Vol.2010-MPS-79 No.9
2010/7/12

Regular Paper

A Light Framework for the Unified Representation and Execution of Variant Tasks in a Grid Based Environment

Mohamed Wahib,[†1] Asim Munawar,[†1]
Masaharu Munetomo[†2] and Kiyoshi Akama[†2]

Grid computing has gained a wide interest from the research community over the past one and a half decade. The immense effort has resulted in mature tools and technologies for grid computing. The utilization of experience and tools of grid computing in the next generation of distributed systems (e.g. cloud computing) is a logical step. However, many problems that come along with grid computing do limit such an effort. Among these problems is the sophistication of each production grid to a specific task type, size and dependency. In other words, grid computing in practice up to the moment can be described as task monolithic in terms of task scope. The approach to tackle this problem in this paper can be viewed as virtualizing the tasks to the middleware in analogy to how resources are virtualized, automatically provisioned and abstracted in current distributed systems trends. The task virtualization in this case refers to the ability of the system to host variant tasks through a generic transparent interface. This paper proposes a light-weight framework using various combined open-source grid tools to establish a grid-based system capable of executing tasks of different types, sizes and dependencies. From the middleware's perspective, designing and implementing such a model is challenged by 1) The need of a unified model for defining and representing the tasks, and 2) A tasks dispatching and execution manager. Experiments for using the framework in an optimization problem solving environment is illustrated. Performance results are presented showing the functional efficiency of the framework.

1. Introduction

Several reasons have contributed in grid computing not expanding to be adopted by the community outside specific scientific research projects. On the other hand, the relatively short legacy in grid computing witnessed a rapid evolution of tools and technologies with many reaching the mature phase. The fade away of grid computing, or limited use in specific projects at best, would be a waste of expertise and technologies if not to be capitalized on by next generation technologies. This paper attempts to capitalize on the existing technologies of grid computing to generate a light-weight framework capable of executing tasks in a distributed computing environment. The framework addresses one of the limitations in grid computing that halted the expansion of grid computing; namely the task monolithic nature of production grids in terms of scope. This limitation's negative effect is oftenly missed compared to the major limitations (i.e. The complication in resources management, security and ownership policies.) attracting all the attention from the designers of new paradigms tackling grid problems. We argue that the limitation discussed in this paper is one of the major problems of grid eventhough being missed occasionly. A quick overview of grid-based projects shows clearly that each project is tailored to a speficic triplet of task $\langle$*type, size, dependency*$\rangle$. Throughout the context of this paper a task triplet refers to a configuration of task type, task size and task dependency associated together (e.g. $\langle$*computation, HPC, independent*$\rangle$ triplet would mean a system hosting computational tasks of High Performance Computing scale where all the tasks are independent). Consequently, each individual grid project required a phase of tuning to the tasks triplet and no model of design-once-use-for-all exists. The main impact of this limitation lead to defining several types of grids to different triplets of tasks, hence being far from generality. To conclude, the simplex model of designing grids hosting specific task types, sizes and dependencies will no more be tolerated in comparison to the innovatve resource and task presentation methods emerging in cloud computing and other state-of-art paradigms. Concequently, to capitalize on grid legacy for designing next generation distributed computing paradigms, there is a need for an innovative model for different task triplets exposure through the same abstractions. The main motivation of this paper is to enable the hosting of different task triplets in grid computing and grid-related paradigms. The implemented framework allows the administrator to dynamically define task types, sizes and dependencies as well as defining task-specific QoS (Quality of Service) attributes to each task type/size. The task-specific QoS attributes are of great importance to later be used for task assignment. The framework proposed can be viewed as a layer residing between

†1 School of Information Science & Technology, Hokkaido University, Sapporo, Japan
†2 Information Initiative Center, Hokkaido University, Sapporo, Japan

the presentation layer and upper middleware layer. This layer can be described to be virtualizing the tasks to the middleware in analogy of how the resources are virtualized to the middleware layer and resource brokers in systems adopting virtualization. Moreover, a task manager that could manage the allocation of such diverse tasks is an essential part.
The rest of this paper is organized as follows; the next section is a brief review on task representation and definition in grid computing. Section 3 discusses the proposed framework. Section 4 discusses the experiments that were conducted using the proposed framework. Finally section 5 concludes and adds insight to future work.

2. Task representation in grids

To review task representation in grid computing, initially a closer look to types, sizes and dependencies of tasks in grid computing environments is essential. The next section will give a close-up to tasks in grid computing. Section 2.2 reviews task representation in grid computing.

2.1 Tasks in grid computing

As a part of a comprehensive grid systems classification,[10] divides the tasks or solutions in traditional and emergent grid systems into four types; computational solutions, data solutions, service solutions and access solutions. The types, along with the associated sizes, are as follows:
a) Computational solutions/tasks are tasks explicitly using CPU cycles. The CPU cycles could be from variant resources including idle desktop computers, servers or equipments and instruments. The size of tasks depends on the temporal factor. Small size computational tasks assume small tasks' sizes with the minimum overhead possible. HPC tasks are applications taking into consideration the performance and thus time is a non-negligible factor. HTC (High Throughput Computing) tasks are massive computational tasks that are motivated by solving previously untackled pre-grid era problems due to resources limitations, and therefore the time factor is of no priority in such tasks. A point to mention here is that HTC applications are usually viewed as a category different from high performance applications from the perspective of grid systems. This paper is basically concerned with high performance tasks and not high throughput tasks. Yet, high throughput tasks were included in the framework to better represent the sizes of computational tasks and offer a sensible level of generality for the framework.
b) Data solutions/tasks (hosted by data grids) are concerned with accessing, storing and moving data in distributed data repositories. There are many key elements upon which the size of the data task fluctuates.[18] introduces a comprehensive taxonomy for data grids, the authors identified the keys elements which define the data grids as organization, data transport, data replication and scheduling. The size used in the context of this paper for data tasks solely depending on the size of the data accessed, modified and transferred. Large volume data flows primarily deal with providing services and infrastructure for distributed data-intense applications dealing with massive datasets stored in distributed storage resources. Large volume data flows also involve massive datasets, but the main target is to stage the data to third party resources that conduct further computing. Thus, the size of handled data in this case is relatively less than sizes in pure data grids. Nevertheless, many projects falling under this category (e.g. computational physics projects) handle massive datasets in comparison to pure data grids. The last size is datasets of relatively small sizes staged to computational or services tasks. Usually in such a case, the grid is considered to be a computation/service task's grid involving data transfers between tasks.
c) Service solutions/tasks are either presentation of data or computation tasks as/through services as in[14], or services offering more sophisticated functionalities (e.g. knowledge discovery systems[21]). software tasks (i.e. tasks exploiting software resources) are considered as services tasks in the context of this paper. Service tasks cannot be expressed through the size metric due to its complex nature. It is more appropriate to describe the differences among service tasks as differences in *behavior* (in this paper behavior for services tasks is the equivalent to size in other task types). Generally, three behaviors can be defined for service tasks. The first includes tasks for services that are explicitly interfacing data and computation resources. The services in this case can be considered as a gateway or proxy to data and computational resources. Next are the tasks for services that are offering some business logic. Even though in the backend the business logic could be using computational and data resources, it is still different from

the previous behavior by having functionalities in the business logic layer that is loosely coupled from the backend dependencies. For the third behavior, the tasks invoke services that have business logic including calls to external (third party) service providers whom are not part of the grid environment.
d) Access solutions/tasks are for grids encompassing distributed input/output devices. The grid in this case is providing multiple access points to those resources. Access tasks' tools and middlewares are relatively not mature compared to the computational, data and services task grids. Also access tasks are not popular and limited only to specific projects. Therefore, access tasks are not addressed in this paper.
As for the dependencies, there are several patterns for the dependencies. The proposed framework is designed to host tasks with different levels of dependences. The framework supports tasks that are either a) individual (i.e. independent from any other task(s)), b) nested (i.e. a parent task invokes another task to be executed and they both run simultaneously) and c) workflows.
Figure 1 includes the most notable sizes, types and dependencies in real practice. Note that the limitation in this case (i.e. not having a comperhensive tasks categorization) is to simplify the analysis and focus the attention on the key point (i.e. design a framework for variant task triplets in grids).

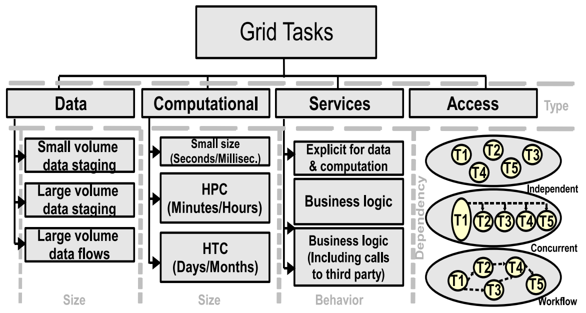

Fig. 1 Classification of grid tasks according to most notable types, sizes and dependencies.

2.2 Task representation in grid computing

This section presents an overview of the various task representation approaches in grid computing. After presenting the approaches, the deficiencies are summarized. Generally, two main approaches for task representation in grids exist, with different methods in each approach.
Approach 1: The most common approach for task representation in grid computing is defining grid types with one-to-one relation to a task triplet. For example data grids are grids designed to execute data tasks. Data grids are further divided into different categories based on the organization. The organization in this case defines the hierarchy which is directly related to the scale of the grid. In other words the task size (i.e. size of datasets in this case) defines how the data grid hierarchy and organization are defined[18]. Service tasks as well have different types of grids, each endorsing specific tools to function in a specific method to run the type of service tasks in hand. The same mechanism prevails for computational tasks, where each grid-based project running computational tasks is tuned to address the specific requirements of the tasks in hand. The point of interest here (i.e. task representation) is project dependant, but the common point is that the task representation takes a simple explicit form. This is expected due to having the grid architecture of each project tailored for handling a specific type of tasks in a predefined environment.
Approach 2: Other forms of task representation are adopted in grids that combine more than one task type. The most popular combination is between computational and data tasks. Workflows are used on a wide scale in grid-based projects to handle two types of tasks (namely data tasks and computational tasks). Workflows started out by adding data staging in-between the computational tasks, then it evolved to represent data tasks and computational tasks in the same model. Generally, workflow managers using DAG(Directed Acyclic Graph) including[11),20)] are more notable compared to the non-DAG managers. Another approach defines hierarchical models for grid services in[5)]. Both numerical and deterministic techniques such as Markov models, graph theory and queuing theory are used in this case to model the services tasks. One more approach to represent tasks in environments with different task types is to use file description. Many file description formats have been proposed and the Open Grid Forum unified the different formats in the Job Submission Description Language (JSDL)[17)]. Overall, the mentioned approaches feature all or some of the

following limitations:
a) Practically all mentioned solutions focus on the operations following task submission (i.e. scheduling, load balancing and resource allocation). Thus, no attention is given to defining different task sizes (e.g. computational tasks rarely have different sizes in the same grid). A drawback in this is case is the insufficient categorization of resources leading to complications in the scheduling process such as utilization imbalance and improper assignment.
b) In terms of Quality of Service (QoS), most of the approaches do not go far beyond defining basic QoS attributes (i.e. time, cost, fidelity, security and availability). This is due to having invariant task types and so the relation of tasks with resources is trivial and no non-functional characters could be defined for the resources (variant tasks leads to defining variant resource types which enables defining application-specific QoS attributes).
c) In many cases, no different parallelism patterns are defined. And so there is sensible limitation on the flexibility of creating tasks with different parallelism patterns.
d) Existing methods for combining variant task types (i.e. workflows) assume task inter-dependencies, thus cannot be utilized for tasks with no dependences.

3. How the framework works

The basic components in the framework are as follows:
Web portal is the entry point to the system, the portal used is gridsphere3.1[2]. New Custom JSR 268 compliant Vine1.0[1] based portlets were implemented and deployed in the portal. The new portlets are for a) task definition, b) QoS function editing and c) task submission. ***Grid services*** are Resource Framework (WSRF) compliant web services running in Globus GT4 Toolkit's[7] container. These services are the core of the framework that orchestrate all the components in the system.
Task manager is responsible for assigning the jobs to the appropriate resources on the Grid. The task assignment procedure will be explained in section 3.3.
Resources pool may include any of the resource types corresponding to the task types mentioned earlier in figure 1 (excluding access resources which are not considered in this paper).

Section 3.1 explains the scenario of what happens in the framework from the administrator's perspective while section 3.2 covers the scenario from the end user's perspective. Section 3.3 explains how task assignment happens in the framework.

3.1 Scenario from administrator's perspective

The administration in the proposed framework is slightly different from the conventional administration in grid systems (i.e. resource management and maintaining the system functionality). The following sections will discuss each of the scenario steps of the how the administrator interacts with the system in details.

3.1.1 Task definition

In the proposed framework, the first thing for the administrator to do is to define the tasks' types, sizes and dependencies for the grid in hand. To be able to do that, a custom portlet was designed (based upon the Vine portlets[1]) to run in the portal's portlet container. Through the *Task definition* portlet the administrator adds the task triplet that will run over the grid. The portlet is simple and includes only the basic information about the task (i.e. task type, task size and the expected dependencies). Such a method is apparently trivial and doesn't support deep description of tasks. Nevertheless, it is anticipated for this part to be extended and to support more task descriptive information that could further improve task categorization.

3.1.2 Resource addition

Normally for resources to be added in a gridsphere portal, the Resources.xml file containing information about the resources available in the grid should be edited. Among the special tags for defining the resources are tags for defining the three types of resources corresponding to the three task types mentioned earlier in the taxonomy. Namely ⟨*ws-gram-resource*⟩ for computational resources, ⟨*gridftp-resource*⟩ for data resources and ⟨*service-resource*⟩ for service resources. The part that Resources.xml tags is missing to be used by the framework is the task size and dependencies. Therefore, the Resources.xml schema was extended in the framework allowing the administrator to define new resources to be added to the grid, where those resources are defined to be used by which task triplet.

3.1.3 Defining and adding task-associated QoS attributes

One of the main advantages of the variant task triplets is the ability to define task-associated QoS attributes and include it in the scheduling process. QoS attributes are non-functional characters describing a process. QoS parameters are divided into two main groups, namely QoS metrics and QoS policies. QoS metrics are used to specify performance parameters including timeliness, precision, accuracy, security requirements and availability. QoS policies, on the other hand, capture application specific policies that govern how an application is treated by the resource manager. In this paper QoS metrics are referred to as basic QoS attributes, while QoS policies are referred to as task-associated QoS attributes. In comparison to task-specific QoS attributes, a huge legacy of scheduling with the basic QoS attributes exists. While appearntly less research effort is directed to task-associated QoS attributes. Adopting task-associated QoS attributes in the proposed framework makes advantage of hosting variant task triplets. This is because the variance in task triplets enables defining new criteria other than the basic ones to enhance the preference of which resources to choose in the task assignment process. For example, a grid application involving service tasks that retrieve images from a service resource could have the color depth and resolution as QoS attributes.

Next is to define a mechanism by which the administrator could: a) represent QoS attributes, b) include them in the scheduling process. Canfora et al.[3] propose service composition binding driven by application-specific QoS. The domain of[3] is different from the work in this paper. This is because the authors are concerned with the composition of services that are initially defined as an abstract workflow, where for each service in the workflow a set of service providers are defined. Then the scheduler/binder next makes a concrete workflow at which each service in the abstract workflow is binded to a service provider. Canfora et al. addressed how application specific QoS attributes can be defined and aggregated to be used for scheduling. A remarkable work was done in implementing a QoS aggregation function definition interface and a QoS definition language, so the administrator through a simple QoS aggregation function editor could define new QoS(s). The QoS definition in this framework here builds on and extends the work done by Canfora though it is defined for a different discipline (i.e. Service Oriented Architecture). It is of no-awkwardness to use a mechanism that was originally defined in SOA service workflows to be used in a grid-based framework as grid computing and SOA have many tangency points. This section will illustrate the QoS attributes representation, while the scheduling part is covered in section3.3. The steps to represent QoS attributes in the framework are as follows:

QoS definition language: The details of the language used is out of the scope of this paper, still a brief insight is given to ease comprehending of all framework parts. For a language to permit specifying new QoS attributes, two things are required; type and scale. The type can be only primitive types (integer, real and Boolean) as in WSLA+ language[16], or include collection types (i.e. a set constituted of sets of atomic values) as Canfora did. The scale limits the set of admissible operations. The language developed by Canfora includes the scales required for our framework, so no change is required in this part. The point of difference here is the set of operators and functions inherited form the Object Constraint Language (OCL)[19] that is used by Canfora. This is due to using those operators and functions in computing overall workflow QoS, while in the proposed framework accepts both inter-dependent task (i.e. workflows) and individual tasks. In the case of workflow tasks, the operators and functions defined by Canfora are sufficient, while in the case of individual tasks the operators are not used due to tasks independency. The next step is to show how the QoS formula specification is supported by a guided editor and type-checker.

QoS aggregation: The QoS aggregator introduced by[3] was implemented in Java using the Java Compiler (JavaCC) parser generator, while for the GUI, JSP was used. The aggregator was adopted here by including the aggregator in a Vine portlet and adding it to the gridsphere portal. Modifications were done to a) drop operators for individual tasks as mentioned earlier, b) associate the newly defined Qos attribute to one of the task type/size pair registered by the administrator and c) adapting the original JSP to work with the Adobe Flex GUI used in gridsphere 3.1. The aggregator includes three basic modules; *QoS aggregation function editor* a portlet that the administrator can use to define new QoS attributes and their aggregation formulae, *Type checker* used at design time for verifying the integrity of the aggregation formulae, *QoS formulae interpreter* that at run time evaluates QoS for a possible workflow/individual task assignment.

3.2 Scenario from enduser's perspective

In the proposed framework, the enduser is unaware of the procedure the administrator is following. The scenario from the user's perspective is as follows:
Task submission: Initially the user selects which task dependency he will use for the task. For the individual tasks, the user can use the *job submission portlet* to submit the task. Nested tasks in the proposed framework use the same portlet as well (i.e. job submissino portlet) for job submission. For the last task dependency type (i.e. workflows), a relatively large number of options and combinations are out there (e.g. Karajan, BPEL, Pegasus, Triana, DAGMan ... etc). For the case of this framework, a simple portlet which allows for workflow executions without complex needs, a simple DAG, was chosen. This was decided to focus the attention on building the framework without being drifted to complications of complex workflows. To our knowledge, no such simple java-based portlet for simple DAG workflows exists. The options are either complex and advanced (e.g. P-Grade portal which is gridsphere based), or needs bridging to be imported in a portlet (e.g. Taverna). The option choosen was to implement a simple custom Vine portlet that uses Java CoG's TaskGraph to create the workflow. The designed portlet does not use nice drag and drop of jobs on a canvas like Triana or P-Grade workflow editor but uses menus to choose from available jobs, their ports and links between them. Additionally, TaskGraph was only used to create the workflow and not to schedule the workflow. The scheduling is carried out by the scheduler described in the next section. One final point to mention about job submission, the user is required to select the of triplet the task he is submitting according to the task triplets supported by the grid as defined earlier by the administrator. A weak point here is the dependency on the user to define the task triplet. As the user might not know which combination to choose. Or even mistakenly choose a wrong task triplet leading to faulty task assignment. This problem rises from the nouvelle approach of offering different task triplets in the same grid leading to switched task submission. Solutions derived from the literature could be designed to solve this problem (e.g. the use of Petri Net queue to estimate the requirements of the tasks as in[6]). Nevertheless, as mentioned earlier the main focus in this paper is to design a framework for many task grids, and so this specific problem is out of the paper scope and user awareness of task size/type is assumed. This specific problem is a potentially challenging point to be addressed separately in future work.
Confirm submission: Finally, the user confirms the acceptance of the grid to the task, and is notified later with the task result via e-mail, later the user could view results log through the job portlet.

3.3 Task associated QoS-based scheduling

3.3.1 Related work

As a starter, a quick overview of the most relevant related work is essential as several projects in the literature attempted multiple QoS job scheduling. All of these projects addressed only the basic QoS attributes in their work; cost, time, availability and fidelity. Those basic QoS attributes are agreed upon in literature to be the metrics for job scheduling in grids. Application specific QoS attributes were not considered in any of them as no service level presentation of resources provided features into which application-specific QoS attributes could be mapped. The few efforts in this area can be summerized as follows.[12],[13] defined the 4 basic QoS attributes as utilities and identified an Integrated Utility Function (IUF) to be used for scheduling. A slight difference between them is that[12] used an iterative scheduling technique by separating the task agents and resources agents.[6] proposed a static scheduling algorithm that uses utility functions for scheduling meta-task with QoS requirements in heterogeneous computing systems. The main aim of this project was to provide resource transparency and not include application-specific QoS attributes.[9] did not define a scheduling method, yet it defined, for the basic QoS attributes, a flexible multidimensional QoS performance measure that could be later aggregated to be used for scheduling. A notable point is that the authors defined a method that is theoretically applicable to model any new QoS attribute by representing the QoS attribute by a feature vector. This approach is generic but if used will shift the entire load to the application developer.

3.3.2 Problem formulation

Suppose that there are n independent users and user i is associated with task T_i, let $T = \left\{T_1, T_2, ..., T_n\right\}$ denote the set of n independent tasks where task T_i is assigned to resource R_j and $R = \left\{R1, R_2, ..., R_m\right\}$. T_i is not the least task unit, as T_i could be a single task, double concurrent task (i.e. in case of nested tasks) or recursively divided to $T = \left\{T_1, T_2, ..., T_l\right\}$ in case of T_i being a workflow. Similarly R_j can be composed of v resources, where $1 \leq v \leq m$. The process of QoS-constrained task scheduling could be summarized in three steps. Step one is resource discovery, which is finding available resources complied by QoS constraints and generate a list of resources. Step two is resource planning, which involves selecting the optimized resources from the available resources list according to the scheduling strategy satisfying user's QoS constrains. The $\Gamma : T \rightarrow R$ denoting the matching function is a NP complete problem4). The third step is task execution. The tasks are scheduled to selected resources to be executed.

In the proposed framework, QoS attributes are described as utility functions. So the integrated utility function is the accumulation of all QoS attributes utility functions. The integrated utility is considered as the objective function of the scheduling algorithm to drive the scheduling of resources and optimizing the task execution with maximum utility. A very important point here is that utility functions have long been used in QoS constrained scheduling, but for this case a new factor requires special handling. The new factor is having different QoS attributes defined according to each task type/size. Thus, the integrated utility function includes a set of mutually exclusive utility functions corresponding to the QoS attributes. The next step, after defining the integrated utility function, is to compute the value of the utility u_k, $1 \leq k \leq d$ where d is the total number of QoS constraints defined by the administrator for all the task types/sizes, what we will call the *dimension of QoS*. The method used for computing the utility functions is the same used by Canfora3) which iteratively update the selection probabilities of the selected resources. For each task T_i a *decision matrix* $Q = (q_{ij})_{m \times d}$ is created, among it, m represents the number of resources that can host T_i, d represents the dimension of QoS attributes considered by the this type of task. Q matrix is not used directly, Q is normalized to make the *normalized matrix* $P = (p_{ij})_{m \times d}$, $j = 1, ..., m, k = 1, ..., d$, where normalizing is done as follows:

$$p_{jk} = \begin{cases} min_j\left(q_{jk}\right)/q_{jk} & \text{if } u_k \text{ is optimally minimized} \\ \\ q_{jk}/max_j\left(q_{jk}\right) & \text{if } u_k \text{ is optimally maximized} \end{cases}$$

Supposing $\omega = (\omega_1, \omega_2, ..., \omega_d)$ is attribute weighting vector, Then the integrated utility function for the evaluation of the selected resources can be defined as:

$$U_{ij}\left(\omega\right) = \sum_{t=1}^{S}\left(C_t \sum_{l=x}^{y} p_{jl}\omega_l\right)$$

where S is the number of task types/sizes defined by the administrator to be used in the system, C_t is a Boolean constant having value 1 only for the QoS attributes subset of the task scheduled (i.e. QoS divided to subsets having a mutually exclusive relation.). x and y denote the start and end of the QoS attributes subset from the set of all QoS attributes defined. By sorting and computing $U_i\left(\omega\right)$, the best resource can be selected. Weights for utility functions in the mentioned related work are calculated by maximizing the deviations in utiliy values. In our case the QoS attributes are defined by the administrator according to the types/sizes of tasks. Therefore, the default weights are defined by the administrator according to the system used. The end user could calibrate the weights according to his interest in which QoS attributes significant for the job he is about to submit.

3.3.3 Scheduling Algorithm

The scheduling process is basically viewed as a combinatorial optimization problem for the integrated utility function. Several approaches are used in the literature to solve this optimization problem, most approaches depend on a heuristic algorithm. Generally, GA (Genetic Algorithms)8) based heuristic algorithms do not impose constraints on the linearity of the QoS composition operators, so they are considered the best option. But because the framework is using a nouvelle technique of having more than one task triplet, the modeling of the problem by a fitness function will be impossible due to having more than one task triplet in the same chromosome with mutually exclusive QoS attributes.

As shown in the previous section, QoS attributes are described as utility functions, The integrated utility function aggregating all utility functions is regarded as an objective function of the scheduling algorithm to drive dynamic scheduling to the resources and optimizing the task execution with maximum utility by accumulating all QoS attributes utility functions. Figure2 shows the algorithm used for scheduling. One important point to note here, the algorithm is designed

Task type	QoS attributes	Individual		Nested		Workflow		
Opimization algorithms as services (Service tasks)	* *Solver to problem adequacy* * *Parallel pattern* * *Fidelity*	Small	50	Small	50	Random size, 20 tasks ↓	🟪	🟩
		HPC	20	HPC	20		🟪	🟩
		HTC	10	HTC	10		🟪	🟩
Optimization algorithms as computational tasks	* *Latency tolerance* * *Data accuracy*	Small	50	Small	50	🟫	🟪	↑ Random size, 20 tasks ↓
		HPC	20	HPC	20	🟫	🟪	
		HTC	10	HTC	10	🟫	🟪	
Data tasks	* *Priority*	Small size staging	50	N/A	N/A	🟫	↑ Random size, 20 tasks	🟩
		Large size staging	20	N/A	N/A	🟫		🟩
		Large size flow	10	N/A	N/A	🟫		🟩

Table 1 Testbed configuration

in a central way, which leads to weak scalability. Yet, the design of centralized scheduling algorithms is an important step towards developing more complex decentralized scheduling algorithms.

```
1- For all $T_i$ if $T_i$ is a workflow or nested tasks, then $T_i = (T_{i1}, T_{i2},\ldots, T_{im})$,
     $m \geq 2$. For each  $T_{i2}$  go to step 1.
2- Use $T_i$ category to create available resources list that can host $T_i$.
3- For all $k \in m$, evaluate QoS attributes of all services in service list.
4- Create decision matrix $Q_i$.
5- Compute the integrate utility function of multiple QoS attributes $U_i(\omega)$.
6- Sort resources according to $U_i(\omega)$.
7- Execute $T_i$ on the selected resource.
```

Fig. 2 Algorithm used for scheduling

4. Experiments and results

Experiments for configuring a grid and then submitting variant tasks with different triplets were done to test the functionality of the proposed model. The experiments were executed on a grid having a dedicated 64 core mini-cluster with 2 x AMD Opteron 2.6 GHz Dual Core 64 bit processors and 2GB RAM for each node, the grid also has 2 dedicated servers each having a 2 x Xeon 2.8 GHz Dual Core with 2GB RAM. The environment on which the framework was tested is Meta Heuristics Grid (MHGrid): a service oriented grid application offering meta heuristics based solvers for global optimization problems. Full details about MHGrid's design and functionalities in[15]. MHGrid was originally designed to offer optimizing algorithms as services. To test the task unified presentation frame-

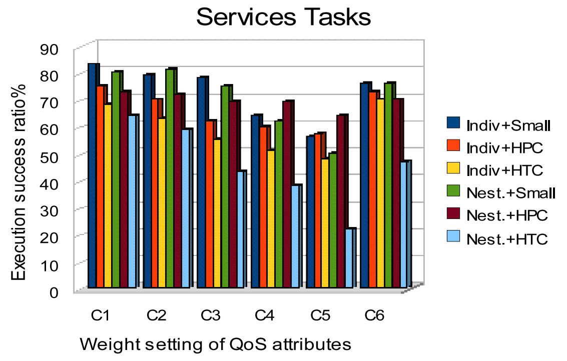

Fig. 3 The effect of services tasks weight vector on execution success ratio

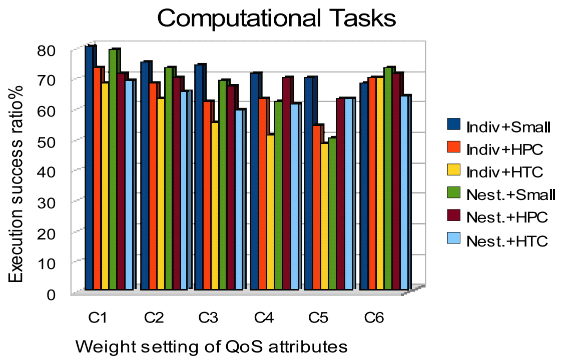

Fig. 4 The effect of computational tasks weight vector on execution success ratio

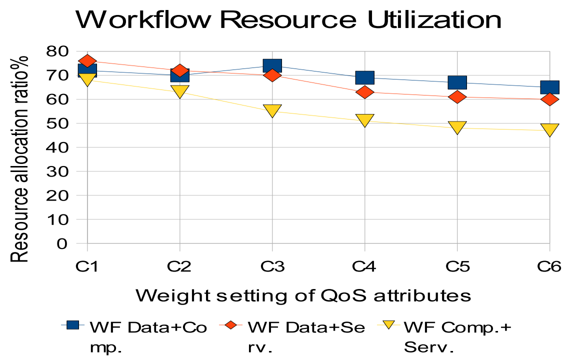

Fig. 6 Resource utilization for workflows with variant tasks weight values

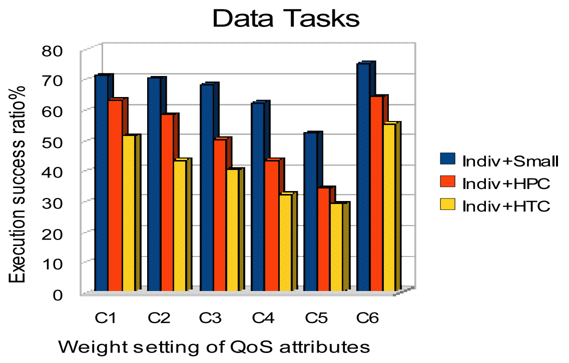

Fig. 5 The effect of data tasks weight vector on execution success ratio

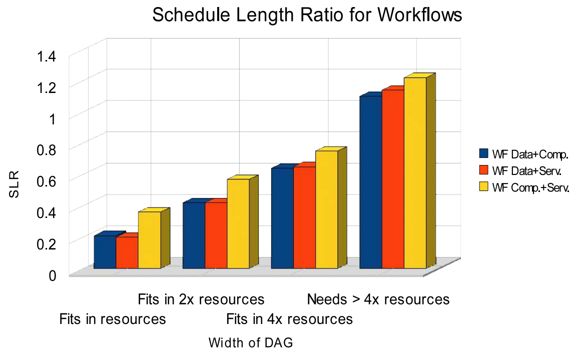

Fig. 7 SLR for the workflows with different DAG width

work, other types of tasks with different sizes and dependencies were introduced. Table 1 shows the task types and sizes used in the experiments along with the number of tasks submitted for each task type/size pair. MHGrid originally exposes the algorithms through services to control the parallel pattern used[15]. To test the proposed framework the following changes in MHGrid were done:
a) Solvers were directly accessible to have computational tasks in the systems along with services tasks.
b) MHGrid originally did not have data tasks, so dummy files of variant sizes were deployed to be staged.
c) Tasks in MHGrid are either individual or nested. So a portlet for simple DAG workflows was implemented and deployed to be used in the experiments.
The experiments were designed to cover possible task triplets. For the service tasks, individual and nested tasks of different sizes were introduced. The size is controlled by the problem length that the algorithms should solve. Three spe-

cific QoS attributes were defined; Solver to problem adequacy, the parallel pattern used and the fidelity or quality of output. These attributes are explained in details in[15]. The same applies for computational tasks but with two different QoS attributes. Long timeouts were added in the solvers to test the latency tolerance with the solvers running in parallel mode. The other QoS attribute is data accuracy which is measured by round off errors resulting from float-point operations intentionally introduced to the solvers. As for the data tasks, dummy files where used to be transferred via GridFTP[7]. The file sizes varied from 100MB to 20TB to represent different task sizes. One QoS attribute (i.e. priority) was defined for the data tasks where each task was assigned priority rank for the scheduler to attempt to meet the priority requirement of those tasks. In addition to that, workflow tasks for different task types where also created by the workflow portlet to be tested. Finally two basic QoS attributes (make span time and cost) were used for all tasks along with the specific QoS attributes defined for each task type.

Execution success ratio is the metric used in the experiments to measure the framework's ability to host and schedule variant task triplets. Other metrics such as resource utilization and task waiting time are important as well, but due to space limitation, results in terms of execution success ratio only are illustrated to evidate that a system hosting variant task triplets can function whilst benefiting from QoS attributes. Figures 3, 4 and 5 show the execution success ratio for the individual and nested tasks on different task types/sizes. The x-axis in all three figures is a configuration state for the QoS attributes considered. The last state (i.e. C6) is the plain state where all weights for task-specific QoS attributes are set to zero and only the basic time and cost QoS attributes with weight = 0.5 are considered. For the other states (i.e. C1 to C5) the weights are having values with an increasing mean from C1 to C5 and also an increasing standard deviation to represent variant weight vector settings. Note that upon moving from C1 to C5 the success rate tends to decrease which is normal as the more the mean of the weight vector increases the more the scheduling process relay on QoS other than the time and cost. This consequently minimizes the set of resource candidates for each task and causes less success rate. Another point to note is that for figure5, C6 gives much better performance than other weight settings. This is because the *priority* QoS defined for data tasks highly effects the scheduling process making it mostly depending on what the user wants. Figure6 shows the resource utilization for workflow tasks. Again the change in weight value does not have a significant effect on the resource utilization and this is because the workflows enforce an order for task execution on the system. Figure7 shows another aspect of the workflows. The figure shows the Schedule Length Ratio (SLR) for the workflows that were created with different DAG widths.

5. Conclusion

This paper is concerned with grid computing based systems at which tasks are defined to be having variant types, sizes and dependencies. First a close-up was given to the nature of tasks in grid computing and a rough classification was introduced. A framework enabling grids to host variant task triplets was proposed. The proposed framework involves representing the tasks, defining QoS attributes for the tasks and scheduling tasks in such an environment. The framework was inspired by a system of application-specific QoS attributes in service composition. While the domain is apparently different, the flexibility offered by the service composition in defining QoS attributes is of great relevance to the requirements of establishing an enviroment that is not limited to hosting specific task triplets. The framework attempts to conceive virtualization of tasks in analogy to virtualization of resources from the middleware perspective. The framework first requires initial definition of task types, sizes and dependencies hosted by the grid. The administrator creates the task-associated QoS attributes through a QoS aggregation function editor enclosed in a portlet. After the administrator registers the categorized resources, the end user can start submitting tasks to the system. With the user unaware, the system schedules the tasks on behalf of the user, and assures the use of task-specific QoS attributes to assign the tasks to the most appropriate resources. Experiments were conducted to assure that the framework is functioning as designed. Many points are promising to be considered as future work. A module for task size/type assurance is of great value to eliminate endusers faulty categorization. Making the best use of the task-associated QoS by incorporating a SLA mechanism is also an interesting point. Investigation of heuristic-based schedulers to develop a heuristic algorithm allowing many task scheduling is as well a challenging task.

References

1) : http://vinetoolkit.org/.
2) : http://www.gridsphere.org.
3) Canfora, G., Penta, M.D., Esposito, R., Perfetto, F. and Villani, M.L.: Service Composition (re)Binding Driven by Application-Specific QoS, *ICSOC*, pp.141–152 (2006).
4) Christensen, T.V.: Heuristic Algorithms for NP-Complete Problems (2007).
5) Dai, Y.-S., Pan, Y. and Zou, X.: A Hierarchical Modeling and Analysis for Grid Service Reliability, *IEEE Transactions on Computers*, Vol. 56, No. 5, pp. 681–691 (2007).
6) Doğan, A. and özgüner, F.: Scheduling of a meta-task with QoS requirements in heterogeneous computing systems, *J. Parallel Distrib. Comput.*, Vol.66, No.2, pp. 181–196 (2006).
7) Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, pp. 2–13 (2006).
8) Jong, K. A.D.: Are Genetic Algorithms Function Optimizers?, *PPSN*, pp. 3–14 (1992).
9) Kim, J.-K., Hensgen, D.A., Kidd, T., Siegel, H.J., John, D.S., Irvine, C., Levin, T., Porter, N.W., Prasanna, V.K. and Freund, R.F.: A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems, *Cluster Computing*, Vol.9, No.3, pp.281–296 (2006).
10) Kurdi, H., Li, M. and Al-Raweshidy, H.: A Classification of Emerging and Traditional Grid Systems, *IEEE Distributed Systems Online*, Vol.9, No.3 (2008).
11) Lee, K., Paton, N.W., Sakellariou, R., Deelman, E., Fernandes, A.A. and Mehta, G.: Adaptive Workflow Processing and Execution in Pegasus, *gpc-workshops*, Vol.0, pp.99–106 (2008).
12) Li, C. and Li, L.: Utility-based QoS optimisation strategy for multi-criteria scheduling on the grid, *J. Parallel Distrib. Comput.*, Vol. 67, No. 2, pp. 142–153 (2007).
13) Li, Y., Zhao, D. and Li, J.: Scheduling Algorithm Based on Integrated Utility of Multiple QoS Attributes on Service Grid, *GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing*, Washington, DC, USA, IEEE Computer Society, pp.288–295 (2007).
14) Middleton, S.E., Surridge, M., Benkner, S. and Engelbrecht, G.: Quality of Service Negotiation for Commercial Medical Grid Services., *Journal of Grid Computing*, Vol.5, No.4, pp.429–447 (2007).
15) Munawar, A., Wahib, M., Munetomo, M. and Akama, K.: *Linkage in Evolutionary Computation*, chapterParallel GEAs with Linkage Analysis over Grid, pp.159–187, Springer Berlin / Heidelberg (2008).
16) Nepal, S., Zic, J. and Chen, S.: WSLA+: Web Service Level Agreement Language for Collaborations, *scc*, Vol.2, pp.485–488 (2008).
17) Reynaud, S. and Hernandez, F.: A XML-based Description Language and Execution Environment for Orchestrating Grid Jobs, *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, IEEE Computer Society, pp.192–199 (2005).
18) Venugopal, S., Buyya, R. and Ramamohanarao, K.: A taxonomy of data grids for distributed data sharing, management, and processing, *ACM Computing Surveys*, Vol.38, p.2006 (2007).
19) Warmer, J. and Kleppe, A.: *The Object Constraint Language: Getting Your Models Ready for MDA*, Addison-Wesley (2003).
20) Wieczorek, M., Prodan, R. and Fahringer, T.: Scheduling of scientific workflows in the ASKALON grid environment, *SIGMOD Rec.*, Vol.34, No.3, pp.56–62 (2005).
21) Zhuge, H. and Liu, J.: A fuzzy collaborative assessment approach for knowledge grid, *Future Gener. Comput. Syst.*, Vol.20, No.1, pp.101–111 (2004).