Bilgisayar Bilimleri Dergisi

Vol:1, No:1, 2016

Design of a Resource Management for GPGPU Supported Grid Computing

D. Emrah, *Inonu University*

Abstract— In this study; we aimed to propose design of a QoS aware resource management infrastructure for a GPGPU supported Grid computing system. This Grid system consists of hybrid (CPU + CPU) and heterogeneous (Nvidia + AMD Radeon) GPGPU computational nodes. It can manage both small scale unit (connections, threads, buffer pools etc.) and large scale unit (whole computing machines). As increasing of the network communication bandwidth and developing powerful computer hardware (CPU, GPU etc.), distributed computing systems acquire more and more attention day by day. Grid computing is as a major player in such kind of distributed system environments like cloud, volunteer, hybrid and etc. Since it supports large scale resource sharing between geographically distributed computer clusters and even single computers. Nowadays, there is another important technology pillar to implement high performance computing rather than CPU, it is known as GPU computing. The GPU systems are ideal especially to data intensive applications; such as image processing, data mining, financial computations etc. Therefore, GPU based grids give an undertaking higher computational performance. GPU processor consists of lots of controllable cores which can be used for high performance demanded applications. Ultimately, the major concerns in grid computing are particularly related to managing QoS requirements, granularity of resources, and heterogeneous resources (both CPU and GPU).

Index Terms— Grid computing, GPGPU, QoS, resource management, data intensive application.

1 Introduction

GRID computing is appearing as a significant technology for data-intensive computations, resource sharing and cluster unification in a distributed manner. Fast and accurate solutions to large-scale scientific, financial problems etc. with unused massive resources of distributed clusters, and higher costs is a significant problem. This problem enabled grid computing to play a major role in commercial domain. There are two main issues; Quality of Service (QoS) and performance becoming a major concern [1].

To satisfy QoS requirements, application level discovery is needed to identify which QoS specifications are suitable to ensure best bandwidth for message packets sent to throughout the network by applications (especially for data intensive ones). QoS requirements are generally set by a QoS broker configured manually or semi-automatically according to pre-defined QoS requirements of application(s). To optimize the QoS requirements for applications, below criterions have to be considered:

- Delay,

- Jitter,
- Throughput,
- Bandwidth, and
- Packet loss

By optimizing each of the above issues dynamically for each application, the QoS requirements will be met at a higher level than manually configuration. Dynamic optimization will provide a QoS aware component in resource management system. Therefore, design of an adaptive component, meeting QoS requirement continuously, should be considered.

On the other hand, to satisfy performance requirements as maximum as possible; CPU, memory, disk and other hardware in cluster nodes also have to be optimized in locally and between nodes. To implement high performance systems, GPGPU (General-Purpose computation on Graphics Processing Units) resources can be used besides CPU resources. The main advantages of the GPGPU are core number of GPU processors, memory speed, and cost. A GPGPU card specialized especially for data intensive tasks. It include thousands of cores worked simultaneously or separately. Each of the core can use memory (or local memory) similar to CPU core's cache memory. There is also a memory (shared memory) on GPGPU card which shared between GPU cores. In short, these memory units are remarkably fast compare to main memory (or global memory) in a computer system. By utilizing such powerful hardware with optimization (such as communication, algorithm etc.), powerful grid clusters can be created. To manage GPGPU supported clusters, it need to be designed more complex management systems compare to only CPU supported ones. In addition, GPGPU systems can be used alongside with CPU, and this kind of computational environment called hybrid computing. Moreover different type of GPU (e.g. AMD Radeon, Nvidia) can be used in same cluster, even in same node, this kind of GPGPU cluster is called heterogeneous GPGPU cluster.

Conventional resource discovery and management designs/frameworks in grid computing tend to work in large scale units. It have fixed non-dynamic principles and deal solely with tangible resource entities e.g. The innate complicacy, heterogeneity and dynamic structures of grid environments constitute some difficulties in managing their capacity to provide that QoS requirements are continually satisfied. Implementation of scheduler (task e.g.) performance evaluation is very rough, even unfeasible in a manageable and repeatable manner. Because, resources and users are distributed throughout multiple organizations with their own principles.

2 RELATED WORKS

There are a number of available studies to configure and optimize resource management in grid systems. Samuel et al. [1] have proposed an approach to design autonomic QoS-aware resource management in Grid computing based on online performance models by making prediction of grid components performance and allocate resources with SLA (Service Level Agreements). Gridkit [2] is a resource management framework that can be used to implement granularity aware resource framework. It is said that, it can manage both fine-grained and coarse-grained resources separately according to necessities of resources. There is another toolkit called GridFlow [6] which is specialized to perform global workflow management and local grid sub-workflow scheduling, and it is based on PACE toolkit which supports a prediction based resource management structure. Globus [8] is a toolkit to implement and manage grid environments. As far as we know, it has not a sub-infrastructure which allows resource management system to manage both smallest and largest units in grid environments.

This study focuses; which design elements should be considered, and how they implemented? What are the potential impacts of such design issues to GPGPU supported hybrid and heterogeneous grid computing environments?

3 COMPUTING INFRASTRUCTURE

Grid Computing: It composes computational and/or resource sharing focused clusters known as Virtual Organizations (VO) to form a global network computing structure. It enables whole registered machines to perform heavy tasks and/or share resources of these machines between separate VO's.

GPGPU: It mainly focuses to perform and accelerate general-purpose scientific and engineering applications and heavy tasks (image processing, data mining) in a short amount of time. It uses a great number of cores of GPU (Graphic Processing Unit) processors which are programmable by using specialized libraries (OpenCL, CUDA etc.)

GPGPU supported Grid Computing: The main aim is utilizing both CPU specialized for serial processes and GPU specialized for parallel processes to perform data intensive tasks (scientific, engineering, financial etc.). In this way, potential power of grid environment has been significantly increased. CPU + GPU combination provides efficient and powerful computation resources by adapting and optimizing applications running on them.

4 DESIGN OF RESOURCE MANAGEMENT

Grid resource management should be considered with regard to aim of grid environment. It can be designed and specialized for data grid, service grid or computational grid. Data grids are used to implement resource sharing between clusters and communication speed is most important parameter in such grids. Service grids are used to provide services to route clients, ensure data, or specific services such as security, protocols etc. Service grids are mostly used as PaaS (Platform as a Service), SaaS (Software as a Service) and IaaS (Infrastructure as a Service) in Cloud computing which is known as commercial pillar of Grid computing. Finally, computational grids are used to provide powerful hardware resources and manage these resources to perform scientific heavy tasks. In this section our main motive is offering a flexible resource management design approach for computational grids.

4.1 Challenges in Resource Management

Grid computing itself has several difficulties; Autonomy, Heterogeneity, and Dynamics. There are several challenges in design of a resource management for Grid computing.

- Satisfactory end-to-end performance through multiple domains,
- Availability of computational resources,
- Handle of conflicts between common resources demand,
- Fault-tolerance,
- Inter-domain compatibility (P2P)

4.2 Design of Resource Management Infrastructure

A general resource management requires three main phase; 1. Resource discovery; locate available resources in reachable points, 2. Selection; allocate suitable resources according to requirements, and 3. Execute jobs; distribute and run tasks, and when tasks are completed release handed resources. While these three phase are worked, there is another issue which should be taken into consideration; security. Security is a major concern in distributed systems especially in geographically distributed ones such as grid computing, volunteer computing, or hybrid computing. But security doesn't deeply researched in this paper, since it is a separate research topic itself. Major concern of this paper is design of an efficient, easy manageable, and modular resource management infrastructure.

In our proposed design approach in figure 1, QoS controller is responsible to predict connection requirements of applications and manage these prediction operations dynamically.

This is done by using application annotations used for QoS requirements. Firstly, QoS controller receives application demands (client demands) and resources are allocated according to requirement predictions of QoS controller by resource manager. Prediction process of demands is done by examining previous demands statistically. Then negotiation with Service Level Agreement (SLA) is implemented. Equation 1 simply models prediction process. Dn_{ApX} is n^{th} demand value of 'X' application. m express starting time point of related demand. Similarly, f express passed time while demanded request is active. $D(n+1)_{ApX}$ is $(n+1)^{th}$ demand value predicted for 'X' application. To perform a fair workload distribution between Grid servers, QoS controller identifies the available non-occupied or non-busy part of the server resources. Then workloads are distributed (e.g. Load Balancing) among these servers dynamically. Dynamically means that, after initial QoS requirements of an application are met, if some modifications of requirements are needed in runtime of application, QoS controller is informed by QoS agents which track applications in grid servers. QoS agents examine the applications periodically in runtime to create information packets about application's status. Controller provides new requirement parameters to the related applications or clients through agents.

$$D(n+1)_{ApX} = Dn_{ApX} + \left(Dn_{ApX} - \frac{\sum_{t=m}^{t=m+f} Dn_{ApX}}{f} \right) \quad (1)$$

Unit size controller component (USCC) manage both the small unit resources such as threads, cache buffers, communication parameters, communication protocols etc. and large unit resources which represent the whole environment (machines, clusters). USCC generates management configurations for both small and large scale units to provide an efficient and easy manageable grid computing environment. USCC enables that resource manager behaves like globally if whole machines are included. It behaves like locally if a specific threads or connections are included. The same mechanism is valid for GPGPU supported hybrid grid servers. If GPGPU cards are taken into consideration resource management behaves as globally. On the other hand, if threads, cache, DMA, shared memory SM's of GPU cards are taken into consideration it behaves as locally.

Global resource management is responsible to perform distribution of available tasks between different nodes. Local resource management is responsible to perform distribution of tasks between GPGPU and CPU cores by using specialized libraries (e.g. OpenCL, CUDA, and OpenMP). Therefore, we can use global resource management infrastructure to manage large scale unit resources. Besides, we can use local resource management infrastructure to manage small scale unit resources; such as, distribution of workloads to local CPUs and GPUs, management of memories etc. These two layered (local & global) resource management infrastructure ensures flexible and efficient computational grid environments.

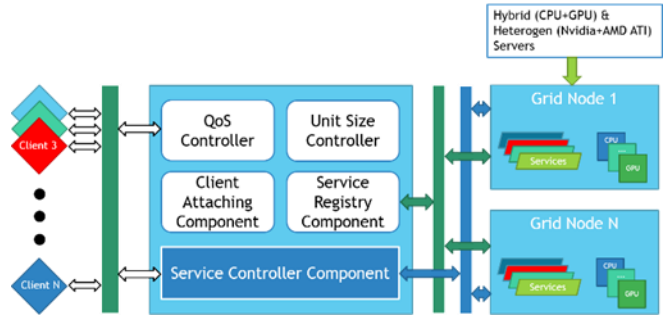

Figure 1 Resource management infrastructure of proposed design

GPGPU supported grid servers can have heterogeneous structure. Different type of GPU cards (AMD Radeon, Nvidia etc.) can be used in same node of a computation cluster. Therefore, heterogeneous GPU supported libraries should be used. OpenCL (Open Computing Library) is an open source GPU programming toolkit that can be used to program GPUs without depending on type of GPUs, whether it is AMD Radeon, Nvidia, or different supported GPU in library. There are of course remarkable usage differences in different type GPUs. There is another powerful toolkit and library called CUDA (Compute Unified Device Architecture) which is developed and specialized for Nvidia GPU cards. It is used to develop parallel applications (or to accelerate data intensive applications). CUDA already supports the OpenCL GPU programming library. It has some additional high level features as advantages by comparing other GPU programming libraries. For instance; GPU-Direct is one of these features which allows us to communicate from one GPU card to another GPU card that are found at the same node (intra-node) or different node (outer/district-node) without CPU intervention. To perform this process it uses DMA (Direct Memory Access) engines located on the GPU cards. CUDA is a good choice for Nvidia GPU cards but we cannot program AMD Radeon GPU cards with CUDA libraries. If we want to communicate two different type GPUs located in same or distinct node then we have to use OpenCL GPU programming library. Since it supports both GPU card type. On the other hand, if communication between different types of GPU is not necessary then using CUDA for Nvidia cards and using OpenCL for AMD Radeon cards separately can be a good choice. Since CUDA supported Nvidia GPU cards can demonstrate more performance than OpenCL support.

In addition, an automatic GPU task distribution and execution feature can be designed in grid servers. Namely as a default library, tasks can be performed with OpenCL for all type of GPUs, if there is Nvidia cards then CUDA supported libraries can be used to maximize Nvidia GPUs performance. Responsible component of job distribution and execution can be automatically provides this switching process between libraries with respect to the type of GPUs. There can be servers which includes GPUs in a homogeneous manner; all GPUs Nvidia or AMD Radeon. For these servers only one library is enough, there is no need for an automatic GPU library switching feature. To be able to detect GPU types in grid servers hardware detection agents are modeled to find out what GPU type the server has; all-N (Nvidia), all-A (AMD Radeon), or all-H (Heterogeneous). After detection processes on available grid servers, optimal resource management configurations for GPUs are determined and assigned to the related servers. For this reasons; OpenCL libraries are used to perform task distribution to GPU cards in resource management as default GPU programming library.

There is a library which allows to perform parallel applications on CPU: OpenMP (Open Multi Processing) is a ‘C’ based API assisting multi-platform and shared memory multi-processing programming language. It comprise of a set of compiler directives, library routines, and environment variables that affect run-time attitude. OpenMP grants programmers a simple and flexible interface by utilizing a scalable, portable model which for developing parallel applications for platforms varying from the standard desktop machine to the super-machine. OpenMP libraries are used to provide parallel programming on CPUs as appropriate to hybrid (CPU + GPU) computing in resource management infrastructure. In addition to parallelization speed of GPU, CPUs are specialized in serialization speed. In other words, GPUs are optimized for parallel applications, on the other hand, CPUs are optimized for serial applications. If we have non-parallelizable applications, algorithm etc. then CPU with OpenMP support can be a good choice to perform these serial applications. Of course OpenMP provides parallelization but this parallelization is not great as in a GPU parallelization. In this instance, it can be said

that CPU provides the acceleration of serial weighted parallel applications (means that there is weak parallelization but powerful serialization) while GPU is providing acceleration of parallel weighted serial applications (means that there is weak serialization but powerful parallelization). By taking into consideration these situations, a manual selecting feature which allows applications to run on CPU or GPU even partially CPU or GPU. For example; there is an image processing algorithm that can be easily run on GPUs to perform process acceleration, but it is just sent another place through a serial process, then it can be said that GPU an CPU are both partially used. This feature is integrated in resource management to determine initial execution specifications related to processor types. These initial specifications include three different parameters; exe-GPU (to select only GPU), exe-CPU (to select only CPU), exe-Hybrid (to select partially between GPU and CPU, w.r.t. requirements). According to the user choices one of these related parameter is taken to implement initial configuration of additional processor selecting feature.

In addition, some processor architectures may have a special feature like Hyper Threading (HT) technology (a processor core virtualization technology to make efficient processes without extra cost) in new generation processors. By using HT technology cost of parallel applications can be reduced and aggregate system performance can be increased. To demonstrate the difference in performance while the HT is used and not used in table 1, an alternative processing feature allowing us to switch between HT and non-HT is added to design. It should be emphasized that HT is generally available for processors being compatible with virtualization architecture.

Table 1 General performance and total communication traffic in grid with HT and without HT

Criteria	HT	Non-HT
Performance	248.965 GFlops	205.673 GFlops
Communication	864Mbit (p.s)	927Mbit (p.s)

Client attaching component adds client to the grid environment to perform that another component takes tasks from these clients, on computational nodes. This attaching component gives unique ID to each of the clients enrolling to the computing environment. So information about related processed tasks are sent to the related clients by using these unique IDs. Before the client addition process, the QoS requirements are requested from QoS controller by client. If requests would be analyzed as appropriate then client negotiation session is reciprocally started by client and QoS controller. After finishing negotiation process client is attached to the computing environment. If negotiation process fails then all negotiation parameters are resettled according to new QoS requirement requested by client. If three attempts to negotiation process fails consecutively, all negotiation parameters are reset then negotiation session is restarted after a defined certain time from the first phase. To prevent anonymous redundant negotiation processes and regulate session negotiation attempts, this feature is integrated as a module which can be extracted from system if it is desired.

Service controller component schedules the service requests after a client session is started. Job distribution management is undertaken by service controller. It arranges the service requests that made by clients to the service requests queue. When a request completed another one is taken from queue to concurrently request available services from grid servers. This requests have been limited with regard to a maximum allowed number of requests defined as a threshold value in service controller component. There is also a dynamic service threshold number

assignment which is maximum service number taken by grid servers in service controller component. It is also limited with respect to maximum saturation value of network QoS parameters like bandwidth. This dynamism is determined by controlling server's real time QoS and performance requirements.

There is a maximum usage limit for resource consumption by services, for example; bandwidth for a specific service can be set a certain value, and same process goes to the other services. At the same time, grid server can satisfy a certain bandwidth value as maximum allowed bandwidth in total communication capacity of server. There isn't a certain threshold number for services, but there is a certain threshold value for services. Namely, grid server can allow any number of services, providing that bandwidth requirements of these services in total value cannot exceed the defined threshold value. Service provider component provide service to the client if this requested service is procurable by grid servers in available thresholds dynamism. If requested server is not available then other available servers are searched in same cluster. If whole cluster including requested server is not available then other available compute clusters are searched to fulfill client's requests.

Services provided by grid servers are registered to resource management through service registry component. This means that grid servers are registered to the management infrastructure and available for requested service in the grid environment. Additional services which are ensured later, can be integrated to the resource management in any time. This feature ensures service attachment flexibility; main aim of such pluggable services is efficiency and prevent redundant complexity in grid environment. When a service have done providing requested jobs it hibernates until a new request arrives. When a service complete its task totally, it plugged system to decrease environment complexity. A plugged service never integrated to the system automatically. If it is required than manual attaching and configuration have to be implemented. Therefore, only the hibernated services can return to the grid environment automatically. In order to perform real time tracing (to eliminate or attach such services) Service Tracing Agents (STAs) are used. STAs are used to obtain information about services in a certain time. Agents inform the management system periodically. These information about services are structured in a simple XML format that can be sent easily to the related component. XML file includes total request number, situation of service connectivity parameters (QoS), average response time, average communication speed between itself and clients, date and time information, service and server known as provider IDs, and some data to make statistical analyses later.

5 Evaluation of Resource Management

The performance measurement of resource management has a number of challenges due to the distributed, hybrid and heterogeneous nature of grid environment. Performance measurement can be done partially through running a data intensive application like stencil2D, Halo exchange, matrix-matrix multiplication, image processing techniques; such as filtering, segmentation, classification, and etc. This partial evaluation suitable for GPGPU performance between same nodes or distinct nodes (in same or distinct clusters). It can be done through Dhrystone; a synthetic benchmark program in terms of CPU performance evaluation. In short, there is not an absolute way to evaluate resource management total performance properly while grid system is under workloads. Since each of the components have typical characteristics that cannot be fitted any other component. If all components is even accurately evaluated, there will be waited problematic issues such as communications, security, and hardware restrictions. For

instance; there may be SSD disks allowing very fast read-write operations in some servers, but there also may be mechanical HDD disks being slower evidently compare to SSD. Such a hardware difference can be easily affect the performance of resource management. On the other hand, measurement of disk performance cannot be made for each node or cluster if disks have not shared.

For these reasons, in addition to above defined tests, a manual evaluation is approved in first stage. To do this several Grid computing middleware and toolkit are examined and strong and weak aspects are demonstrated by comparing the proposed resource management system. Globus known commonly a grid computing toolkit used to create powerful Grid environment. Globus can change application interceder, but the system has to be restarted, proposed system can manage these process without any performance loss that similar to the study done in [2]. Proposed resource management system integrates both large and small scale units literally. There are other middleware and toolkits but they allows such process with additional restrictions.

In our design, GPGPU and CPU are both supported to maximize performance. In addition, heterogeneous GPGPU structure on the same node is supported with flexible switching between GPUs. On the other hand, a manual processor selecting feature is added to perform executing of application according to user choices. By using this feature, user can partially make selection between GPU and CPU. Proposed design supports both CUDA and OpenCL GPU programming languages. Besides, OpenMP is supported for parallel executions on CPU. Tests for resource management have been done with HT and without HT to see differences in performance. In proposed design, QoS requirements are satisfied in real-time dynamically while computational works are sustained. Service attaching process can be done without any system shut-down.

Pluggable services have been gained the flexibility and maintainability to the responsible service registry component in resource management. Pluggable components are designed in a study being done by Geoff et al. They are designed their system with pluggable components in resource management, but there is no a service plugging feature. These all structures have been gained a new acceleration to this area of study. Since these all features are considered to perform together in a grid resource management infrastructure, firstly. And there are new approaches which are not totally or sufficiently taken into consideration in other studies. Ultimately, we have made a test for three criteria demonstrated in table 2. These test criteria are: I. Error rate of job distribution that is performed by whether global or local resource management. II. Error rate of task complete process. III Energy efficiency per node. Proposed grid management system provides better performance/efficiency rate compare to other known management infrastructures as seen in this table.

Table 2 Comparison of different Grid Environments

Grid Computing Environment	Err. Rate of Job Dis. (s)	Err. Rate of Task C. (s)	P.N Energy Efficiency
Offered Res. Man.	6,18%	2,13%	724W
Globus Res. Man.	11,45%	4,81%	740W
Gridflow	13,86%	4,20%	702W
Gridkit	15,21%	5,63%	734W

6 CONCLUSION AND FUTURE WORKS

Grid computing is an important player in terms of offering solutions to large scale scientific and industrial problems. It can be said that there are several issues which have not been developed fully in grid management, yet. For instance; the development of network communication technologies effect this environments in a large scale. Therefore such problems will be reduced to an admissible level with advancing in critical areas.

In this study, a flexible, easy manageable QoS aware, hybrid (CPU+GPU) supported resource management for GPGPU supported Grid computing is proposed. On the other hand, it can manage both large and small scale resources. This resource management infrastructure supports heterogeneous GPUs on same node (Nvidia and AMD Radeon). Both can be programmable through OpenCL library. In addition, CUDA optimization is integrated to take best performance from Nvidia cards. Design challenges in both Grid and resource management infrastructures are introduced. Design of resource management is modified to obtain performance and efficiency as maximum as possible. Grid computing will gain more and more importance with the development of the network communication technologies. For this reason, instead of hardware novation, the maximum utilization from the available hardware should be essence for computational environments.

As a future work reliability measurement of resources will be able to be integrated to measure trustworthy of different type of resources by taking periodical information from these resources through specialized tracking agents. As mentioned above an automatic processor selecting in run-time integration of module being responsible for management of neglected processes by GPU or CPU and then load these processes to the available GPU or CPU to the resource management is considered. This alternative automatic selection algorithm between GPU and CPU, has been considered. Rather than performing this selection according to application requirements, it looks availability of processors. In this algorithm, when a GPU is unavailable (cannot proceed computations), CPU will undertakes the neglected processes by GPU or vice versa. When a CPU is unavailable (cannot proceed computations), GPU will undertakes the neglected processes by GPU. But there is several issues; how can be such an operation done; how can a parallel application using a number of GPU processor cores which exceeds the number of CPU cores on a server, be ported to a CPU, and how can an algorithm be modified to perform this adaption process? Can all applications, working on GPU, work on CPU or vice versa? This is a future concept feature considered to add to the resource management systems when above issues solved.

ACKNOWLEDGMENT

We are grateful to volunteer organizations and volunteers who allow middleware software to perform tests in their computers, firstly. We thank especially grid service providers allowing integration of resource management infrastructure to their platforms.

REFERENCES

[1] S. Kounev, R. Nou, and J. Torres, Autonomic QoS-Aware resource management in grid computing using online performance models. the 2nd international conference on Performance evaluation methodologies and tools (ValueTools '07). ICST [Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering], ICST, Brussels, Belgium, Belgium, Article 48, 2007, 10 p.

[2] W. Cai, G. Coulson, P. Grace, G. Blair, L. Mathy, and W. K. Yeung, The Gridkit Distributed Resource Management Framework. European Grid Conference, EGC., Springer Verlag, 2005, pp. 786-796

[3] A. Younes, M. Essaaidi, A. El moussaoui, and A. Bendahmane, Grid computing middleware information systems: Review and synthesis study, Multimedia Computing and Systems, 2009. ICMCS '09. International Conference on, vol., no., pp.530-534, 2-4 April 2009B. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.
[4] T.-Y. Liang and Y.-W. Chang, GridCuda: A Grid-Enabled CUDA Programming Toolkit, Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on , vol., no., pp.141,146, 22-25 March 2011
[5] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid, High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on , vol.1, no., pp.283,289 vol.1, 14-17 May 2000
[6] J. Cao, S.A. Jarvis, S. Saini, G. R. Nudd, GridFlow: workflow management for grid computing, Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on , vol., no., pp.198,205, 12-15 May 2003
[7] M. Murshed, and R. Buyya, Using GridSim Toolkit for Enabling Grid Computing Education, 2001
[8] V. Sahota, L. Maozhen, and G. Wenming, Resource Monitoring with Globus Toolkit 4, Semantics, Knowledge and Grid, 2006. SKG '06. Second International Conference on , vol., no., pp.79,79, 1-3 Nov. 2006
[9] H. Lee; D. Park; M. Hong; S. Yeo; S. Kim; Sk. Kim, "A Resource Management System for Fault Tolerance in Grid Computing," in Computational Science and Engineering, 2009. CSE '09. International Conference on , vol.2, no., pp.609-614, 29-31 Aug. 2009
[10] Z. Wei; L. Minghao; L. Jinxia; T. Yuanming; T. Ma, "Design and Implementation of National Meteorological Computing Resource Management System Based on Grid," in Information Science and Engineering (ICISE), 2009 1st International Conference on , vol., no., pp.182-185, 26-28 Dec. 2009
[11] M. Fukuda; Ngo Cuong; E. Mak; J. Morisaki,, "Resource Management and Monitoring in AgentTeamwork Grid Computing Middleware," in Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on , vol., no., pp.145-148, 22-24 Aug. 2007
[12] F. Li; D. Qi; L. Zhang; X. Zhang; Z. Zhang, "Research on Novel Dynamic Resource Management and Job Scheduling in Grid Computing*," in Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums on , vol.1, no., pp.709-713, 20-24 June 2006

Emrah Dönmez is a Ph.D. student in the Computer Engineering Department at the Inonu University. His research interests are: image analysis, robotic, machine learning, pattern recognition, cloud computing, data mining with HPC systems, GPGPU computing and acceleration, global, grid, volunteer and hybrid computing. He is a student member of IEEE.