Grid Management System and Information System for Semantic Grid Middleware

Hyung-Lae Kim[1], Byong-John Han[2], In-Yong Jeong[2] and Chang-Sung Jeong[2]
[1]School of Visual Information Processing, Korea University
Anamdong 5-ga, Sung-buk gu, Seoul 137-710, Korea
[e-mail: nolight@korea.ac.kr]
[2]School of Electronics and Computer Engineering, Korea University
Anamdong 5-ga, Sung-buk gu, Seoul 137-710, Korea
[e-mail: {guru1013, dekarno, csjeong}@korea.ac.kr]
*Corresponding author: Chang-Sung Jeong

Received July 5, 2010; revised September 21, 2010; accepted September 30, 2010; published December 23, 2010

Abstract

Well organized and easy usable Grid management system is very important for executing various Grid applications and managing Grid computing environment. Moreover, information system which can support Grid management system by providing various Grid environment related information is also one of the most interesting issue in the Grid middleware system area. Effective cooperation between Grid management system and information system can make a novel Grid middleware system. Especially, service oriented architecture based Grid management system is flexible and extensible for providing various type of Grid services. Also, information system based on data mining process which comprises various different kinds of domains such as users, resources and applications can make Grid management system more precise and efficient. In this paper, we propose semantic Grid middleware system which is a combination of Grid management system and semantic information system.

Keywords: Grid, middleware, semantic, data mining, resource allocation

This research was supported by the Industry Core Technology Development Project supervised by the KEIT(Korea Evaluation Institute of Industrial Technology)(KEITK1002153), the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)(NIPA-2010-C1090-1001-0008), the Seoul Research and Business Development Program, Smart City Consortium(10561), Future-based Technology Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0020-732), and Samsung Electronics.

DOI: 10.3837/tiis.2010.12.006

1. Introduction

To make efficient use of a Grid environment, an easy-to-use and stable Grid management system must be necessary for providing various Grid services such as resource allocation, authentication, data transferring, scheduling and monitoring. Moreover, information system which reflects real time information of Grid environment for searching most suitable services is another necessity of Grid middleware system. Many Grid middleware systems have their own service implementations and information broker to meet the minimum requirements of Grid middleware system. Globus is one of the most well-known Grid middleware systems that contain services such as authentication, resource allocation, and data management. Recently, Globus provides a way to implement services as a form of web services so that user can easily access to the Globus through the various system environment. However, Globus is not easy to be managed because each of services is heavy and implemented separately. Moreover, due to its lack of information service, Globus is hard to provide intellectual services considering network and feature of Grid applications. To overcome these limitations, we already proposed agent-based Grid management system which encapsulated various Grid services into web services and semantic information system that provides high level service to Grid users by analyzing and inferring Grid environment information. In this paper, we will introduce an advanced semantic Grid middleware system for effective scheduling and monitoring services based on multiple semantic information sources. This paper is organized as follows. Section 2 introduces our system architectures. In section 3, we explain evaluations and experiments of semantic Grid middleware system with an example of labeling algorithm application. Finally, we summarize our work and conclude this paper in section 4.

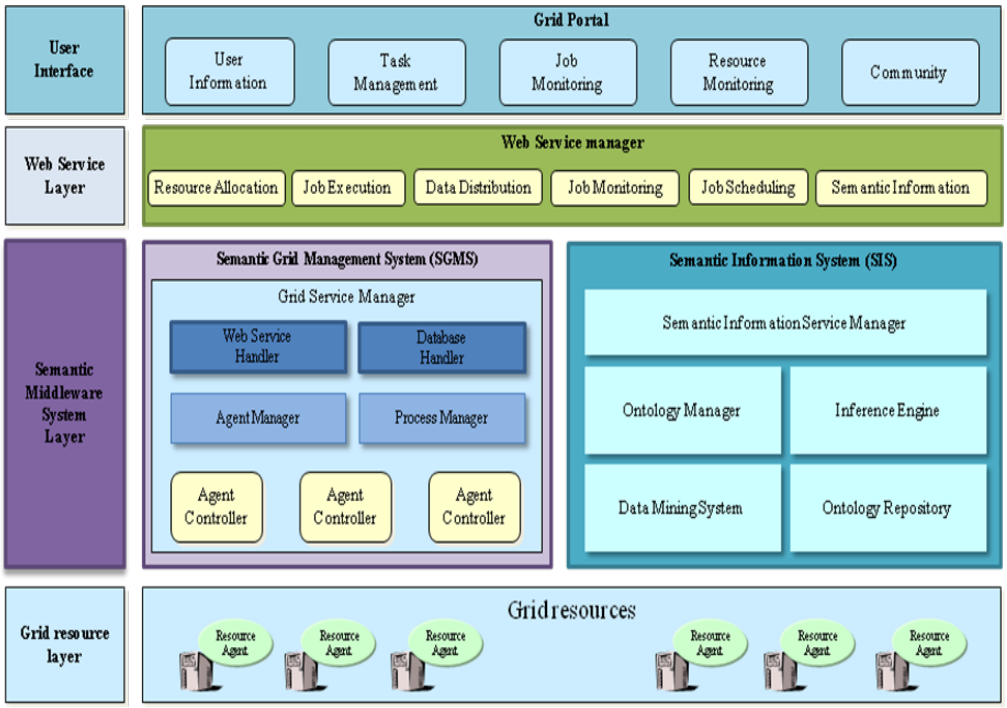


Fig. 1. Overall System Architecture

2. System Architecture

Our semantic Grid middleware system consists of three sub systems. Semantic Grid Management System (SGMS) [1] not only takes charge of various Grid services such as job executing, resource allocation, and data distribution, but also contains functions for resource information extracting and monitoring service. Semantic Information System (SIS) [2] makes ontology which contains information about Grid resources and application features by interacting with SGMS, and provides semantic information for efficient Grid services. Grid Portal System is web service based user interface for two semantic Grid middleware systems to provide easy accessibility to Grid users. Users can easily access to the Grid services through the Grid Portal System, and also can monitor resources and application status in the Grid environment. We will explain features and components of each system in more detail in the next sub sections.

2.1 Semantic Grid Management System (SGMS)

SGMS is a centralized Grid service middleware system for Grid services by using agent-based resource management technology. An agent-based approach can be employed for integrating and coordinating distributed resources in computational Grid environments. Also, this approach can improve scalability, heterogeneity, and interoperability. SGMS consists of four sub components: Grid Service Manager, Process Manager, Agent Manager, and Resource Agent. Grid Service Manager is a kernel of our Grid service management system which is bound with web service interface. Process Manager controls life cycle of Grid tasks, and takes charge of scheduling and monitoring services. Agent manager controls a virtual resource pool which is made up of combination of Agent Controllers. Agent Controller is a proxy component of Resource Agent. Resource Agent is an actual component which executes services such as data management, remote execution, and resource information extraction, and located in each of Grid resources. We will describe each of components more in detail.

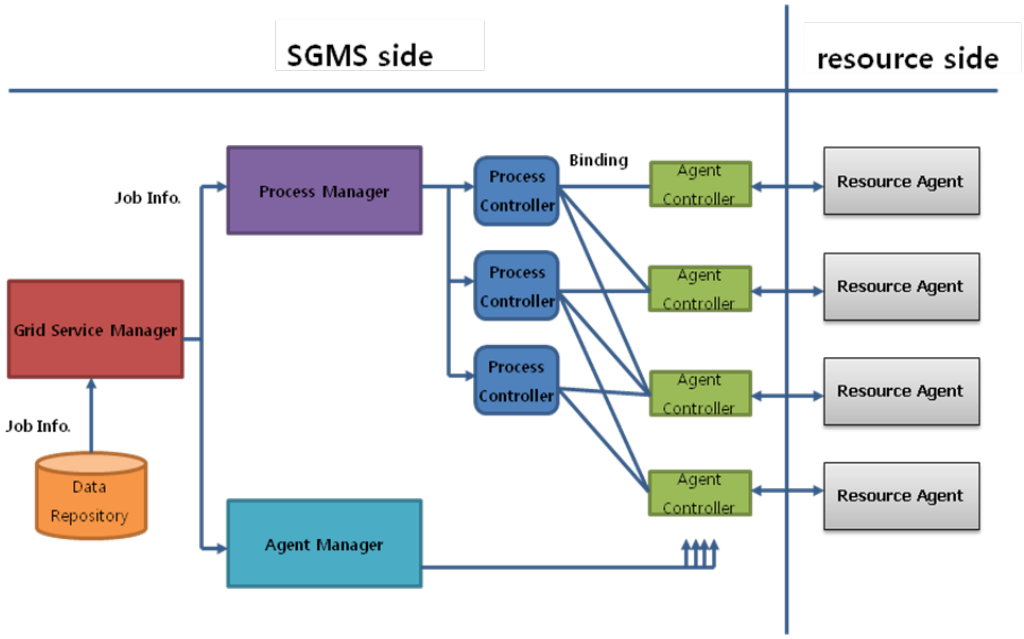


Fig. 2. SGMS components and their relations

2.1.1 Grid Service Manager (GSM)

Grid Service Manager is a kernel of SGMS. It has communication modules that interact with web services and databases. And it has Agent Manager and Process Manager for controlling Grid resource and applications.

Grid Service Manager has two interface handlers: web service handler and database handler. Web service handler parses and disposes requests of service from Grid portal or web service client. Grid Service Manager disposes requests based on the type of service. If the request is a Grid service from the Grid portal like job execution or data distribution, Grid Service Manager passes the request to Agent Manager as a translated format of command. When the request is for resource information from web service client or SIS, Grid Service Manager collects resource information from Agent Manager and database, and returns it to the service requester. This web service handler can provide flexibility and extensibility of service implementations, and also give a uniform interface to Grid Portal System or web service client.

Database handler takes charge of storing and managing information of users, tasks, and resources which used in Grid management services. When web service handler receives request of service, Grid Service Manager queries information related to the request from database through the database handler to build a service command to send it to Agent Controllers for executing Grid services. Database handler maintains connection of database between database server and Grid Service Manager, and regulates access level to the database.

2.1.2 Resource Agent (RA)

RA is a agent component which can be installed in Grid resources. RA has encapsulated Grid services such as data distribution, job execution, and information monitoring. Any Grid resource owner who wants to join Grid resource pool can obtain the RA which is opened to the public. When resource providers get a RA, they can install it to their own Grid resource with a particular configuration. This configuration contains information of user, security, resource properties, restrictions of utility, and so forth. This kind of Grid resource information is passed to Agent Controller which is located in Agent Manager at Grid management server, and the Agent Controller stores passed information to its local memory. Grid management system can control each of Grid resource based on information in Agent Controller for providing various Grid service individually.

Most of Grid management system provides various Grid services such as security, data management, remote execution and monitoring. These services are usually separated in different demons. In the case of Globus toolkit [3], for example, each Grid services main body such as GSI [4] for security, GridFTP for data management, GRAM [5] for execution management, and MDS for information service are packed in different components. The separation of service component can provide a flexibility of service selection, but it can make sharing of Grid resource more complex. To give easy use environment to resource provider, our Resource Agent offers a uniform interface for lots of Grid services within single communication channel. Resource Agent capsules different Grid services into one single component to give a simple installation environment and maintain cost.

Resource Agent is located in each Grid resources, active as a demon, connected with Agent Controller. Resource Agent controls operations Grid services, and executes Grid services such as data transferring, job executing, user authenticating, and resource monitoring. Also, Resource Agent collects information of Grid resource such as CPU clock, storage, memory, library which its resource have, and system properties. This information is transmitted to Agent Controller for providing resource information service. Grid management system

provides dynamic resource information service as a form of web service by using the information in Agent Controller.

2.1.3 Agent Manager and Agent Controller

Agent Manager is a master controller of Resource Agent that makes Grid resources as one single virtual Grid resource pool. It manages Agent Controllers and their life cycles. It waits for request of connections from Grid resource, and when it receives connection request, Agent Manager generates a new Agent Controller which is bound to the Grid resource. And then, Agent Manager adds the Agent Controller to agent list for expanding current Grid resource group. After then, each Grid resources can communicate with other Grid resources through Agent Manager.

Agent Controller is a proxy of Resource Agent which is managed by Agent Manager, and it has a one-to-one relation with Resource Agent. Grid Service Manager controls Grid resources by sending commands to Agent Manager, and this Agent Manager passes it to the correspondinig Agent Controllers to execute Grid services. Agent Controllers send service commands or transfer data to Resource Agent when it receives a Grid service command from Agent Manager.

2.1.4 Process Manager and Process Controller

Process Manager is a master component of Grid applications, which is created by Grid Service Manager. Process Manager receives application data from Grid portal, and generates Process Controllers which take charge of each of life cycles of Grid application. Each Process Controllers is placed under the Process Manager control before application execution being completed. Process Manager not only takes charge of life cycles of Process Controllers, but also provides scheduling and monitoring services.

Process Controllers are generated by Process Manager based on application data passed through Grid portal. Process Controller has a one-to-one relationship with Grid application, and also contains information about Grid application such as process procedure and allocated resources. Process Controller collects Agent Controllers based on Grid application information from the Agent Manager, and it orders Agent Controllers to execute Grid applications.

Our proposed Grid management system architecture has a flexibility and extensibility due to the separation of Process Manager and Agent Manager. In other words, Grid Service Manager controls Grid resources and applications independently, but it can enable lots of combinations of Grid resources and application easily. Moreover, to select Grid resources for executing Grid application, one Grid application can flexibly access to the Agent Controllers through the Agent Manager without resource dependency. **Fig. 2** shows that one Process Controller which takes charge of execution of Grid application can make virtual Grid resource pool by comprising several Agent Controllers in Agent Manager. From a point of view of Grid resource, one Grid resource does not depend on specific Grid application, and can be shared by one or more Process Controllers simultaneously. So this architecture can make high usability, sharability, and extensibility as well. Especially, to make high usability, Process Manager must have adjustment function for the competitions between Process Controllers. Scheduling and monitoring services based on semantic information can solve this competition condition for effective usability of Grid environment. Section 3 shows the detailed description of scheduling and monitoring services.

2.2 Semantic Information System (SIS)

SIS provides high level information to Grid management system or Grid users for effective utilizing Grid environment based on multi-dimensional information sources. Different kinds of information sources provide more rich Grid environment information to the SIS. **Fig. 3** shows components and their relation of SIS.

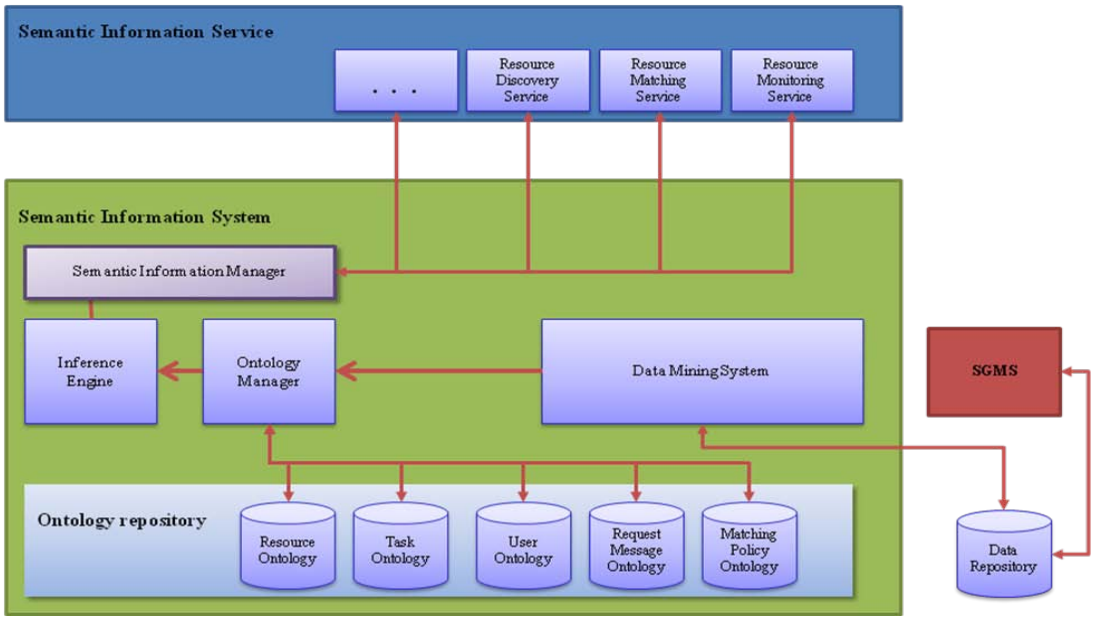


Fig. 3. SIS components and their relations

2.2.1 SIS Components

Ontology Repository is a storage which contains ontology files described with semantic web language such as OWL [6] or RDF [7]. These files consists of ontology schemas and instances about Grid environment information. Ontology Manager takes charge of maintaining ontology files in the Ontology Repository, and communicates with Data Mining System and inference engine. Inference Engine is a reasoning component based on ontology data. Ontology files have just basic information about Grid environment, and inference engine can generate additional information by extending information relations. Semantic Information Manager deploys semantic information services as a form of web service. Semantic Information Manager receives a request of information services and handles the request by using inference engine.

2.2.2 Data Mining System (DMS)

DMS is a sub system of SIS. The main role of DMS is a reflection of real time Grid environment information by using information modeling and analyzing Grid resources and applications. DMS collects Grid environment information from SGMS and database, and reports useful information or updated data of Grid environment to Ontology Manager. Ontology Manager updates ontology files in the Ontology Repository for providing more accurate semantic information service. DMS can make SIS more strong and reliable by reflecting real time Grid environment changes.

2.3 Multiple Information Sources

Our middleware system has various information sources for making accurate information services. Each of information sources has different kinds of information types. One is Resource Agent that can collect static and dynamic information about Grid resources. Static Grid resource information consists of such hardly changeable data as CPU clock, storage space, memory size, etc. Dynamic resource information is frequently changed data such as CPU and memory utilization. Another information source is Process Controllers. Process Controllers have Grid application information. Status of application, time stamps of execution, and other application related information is stored in the Process Controllers. Process Controller can provide application information with its related resource information by communicating with Agent Controllers for determining another application's resource matching service. And the other information source is DMS. DMS is a higher level of information source by executing three steps of information processing: collecting, modeling, and reporting. DMS collects information from the Resource Agents or database. And it makes models of its related information for analyzing resources and application. When modeling process has done, it reports its result to the Ontology Manager which updates ontology files in the Ontology Repository.

Our system has various information sources for semantic information services. Each of information sources has different kind of information, which enables us to improve accuracy of information service.

3. Evaluation

3.1 Evaluation Overview

Our semantic Grid middleware has a strong point for an application executing environment. Grid application users can easily execute their applications by describing job descriptions and service level agreements of resources through Grid Portal System. And also, our Semantic Grid Management System provides various Grid services as a form of web service and encapsulates these services into a single Grid service component. Thus, user can easily access to the Grid middleware system and take their application execution result through the Grid Portal System. But, most of Grid middleware systems are focusing their system purpose on executing environment, since it is not easy to improve application performance by using Grid middleware system services. Similarly, the main object of our middleware system is providing easy usable Grid services and high level information services. So our Grid management system can provide easy usable application execution environment, but it does not consider the performance improvement of Grid application. From a point of view of Semantic Information System, the main object of this system is providing additional information about applications, resources, and users for making better performance of Grid application by using semantic information. These semantic information processing has many analyzing processes for various domains such as resources, applications, and users. One example of semantic information processing is a resource analysis. The resource analysis collects information of resources which include both dynamic and static values. Semantic Information System can identify resource specification based on analyzed resource information. So, Semantic Information System can classify Grid resources according to application dependency. Another example is a task analysis. The task analyzing is classifying application types in related with what kinds of hardware elements are strongly dependent for the performance of Grid applications based on executing performance on various difference Grid resources. So, in this paper, we will try to

show the process of semantic information processing about task feature analysis by using resource matching and performance comparison. Especially, we will consider calculation power dependent application which can be solved using either CPU or GPU to show which hardware elements are more significant for the performance of our sample application.

3.2 Fast Parallel Connected-Component Labeling Algorithm

Connected-Component labeling algorithm is calculation for giving unique labels to the all of connected pixels which form a one single object. This labeling algorithm is one of the basic skills that are adopted in pattern recognition, computer vision and simulation area. Especially, 2-dimensional connected-component labeling needs fast processing speed for the real time image processing. Many already known algorithms are improved in various researches, but most of previous algorithms use a serial approach based on CPU. To improve connected-component labeling performance, we developed 2-dimensional parallel labeling algorithm suitable to GPGPU application by using NVIDIA CUDA technique. The CUDA (Computer Unified Device Architecture) [8] which is released by NVIDIA can support latest SIMD structured graphic devices with common calculation process through the C language type syntaxes and libraries. SIMD structured GPGPU can access and handle multiple pixels simultaneously by using identical index of threads through the hardware. For executing 2-dimensional labeling process, there must be an additional array space with the same size of original input image for storing label equivalence and output image. **Fig. 4** shows GPGPU based 2-dimensional labeling algorithm diagram.

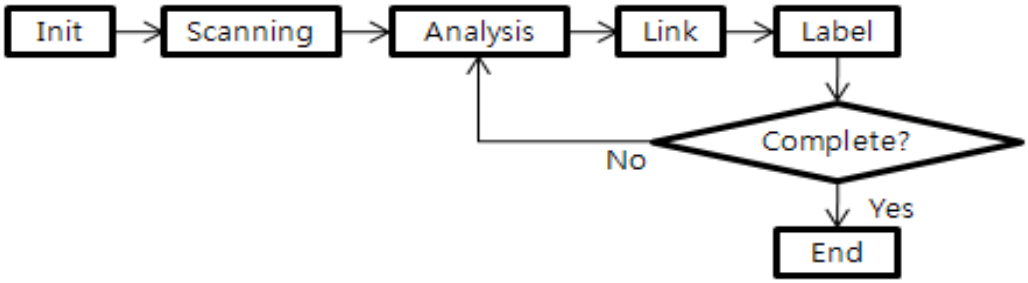


Fig. 4. GPGPU based 2-dimension labeling algorithm diagram

In the initial phase, set initial label values to all index label array. If pixel is located in background area, very large integer value is assigned to this pixel. If pixel is located in the object area, that pixel is set with thread index value. In the scanning phase, execute scanning process to the neighbor pixels by using some masked filters based on initialized labels in the label array. We ignore background pixel in neighbor pixels, and find the minimum value in the masked area including origin pixel, store it in the equivalence array. In the analysis phase, based on label information in the equivalence array, execute a search process until reaching the point where the two values between pixel index and label value are matched. When the search is over, each thread stores start point label values into the pixels which the threads took process. In the link phase, to link one or over two sub-regions which is made through analysis process, execute scanning process to all neighbor pixels of each pixel in the equivalence array. If the founded sub-region label value is smaller than self pixel, thread set the previous start value of sub-region into a newly founded sub-region start value. In the label phase, execute a final labeling by passing start value of label of each pixel in the equivalence array to the label array. When the label phase is over, to confirm the labeling process, our algorithm tries to scan once again for all neighbor pixels in the label array. If the labeling still remains the same, go

back to the analysis step for processing the remaining area. **Fig. 5** shows the example of this labeling algorithm process.

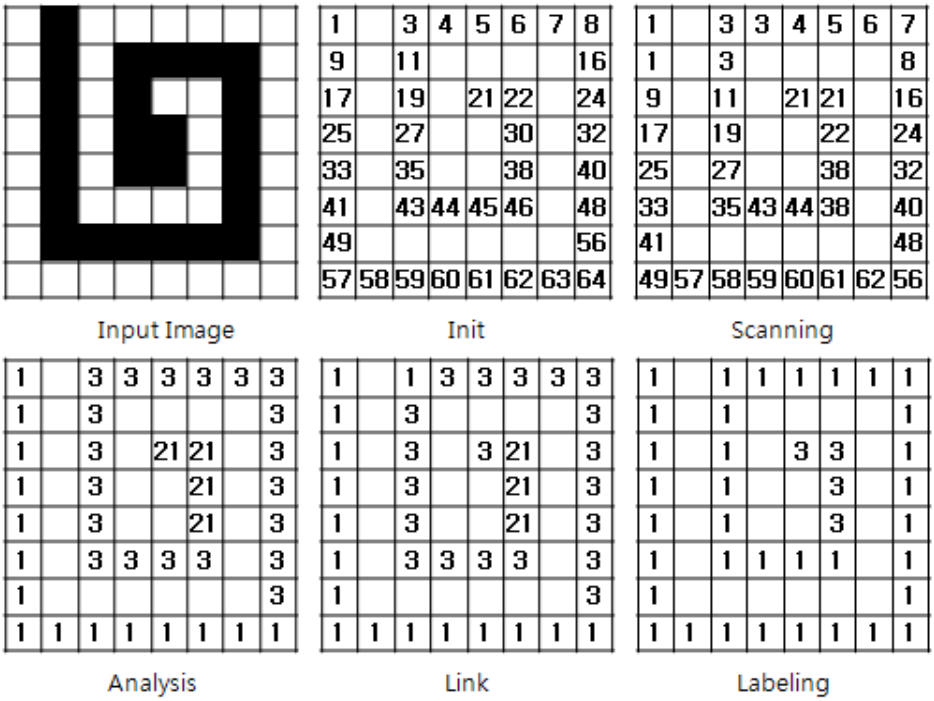


Fig. 5. GPGPU based labeling algorithm example

3.3 Resource Analysis

Due to the heterogeneity of Grid computing environment, each of resources in the resource pool has difference hardware features. Basically, our Resource Agent in Semantic Grid Management System has a function for extracting resource information such related information as CPU, memory, and storage. And also, Resource Agent can collect dynamic resource information which has easily changable tendency in real time execution process. But, it is not easy to extract all of the resource features through one agent component because each hardwares has various sub categories. To solve this problem, we use one of the most famous resource information extracting program whose name is Everest. The Everest software released by Lavalys Consulting Group is a powerful hardware analysis program that contains lots of useful services such as benchmarking between difference vendors, performance simulation based on CPU power dependent algorithm, and easy profiling and reporting services. Especially, Everest can be applied at all kinds of hardwares in the current market, and support frequent update for following a fast changable trend in the hardware market. We can collect wide range of hardware feature for each of sample resources, but we had to limit the hardware elements for effective analysis. The elements of hardware that needs to be considered in case of executing application is CPU, GPU, memory, and storage. In case of memory, there are four sub elements of hardware property that is related with speed performance of memory. We will analyze the speed of read, write, copy, and delay of memory in each resource. CPU has two types of performance features. One is common execution speed, and another is floating point execution speed. For testing common execution speed, Everest applies some well known calculation dependent algorithm such as Queen or PhotoWorxx. To

check the performance of floating point execution, Everest simulates Julia and Mandel algorithm. In case of GPU, there are two key elements of hardware performance. One is GPGPU core number, and another is GPGPU clock. The number of GPGPU cores tell the maximum number of thread which that kind of GPU can make at one time, so usually the higher number of GPGPU cores shows the better performance of the GPGPU. The GPGPU clock indicates the speed of processor in the GPU hardwares, but this values is tightly related with the performance of GPGPU, so it may be considered as some reference features. In case of storage performance, most important performance factor of HDD is speed of read and write data. Unfortunately, Everest does not support enough test mode for measuring HDD performance, we use another software for testing HDD performance of each resource. We used CrystalDiskInfo for evaluating HDD performance for the case of sequence read and write, large block(512K) read and write, and small block(4K) read and write. The CrystalDiskInfo provides easy interface and simulation method for measuring various HDD performance, so we can get a performance result through the CrystalDiskInfo software. The details of hardware and software specifications are shown in **Table 1**, and the result of resource analysis of our four sample resources is shown in **Table 2**.

Table 1. Sample resource specifications

	M1	M2	M3	M4
OS	Windows Server 2003 Enterprise	Windows 7 Enterprise 64bit	Windows 7 Enterprise 32bit	Windows XP Professional
CPU_type	Intel Core 2 Duo E6550	AMD Athlon 64 3500+	Intel Core i5 750	Intel Core 2 Quad Q6600
CPU_core	2	1	4	4
CPU_clock	2333 (MHz)	2200 (MHz)	2800 (MHz)	2400 (MHz)
Mem_type	DDR2	DDR	DDR3	DDR2
Mem_size	2048 (Mb)	512 (Mb)	4096 (Mb)	4096 (Mb)
Mem_clock	333 (MHz)	200 (MHz)	609 (MHz)	400 (MHz)
GPU_type	nVIDIA GeForce 9800 GT	nVIDIA GeForce 8800 GT	nVIDIA GeForce 9600 GT	nVIDIA GeForce GTX 260
GPU_uProc /core	14/112	14/112	8/64	27/216
GPU_clock	600 (MHz)	602 (MHz)	650 (MHz)	602 (MHz)
GPU_Mem	512 (Mb)	512 (Mb)	512 (Mb)	896 (Mb)
HDD_space	99998 (Mb)	114400 (Mb)	200000 (Mb)	117200 (Mb)
MB_chipset	Intel G33	nVIDIA nForce4	Intel P55	Intel P35
CUDA	2.2	2.2	2.3	2.2

Producing a resource analysis result is not a difficult process. The values in **Table 2** are actually absolute values that does not contain relative information between each resource. To compare the hardware efficiency between each of sample resources, we have to normalize all the values in **Table 2**. Normalization can reduce the range of each values, thus we can execute easy performance comparison for the sample resources. We used a maximum value pivot normalization for calculating each value of resource performance. This method calculates a proportion of each value in the same region by measuring relative values to maximum value. This method is easy and fast, and also can be applied to various hardware feature domains at the same time. **Table 3** shows the result of normalization from the analysis result for our sample resources. This table shows that M3 is the most powerful performance in lots of hardware domains except GPGPU cores. M2 has poor performance in both GPU and GPGPU

domains. M1 has average performance throughout the hardware domains. M4 has high average performance in CPU domain. Especially, M4 has most powerful performance in GPGPU domain.

Table 2. Resource analysis result

	M1	M2	M3	M4
MEM read(MBps)	6393	5878	12540	6473
MEM write(MBps)	6054	4059	9643	4838
MEM copy(MBps)	5827	4586	13388	5377
MEM delay(ns)	78.7	55.9	56.1	74.8
CPU Queen	9978	3850	19863	17190
CPU PhotoWorxx	11389	3758	28761	16626
CPU Zlib(KBps)	31058	12980	64940	63430
CPU AES	8713	2866	21485	17895
FPU Julia	4380	933	10988	8856
FPU Mandel	2189	533	5777	4383
FPU SinJulia	1092	491	3441	2244
GPGPU core	112	112	64	216
GPGPU clock(MHz)	1512	1500	1625	1296
HDD seq read	73.04	46.86	91.71	70.42
HDD seq write	74.94	46.29	90.63	65.48
HDD 512k read	37.81	24.89	42.16	30.55
HDD 512k write	54.59	28.61	77.58	32.24
HDD 4k read	0.565	0.411	0.562	0.403
HDD 4k write	1.767	1.433	1.827	1.211

Table 3. Resource analysis normalization result

	M1	M2	M3	M4
MEM read(MBps)	0.509808612	0.468740032	1	0.516188198
MEM write(MBps)	0.627812921	0.420927097	1	0.501711086
MEM copy(MBps)	0.435240514	0.342545563	1	0.401628324
MEM delay(ns)	1	0.710292249	0.712833545	0.950444727
CPU Queen	0.502341036	0.19382772	1	0.865428183
CPU PhotoWorxx	0.395987622	0.130663051	1	0.578074476
CPU Zlib(KBps)	0.478256852	0.199876809	1	0.976747767
CPU AES	0.405538748	0.133395392	1	0.832906679
FPU Julia	0.398616673	0.084910812	1	0.805970149
FPU Mandel	0.378916393	0.09226242	1	0.758698286
FPU SinJulia	0.317349608	0.142691078	1	0.652136007
GPGPU core	0.518518519	0.518518519	0.296296296	1
GPGPU clock(MHz)	0.930461538	0.923076923	1	0.797538462
HDD seq read	0.796423509	0.510958456	1	0.767855196
HDD seq write	0.826878517	0.510758027	1	0.722498069
HDD 512k read	0.896821632	0.590370019	1	0.724620493
HDD 512k write	0.703660737	0.368780614	1	0.415571023
HDD 4k read	1	0.727433628	0.994690265	0.713274336
HDD 4k write	0.967159278	0.784345922	1	0.662835249

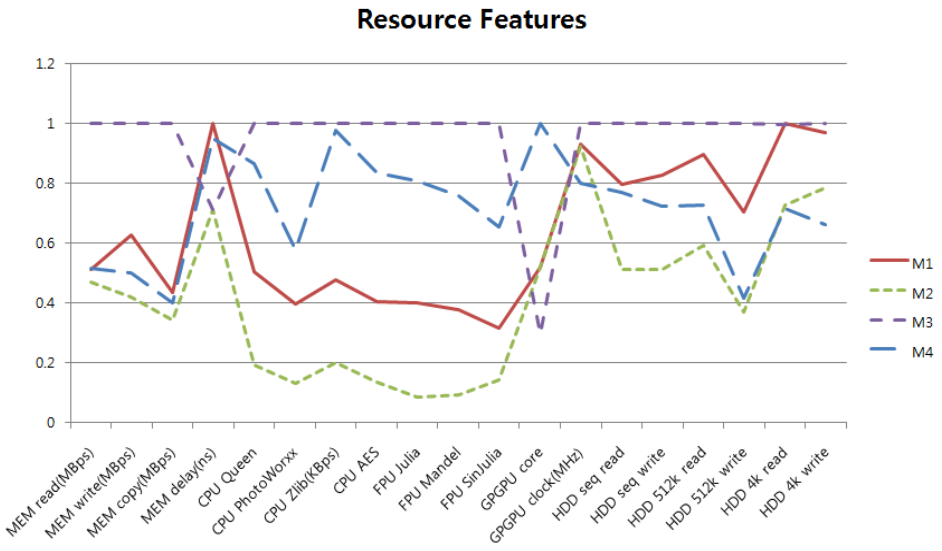


Fig.6. Resource feature normalization result

Table 3 shows relative performance between each of sample resource. However, graphical analysis is much more efficient for task analysis and performance evaluation. With a graphical analysis result, we can easily verify features of each of resources, and it can be useful for calculating task features. **Fig. 6** shows graphical analysis result of our four sample resources. The most important hardware domain is CPU and GPGPU area. For example, in case of CPU, M3 has most powerful performance though all CPU domain, and M2 has the poorest performance. But in case of GPU, M2 has better analysis result than M3. Based on this graphical resource analysis result, we can utilize the resource analysis for calculating task features by comparison of performance margin.

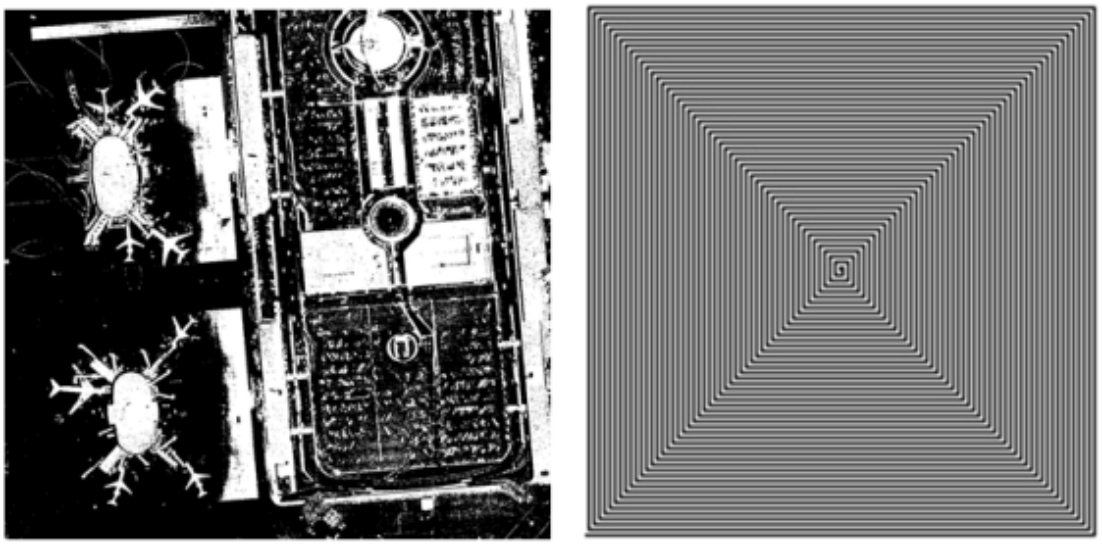


Fig.7. Test images for labeling algorithm
Airport(left) with 1024x1024 and Spiral(right) with 4084x4084

3.4 Performance Analysis

As already explained in section 3.2, we developed a parallel connected-component labeling application. This application shows a better performance in GPGPU environment, but it can be also executed in CPU environment. The purpose of executing this application compares a performance difference between sample resources. If we can get performance variation in resource for each application, we can extract application features. For the preprocessing of task analysis, we executed our labeling algorithm application on each of the sample resources with two sample images. **Fig. 7** shows our two sample images.

Table 4 and **table 5** shows the result of executing our connected-component labeling algorithm application. **Table 4** shows the execution time for each of input images by using both CPU and GPGPU. Due to the difference of test image input size, airport image execution result does not show clear result in case of CPU and CPU except a M4 GPU case. The execution result in spiral4096 shows a distinct difference in both CPU and GPU cases. Especially the result of M4 with GPU shows significant reduction of execution time due to the its outstanding hardware performance. **Table 5** indicates the performance factor of CPU and GPU for the two input images. These values shows relative hardware efficiency in case of CPU and GPU.

Table 4. Application execution result

	M1	M2	M3	M4	MAX
spiral4096					
CPU(ms)	693	979	595	687	979
GPU(ms)	553	625	478	124	625
airport1024					
CPU(ms)	40	55	40	39	55
GPU(ms)	27	25	20	8	27

Table 5. Application execution performance

	M1	M2	M3	M4	MAX
spiral4096					
CPU	0.0014430	0.0010214	0.0016807	0.0014556	0.0016806
GPU	0.0018083	0.0016012	0.0020921	0.0080645	0.0080645
airport1024					
CPU	0.0250000	0.0181818	0.0250000	0.0256410	0.0256410
GPU	0.0370374	0.0400000	0.0500000	0.1250000	0.1250000

3.5 Task Feature Analysis

The main purpose of our experiment is analyzing specific tasks by using resource analysis and performance variation. Most of Grid application usually has a repeatable execution feature with different parameters. For example, gene pattern searching application has a simple process algorithm with different gene input data area. Also, in case of volume rendering, without the change of algorithm and input data, user usually want to get a different output images based on the point of view value. So, it is important issue for allocating these kinds of repeatable application for a most suitable resources.

To find the most recommendable resource for a Grid application, analyzing input task feature is very important issue in this problem. In this paper, we propose a reverse task feature analyzing method by using resource analysis and performance variation. In the previous section, we already explained the resource analysis and performance analysis process. In the

section 3.3, we already got a resource analysis data from Resource Agent. As we explained in section 3.4, we also had our labeling application performance result. The main idea of our task feature analyzing method is performance variation comparison between two different sample resources. We know all the features of test bed resources. And if there is some variation in both the resources and result performance, we can inference what kind of hardware elements are strongly or loosely effected to the application performance by comparing with all case of combination between resources. **Table 6** shows the resource variation for each of combination of our four sample resources. This table shows what kind of hardware elements are changed between two resources. **Table 7** shows the result of performance variation between each of combination of our sample resources. Based on the data in **Table 6** and **Table 7**, we can extract strongly coupled hardware elements with our labeling application.

Table 6. Resource variations between each of sample resources

	M1-M2	M1-M3	M1-M4	M2-M3	M2-M4	M3-M4
MEM read(MBps)	0.04107	-0.4902	-0.0064	-0.5313	-0.0474	0.48381
MEM write(MBps)	0.20689	-0.3722	0.1261	-0.5791	-0.0808	0.49829
MEM copy(MBps)	0.09269	-0.5648	0.03361	-0.6575	-0.0591	0.59837
MEM delay(ns)	0.28971	0.28717	0.04956	-0.0025	-0.2402	-0.2376
CPU Queen	0.30851	-0.4977	-0.3631	-0.8062	-0.6716	0.13457
CPU PhotoWorxx	0.26532	-0.604	-0.1821	-0.8693	-0.4474	0.42193
CPU Zlib(KBps)	0.27838	-0.5217	-0.4985	-0.8001	-0.7769	0.02325
CPU AES	0.27214	-0.5945	-0.4274	-0.8666	-0.6995	0.16709
FPU Julia	0.31371	-0.6014	-0.4074	-0.9151	-0.7211	0.19403
FPU Mandel	0.28665	-0.6211	-0.3798	-0.9077	-0.6664	0.2413
FPU SinJulia	0.17466	-0.6827	-0.3348	-0.8573	-0.5094	0.34786
GPGPU core	0	0.22222	-0.4815	0.22222	-0.4815	-0.7037
GPGPU clock(MHz)	0.00738	-0.0695	0.13292	-0.0769	0.12554	0.20246
HDD seq read	0.28547	-0.2036	0.02857	-0.489	-0.2569	0.23214
HDD seq write	0.31612	-0.1731	0.10438	-0.4892	-0.2117	0.2775
HDD 512k read	0.30645	-0.1032	0.1722	-0.4096	-0.1343	0.27538
HDD 512k write	0.33488	-0.2963	0.28809	-0.6312	-0.0468	0.58443
HDD 4k read	0.27257	0.00531	0.28673	-0.2673	0.01416	0.28142
HDD 4k write	0.18281	-0.0328	0.30432	-0.2157	0.12151	0.33716

Table 7. Performance variations between each of sample resources

spiral4096	M1-M2	M1-M3	M1-M4	M2-M3	M2-M4	M3-M4
CPU(ms)	0.00042	-0.0002	-0.00000	-0.00070	-0.00040	0.00023
GPU(ms)	0.00021	-0.0003	-0.00630	-0.00050	-0.00650	-0.00600
airport1024	M1-M2	M1-M3	M1-M4	M2-M3	M2-M4	M3-M4
CPU(ms)	0.00682	0.00000	-0.00060	-0.00680	-0.00750	-0.00060
GPU(ms)	-0.00300	-0.01300	-0.08800	-0.01000	-0.08500	-0.07500

After collecting the resource and performance variations between each of sample resources, we can find a weight value for each of hardware elements. These hardware weight values indicate effects which are contributions of each elements to the application performance result. By executing multiplication operation to the variation of each of resources and performance, we can find a hardware weight value for each element. Based on these hardware weight

Table 8. Hardware element weight values on CPU with spiral image

TASK1(CPU)	TR1	TR2	TR3	TR4	TR5	TR6	AVR
MEM read(MBps)	1.73125E-05	0.000116504	8.03996E-08	0.000350218	2.05998E-05	0.000108891	10.22676151

MEM write(MBps)	8.72129E-05	8.8458E-05	-1.58922E-06	0.000381737	3.50727E-05	0.000112149	11.71734751
MEM copy(MBps)	3.90756E-05	0.000134227	-4.23602E-07	0.000433408	2.5651E-05	0.000134674	12.77687714
MEM delay(ns)	0.000122127	-6.82511E-05	-6.24527E-07	1.67528E-06	0.000104263	-5.34787E-05	1.761843282
CPU Queen	0.000130054	0.000118279	4.57585E-06	0.000531446	0.000291578	3.02878E-05	18.43701529
CPU PhotoWorxx	0.000111848	0.000143556	2.29477E-06	0.000573086	0.000194245	9.4962E-05	18.66653181
CPU Zlib(KBps)	0.000117351	0.000124003	6.2823E-06	0.000527459	0.000337281	5.23334E-06	18.62683558
CPU AES	0.000114722	0.000141286	5.38596E-06	0.000571285	0.000303695	3.76074E-05	19.56636268
FPU Julia	0.000132243	0.000142931	5.13373E-06	0.000603247	0.000313051	4.36699E-05	20.67125382
FPU Mandel	0.000120839	0.000147613	4.78625E-06	0.0005984	0.000289336	5.43093E-05	20.25473769
FPU SinJulia	7.36275E-05	0.000162246	4.21919E-06	0.000565157	0.000221177	7.82931E-05	18.41199941
GPGPU core	0	-5.28157E-05	6.06793E-06	-0.000146494	0.000209037	-0.000158381	-2.37643225
GPGPU clock(MHz)	3.11299E-06	1.65273E-05	-1.67518E-06	5.07094E-05	-5.4503E-05	4.55677E-05	0.995652005
HDD seq read	0.000120338	4.83842E-05	-3.60036E-07	0.000322387	0.000111533	5.22484E-05	10.9088357
HDD seq write	0.000133261	4.11459E-05	-1.31547E-06	0.000322519	9.19277E-05	6.24569E-05	10.83324837
HDD 512k read	0.000129185	2.45225E-05	-2.17019E-06	0.000270037	5.82853E-05	6.19792E-05	9.030646553
HDD 512k write	0.000141169	7.04312E-05	-3.63069E-06	0.000416114	2.03142E-05	0.000131536	12.93222899
HDD 4k read	0.000114901	-1.26197E-06	-3.6135E-06	0.000176181	-6.14731E-06	6.33378E-05	5.72328376
HDD 4k write	7.70651E-05	7.80528E-06	-3.83528E-06	0.000142164	-5.27543E-05	7.58851E-05	4.105496363

Table 9. Hardware element weight values on GPU with spiral image

TASK2(GPU)	TR1	TR2	TR3	TR4	TR5	TR6	AVR
MEM read(MBps)	8.55534E-06	0.000139083	3.99E-05	0.00026141	0.000307	-0.00289	-35.5644
MEM write(MBps)	4.30981E-05	0.000105601	-0.00079	0.00028493	0.000522	-0.00298	-46.8178
MEM copy(MBps)	1.93101E-05	0.00016024	-0.00021	0.0003235	0.000382	-0.00357	-48.3174
MEM delay(ns)	6.03514E-05	-8.14783E-05	-0.00031	1.2504E-06	0.001552	0.001419	44.02817
CPU Queen	6.4269E-05	0.000141202	0.002272	0.00039668	0.004342	-0.0008	106.859
CPU PhotoWorxx	5.5272E-05	0.000171378	0.001139	0.00042776	0.002892	-0.00252	36.09902
CPU Zlib(KBps)	5.79916E-05	0.000148035	0.003119	0.0003937	0.005022	-0.00014	143.3601
CPU AES	5.66924E-05	0.000168668	0.002674	0.00042641	0.004522	-0.001	114.1586
FPU Julia	6.53507E-05	0.000170632	0.002548	0.00045027	0.004661	-0.00116	112.2867
FPU Mandel	5.97153E-05	0.000176221	0.002376	0.00044665	0.004308	-0.00144	98.75998
FPU SinJulia	3.63846E-05	0.00019369	0.002094	0.00042184	0.003293	-0.00208	66.03521
GPGPU core	0	-6.30515E-05	0.003012	-0.0001093	0.003113	0.004203	169.254
GPGPU clock(MHz)	1.53835E-06	1.97303E-05	-0.00083	3.785E-05	-0.00081	-0.00121	-46.5536
HDD seq read	5.94676E-05	5.77612E-05	-0.00018	0.00024063	0.001661	-0.00139	7.556148
HDD seq write	6.58537E-05	4.91201E-05	-0.00065	0.00024073	0.001369	-0.00166	-9.76489
HDD 512k read	6.38395E-05	2.9275E-05	-0.00108	0.00020156	0.000868	-0.00164	-25.9914
HDD 512k write	6.97616E-05	8.40809E-05	-0.0018	0.00031059	0.000302	-0.00349	-75.4319
HDD 4k read	5.67806E-05	-1.50654E-06	-0.00179	0.0001315	-9.2E-05	-0.00168	-56.3219
HDD 4k write	3.80834E-05	9.31796E-06	-0.0019	0.00010611	-0.00079	-0.00201	-75.8268

elements in each of combination of sample resource, we can find a influence proportion at every case of resource changes. But, it is not a common impact ratio for all kind of resource changes. So, for calculating common weight values of each hardware element in all cases, we have to apply some compensation method for reducing a possibility of exceptional case. Most general method is applying some kinds of average calculation metric such as median, mean, and average numerical formula. In this paper, we used average calculation formula for determining common weight values of each hardwares elements, due to the its accuracy and conveniency. **Table 8** shows the result of hardware elements weight value on CPU with a spiral image. **Table 9** shows the result of hardware elements weight value on GPU with a spiral image.

The most important attribute is the average value of each hardware element. This value shows how strongly the hardware factors give an influence to the application performance in common case. As already mentioned in this section, finding a task feature is the main purpose of our experiment for the efficiency of executing applications. By analyzing resource features and collecting performance variations between the each compbination of our sample resources,

we can calculate an impact factor of each hardware element for the performance of application. As a result of our experiment for the connect-component labeling application, finally we produce our application feature analysis result shown in **Fig. 8** and **Fig. 9**.

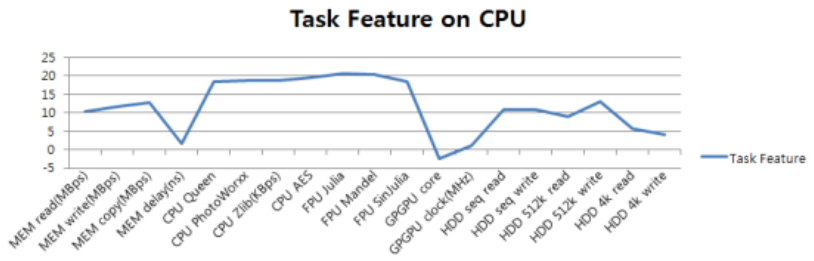


Fig.8. Feature analysis result of labeling application on CPU

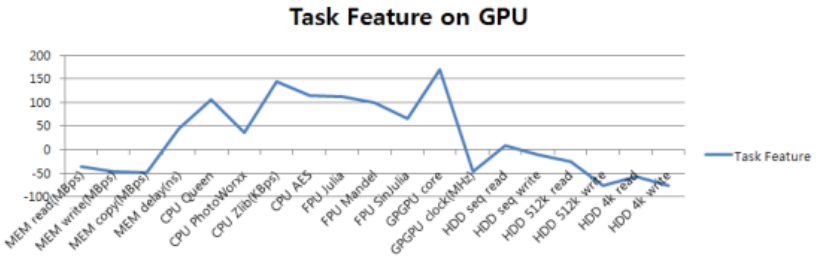


Fig.9. Feature analysis result of labeling application on GPU

3.6 Matching Score and Resource Allocation

Throughout the section 3, we introduced fast connected-component labeling application. Moreover, we already showed information processing steps which include resource analysis, application performance analysis and taks feature analysis. The main purpose of our semantic information processing is finding a relation between resources and applications. Semantic information system can find a suitability of secured resouces for executing Grid applications, so it can provide a high system usability by making a resource allocation order. To indicate a suitability of resources for Grid application, which result from semantic information processing, we introduce a matching score. Matching scores are numercal values of computing resources which indicate a suitability for Grid application. Matching score is not a fixed value of computing resource but a relative value from the secured Grid resources. It can be different at any instance of semantic information processing based on Grid applications or applied algorithms.

In section 3.3 and 3.5, we already performed resource analysis and task feature analysis processes. We can get an application executing efficiency values of each resource by using each of hardware element weight values of Grid applications and resources based on resource analysis normalization result and feature analysis result. Semantic information system can apply many different kinds of algorithms for calculating matching scores, but in this paper, we appied multiplication operation to each of hardware elements of Grid resources and application feature, and we used an average filter to reduce the dimension of feature elements. **Table 10** shows matching scores of each resouce for connected-component labeling appplication. Each value means an efficiency of computing resource for executing our connected-component labeling application. Resources with higher value are more efficient

than resources of lower values. We can see that M3 has highest value in case of CPU execution while M4 is the most efficient for GPU based execution. As a result, we can get the best performance by allocating resources in order of M3, M4, M1, and M2 in case of CPU execution. Allocating resources in order of M4, M3, M1 and M2 can make an optimal performance in case of GPU based application execution. As we can see in **Table 10**, semantic information system can offer resource allocation order for optimal application execution based on various information sources such as computing resource specifications, application algorithms and features.

Table 10. Matching scores of sample resources for the connected-component labeling application in case of CPU and GPU based execution

	M1	M2	M3	M4
CPU execution	119.380	62.228	224.406	156.578
GPU execution	97.786	24.220	346.357	527.800

4. Conclusion

In this paper, we have proposed semantic Grid middleware system for various Grid services based on semantic information process. Our system consists of semantic Grid management system and semantic information system. Each of our Grid middleware system has flexible and extensible components for various Grid services such as scheduling, monitoring and application executions. With the connected-component labeling application, we have shown the detailed semantic information process for resource allocation service by using resouece analysis and application feature analysis. Taking advantage of our semantic Grid middleware system based on data mining service embedded semantic information system and flexible Grid management system, application users can easily access to the various Grid services with an optimal application performance expectation.

References

[1] Hyung-Lae Kim, Tae-Nyon Kim and Chang-Sung Jeong, “Grid resource management system and semantic information system,” in *Proc. of IEEE 22nd International Conf. on Advanced Information and Networking Applications Workshops*, pp.1666-1671, 2008. Article (CrossRef Link)

[2] Tae-Nyon Kim, Hyung-Lae Kim, Ki-Ho Yi and Chang-Sung Jeong, “WebSIS: semantic information system based on web service and ontology for Grid computing environment,” in *Proc. of IEEE 7th International Conf. on Computer and Information Technology*, pp. 247-252, 2007. Article (CrossRef Link)

[3] J. Frey, S. Graham and C. Kesselman “Grid Service Specification,” *Open Grid Service Infrastructure WG, Global Grid Forum, Draft 2*, Jul. 2002.

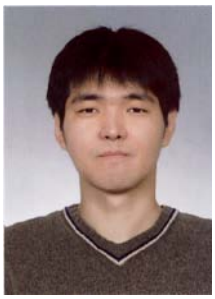
[4] G.T. Ian Foster, Carl Kesselman and S.Tuecke. “A Community authorization service for group collaboration,” in *Proc. of Fifth ACM Conf. on Computers and Communications Security*, Nov. 1998. Article (CrossRef Link)

K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W.Smith, and S. Tuecke. “A resource management architecture for metacomputing system,” *Lecture Notes in Computer Science*, no. 1459, pp. 62-82, 1998. Article (CrossRef Link)

[5] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. *OWL web ontology language reference. W3C*, Feb. 2004.

[6] RDF, Resource Description Framework, http://www.w3g.org/RDF

[7] CUDA (Computer Unified Device Architecture), http://nvidia.com/object/cuda_what_is.html



Hyung-Lae Kim received a B.S. degree in Electrical Engineering from Korea University, Seoul, South Korea, in 2004. He has experiences in many industrial works such as distributed rendering service, implementation of online education system, and customer management service about financial business. He is currently a Ph.D. candidate in the School of Visual Information Processing from Korea University, Seoul, South Korea. His research interests include Grid computing environment and middleware software.

Byong-John Han received a B.S degree in Electrical Engineering from Korea University, Seoul, South Korea, in 2007. He has involved in some projects such as Intellectual Unmanned Vehicle and Semantic Information System in Grid middleware system. He is in the doctoral course in Electrical Engineering from Korea University, Seoul, South Korea. His current research is Intellectual management in Grid middleware system.

In-Yong Jeong received a B.S. degree in Electrical Engineering from Korea University, Seoul, South Korea, in 2008. He is currently working toward the Ph.D. degree in Electrical, Electronic, Computer Engineering at the Korea University. His research interests include distributed/parallel computing and GPGPU computing

Chang-Sung Jeong is a professor at the Department of Electrical Engineering at Korea University. He received his M.S.(1985) and Ph.D(1987) from Northwestern University, and B.S.(1981) from Seoul National University. Before joining Korea University, he was a professor at POSTECH during 1982-1992. He was on editorial board for Journal of Parallel Algorithms and Application in 1992-2002. Also, he has been working as a chairman of Collaborative Supercomputing Group in GFK(Global Forum in Korea) since 2001, and a chairman of Computer Chapter at Seoul Section of IEEE region 10. His research interests include distributed concurrent computing, grid computing, and collaborative ubiquitous computing.