading (including negotiation of access cost), the binding of application, data, and hardware resources, the initiation of computations, any required adaptation to changes in the Grid resources, and the collection of results.

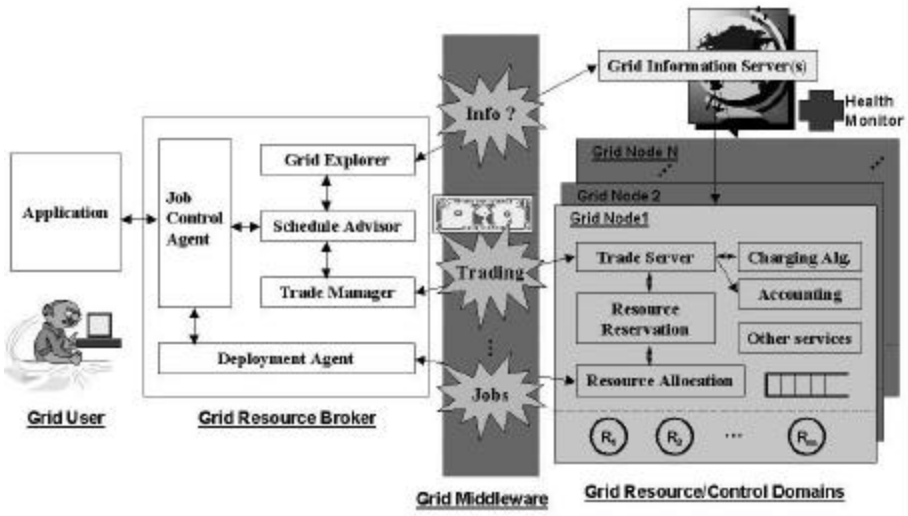

Figure 1. Market/Economy Model for Grid Resource Management.

The Grid *middleware* offers services that help in coupling a Grid user through resource broker or Grid enabled application and (remote) resources. It offers core services [2][11] such as remote process management, co-allocation of resources, storage access, information publication (directory), security, authentication, and

Quality of Service (QoS) such as resource reservation for guaranteed availability and trading for minimising computational cost.

The components that are specifically responsible for managing economy of computations on the Grid are the *schedule adviser*, *trade manager*, and *trader server*. The schedule adviser uses services of Grid Explorer for resource discovery, trade server for negotiating access costs from trader server, and scheduling algorithms for identifying mappings (jobs to resources) that meet user requirements (deadline and cost minimisation). The trade server decides access costs based on resource-owner defined charging algorithms/policies and interacts with accounting system for recording usage details and billing as per negotiation.

Nimrod/G Resource Broker

The Nimrod/G resource broker (a.k.a. superscheduler, or metascheduler) is a global resource management and scheduling system (see Figure 2) that supports deadline and economy-based computations in Grid computing environments [1][3] for parameter sweep applications. It supports a simple *declarative* parametric modeling language for expressing parametric experiments. The domain experts (application area experts/users) can easily create a *plan* for parameter studies and use the Nimrod/G broker to handle all issues related to seamless management issues including resource discovery, mapping jobs to appropriate resources, data and code staging and gathering results from multiple Grid nodes.

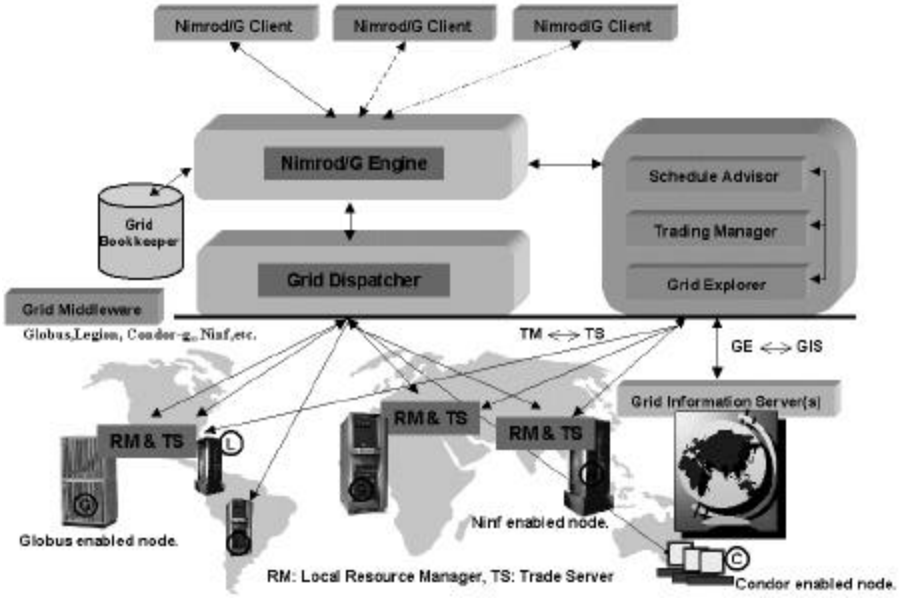

Figure 2. Nimrod/G Resource Broker.

Initially Nimrod/G was targeted for Globus, but the latest version is being abstracted and is now capable of deploying computations on Grids using other middleware systems such as Legion and Personal Condor. With a minimal effort Nimrod/G can be enabled to use services of other middleware systems including

Ninf and NetSolve leading to *federated Grid computing*[1]. However, it should be noted that most of these (as of today) do not offer on-demand/online trading services for low-cost access to resources. This limitation is overcome by our GRid Architecture for Computational Economy (GRACE) [4] middleware infrastructure that can co-exist with system-level middleware toolkit (like Globus) services. The higher-level Grid services or tools like Nimrod/G broker can use them for dynamic online negotiation for access to Grid resources at lower cost and also to make deadline vs. cost trade-off during scheduling. This paper presents the usage of GRACE services in Nimrod/G and its evaluation through the execution of simulated parameter sweep applications for various user defined constraints (such as limited budget and time) in a controlled and repeatable/reproducible manner.

3. SCHEDULING ALGORITHMS

Parameter studies (p-studies) involve the execution of a large number of (independent) tasks over a range of parameters. Scheduling of such applications appears simple, but complexity arises when users place QoS constraints like execution time and computation cost limitations. Such a guarantee of service is hard to provide in a Grid environment as its resources are shared, heterogeneous, distributed in nature, and owned by different organisations having their own policies and charging mechanisms. In addition, scheduling algorithms need to adapt to the changing load and resource availability conditions in the Grid in order to achieve performance and at the same time meet cost constraints. In our Nimrod/G application level resource broker (also called an application level scheduler) for the Grid, we have incorporated three adaptive algorithms for scheduling:

- Time Minimisation, within time and budget constraints,
- Cost Minimisation, within time and budget constraints,
- None Minimisation, within time and budget constraints.

Table 1: Adaptive Scheduling Algorithms.

Adaptive Scheduling Algorithms	***Execution Time (not beyond deadline)***	***Execution Cost (not beyond budget)***
Time Minimisation	Minimise	Limited by budget
Cost Minimisation	Limited by deadline	Minimise
None Minimisation	Limited by deadline	Limited by budget

[1] *Federated grids* can couple private, enterprise, state, national, and/or international grids each powered using different grid technologies. Global resource brokers like Nimrod/G can simultaneously use all these grids for solving very large-scale problems economically. In this kind of environment the management of heterogeneity, scalability, transparency, security and open standards will be major issues.

The Time Minimisation algorithm attempts to complete the experiment as quickly as possible, within the budget available. A description of the core of the algorithm follows:

1. For each resource, calculate the next completion time for an assigned job, taking into account previously assigned jobs.
2. Sort resources by next completion time.
3. Assign one job to the first resource for which the cost per job is less than or equal to the remaining budget per job.
4. Repeat all steps until all jobs are assigned.

The Cost Minimisation algorithm attempts to complete the experiment as economically as possible within the deadline.

1. Sort resources by increasing cost.
2. For each resource in order, assign as many jobs as possible to the resource, without exceeding the deadline.

A final algorithm ("None Minimisation") attempts to complete the experiment within the deadline and cost constraints without minimising either.

1. Split resources by whether cost per job is less than or equal to the budget per job.
2. For the cheaper resources, assign jobs in inverse proportion to the job completion time (e.g. a resource with completion time = 5 gets twice as many jobs as a resource with completion time = 10).
3. For the dearer resources, repeat all steps (with a recalculated budget per job) until all jobs are assigned.

Note that the implementations of all the above algorithms contain extra steps for dealing with the initial startup (when average completion times are unknown), and for when all jobs cannot be assigned to resources (infeasible schedules).

4. EXPERIMENTATION AND EVALUATION

In addition to accessing real computational resources, Nimrod can also simulate the execution of jobs on a test queue. These simulated queues are useful for testing the scheduling algorithms, since their behaviour can be controlled very precisely. A test queue runs each submitted job in succession and the apparent wallclock time and reported CPU usage can be controlled exactly.

For this simulation, we created experiments containing 100 jobs, each with a 90 second running time, giving a total computation time of 9000 seconds. For each experiment, we created 10 test queues with different (but fixed) access costs of 10, 12, 14, 16, 18, 20, 22, 24, 26, and 28 units/CPU-second. The optimal deadline for this experiment is achieved when each queue runs 10 jobs in sequence, giving a running time of 900 seconds for the 100 jobs.

We selected three deadlines: 990 seconds (the optimal deadline plus 10%), 1980 seconds (990 x 2), and 2970 seconds (990 x 3). The 10% allowance allows for the fact that although the queues are simulated, and behave perfectly, the standard scheduler has some delays built in.

We selected three values for the budget. The highest is 252000 units, which is the amount required to run all jobs on the most expensive queue. Effectively, this allows the scheduler full freedom to schedule over the queues with no consideration for the cost. A budget of 171000 units is the budget required to execute 10 jobs on each of the queues. Finally, the lowest budget of 126000 units is the budget required to execute 20 jobs on each of the 5 cheapest queues. Note that for this value, the deadline of 990 seconds is infeasible, and the deadline of 1980 seconds is the optimal deadline plus 10%.

Table 2 shows a summary of results for each combination of scheduling algorithm, deadline and budget, and the resulting percentage of completed jobs, the total running time, and the final cost. The jobs marked "infeasible" have no scheduling solution that enables 100% completion of jobs. The jobs marked "hard" have only one scheduling solution.

Table 2. Behaviour of Scheduling Algorithms for various scenarios on the Grid.

Algorithm	*Deadline*	*Budget*	*Completed*	*Time*	*Cost*	*Notes*
cost	990	126000	85%	946	125820	infeasible
cost	990	171000	84%	942	139500	hard
cost	990	252000	94%	928	156420	hard
cost	1980	126000	97%	1927	124740	hard
cost	1980	171000	99%	1918	128520	
cost	1980	252000	98%	1931	127620	
cost	2970	126000	98%	2931	116820	
cost	2970	171000	98%	2925	116820	
cost	2970	252000	100%	2918	118800	
none	990	126000	78%	919	120060	infeasible
none	990	171000	99%	930	168480	hard
none	990	252000	100%	941	171000	hard
none	1980	126000	97%	1902	125100	hard
none	1980	171000	100%	1376	160740	
none	1980	252000	100%	908	171000	
none	2970	126000	99%	2928	125100	
none	2970	171000	100%	1320	161460	
none	2970	252000	100%	952	171000	
time	990	126000	36%	955	50040	infeasible
time	990	171000	100%	913	171000	hard
time	990	252000	100%	930	171000	hard
time	1980	126000	80%	1968	101340	hard
time	1980	171000	100%	909	171000	
time	1980	252000	100%	949	171000	
time	2970	126000	100%	2193	126000	

time	2970	171000	100%	928	171000	
time	2970	252000	100%	922	171000	

We analyse the behaviour of the queues by examining the usage of the queues over the period of the experiment. For the Cost Minimisation algorithm, Figure 3 shows the node usage for a deadline of 1980 seconds. After an initial spike, during which the scheduler gathers information about the queues, the scheduler calculates that it needs only use the 45 cheapest queues in order to satisfy the deadline. (Actually, it requires exactly 5, but the initial spike reduces the requirements a little.) Note that the schedule is similar, no matter what the allowed budget is. Since we are minimising cost, the budget plays little part in the scheduling, unless the limit is reached. This appears to have happened for the lowest budget, where the completion rate was 97%. The budget of 126000 units is only enough to complete the experiment if the cheapest 5 nodes are used. Because of the initial spike, this experiment appears to have run out of money. The other experiments also did not complete 100% of the jobs, but this is mainly because in seeking to minimise cost, the algorithm stretches jobs out to the deadline. This indicates the need for a small margin to allow the few remaining jobs to complete close to the deadline.

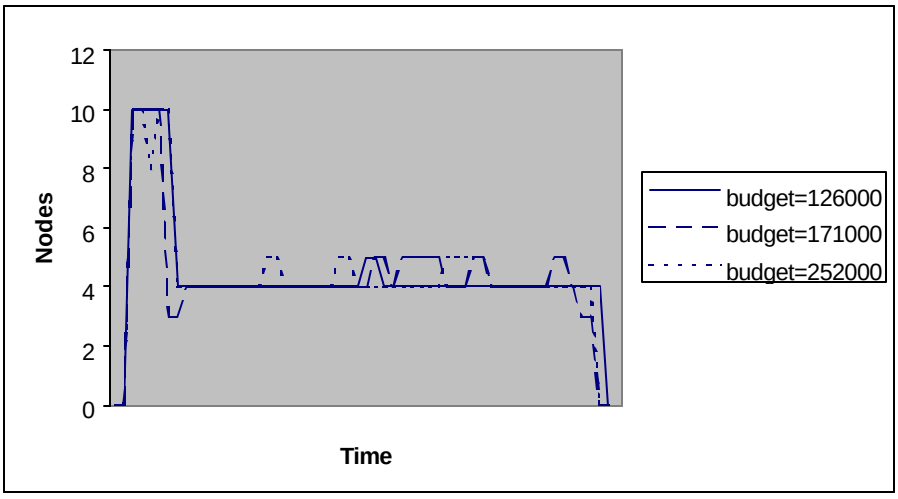

Figure 3. Cost Minimsation scheduling behaviour for various budgets.

The equivalent graph for the Time Minimisation algorithm is shown in Figure 4. Here we see that except for the case of a limited budget, we get a rectangular shape, indicating the equal mapping of jobs to each resource. Only the experiment with a very limited budget follows the pattern experienced above.

Looking at the equivalent graph for the None Minimisation algorithm, we see a lot more variation in the schedules chosen for different budgets. The schedule with a very large budget is equivalent to the Time Minimisation algorithm. The schedule with the low budget is almost the same as the Cost Minimisation algorithm.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed an economy-based model for resource management and scheduling in the Grid through Nimrod/G and GRACE services. The adaptive scheduling algorithms have been evaluated for various application scenarios and user constraints and have demonstrated the capabilities of our resource broker and computational economy model. Future work focuses on the use of a resource reservation model in scheduling and competitive charging algorithms that enables Nimrod/G to guarantee the user up-front when application processing can complete and how much it is going to cost.

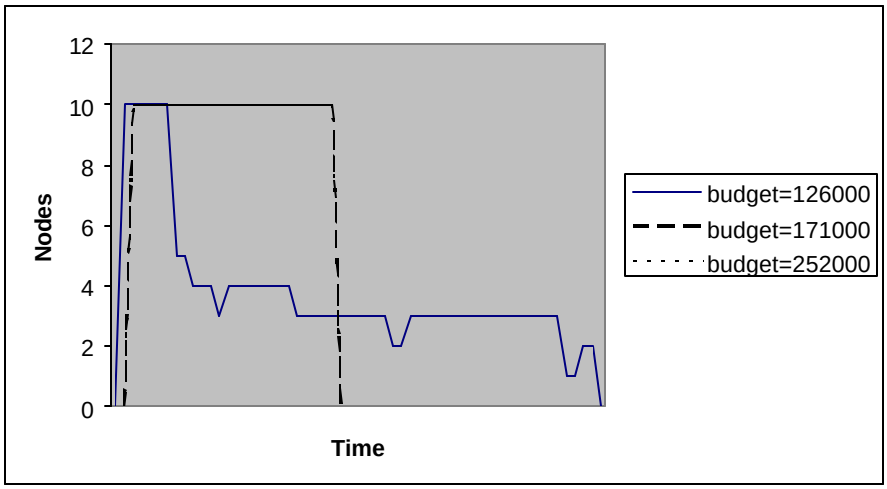

Figure 4. Time Minimisation scheduling behaviour for various budgets.

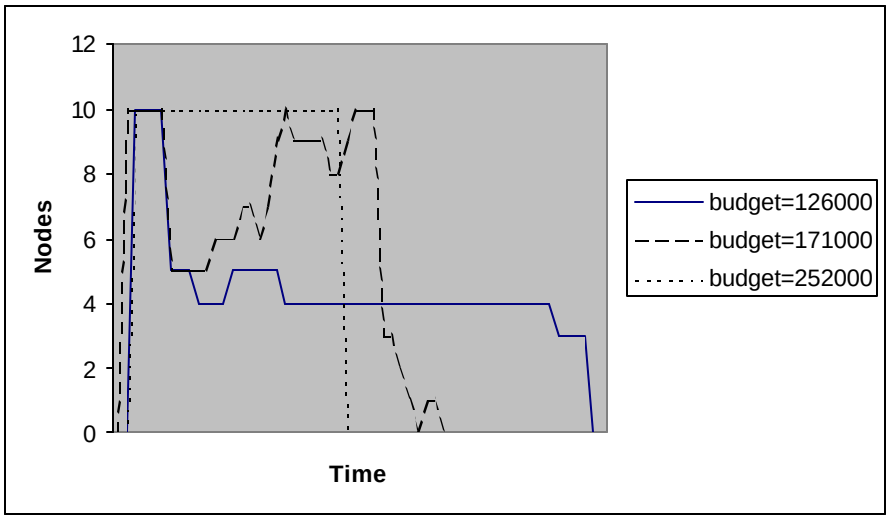

Figure 5. None Minimisation scheduling behaviour for various budgets.

6. REFERENCES

[1] Abramson, D., Giddy, J., and Kotler, L., *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?*, International Parallel and Distributed Processing Symposium (IPDPS 2000), Mexico.

[2] Baker M., Buyya R., Laforenza D., *The Grid: International Efforts in Global Computing*, Intl. Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR'2000), Italy, 2000 (to appear).

[3] Buyya, R., Abramson, D., and Giddy, J., *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, HPC ASIA 2000, China, IEEE CS Press, USA, 2000.

[4] Buyya R, Abramson D, and Giddy J, *Economy Driven Resource Management Architecture for Computational Power Grids*, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000), Las Vegas, USA.

[5] Buyya R., Chapin S., DiNucci D., *Architectural Models for Resource Management in Global Computational Grids*, http://www.buyya.com/ecogrid/

[6] Gibbs W., Cyber View—World Wide Widgets, Scientific American, San Francisco, USA - http://www.sciam.com/0597issue/0597cyber.html.

[7] Grid Computing Infoware (Info Centre) - http://www.gridcomputing.com/

[8] Globus - http://www.globus.org/

[9] Globus Testbeds - http://www-fp.globus.org/testbeds/

[10] Foster I. and Kesselman C., *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[11] Foster I. and Kesselman C. (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[12] Dongarra J., *An Overview of Computational Grids and Survey of a Few Research Projects*, Symposium on Global Information Processing Technology, Japan, 1999.

[13] Legion - http://legion.virginia.edu/

[14] Casanova H. and Dongarra, J., *NetSolve: A Network Server for Solving Computational Science Problems*, Intl. Journal of Supercomputing Applications and High Performance Computing, Vol. 11, No. 3, 1997.

[15] AppLeS Project — http://apples.ucsd.edu

[16] Hawick K. et al, *DISCWorld: An Environment for Service-Based Metacomputing*, Future Generation Computing Systems (FGCS), Vol. 15, 1999.

[17] Condor - http://www.cs.wisc.edu/condor/

[18] SETI@Home – http://setiathome.ssl.berkeley.edu/

[19] Distributed.Net – http://www.distributed.net/

[20] Ninf – http://ninf.etl.go.jp/

[21] NASA IPG – http://www.ipg.nasa.gov

[22] JaWS – http://roadrunner.ics.forth.gr:8080/

[23] EcoGRID – http://www.csse.monash.edu.au/~rajkumar/ecogrid/