A FRAMEWORK FOR SLA-AWARE EXECUTION OF GRID-BASED WORKFLOWS

DISSERTATION

Schriftliche Arbeit zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften an der Fakultaet fuer Elektrotechnik, Informatik und Mathematik der Universitaet Paderborn.

Dang Minh Quan

November 2006

© Copyright by Dang Minh Quan 2007
All Rights Reserved

Preface

Service Level Agreements (SLAs) are currently one of the major research topics in Grid Computing, as they serve as a foundation for reliable and predictable Grids. SLAs define an explicit statement of expectations and obligations in a business relationship between provider and customer. Thus, SLAs should guarantee the desired and a-priori negotiated Quality of Service (QoS), which is a mandatory prerequisite for the Next Generation Grids. This development is proved by a manifold research work about SLAs and architectures for implementing SLAs in Grid environments. However, this work is mostly related to SLAs for standard, monolithic Grid jobs and neglects the dependencies between different steps of operation.

The complexity of an SLA-specification for workflows grows significantly, as characteristics of correlated sub-jobs, the data transfer phases, the deadline constraints and possible failures have to be considered. Thus, an architect for an SLA-aware workflow implementation needs sophisticated mechanisms for specification and management, sub-job mapping, data transfer optimization and fault reaction.

Therefore, this dissertation presents a system for SLA-aware Grid workflows. The main contributions include an improved specification language for SLA-aware workflows, three mapping and optimization algorithms for sub-job assignment to Grid resources, an error recovery mechanism, and a prototype implementation using standard middleware. Experimental measurements prove the quality of the development.

Acknowledgements

This dissertation cannot be finished without the help of a number of people:

First of all, I would like to especially thank Professor Odej Kao, who is my great supervisor. He first brought me to Germany and gave me the chance to do research in a modern academic environment. I would like to acknowledge his valuable supports, comments and suggestions during my research process. My knowledge has been enriched by his judgment and wisdom. I never forget all these helps.

I deeply thank my parent. Their support and understanding have helped me to overcome the most difficult period, encouraging me to forget the bad things and giving me a new hope when I am down. They are always supporting and believing in whatever I do. Without their love and supports, I would never be able to finish this dissertation.

I am grateful for the helps and co-operations of many researchers in the working group of Professor Odej Kao, who have provided valuable opinions for my research topic. Many thanks are to the secretary of Professor Odej Kao, namely Irene Roger, for her kind helps and assistance.

Last but not least, I would like to thank system administrators in Computer Science department, University of Paderborn for their help in establishing experiments system.

Paderborn, 2th January 2006

Dang Minh Quan

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Grid computing is viewed as the next phase of distributed computing. Built on Internet standards, grid computing enables organizations to share computing and information resources across departments and organizational boundaries in a secure and highly efficient manner. According to Ian Foster, a Grid is a system that "coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of services" [38]. In general, from the view point of the application layer, a Grid can exist in one of three types: Computational Grid, Scavenging Grid and Data Grid [63]. A computational Grid is a Grid that focuses on setting aside resources particularly for enhancing computing power. A scavenging Grid usually includes large numbers of desktop machines which are scavenged for available CPU cycles and other resources. A data Grid is responsible for housing and providing access to data across multiple organizations.

There are a lot of Grid users who have high demand of computing power to solve large scale problems such as material structure simulation, weather forecasting, fluid dynamic simulation, etc. Beside vast number of single-program applications, which has only one sequential or parallel program, there exist many applications, which require the co-process of many programs following a strict processing order. Since those applications are executed on the Grid, they are called Grid-based workflows. The concept of workflow first arose in business environments, e.g. [34], and has drawn

a lot of attention in the database and information system research and development communities [35, 46, 60]. Although business applications play a significant role in research and development, Grid-based workflows are also important as computing expands into the routine activities of scientists. A Grid-based workflow is distinguished from a conventional single application in two main characteristics. Firstly, it includes many sub-jobs, each of which can be a sequential or a parallel program; secondly, data dependency exists among sub-jobs.

Traditionally, to run the application, users submit it to a Grid system and the system will try to execute it as early as possible [36, 80, 5, 78, 30, 19, 109]. That best-effort mechanism is not suitable when users need the running result at a specific time, and hence require that the application must be run at a specific period. This requirement must be agreed on by both users and the Grid system before the application is executed. This task can be done by a Service Level Agreement (SLA) [113]. The purpose of the SLA is to identify the shared goals and objectives of the concerned parties. A good SLA is important as it sets boundaries and expectations for the following aspects of a service provisioning. An SLA clearly defines what the user wants and what the provider promises to supply, which helps to reduce the chances of disappointing the customer. The provider's promises also help the system stay focused on customer requirements and assure that the internal processes move in the right direction. An SLA describes a clear, measurable standard of performance. Based on this description, internal objectives become clear and measurable. An SLA defines penalties. This criterion makes the customer understand that the service provider truly believes in its ability to achieve the set of performance levels. It makes the relationship clear and positive. In this context, an SLA sets the expectations between the consumer and the provider, and defines the relationship between the two parties.

Only computational Grid is suitable to run sub-jobs of the workflow within SLA context as it brings many important advantages for this task.

- The computational Grid connects many High Performance Computing Centers (HPCCs) all over the world. Only these centers can handle the high computing demand of scientific applications.

- The cluster or super computer in a HPCC is relatively stable and well maintained. This is an important feature to ensure finishing the sub-job within a specific period of time.

- The HPCCs usually connect to the worldwide network by high speed links, whose broad bandwidth makes the data transfer among sub-jobs easier and faster.

Scavenging Grid and data Grid cannot fully satisfy all of the above features. Scavenging Grid includes many desktop computers distributed over the internet. Those computers are connected by heterogeneous network infrastructure and thus, cannot ensure to provide a good performance to the parallel application especially the one having great communication among tasks. Data Grid concentrates to manage and store data over the network. Its primary function is not executing high performance applications. Therefore, they are not considered in this work. In computational Grid, each resource joining to the system supporting SLA for workflow is a HPCC, which usually has a set of computing nodes, a mass storage and a number of experts for technical support. The resources in each HPCC are managed by software called local Resource Management System (RMS). In this dissertation, the acronym "RMS" is used to represent the cluster/super computer as well as the Grid services provided by the HPCC.

Supporting SLAs for the Grid-based workflow that mainly aims at finishing the workflow execution on the Grid within a pre-determined period of time faces several problems.

The first problem is the lack of effective mapping mechanism to map each sub-job of the workflow to resources in a manner that can satisfy two main criteria: being able to finish workflow execution on time and being able to optimize the running cost. The first criterion is quite clear because it is the main reason for an SLA system to exist. The latter criterion is derived from the business aspect of an SLA. If a customer wants to use a service, he must pay for the service usage and has the right to receive it with an appropriate quality. An automated mapping is necessary as it frees users from the tedious job of assigning sub-jobs to resources under many constraints such

as workflow integrity, on time condition, optimal condition, etc. Additionally, a good mapping mechanism will help users to save money and to increase the efficiency of using Grid resources.

Secondly, to ensure the SLA for workflow requires the co-operation of many components in the Grid. The co-operating procedure to reach the agreement about providing the service among those components does not exist in the literature. Without such a procedure, users must negotiate SLA for each sub-job in the workflow. This action also creates a significant problem for the user's patience, especially when the number of sub-jobs in the workflow increases to large values.

Another problem is related to errors which may occur during the execution of the workflow. Randomly appearing errors may damage the workflow completion as well as the negotiated SLA. Thus, this demands building an error recovery mechanism for a workflow in order to eliminate the affection of error to users and to make the Grid system more stable and reliable.

Finally, a system which can execute a real Grid-based workflow within the SLA context needs to be built. The system must integrate all of the above features in order to help users to minimize the works. It should do the task by itself.

Supporting SLAs for a workflow in the Grid environment is a new problem and still in the initial phase of the exploiting process. The work in this dissertation will fill the gap by building a skeleton architecture which can be the basis for deploying the real service. Within the scope of this dissertation, I concentrate on the selected core problems of a system, which supports SLA for a workflow in the Grid environment, as stated above. Other important issues such as security, charging/accounting, risk management and so on are not considered in this thesis. The main contributions of this dissertation are:

- A new problem statement which supports executing a workflow on reserved Grid resources within the scope of a business contract.

- A mapping mechanism, which includes several sub optimization algorithms, to map sub-jobs of the workflow to the Grid resources within SLA context to satisfy the specific user's runtime requirement and to optimize the cost.

At present, the size of the Grid is still small. For example, the Distributed European Infrastructure for Supercomputing Applications (DEISA) includes only 11 sites. Based on that fact, a distributed mapping model for very large size Grid is not an urgent requirement at the present and thus, it is not focused in this work. The proposed central mapping mechanism is the heart of the system. The goal of this main work is to provide fast response solution while ensuring the QoS for customers. Thus, it reduces the overhead of the workflow execution time and encourages users to utilize the services.

- An SLA negotiation protocol for workflows. This protocol is the core procedure to co-operate many components of the Grid to ensure the SLA for the workflow.
- An error recovery mechanism for workflows within an SLA context. The mechanism takes into consideration the catastrophic failure when one or several Grid resources are detached from the Grid system. Other mechanisms to increase the reliability of the system such as risk assessment, slack-based scheduling, security of the data connection are not included in this dissertation.
- A prototype system to realize all the proposed theories.

All the research results presented in this dissertation were also published in [97, 98, 99, 100, 101, 102].

Chapter 2

Background and review of the state of the art

2.1 Introduction to Grid-based workflow

Workflows received enormous attention in the databases and information systems research and development community [35], [46], [60]. According to the definition from the Workflow Management Coalition (WfMC) [125], a workflow is *"The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules."* Although business workflows have great influence on research and development, another class of workflows emerges naturally in sophisticated scientific problem-solving environments called Grid-based workflow [82, 11, 105]. A Grid-based workflow differs slightly from the WfMC definition as it concentrates on intensive computation and data analyzing but not the business process. A Grid-based workflow is characterized by following features [112, 129].

- A Grid-based workflow usually includes many applications which perform data analysis tasks. However, those applications, which are also called sub-jobs, are not executed freely but in a strict sequence.
- A sub-job in the Grid-based workflow depends tightly on the output data from

previous sub-job. With incorrect input data, the sub-job will produce wrong result and damage the result of the whole workflow.

- Sub-jobs in the Grid-based workflow are usually computationally intensive tasks, which can be sequential or parallel programs and require long runtime.
- Grid-based workflows usually require powerful computing facilities such as super computers or cluster to run on.

It can be seen that the Grid-based workflow and the business workflow have the same primary characteristic as they both have a procedure that applies a specific computation into selected data according to certain rules. Each Grid-based workflow is defined by three main factors.

- **Tasks**. A task in the Grid-based workflow is a sub-job which is a specific program doing a specific function. Within a Grid-based workflow, a sub-job can be a sequential program or a parallel program. The sub-job usually has long running period and needs powerful computing resources. Each sub-job requires specific resources for the running process such as operating system (OS), amount of storage, CPU, memory, etc.
- **Control aspect**. The control aspect describes the structure and the sequence in processing of sub-jobs in the workflow.
- **Information aspect**. The information aspect of the Grid-based workflow is presented by data transmissions. The dependency among sub-jobs can also be identified by the data transmission task. A sub-job is executed to produce a number of output data, which are the input data for the next sub-job in the sequence. These data must be transferred to the place where the next sub-job is executed. Within a Grid-based workflow, the quantity of data to be transferred between two sub-jobs varies from several KB to a hundred GB depending on the type of application and its scope.

Figure 2.1 depicts a sample scenario by considering weather forecasting as an example Grid-based workflow. The main requirement is that a three hours forecast

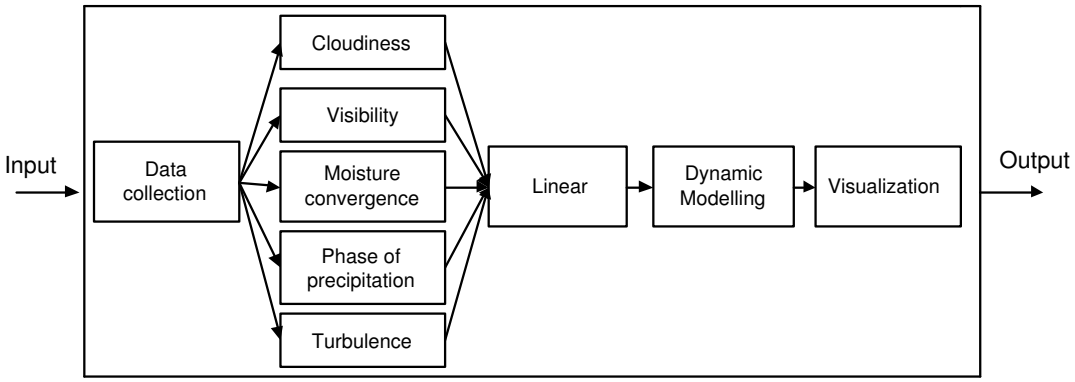

Figure 2.1: Weather forecast processing workflow

should be available within 20 minutes after all necessary data have been collected. In the workflow, the data collection module collects the input data from various sources such as radars, satellites, lightning detectors, etc. Then the data are processed by several modules to get information about cloudiness, visibility, moisture convergence, phase of precipitation, etc. Thereafter, the results are processed by the linear module to interpolate between the field analysis and the forecast of the numerical weather prediction models. This data is used in dynamic modeling to build high-resolution models with special physical parameterization schemes for a precise prediction of weather events. Finally, the weather information is visualized by the dedicated module.

Because the data and computing resource may spread in a physically distributed environment, running the workflow needs a mechanism to handle the data transfer and to invoke the computational tools over a distributed and heterogeneous platform. This mechanism is exactly the goal that Grid computing technology attempts to achieve in scientific environments.

2.2 Introduction to Grid computing

Built on Internet infrastructure, Grid computing enables organizations to share computing and information resources across departmental and organizational boundaries in a secure, highly efficient manner. Organizations around the world are utilizing Grid

computing today in such diverse areas as collaborative scientific research, drug discovery, financial risk analysis, and product design. Grid computing enables research-oriented organizations to solve problems that were not feasible to solve due to computing and data-integration constraints. Grids also reduce costs by means of automation and improving IT resource utilization. Finally, Grid computing can increase an organization's agility enabling more efficient business processes and greater responsiveness to change. Over time grid computing will enable a more flexible, efficient and utility-like global computing infrastructure.

2.2.1 Grid computing definition

An exact and complete definition of Grid is still under discussion. Informally, Ian Foster proposed three criteria which a Grid system should satisfy [38]. A Grid is a system that:

- *"coordinates resources that are not subject to centralized control - (A Grid integrates and coordinates resources and users that live within different control domains for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.)"*

- *"using standard, open, general-purpose protocols and interfaces - (A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As I discuss further below, it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application-specific system.)"*

- *"to deliver nontrivial qualities of service - (A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands,*

so that the utility of the combined system is significantly greater than that of the sum of its parts.)"

2.2.2 Grid architecture description

To co-operate multiple distributed resources, it is necessary to have a common protocol that all components in the grid must follow for a uniform working mechanism. A protocol definition specifies how distributed system elements interact with one another in order to achieve a specified behavior, and the structure of the information exchanged during this interaction. Standard protocols have emerged as important and essential means of achieving the inter-operability that Grid systems depend on. In [39], Foster et al present a Grid taxonomy which has a great influence on the research and development of Grid systems. According to [39], the modern Grid system is built on two main concepts: Virtual Organization (VO) and service.

VO is a set of individuals and/or institutions sharing resources under clearly and carefully defined rules about what is shared, who is allowed to share, and the conditions under which a share occurs. In reality, VOs vary tremendously in the size, types of resource sharing, number of participants, sharing rules, etc. VOs enable many individuals, groups, and institutions to share resources in a controlled fashion so that members can co-operate to achieve a common goal.

In Computer Science, a service is a software serving a specific purpose. More specifically, as stated in [39], *"A service is a network-enabled entity that provides a specific capability, for example, the ability to move files, create processes, or verify access rights. A service is defined in terms of the protocol one uses to interact with it and the behavior expected in response to various protocol message exchanges (i.e., "service = protocol + behavior.")."*

An overall system architecture, which identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another, is depicted in Figure 2.2. Figure 2.2 also presents the correlative functions between layers of the Grid protocol and the internet protocol.

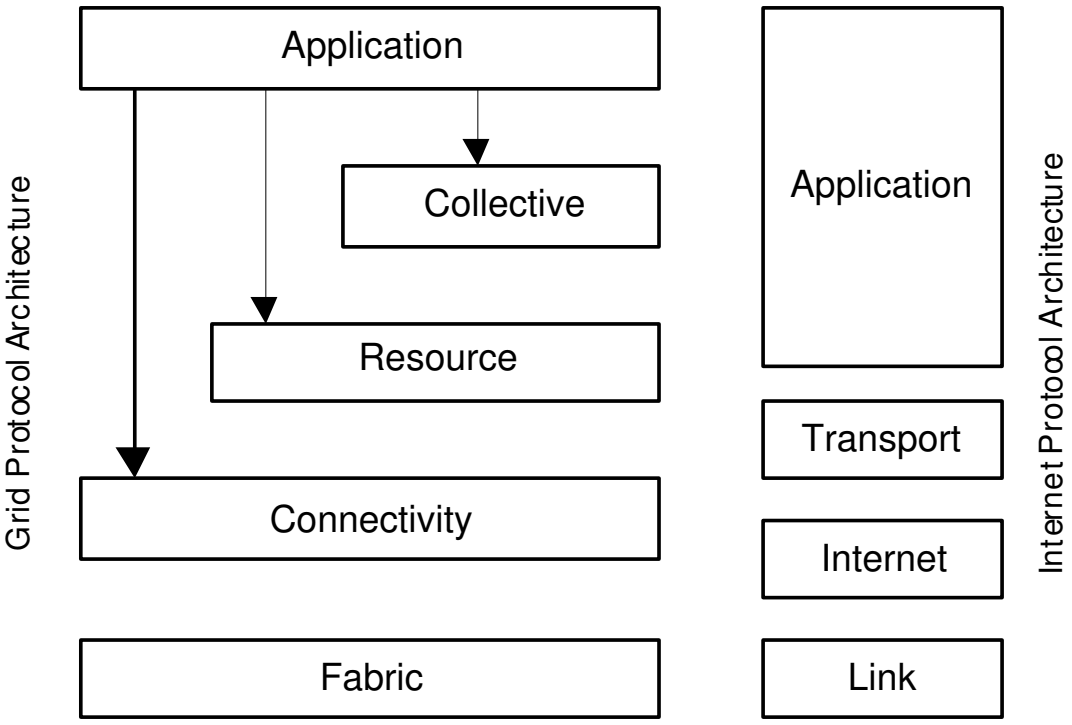

Figure 2.2: The layered Grid architecture and its relationship to the Internet protocol architecture [39]

Fabric layer

The fabric layer contains physical resources that people want to share and access. Those resources can be computational, storage or network resources, code repositories, databases, etc. The physical resources have all characters of Grid hardware, great in number, heterogeneous in configuration, distributed in location, and various in usage policy.

In each site, the physical resources are managed by software, which handles the management task locally. The management software does scheduling, makes allocation, controls the sate of resources, and provides access service for local users. Sample of those managements software includes PBS [93], CCS [58], etc.

The layer also has some software components to provide functions supporting higher layer features such as remote access operation, sharing operation, etc.

Connectivity layer

The connectivity layer is characterized by two main features: the core communication and authentication protocols required for Grid-specific network transactions.

The communication protocol defines the way in which data are transferred between different resources in the Fabric layer and based on the existing internet data communication technology, the TCP/IP protocol.

The authentication protocol provides secure mechanisms for verifying the identity of users and resources. For a distributed resource environment like Grid, which provides virtual organization interface to user, the authentication protocol should have the following features.

- **Single sign on**. Users log on to system once and can access many resources in the Fabric layer without entering an additional username or password.
- **Delegation**. Users have the ability to assign their power to a program so that it can also access the resources that they have the right to access. Beside that, these programs can also assign the right to other programs.
- **Integration with various local security solutions**. The Grid security solution must be able to co-operate many different security policies from different organizations in the Grid.
- **User-based trust relationships**. Users should have the ability to use resources from many sites together without the need of interaction among those sites.

Resource layer

The Resource layer is an intermediate layer coping with the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. This layer does the task by calling functions provided by the Fabric layer. Resource layer protocol can be divided into two classes.

Information protocols are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy.

Management protocols are used to negotiate access to a shared resource. Negotiating information includes resource requirement, quality of service and the operation to be performed.

Collective layer

While the Resource layer is focused on interactions with a single resource, the Collective layer works on global scale and captures interactions across collections of resources. The scope of function in this layer is wide and flexible such as directory services, co-allocation services, data replication services, etc.

Applications layer

At the top of any Grid system are user applications which are constructed in terms of, and call on, the components in any other layer. User applications range from material simulation to image processing, etc.

The theory structure as described above was implemented following Open Grid Services Architecture (OGSA) [40] and realized by Globus Toolkit [48].

2.3 Introduction to Service Level Agreement

The main purpose of an Information Technology organization is to provide a computing service which satisfies the customers' business requirements. To achieve this goal, the organization needs to understand those requirements and to evaluate its own capability of providing the service and measures the service delivered. To enable the realization of the process, the service and level of delivery required must be identified and agreed between the organization and its users. It is usually done by Service Level Agreements (SLAs), which are contracts developed jointly by the organization and the customers.

2.3.1 Definition

A Service Level Agreement (SLA) identifies the agreed upon services that will be provided to a customer in order to ensure that they meet the customer's requirement. The SLA identifies customers' expectations and defines the boundaries of the service, stating agreed-upon service level goals, operating practices, and reporting policies. Webopedia defines the SLA as *"Abbreviated SLA, a contract between an ASP (Application Service Provider) and the end user which stipulates and commits the ASP to a required level of service. An SLA should contain a specified level of service, support options, enforcement or penalty provisions for services not provided, a guaranteed level of system performance as relates to downtime or uptime, a specified level of customer support and what software or hardware will be provided and for what fee."* [123]

2.3.2 Structure

A common SLA contains following components:

- Parties joining the agreement. The agreement is made between the service provider and the service user. Two participants should exist as individuals, either by name or by title. Both sides must sign the document.
- Type and the time window of the service to be provided. The SLA must state clearly which service will be provided and the time window during which the service is provided to the user. In fact, there are a lot of system components contributing to the type definition of the service. They can be the number of processors, processor speed, amount of memory, communication library, etc.
- The guaranty of the provider to provide the appropriate service and performance. The SLA must state clearly how well the service will be provided to the user as Quality of Service. Penalties must also be figured out if a certain QoS cannot be satisfied.
- The cost of the service. Business users wishing to use any service have to pay for the usage. The cost depends on the quantity of service usage and how long

the user uses it.

- The measurement method and reporting mechanism. The SLA defines which parameter will be measured and how it will be measured. Data collected from the monitoring procedure are important as they help user and provider check the validity of the SLA.

2.3.3 Applying the Service Level Agreement

Applying the SLA requires the co-operation from both service provider and customers. The provider is responsible for preparing the infrastructure and deploying the service. The provider should train customers on SLA so they can understand and are willing to use the new form of service. The following will describe some remarkable notes for applying the SLA.

- The provider should deploy a high visibility, a well-understood application and target only the things that can be measured.

- The provider must recognize the relationship between the architecture and what the maximum levels of availability are. Thus, an SLA cannot be created in a vacuum. An SLA must be defined with the infrastructure in mind. If not, the provider cannot meet the user's requirement and violates the SLA.

- A relationship exists between the levels of availability and the related cost. Some customers need higher levels of availability and are willing to pay more. Therefore, having different cost policies with different level of service quality is a common approach.

- The provider commits to reporting measurements clearly and accurately, and to take actions to avoid service degradation.

- The customer should forecast workload as accurate as possible. This is important for both customer and provider. Accurately estimating workload helps the provider reserve appropriate amount of resource for the task. If less resource

than required is reserved, the work cannot be done. If more resource than required is reserved, the resource is wasted and user must pay more.

- The customer must represent accurately the service requirement.
- The customer should describe details on how and who receives the reports.

2.4 Review of the state of the art

The work in this dissertation is building a system supporting QoS for the Grid-based workflow. Therefore, in this section, we will discuss other related projects appeared in the literature. Nearly current with the work in this dissertation, there are some other efforts related to supporting QoS for workflow [86, 130, 8]. They are distinguished from each other in the architecture, the workflow description language and the mapping algorithm.

2.4.1 Architecture

Although having different architectures, all of them build services based on the infrastructures which support resources reservation.

Imperial College e-Science Network Infrastructure (ICENI) is an end-to-end Grid middleware system developed at the London e-Science Centre [86]. It supports users of the Grid, and adds value to their Grid experience, by enabling a complete workflow pipeline in a transparent manner. The main feature of ICENI is running a workflow over reserved resources to decrease the variance of execution. The overall architecture of ICENI is presented in Figure 2.3.

Abstract workflows enter the scheduler and are translated into concrete workflows detailing the resources to be used. The concrete workflow is then distributed to the relevant resources through the launching services. The scheduler, which is used to select the resources, and the software implementations to execute, may interact with multiple Launchers where each Launcher is used as a mechanism to deploy work onto one or more resources. Each Launcher may be associated with one Reservation

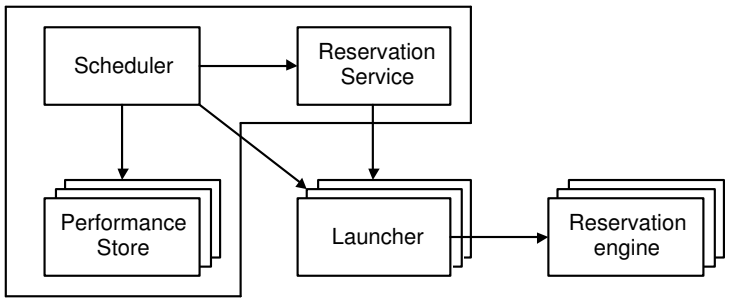

Figure 2.3: The architecture of ICENI system [86]

Engine, if reservation is possible. The Reservation Engine provides an abstraction of the underlying DRM's reservation system. The Reservation Service provides the ability to co-reserve multiple resources so that all the resources involved in a workflow can be reserved at the appropriate time. The Scheduler interacts with the Reservation service, which in turn communicates with the Reservation Engines through the Launchers. There may be multiple Performance Stores, each of which can be interrogated by the Scheduler. Once a workflow is instantiated, it will have an Application Service, which exists until the workflow terminates.

AgFlow is a middleware platform that enables the quality-driven composition of Web services [130]. In AgFlow, the QoS of Web services is evaluated by means of an extendable multi-dimensional QoS model, and the selection of component services is performed in such a way as to optimize the composite service's QoS given a set of user requirements (i.e., constraints on QoS) and a set of candidate component services. Furthermore, AgFlow adapts to change that occurs during the execution of a composite service, by revising the execution plan in order to conform to the user's constraints on QoS. The overall architecture of AgFlow is presented in Figure 2.4.

There are three distinct components in the AgFlow system, namely, Web services, service broker, and service composition manager. The service broker allows providers to register their service descriptions in an UDDI registry. A service description contains meta-data that describe, among others, the capabilities and QoS of a Web service. The service composition manager is made up of an execution planner and an execution engine. When an instance of a composite service is initiated, the

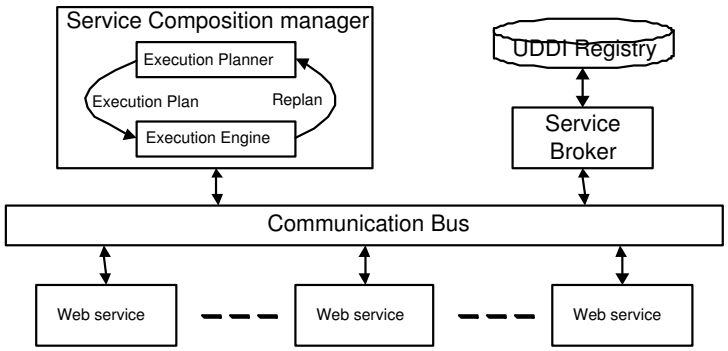

Figure 2.4: The architecture of AgFlow system [130]

execution planner contacts the service broker to search for the candidate component services, and based on the candidate services retrieved, it generates an execution plan, i.e., an assignment of component services to the tasks in the schema of the composite service. Based on the execution plan, the adaptive execution engine then orchestrates the component services to execute the instance of the composite service. At runtime, the execution engine also monitors the component services. When the current status of the execution violates the execution plan, the execution engine triggers the execution planner to revise the current plan and resumes the new plan to orchestrate the execution.

QoS-aware Grid Workflow is a project, which aims at extending the basic QoS support developed within VGE [8] and GEMSS [9, 44] to Grid workflow applications. In order to enable QoS-aware service composition and workflow execution, the project extends VGE environment with a QoS-aware Grid Workflow Language (QoWL) and a QoS-aware Grid Workflow Engine (QWE) [16]. The overall architecture is presented in Figure 2.5.

The XML parser and un-parser generates the intermediary representation of the QoWL workflow. The QoS negotiator queries the registries, generates necessary QoS requests and receives offers from services. The Workflow planning component calculates a workflow execution plan. The Service Deployer exposes a QoWL workflow as a Web service and the Workflow Executer starts the execution of the QoWL workflow. In case of dynamic planning strategy the Workflow Planner and Workflow Executer

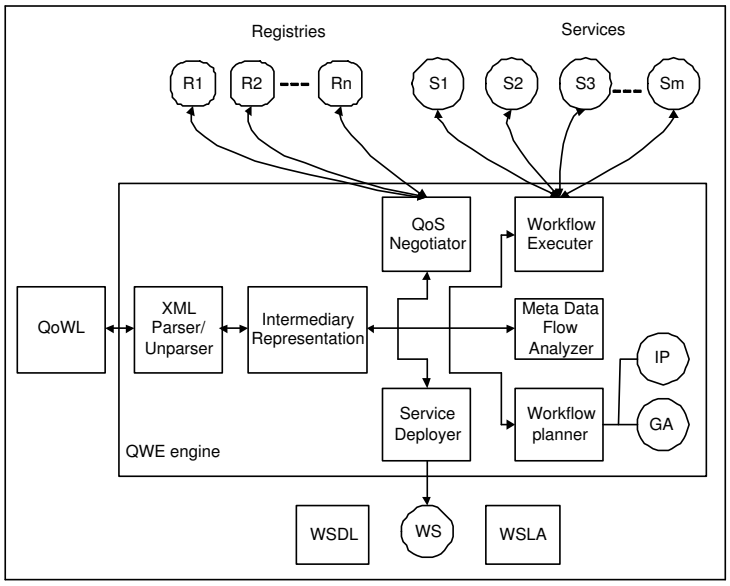

Figure 2.5: The architecture of QoS-aware Grid Workflow system [16]

are invoked in an alternating way.

2.4.2 Workflow language

ICENI uses an XML based language to describe the workflows that are submitted for execution. The workflow describes a collection of components and the links between them. These workflows are called Execution Plans (EP). When EPs are submitted to the ICENI environment they are abstract in nature. The detailed language uses the workflow definitions from the NeSC e-Science Workflow Services Workshop [33]. The communication between Scheduler and Launcher uses Job Description Markup Language (JDML) [85] as the means of transport.

AgFlow uses statecharts to represent these dependencies.

In QoS-aware Grid Workflow project, for the specification of QoS constraints for Grid workflow applications, a QoS-aware Grid Workflow Language (QoWL) is used. The language utilizes a subset of the Business Process Execution Language (BPEL) [15] with extensions for expressing QoS constraints. QoS extensions are necessary to express both the requested QoS constraints of a workflow before the QoS negotiation

and the offered QoS of a workflow after the negotiation with the services since the offered QoS may differ from the requested one. Local constraints usually address QoS constraints of single tasks invoking external services. Global constraints address the QoS of the overall workflow or of composite activities. QoS constraints of a workflow and of the underlying tasks are expressed in terms of $qos - constraints$ elements, which may contain several $qos - constraint$ elements and several $candidate - registry$ elements. Each $qos - constraint$ element defines a name/value pair and a weight of a QoS constraint.

2.4.3 Mapping mechanism

The ICENI project supposes that each task of the workflow can be performed by one resource and each resource of the Grid can perform one task at a time. The scheduling algorithm is used to determine the resources to be used to perform the task and the implementation of the software to be used on these resources. A number of algorithms such as simulated annealing, complete information game theory and best of N random [128] are used to do mapping with time optimization. The scheduling algorithms produce several potential concrete workflows. To determine which of these will be run, the Scheduler will send them to the Reservation Service, which will return one of the workflow for which it was able to obtain reservation.

AgFlow also supposes one task can be performed by one web service and one web service can handle one task at a time. AgFlow uses a global planning approach to map tasks to web services. The scheduling method is based on Integer Programming (IP) [67] for selecting an optimal execution plan without generating all possible execution plans. There are three inputs in an IP problem: a set of variables, an objective function, and a set of constraints where both the objective function and the constraints must be linear. IP attempts to maximize or minimize the value of the objective function by adjusting the values of the variables while enforcing the constraints. The output of an IP problem is the maximum (or minimum) value of the objective function and the values of variables at this maximum (minimum). To apply IP, AgFlow defined several constraints for the problem.

- Allocation constraint: For each task t_j , there is a set of Web services S_j that can be assigned (allocated) to it. However, for each task t_j, one Web service should be selected to execute this task.

- Constraints on Execution Duration, Price, and Reputation: The execution duration of a given task t_j must be the execution duration of one of the Web services. If task t_k is a direct successor of task t_j , then the execution of t_k must start after task t_j has been completed. The execution of a composite service plan is completed only when all the tasks in the plan are completed. The execution price of the composite service should not be greater than user's expectation. The reputation of all selected services must equal to a pre-determined value.

- Constraints on Success Rate and Availability: The success rate and the availability of the execution plan must equal to a pre-determined value.

- The expected execution duration is an optimization criteria.

When using the global planning approach, an execution plan is built at the beginning of the execution of the composite service. Once the execution has started, several contingencies may occur, e.g., a component service becomes unavailable or the QoS of one of the component services changes significantly. In these situations, a re-planning procedure may be triggered in order to ensure that the QoS of the composite service execution remains optimal.

The mapping module in QoS-aware Grid Workflow project includes two main methods: static scheduling and dynamic scheduling. The former employs the Integer Programming method from the work of AgFlow. The latter is similar to the working mechanism of Condor DAG Man [79] except based on resource reserved infrastructure.

Chapter 3

Problem statement

Most of existing Grid-based workflows [82, 11, 105] can be presented under Directed Acyclic Graph (DAG) form so that only the DAG workflow is considered in this work. Figure 3.1 presents a sample of a workflow as a material for presentation.

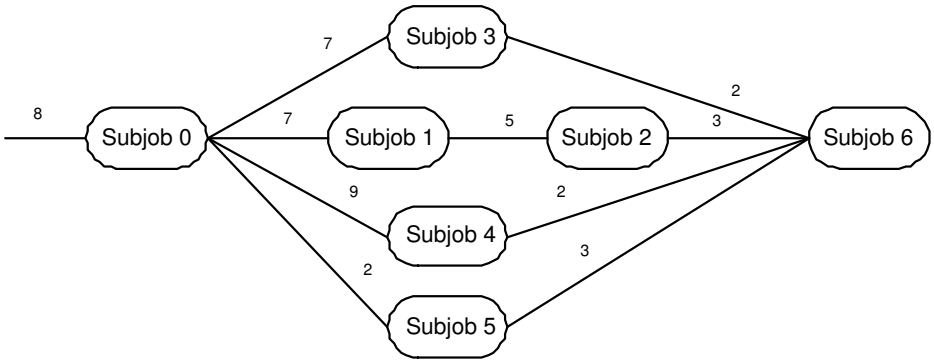

Figure 3.1: A sample Grid-based workflow

Traditionally, users let the Grid system run the whole workflow in best effort manner [36, 80, 5, 78, 30, 19, 109, 20]. The Grid system tries to find suitable Grid resources to run sub-jobs of the workflow as soon as possible. In this way, the runtime of the workflow varies depending on the state of Grid resources. When there are a lot of free Grid resources at runtime, the finish time of the workflow can be very short, and vice versa. With some users, this model works fine but with some other users, this is not good enough. For example, with the case of the weather forecasting

workflow [105], users want to receive the final result within 20 minutes from the start of execution. With those users, finishing the running of the workflow within a specific period of time is a mandatory requirement. Thus, they want to have an agreement with the Grid system to ensure that the system will provide the service meeting the requirement. This task can be defined with a Service Level Agreement (SLA).

To perform the SLA, a user must give his requirements and the system will check if it can satisfy them. Among many requirement parameters, a system supporting SLA requires the estimation of the sub-job's runtime to do resource reservation. Thus, the sub-job can be run on dedicated resources within a reserved time frame to ensure the QoS (this is the runtime period). The runtime period of a sub-job can be estimated from statistical data. The user usually runs a sub-job many times with different resource configurations and different amount of input data before integrating it to the workflow. The data from those running is a dependable source for the estimation.

Before the work in this dissertation, there have only been some works aimed at supporting SLA for single Grid jobs [18, 68]. The literature has recorded some recently proposed solutions to support QoS for workflow such as [86, 130, 16]. Most of the proposed mechanisms suppose that a workflow includes many sub-jobs, which are sequential programs, and a Grid service having the ability to handle one sub-job at a time. This is not sufficient as sub-jobs in many existing workflows [82, 11, 105] are parallel programs and many HPCCs provide computing services under a single Grid service [18]. It is obvious that an HPCC can handle many sub-jobs at a time. With different workload models and resource models, the system supporting SLA for Grid-based workflow must solve the following problems.

3.1 Optimal cost

The system must have an efficient mapping mechanism to map sub-jobs of the workflow to the Grid resources. The mapping should find a solution that meet the user's requirements and is as inexpensive as possible. This is a great challenge because of various parameters of workflow and Grid resources. The workflow could have greatly

different sub-job's resource requirements. The Grid resources also have various configurations and have different free resources over time. The following parts will describe this problem.

3.1.1 Workflow description

To describe the workflow information, users describe specifications about required resources, data transfer among sub-jobs, and the estimated runtime of sub-jobs as well as the expected runtime of the whole workflow. In more detail, we will look at a concrete example, a simple Grid workflow presented in Figure 3.1. The time is computed in slot. Each slot equals to a specific period of real time, usually from 2 to 5 minutes. The main requirement of this workflow is described as follows.

- Each sub-job having different resource requirements about hardware and software configurations. Important parameters such as the number of CPU, the size of storage, the number of experts, and the estimated runtime for each sub-job in the workflow are described in table 3.1.
- The number above each edge describes the number of data to be transferred between sub-jobs.

Sj_ID	CPU	Storage	exp	runtime
0	51	59	1	21
1	62	130	3	45
2	78	142	4	13
3	128	113	4	34
4	125	174	2	21
5	104	97	3	42
6	45	118	1	55

Table 3.1: Sub-jobs' resource requirements of the workflow in Figure 3.1

In the resource requirements of a sub-job, there are two types of resources: adjustable and nonadjustable. The nonadjustable resources are type of RMS, OS, and communication library. If a sub-job requires a supercomputer it cannot run on a

cluster. If a sub-job requires Linux OS it cannot run on Windows OS. Other types of resources are adjustable. For example, a sub-job which requires a system with CPU 1Ghz can run on the system with CPU 2 Ghz; a sub-job requiring a system with 2GB RAM can run on the system with 4GB RAM. In the common case, all sub-jobs in a workflow have the same nonadjustable resources and different adjustable resources.

The distinguishing character of the workflow description within SLA context lies in the time factor. Each sub-job must have its estimated runtime correlative with specific resource configuration to run on. If these parameters exceed the pre-determined limitation, the SLA will be violated. Within the SLA context, the resources are reserved over time. If a sub-job runs out of an estimated time period, it will occupy the resource of other reserved sub-job. This is not allowable in an SLA system.

3.1.2 Character of RMSs

In the Grid environment, there are a lot of RMSs. Each RMS has its own resource configuration and this configuration is usually different from one RMS to another.

To ensure that the sub-job can be executed within a dedicated time period, the required CPU, the storage, the expert in the RMS must be ready at that time. This mechanism can be done only if the RMS supports advance resource reservation, for example CCS [58]. Figure 3.2 depicts a sample CPU reservation profile in such a RMS. Queuing-based RMSs are not suitable for our requirement, as no information about the starting time is provided. In our system, we reserve three main types of resource: CPUs, storages and experts.

For present purposes, suppose that we have three involved RMSs executing the sub-jobs of the workflow, reservation information of the resources is presented in Table 3.2. Each RMS represented by an ID_hpc value has different number of free CPU, memory and expert during a specific period of time. The sample resource reservation profiles of the RMSs are empty.

If two sequential sub-jobs are executed in the same RMS, it is not necessary to do data transfer, and the time used for this work equals to 0. Otherwise, the data transfer between them must be performed. As this is a common task in the Grid,

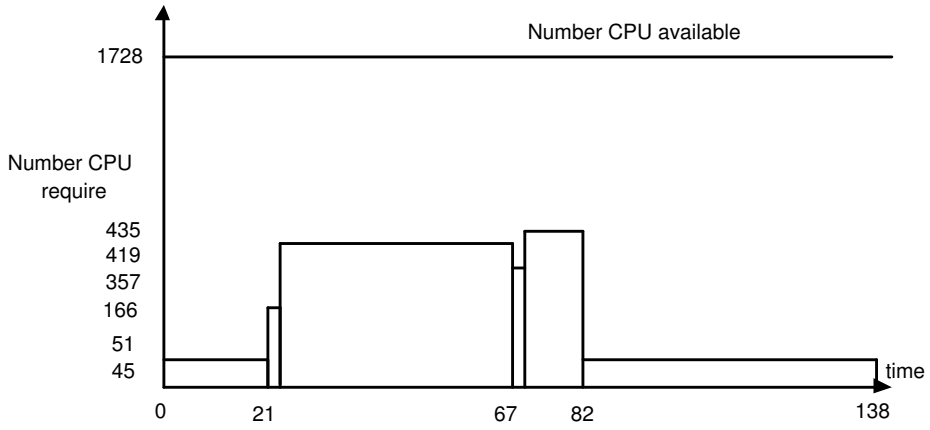

Figure 3.2: A sample CPU reservation profile of a RMS

ID	ID_hpc	CPUs	mem	exp	start	end
31	2	128	256000	8	0	1000000
23	0	128	256000	9	0	1000000
30	1	128	256000	6	0	1000000

Table 3.2: RMSs resource reservation

the bandwidth of the link between two local RMSs also plays an important factor in contributing to ensure SLA for the workflow. To make sure that a specific amount of data will be transferred within a specific period of time, the bandwidth must also be reserved. Unfortunately, up to now there has been no mechanism responsible for that task in the worldwide network. Here, to overcome this elimination, we use a central broker mechanism. The link bandwidth between two local RMSs is determined as the average bandwidth between two sites in the network which has a different value with each different couple of RMSs. Whenever having a data transfer task on a link, the SLA broker will determine which time slot is available for the task. During the specified period, the task can use the whole bandwidth while other tasks must wait. Using this principle, the bandwidth reservation profile of a link will look similar to the one depicted in Figure 3.3. A more correct model with bandwidth estimation [121, 122] can be used to determine the bandwidth within a specific time period instead of the average value. In both cases, the main mechanism is unchanged.

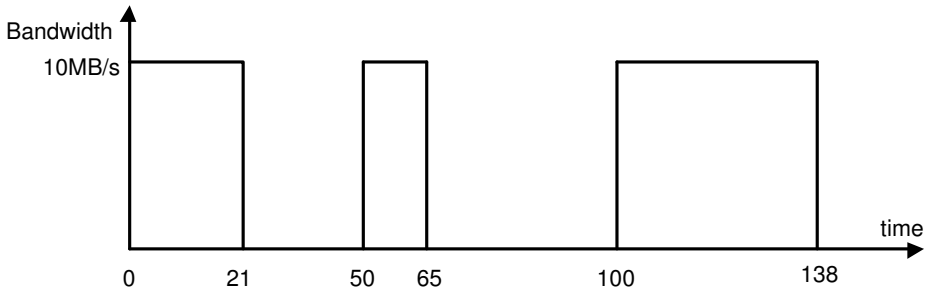

Figure 3.3: A sample bandwidth reservation profile of a link between two RMSs

3.1.3 Formal requirement statement

The formal specification of the described problem includes following elements:

- Let R be the set of Grid RMSs. This set includes a finite number of RMSs, which provide static information about controlled resources and the current reservations/assignments.
- Let S be the set of sub-jobs in a given workflow including all sub-jobs with the current resource and deadline requirements.
- Let E be the set of edges in the workflow, which express the dependency between the sub-jobs and the necessity for data transfers between the sub-jobs.
- Let K_i be the set of resource candidates of sub-job s_i. This set includes all RMSs, which can run sub-job s_i, $K_i \subset R$.

Based on the given input, a feasible and possibly optimal solution is sought, which allows the most efficient mapping of the workflow in a Grid environment with respect to the given global deadline. The required solution is a set defined in Formula 3.1.

$$M = \{(s_i, r_j, start\_slot) | s_i \in S, r_j \in K_i\} \tag{3.1}$$

If the solution does not have start_slot for each s_i, it becomes a configuration as defined in Formula 3.2.

$$a = \{(s_i, r_j | s_i \in S, r_j \in K_i\} \tag{3.2}$$

A feasible solution must satisfy following conditions:

- **Criteria 1:** The finished time of the workflow must be smaller or equal with the expected deadline of the user.

- **Criteria 2:** All $K_i \neq \emptyset$. There is at least one RMS in the candidate set of each sub-job.

- **Criteria 3:** The dependencies of the sub-jobs are resolved and the execution order remains unchanged.

- **Criteria 4:** The capacity of an RMS must equal or greater than the requirement at any time slot. Each RMS provides a profile of currently available resources and can run many sub-jobs of a single flow both sequentially and in parallel. Those sub-jobs, which run on the same RMS, form a profile of resource requirement. With each RMS r_j running sub-jobs of the Grid workflow, with each time slot in the profile of available resources and profile of resource requirements, the number of available resources must be larger than the resource requirement.

- **Criteria 5:** The data transmission task e_{ki} from sub-job s_k to sub-job s_i must be taken place in a dedication time slots on the link between RMS running sub-job s_k to RMS running sub-job s_i. $e_{ki} \in E$.

In the next phase the feasible solution with the lowest cost is sought. The cost C of running a Grid workflow is defined in Formula 3.3. It is the sum of four factors: money for using CPU, money for using storage, cost of using experts knowledge and finally money for transferring data between the involved resources.

$$C = \sum_{i=1}^{n} s_i.r_t * (s_i.n_c * r_j.p_c + s_i.n_s * r_j.p_s + s_i.n_e * r_j.p_e) + \sum e_{ki}.n_d * r_j.p_d \tag{3.3}$$

with $s_i.r_t, s_i.n_c, s_i.n_s, s_i.n_e$ being the runtime, number CPU, number storage, number expert of sub-job s_i respectively. $r_j.p_c, r_j.p_s, r_j.p_e, r_j.p_d$ are the price of using CPU, storage, expert, data transmission of RMS r_j respectively. $e_{ki}.n_d$ is the number of data to be transferred from sub-job s_k to sub-job s_i.

If two sequential sub-jobs run on the same RMS, the cost of transferring data from the previous sub-job to the later sub-job is neglected.

The ability to find a good solution depends mainly on the resource state at the expected period when the workflow runs. During that period, if the number of free resources in the profile is large, there are lots of feasible solutions and we can choose the cheapest one. But if the number of free resources in the profile is small, simply finding out a feasible solution is difficult. Thus, a good mapping mechanism should find out a cheap solution when there are a lot of free resources and find out a feasible solution when there are few free resources in the Grid.

Supposing the Grid system has m RMSs, which can satisfy the requirement of n sub-jobs in a workflow. As a RMS can run several sub-jobs at a time, finding out the optimal solution needs (m^n) loops. It can be shown easily that the optimal mapping of the workflow to Grid RMS as described above is an NP hard problem [12].

From the above description, we can see that this is a scheduling problem. However the problem has many distinguished characteristics.

- An RMS can handle many sub-jobs of the workflow at a time. The RMS supports resource reservation.
- A sub-job is a parallel application.
- The destination of the problem is optimizing the cost. User imposes some strict requirements on the Grid system and pays for the appropriately received service. It is obvious that the user prefers a good service with the cost as low as possible. The cost of running a workflow includes the cost of using computation resources and the cost of transferring data among sub-jobs.

As no previous work has similar context, a new strategy must be developed to handle the new requirements. An efficient mapping mechanism will satisfy that preference and also increase the efficiency of using Grid resources.

3.2 Automated negotiation

The system must have an automated negotiation mechanism. "Automated" means that the user has little interference in the negotiation process. The complexity of the SLA negotiation protocol grows significantly if a workflow consisting of multiple, dependent sub-jobs is considered. In a normal case, the user needs only submit the workflow and receives the negotiation result. The Grid service providers locate distributed RMSs over the worldwide network. The negotiation mechanism co-operates them to reach an agreement of providing services to the user.

Figure 3.4 depicts a common scenario when running a workflow in the Grid environment. The difference in the running scenario leads to more challenges for the protocol of SLA workflow negotiation comparing to the formal single job.

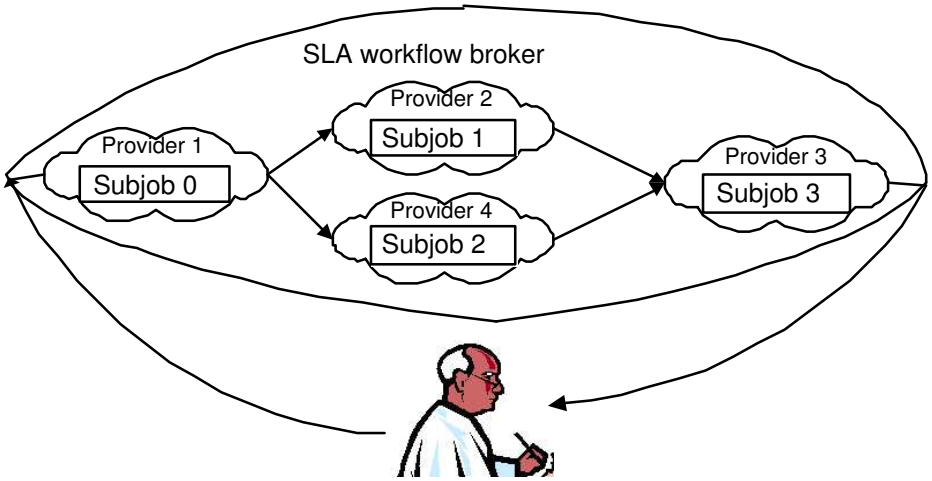

Figure 3.4: Scenario of running a workflow in the grid environment

- Because of a more complex structure, the SLA content will be also more complex.
- Besides two formal participants, the consumer and the provider, here appears a new component. It is the SLA workflow broker. Without it, the consumer needs a separate SLA for each sub-job, which is a tedious and time-intensive task.

- With the data dependency between sub-jobs in the workflow, inter-provider SLA negotiation is necessary.

This issue has not been addressed before. The existing system works only with each individual sub-job and it cannot handle the whole workflow. The workflow usually has a complex structure with many sub-jobs working in a strict manner. To negotiate SLA with the service provider, the user must trace the workflow following the process sequence to do the negotiation of running each sub-job. This task takes a lot of time. With a big workflow consisting of 20 sub-jobs, the negotiation process is a tedious job with many repeated tasks, which easily leads to confusion. The patience of the user will also be challenged if, at the end, the workflow's runtime is not within the desired time period. The work described in [130] also aims at automating the negotiation process for workflow. However, their work is mainly for Web services and the aspects of SLA is not fully considered.

Having a mechanism to automate the SLA negotiation process will help the user save a lot of time. It also frees users from annoying stuff and provides them an easy way to use the service. This is important as it encourages users to utilize the service and the system itself has chance to develop.

3.3 Error recovery

The system must have an effective error recovery mechanism to deal with errors that can happen at any time in a large and complex distributed system like the Grid. We concentrate on the serious error that one or several Grid service providers are detached from the system while executing sub-jobs of the workflow. The error recovery mechanism has to move sub-jobs of the affected workflow to other healthy RMSs in a way that minimizes the workflow runtime.

When one RMS is detached out of the Grid system, all running/waiting sub-jobs from several workflows in the RMS are considered as having failure because the system cannot control the state and collect the result from them. Checkpoint images of all sub-jobs in the failing RMS are not available to restart them in other healthy

RMSs. Besides that, the output data from finished sub-jobs in the failing RMS are also not available. Therefore, several waiting sub-jobs in healthy RMSs cannot be run because of no input data. In case of having to cancel the workflow because of error, the system will be fined seriously as stated in the SLA. Thus, the system has no way but to try to finish executing the workflow by rerunning all failing sub-jobs. However, this task faces two major problems.

- Mapping and re-executing only failed sub-jobs in other healthy RMSs are not sufficient. Workflow requires a strict execution order to ensure integrity. Only considering the failed sub-jobs and forgetting others will lead to the potential of breaking integrity. This problem has not been fully considered in the literature. The existing systems consider only the individual sub-jobs so that only the affected sub-job is re-mapped to other healthy RMSs. Unfortunately, this action breaks the integrity of the workflow. For example, with the running scenario as presented in Figure 3.4, sub-job 1 is planned to run from time slot 10 to 15, while sub-job 3 from time slot 18 to 23. If the resource running sub-job 1 fails at times slot 14, the system will restart sub-job 1 at another healthy resource from time slot 16 without considering the sub-job 3. When the time to execute sub-job 3 comes, it does not have the input data from sub-job 1 to run and consequently fails. Thus, the failure of sub-job 3 will lead to the failure of the whole workflow. Thus, determining all sub-jobs which are needed for the continued workflow execution is the mandatory requirement.

- When sub-jobs in the workflow must be re-executed, the ability to finish the workflow execution on time as stated in the original SLA is very low and the ability to be fined because of not fulfilling the SLA is nearly 100%. Within the SLA context, which relates to business, the fine is usually very high and increased with the lateness of the finished time. Thus, those sub-jobs must be mapped to the healthy RMSs in a way that minimizes the finished time.

An effective error recovery makes the Grid system more stable and reliable, important characters of a system supporting SLA.

Chapter 4

Mechanism of mapping workflow to Grid resources

This chapter presents the complex mechanism to map a workflow to the Grid resources. The mechanism includes several sub-algorithms to handle different workload and resource scenarios.

4.1 Related works

The mapping of jobs to suitable resources is one of the core tasks in Grid Computing. However, the majority of the research is related to the mapping of singular jobs, which do not exhibit dependencies to other jobs regarding input/output data. The mapping of workflows, where a single job is divided into several sub-jobs, is the next research step. In the literature, there are many attempts at this issue such as [30, 29, 19, 114, 81]. However, all those mechanisms work to map a workflow to the Grid resources in best effort manner.

Cao et al. presented an algorithm that maps each sub-job separately on an individual Grid RMS [20]. The algorithm processes one sub-job at a time, schedules it to a suitable RMS with a start time slot not conflicting with the dependency of the flow. The selection of the destination resources is optimized with respect to a minimal completion time. When applying this strategy to the specified problem, each sub-job

will be assigned separately to the cheapest feasible RMS. This strategy allows fast computation of a feasible schedule, but it does not consider the entire workflow and the dependencies among the sub-jobs.

The mapping of Grid workflows onto Grid resources based on existing planning technology is presented in [30, 29]. This work focuses on coding the problem to be compatible with the input format of specific planning systems and thus transfers the mapping problem to a planning problem. Although this is a flexible way to gain different goals, significant disadvantages regarding the huge resource usage, long response times. For example, with a workflow including 20 sub-jobs and the Grid including 10 RMSs, a planning system cannot solve the problem on a desktop computer because of not enough memory [100]. If the size of the problem is slightly smaller, the planning system needs several hours to find out the high quality solution. The latter character is the main cause that a planning system should not be used for a broker.

In two separated works [130, 16], Zeng et al and Iwona et al built systems to support QoS features for Grid-based workflow. In their work, a workflow includes many sub-jobs, which are sequential programs, and a Grid service has ability to handle one sub-job at a time. To map the workflow on to the Grid services, they used the Integer Programming method. Applying Integer Programming to our problem is impossible. The first is the flexibility in runtime of the data transfer task. The time to complete the data transfer task depends on the bandwidth and the reservation profile of the link which varies from link to link. The second is that an RMS can handle many parallel programs at a time. Thus, we do not know how many, which and when sub-jobs will be run in an RMS and we cannot present the constraint of available capacity as described in Criteria 4.

Our problem has a close relation to the classical job shop scheduling problem (JSSP) [17]. The DAG form of our workflow is similar to the graph representing the sequence processing of JSSP. Each sub-job in our problem is correlative to each operation in JSSP. As job shop scheduling problem is a NP hard problem, two main methods to solve this problem – complete and incomplete method – exist. A complete method explores systematically, though very often implicitly, the whole search space. To do this, most complete methods construct in a "step by step" way a solution and

backtrack in case of failure [74]. These methods usually use various heuristics [106, 74, 45] to guide the choice of the next variable to be instantiated and its value, and employ powerful filtering techniques to achieve different levels of consistency. Similarly, exact methods for constraint optimization are usually based on the branch and bound principle [74] and try to eliminate heuristically as many as possible solutions not leading to an optimum one. Complete and exact methods have in general exponential time complexity. The solving time required by such a method may consequently become prohibitive for large-sized problems.

An incomplete (non-exact) method does not explore systematically the whole search space. Instead, it tries to examine as rapidly as possible a large number of search points according to a selective or random strategy. Local search is one of the most popular examples of this family of methods. In general, these methods do not guarantee the completeness of the resolution, but require no exponential time complexity. They constitute in fact a very interesting alternative for the practical solving of many hard and large-sized problems. The famous method in this way include Tabu Search [49, 50], Simulated Annealing [70, 32], GA [65, 66], etc.

In the literature, when applying local search for the problem, the moving neighborhood is the most important factor in determining the speed of the algorithm and the quality of the solution. Many effective neighborhood structures N1, N2, N3, N4 [17] and N5 [90, 91] were proposed. The primary activity of these neighborhoods is changing the process sequence of two operations in the same machine. However, while each operation in the JSSP can be mapped to only one machine and one machine can process only one operation at a time, each sub-job in our problem can be mapped to several RMSs and each RMS can process several sub-jobs at a time. Thus, such process sequence changing does not exist in our solution.

The flexible job shop scheduling problem (FJSSP) extends the JSSP by assuming that, for each given operation, there exist several instances of the machine type necessary to perform it. The FJSSP is far more complicated than the classical JSSP with two main problems, that of assigning each operation to an appropriate machine, and that of sequencing the operation in each machine. To solve the FJSSP, the research community applied several local search methods and proposed several

techniques [84, 64] to decrease the very large search space. Those techniques, based strictly on the character of each machine, can do one operation at a time. Our problem differs from the FJSSP in that while each machine in FJSSP can process only one operation at a time, each RMS in our problem can process several sub-jobs at a time. Thus, we cannot apply the proposed techniques to our problem because of the differences in characteristic.

Related to mapping task graph to resources, there is also the multiprocessor scheduling precedence-constrained task graph problem [43, 71]. As this is a well-known problem, the literature recorded a lot of methods for this issue, which can be classified into several groups [77]. The classical approach is based on the so-called list scheduling technique [1, 24]. More recent approaches are UNC Scheduling [47, 107, 69, 127, 126, 76], BNP Scheduling [1, 72, 110, 7], TDB Scheduling [25, 73, 23, 22, 3], APN Scheduling [104, 111, 88, 75], genetic [108, 59]. Our problem differs from the multiprocessor scheduling precedence-constrained task graph problem in many factors. In the multiprocessor scheduling problem, all processors are similar, but in our problem, RMSs are heterogeneous. Each task in our problem can be a parallel program, while each task in this problem is a strictly sequential program. Each node in this problem can process one task at a time while each RMS in our problem can process several sub-jobs at a time. For those reasons, applying a local search approach for this problem is very rare. In our solution, modified Tabu search is used and has proven to be very efficient.

In the SLA context, the problem of mapping a workflow to Grid resources is greatly different from the formal one both in character of sub-job and character of Grid resource in RMSs as presented in section 3.1. Recently appeared algorithms such as x-DCP [83], minmin, maxmin, suffer [10, 21], GRASP [13] concentrate on scheduling the workflow with parameter sweep tasks on Grid resources. The common destination of those algorithms is optimizing the finished time of the workflow. Subtasks in this kind of workflow can be grouped into layers and there is no dependency among subtasks in the same layer. All proposed algorithms assume each task as a sequential program and each resource as a compute node. By using several heuristics, all those algorithms do the mapping very fast but the solution quality is not sufficient. Our workflow with

DAG form can also be transformed to the workflow with parameter sweep tasks type. A detailed description of those approaches that apply for our problem can be found in following sections.

Minmin algorithm

```
For each layer in sequence {
  while set of sub-jobs in the layer not empty {
    For each sub-job in the layer {
              For each RMS in the candidate set {
                compute end_time of sub-job
                store (RMS,end_time) in a list}
              Select the minimum end_time
              Store (sub-job, RMS, end_time) in a list}
  Pick (sub-job, RMS, end_time) with min
  end_time
  Drop the selected sub-job out of layer
 }}
```

Figure 4.1: Minmin algorithm

Min-min uses the Minimum MCT (Minimum Completion Time) as measurement, meaning that the task that can complete the earliest is given priority. The motivation behind Min-min is that assigning tasks to hosts that will execute them the fastest will lead to an overall reduced finished time [10, 21]. To adapt minmin algorithm to our problem, we analyze the workflow into a set of sub-jobs in sequential layers. Sub-jobs in the same layer do not depend on each other. With each sub-job in the sequential layer, we find the RMS, which can finish sub-job the earliest. The sub-job in the layer which has the earliest finish time, will be assigned to the determined RMS. The overall algorithm is presented in Figure 4.1.

Maxmin algorithm

Max-min's metric is the Maximum MCT. The expectation is to overlap long-running tasks with short-running ones [10, 21]. To adapt maxmin algorithm to our problem, we analyze the workflow into a set of sub-jobs in sequence layers. Sub-jobs in the same layer do not depend on each other. With each sub-job in the sequential layer, we find the RMS, which can finish sub-job the earliest. The sub-job in the layer which has the latest finish time, will be assigned to the determined RMS. The overall algorithm is presented in Figure 4.2.

```
For each layer in sequence {
  while set of sub-jobs in the layer not empty {
    For each sub-job in the layer {
            For each RMS in the candidate set {
              compute end_time of sub-job
              store (RMS, end_time) in a list}
            Select the minimum end_time
            Store (sub-job, RMS, end_time) in a list}
  Pick (sub-job, RMS, end_time) with max
  end_time
  Drop the selected sub-job out of layer
 }}
```

Figure 4.2: Maxmin algorithm

Suffer algorithm

The rationale behind sufferage is that a host should be assigned to the task that would "suffer" the most if not assigned to that host. For each task, its sufferage value is defined as the difference between its best MCT and its second-best MCT. Tasks with higher sufferage value take precedence [10, 21]. To adapt a suffer algorithm to our problem, we analyze the workflow into set of sub-jobs in sequence layers. Sub-jobs in the same layer do not depend on each other. With each sub-job in the sequential layer, we find the earliest and the second-earliest finish time of the sub-job. Sub-job

in the layer, which has the highest difference between the earliest and the second-earliest finish time, will be assigned to the determined RMS. The overall algorithm is presented in Figure 4.3.

```
For each layer in sequence {
   while set of sub-jobs in the layer not empty {
     For each sub-job in the layer {
                For each RMS in the candidate set {
                  compute end_time of sub-job
                  store (RMS,end_time) in a list}
        suffrage value=the earliest end_time - the
        second earliest end_time
        Store (sub-job, RMS, suffrage value) in a list
     }
     Pick (sub-job, RMS, suffrage value) with min
     suffrage value
     Drop the selected sub-job out of layer
}}
```

Figure 4.3: Suffer algorithm

GRASP algorithm

In this approach a number of iterations are made to find the best possible mapping of jobs to resources for a given workflow. In each iteration, an initial allocation is constructed in a greedy phase. The initial allocation algorithm computes the tasks whose parents have already been scheduled on each pass, and consider every possible resource for each such task. The overall algorithm is presented in Figure 4.4

w-DCP algorithm

The DCP algorithm is based on the principle of continuously shortening the longest path (also called critical path (CP)) in the task graph, by scheduling tasks in the current CP to an earlier start time. This principal was applied for scheduling workflows

```
Repeat until time limit is reached
   concreteWF=CreateMapping(workflow)
If concreteWF has lower finished_time than bestConcreteWF
   bestConcreteWF=concreteWF

procedure CreateMapping(workflow)
   While all jobs in workflow are not mapped do
      Find availJobs=unmapped jobs with every parent mapped;
      Map (availJobs)

procedure Map(availJobs)
   While all availJobs not mapped do
      For each job j do
         For each resource R do
            Calc ECT(j,R)
   l-min = min finished_time increase over all j and R
   l-max = max finished_time increase over all j and R
   s.t finished_time increase <= l-min + a*(l-max - l-min)
   (J*, R*) = random choice from availPairs
   map(j*, R*)
   Update EAT(j*, R*)
```

Figure 4.4: GRASP algorithm

with parameter sweep tasks on global Grids by Tianchi Ma et al in [83]. Directly applying this algorithm to our problem faces many difficulties especially in determining parameter AEFT (absolute earliest finished time), ALFT (absolute latest finished time). For this reason we propose a new algorithm called w-DCP as presented in Figure 4.5. Several procedures such as computing the time table and determining the critical path were employed from H-Map algorithm, which is described in section 4.2.3.

```
Initialize initial solution, compute time table,
Loop until the finished_time can not be decreased {
  While all sub-jobs in the critical path is not
  marked {
    Determine the critical path
    For each unmarked sub-job in the critical path{
      For each RMS candidate of that sub-job {
        if first sub-job then compute end_time of the
            sequence data transfer task
        else compute the end_time of the sub-job
      Push tuple (sub-job, RMS, end_time) to list
    }
    Pick the result with min end_time
    Marked the sub-job
}}}
```

Figure 4.5: w-DCP algorithm

4.2 Mapping with standard metaheuristics

Our problem is a specific case of combinatorial optimization (CO) problem and metaheuristics are usually used to solve CO problems especially in mapping and scheduling. Therefore, in the first step, we directly apply many famous metaheuristics to our problem to see the advantages and disadvantages of each method. The selected metaheuristics include Tabu Search[49, 50], Simulated Annealing[70, 32], Iterated Local Search[116, 117], Guided Local Search[119, 120], Genetic Algorithm[65, 66], and Estimation of Distribution Algorithm[94, 95]. Other methods such as Variable Neighborhood Search [53], Ant Colony Optimization [31], etc, face many difficulties when applied to our problem. For example, Ant Colony Optimization is specific for Traveling Sale Man problem, thus converting to our case requires major modifications, which can decrease the performance of the original algorithm.

4.2.1 General strategy

The input of the mapping procedure includes information about workflow and about RMSs. The user provides the information of workflow through the SLA text, which is described in section 5.3. The parser will extract information about workflow to a file describing sub-jobs and a file describing the dependence. The file describing sub-jobs contains the various sub-jobs' resource requirements and estimated runtime. The file describing the dependence among sub-jobs contains the source sub-job, the destination sub-job and the number of data transfer. Information about RMSs is stored in a relational database which is specifically designed for the system. They include the description of the resource configuration in each RMS, the resource reservation profile of each RMS, and the bandwidth reservation profile of each link. Data about RMS is collected from RMSs by the monitoring module. Based on this information, the system will do the mapping.

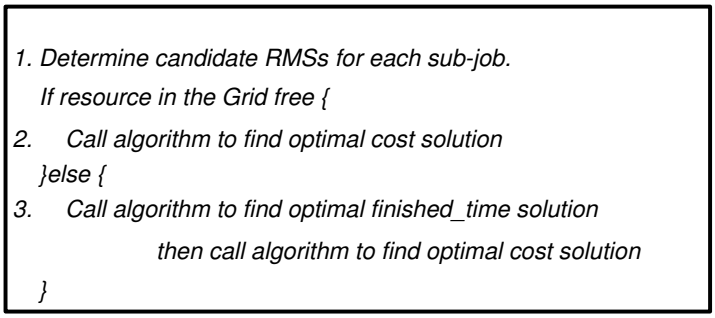

Figure 4.6: Mapping mechanism overview

Figure 4.6 presents the basic strategy of the mapping mechanism. Each sub-job has different resource requirements about the type of RMS, the type of CPU, etc. There are a lot of RMSs with different resource configurations. The first action is finding among those heterogeneous RMSs the suitable RMSs which can meet the requirement of the sub-job. Data of each resource parameter of RMS is represented by a number value and is stored in a separate column in the database table. The co-relative resource requirement parameter of sub-job is also represented by number value in the same manner. Thus, the matching between sub-job's resource requirement and RMS's resource configuration is done by several logic checking conditions in the

WHERE clause of the SQL SELECT command. This work will satisfy Criteria 2. With the case of our example, each sub-job of the workflow depicted in Figure 3.1 has three candidate RMSs as presented in Table 4.1.

Sj_ID	RMS	RMS	RMS
sj0	R1	R2	R3
sj1	R1	R2	R3
sj2	R1	R2	R3
sj3	R1	R2	R3
sj4	R1	R2	R3
sj5	R1	R2	R3
sj6	R1	R2	R3

Table 4.1: RMSs candidate for each sub-job

The second and the third actions mean that we need two different algorithms. The first is used to find cost optimal solution while the second is used to find the finished time optimal solution. The following sections will describe the tailoring of different metaheuristics to form two algorithms.

4.2.2 Tabu Search (TS)

TS [49, 50] is among the most cited and used metaheuristics for CO problems. The standard TS uses a tabu list that keeps track of the most recently visited solutions and forbids moving toward them. This allows Tabu Search both to escape from the local minima and to implement an explorative strategy.

Finding the minimal finished time with TS

The TS algorithm to find the minimal finished time of a workflow within an SLA context is presented in Figure 4.7. Starting from initial solution, the algorithm checks for the best configuration in the neighborhood of the current configuration in each iteration. The best neighbor that does not violate Tabu rule is sought to replace the current configuration even if it is no better than the current configuration in terms of the finished time of the workflow. If the best neighbor has the finished time smaller

than all configurations found so far, it will replace the current configuration even it violates Tabu rule.

```
Pick randomly an initial configuration a
Compute finished_time p of a
a"=a  // a" is the final configuration, p" is the finished_time of a"
while(num_mv<max) {
  for each solution in the neighborhood set of a {
        compute the finished_time
        store tuple (sub-job, RMS, finished_time) to a list
  }
  Sort the list according to finished_time criteria
  Pick the best solution, which has finished_time<p" or not affect Tabu rule
  Assign tabu_number to the selected RMS
  if the selected solution has finished_time<p" then store in a"
  Store the selected solution in a
  num_loop++
}
```

Figure 4.7: TS algorithm to find the minimal finished time of the workflow

Besides the standard components of Tabu Search, there are some components specific to the workflow problems.

The neighborhood set structure: A configuration can also be presented as a vector. The index of the vector represents the sub-job, value of the element represents the RMS. With a configuration a, a=$a_1a_2...a_n|$ with all $a_i \in K_i$, we generate n*(m-1) configurations a' as in Figure 4.8. We change the value of x_i to every value in the candidate list which is different from the present value. With each change, we have a new configuration. After that we have set A, $|A|$=n*(m-1). A is the set of neighborhoods of a configuration. A detail neighborhood set for the case of our example is presented in Figure 4.9.

The assigning sequence of the workflow: When the RMS to execute each sub-job, the bandwidth among sub-jobs was determined, the next task is determining a time slot to run sub-job in the specified RMS. At this point, the assigning sequence

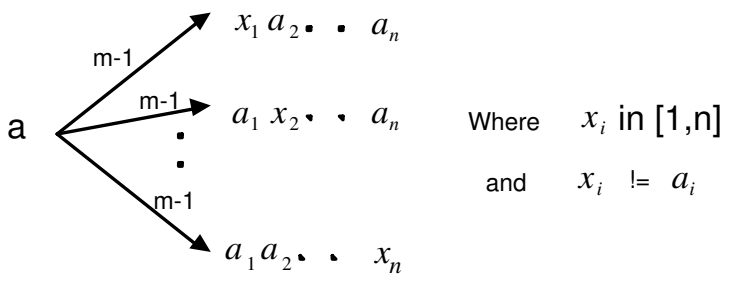

Figure 4.8: Neighborhood structure of a configuration

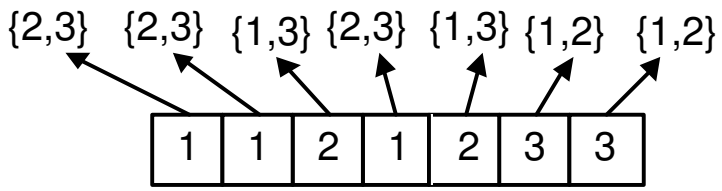

Figure 4.9: Sample neighborhood structure of a configuration

of the workflow becomes important. The sequence of determining runtime for sub-jobs of the workflow in RMS can also affect the final finished time of the workflow especially in the case of having many sub-jobs in the same RMS.

In general, to ensure the integrity of the workflow, sub-jobs in the workflow are assigned based on the sequence of the data processing. However, that principal does not cover the case of a set of sub-jobs, which have the same priority in data sequence and do not depend on each other. To examine the problem, we determine the earliest and the latest start time of each sub-jobs of the workflow in ideal condition. The time period to do data transfer among sub-jobs is computed by dividing the amount of data to a fixed bandwidth. The earliest and latest start and stop time for each sub-job and data transfer depends only on the workflow topology and the runtime of sub-jobs but not the resources context. These parameters can be determined using conventional graph algorithms. A sample of these data for the workflow in Figure 3.1, in which the number above each link represents number of time slots to do data transfer, is presented in Table 4.2.

The ability of finding a suitable resource slot to run a sub-job depends on the

Sub-job	Earliest start	Latest start
0	0	0
1	28	28
2	78	78
3	28	58
4	30	71
5	23	49
6	94	94

Table 4.2: Valid start time for sub-jobs of workflow in Figure 3.1

number of resource free during the valid running period. From the graph, we can see sub-job 1 and sub-job 3 having the same priority in data sequence. However, from the data in table 4.2, sub-job 1 can start at max time slot 28 while sub-job 3 can start at max time slot 58 without affecting the finished time of workflow. Suppose that two sub-jobs are mapped to run in the same RMS and the RMS can run one sub-job at a time. If sub-job 3 is assigned first and in the worst case at time slot 58, sub-job 1 will be run from time slot 92, thus the workflow will be late minimum 64 time slots. If sub-job 1 is assigned first at time slot 28, sub-job 3 can be run at time slot 73 and the workflow will be late 15 time slots. Here we can see the latest time factor is the main parameter for evaluating the full affection of the sequential assigning decision. It can be seen through the affection, mapping sub-job having smaller latest start time first will make the latency smaller. Thus, the latest start time value determined as above can be used to determine the assigning sequence. The sub-job having the smaller latest start time will be assigned earlier. This procedure will satisfy Criteria 3.

Determining the timetable of the workflow: To determine the finished time of the present solution, we have to determine the specific runtime period for each sub-job and each data transfer task. The start time of a data transfer task depends on the finish time of the source sub-job and the state of the link's reservation profile. We use the min_st_tran variable to present the dependence on the finish time of the source sub-job. The start time of a sub-job depends on the latest finish time of the related data transfer tasks and the state of the RMS's reservation profile. We use the min_sj_tran variable to present the dependency on the latest finish time of the

related data transfer tasks. The task to determine the timetable for the workflow is done with the procedure in Figure 4.10.

```
For each sub-job k following the assign sequence {
  For each link from determined sub-jobs to k{
    min_st_tran=end_time of source sub-job
    Search reservation profile of the link
    start_tran > min_st_tran
    end_tran = start_tran+num_data/bandwidth
    Store end_tran in a list
  }
  min_st_sj=max (end_tran)
  Search in reservation profile of RMS running
  k the start_job > min_st_sj
  end_job= start_job + runtime
}
```

Figure 4.10: Procedure to determine time table for workflow

For each sub-job of the workflow in the assigning sequence, firstly, we find all the runtime periods of data transfer tasks from previous sub-jobs to the current sub-job. This period must be later than the finish time of the source sub-job. Note that with each different link the transfer time is different because of different bandwidth. Then, we determine the runtime period of the sub-job itself. This period must be latter than the latest finish time of previous related data transfer task. The whole procedure is not so complicated but time consuming. The most time consuming steps are located in the searching reservation profiles. This procedure will satisfy the Criteria 4 and 5.

Finding the minimal cost with TS

The TS algorithm to find minimal cost solution is presented in Figure 4.11.

The TS algorithm for finding minimal cost of the workflow is similar to the algorithm presented in Figure 4.7 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. Cost value

```
Repeat  {
   Pick randomly an initial configuration a
   Compute finished_time p of a
} until p meets Criteria 1
Compute cost c of a
a"=a  // a" is the final configuration, c" is the cost of a"
while(num_mv<max) {
  for each solution in the neighborhood set of a {
      compute the finished_time and cost
      if finished_time meets Criteria 1
         store tuple (sub-job, RMS, cost) to a list
  }
  Sort the list according to cost criteria
  Pick the best solution, which has cost<c" or not affect Tabu rule
  Assign tabu_number to the selected RMS
  if the selected solution has cost<c" then store in a"
  Store the selected solution in a
  num_loop++
}
```

Figure 4.11: TS algorithm to find the minimal cost

is computed as described in Part 3.1.3. The second difference is that after choosing randomly or taking a configuration a from the neighborhood structure, the module computing timetable is called on to check if the finished time of the configuration is still within the user's limitation. If the finished time of a exceeds the user's limitation, new configuration will be selected. This action will ensure Criteria 1.

4.2.3 Simulated Annealing (SA)

SA is an advanced local search method which finds its inspiration from the physical annealing process studied in statistical mechanics [70, 32]. An SA algorithm repeats an iterative repairing procedure, which looks for better solutions while offering the possibility of accepting in a controlled manner worse solutions. The second feature

allows SA to escape from the local optima.

Finding the minimal finished time with SA

The SA algorithm to find the minimal finished time of a workflow within an SLA context is presented in Figure 4.12.

```
Pick randomly an initial configuration a
Compute the finished_time p of a
a"=a  // a" is the final configuration, p" is the finished_time of a"
Assign initial temperature t
Assign initial step length l
while(num_mv<max) {
  while(l_count<l) {
     Pick randomly a' in the neighborhood set of a
     Compute finished_time p' of a'
     if((p'-p<0)or((p'-p>0) and (Probability exp(-(p'-p)/t) is verified))){
        a=a'
        if(p'<p")
           a"=a'
        num_mv++}
     l_count++; n_ite++}
  t=t*(1-A/n_ite)
  l=l*(1-A/n_ite) }
```

Figure 4.12: SA algorithm to find the minimal finished time

The neighborhood structure, the assigning sequence, and the procedure to determine timetable are the same as those described in Tabu Search section.

Temperature t, step length l and A are the parameters of the SA algorithm. Increasing t will lead to increasing the possibility of accepting a worse solution. Decreasing l will make the number of iterations for a move decrease. To determine the value of t, A and l, we run the algorithm starting from the existed recommendation values [52]. Then, we change the value of them and see the difference in the quality of the solution. After several studies, we found with the initial values t=2.5; A=100;

l=100 the algorithm can find high quality solution than others.

Starting from initial solution, the algorithm check for a better configuration in the neighborhood of the current configuration in each iteration. When number of iteration increases, the temperature increases, the possibility to accept bad configurations increases and the number of checking a new configuration within a temperature decreases.

Finding the minimal cost for workflow with SA

The SA algorithm to find minimal cost solution is presented in Figure 4.13.

The SA algorithm for finding the minimal cost of a heavy workflow is similar to the algorithm presented in Figure 4.12 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. The cost value is computed as described in Part 3.1.3. The second difference is that after choosing randomly configuration a', the module computing timetable is called on to check if the finished time of a' is within the user's limitation. If the finished time of a' exceeds the user's limitation, a new configuration will be selected.

4.2.4 Iterated Local Search (ILS)

ILS is a simple but powerful metaheuristic algorithm [**?**, **?**]. It applies a local search to an initial solution until it finds a local optimum. Then it perturbs the solution and restarts the local search. If perturbation is too small, the algorithm might not escape from the local area. If perturbation is too strong, the algorithm will be similar to a random restart local search.

Finding the minimal finished time with ILS

The ILS algorithm to find the minimal finished time of a workflow within SLA context is presented in Figure 4.14.

The neighborhood structure, the assigning sequence and the procedure to determine timetable are the same as those described in the Tabu Search section.

```
Repeat {
   Pick randomly an initial configuration a
   Compute the finished_time p of a
} until p meets Criteria 1
Compute cost c of a
a"=a // a" is the final configuration, c" is the cost of a"
Assign initial temperature t
Assign initial step length l
while(num_mv<max) {
  while(l_count<l) {
     Repeat {
        Pick randomly a configuration a' from the neighbor hood of a
        Compute the finished_time p' of a'
     } until p' meets Criteria 1
     Compute cost c' of a'
     if((c'-c<0)or((c'-c>0) and (Probability exp(-(c'-c)/t) is verified))){
        a=a'
        if(c'<c")
           a"=a'
        num_mv++}
     l_count++; n_ite++}
  t=t*(1-A/n_ite)
  l=l*(1-A/n_ite) }
```

Figure 4.13: SA algorithm to find the minimal cost

Local search algorithm is used to find the configuration which is the local optimum in finished time. The basic steps of this algorithm include:

- Starting from the current configuration, we compute the finished time of all configuration in the set of neighborhood in order to choose the best one.
- If the best found configuration has a smaller finished time than the current configuration, we then replace the current configuration with the new one and repeat the process.

```
Pick randomly an initial configuration a
a<- LocalSearch(a)
Compute the finished_time p of a
a"=a  // a" is the final configuration, p" is the finished_time of a"
m_loop=0  // m_loop stores times the configuration not improve
while(num_mv<max) {
  a' <- Perturbation(a)
  a'<- LocalSearch(a')
  Compute finished_time p' of a'
   if(p'<p){
      a=a'
      m_loop=0
      if(p'<p"){
          a"=a'; p"=p' }
   }
   else{
      m_loop++ }
   if(m_loop<max_val)
      a=a'
}
```

Figure 4.14: ILS algorithm to find the minimal finished time

- If not, we stop the search process.

The perturbation procedure creates a new configuration from the current configuration by changing the assigned RMS of a number of sub-jobs. The candidate sub-job is selected randomly and the new RMS is selected randomly in the candidate set. After a number of studies, we have found that the number of changes equals 4 bringing a good solution and we use this value for every case in the experiment.

The ILS algorithm starts with an initial configuration. It does perturbation to create the new one and call local search to improve it as much as possible. If the finished time of the new configuration is smaller than the current one, it will replace the older. If we cannot find a better one after a number of perturbations, the current

configuration will also be replaced with the new bad one.

Finding the minimal cost for workflow with ILS

The ILS algorithm to find the minimal cost solution is presented in Figure 4.15.

The ILS algorithm for finding the minimal cost of the heavy workflow is similar to the algorithm presented in Figure 4.14 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. Cost value is computed as described in Part 3.1.3. The second difference is that after having the new configuration a' by choosing random configuration, doing perturbation or selecting from the neighborhood set, the module compute timetable is called on to check if the finished time of a' is within the user's limitation. If the finished time of a' exceeds the user's limitation, the new configuration will be selected.

4.2.5 Guided Local Search (GLS)

The basic GLS principle [119, 120] is to help the search to gradually move away from local minima by changing the search landscape. In GLS, the set of solutions and the neighborhood structure is kept fixed, while the objective function f is dynamically changed with the aim of making the current local optimum "less desirable".

Finding the minimal finished time with GLS

The GLS algorithm to find the minimal finished time of a workflow within SLA context is presented in Figure 4.16.

The neighborhood structure, the assigning sequence, and the procedure to determine timetable are the same as those described in Tabu Search section. The local search module used in GLS algorithm is similar to the one in the ILS algorithm. The objective function $f(a)$ is determined as follow:

$$f(a) = \lambda * \Sigma(k_{ij} * I_{ij}(a)) \tag{4.1}$$

Where a is the candidate configuration, λ is a parameter to the GLS algorithm.

```
Repeat  {
    Pick randomly an initial configuration a
    Compute finished_time p of a
} until p meets Criteria 1
a<- LocalSearch(a)
Compute cost c of a
a"=a  // a" is the final configuration, c" is the cost of a"
m_loop=0  // m_loop stores times the configuration not improve
while(num_mv<max) {
    Repeat  {
        a' <- Perturbation(a)
        Compute finished_time p' of a'
    } until p' meets Criteria 1
   a'<- LocalSearch(a')
   Compute cost c' of a'
    if(c'<c){
        a=a'
        m_loop=0
        if(c'<c"){
            a"=a'; c"=c' }
    }
    else{
        m_loop++ }
    if(m_loop<max_val)
        a=a'
}
```

Figure 4.15: ILS algorithm to find the minimal cost

```
Pick randomly an initial configuration a
Initialize penalty kij=0  //  each candidate RMS rj of sub-job i has a kij
a"=a  // a" is the final configuration, p" is the finished_time of a"
while(num_mv<max) {
  Compute finished_time p of a
  f'=p + f(a)
  LocalSearch(a) finds local optima a according to f' function and
  records local optima a' according to p
  Compute finished_time p' of a'
  if(p'<p"){
     a"=a'
     p"=p' }
  Compute Util(a,rj,kij)
  kij++ if Util(a,rj,kij) max
}
```

Figure 4.16: GLS algorithm to find the minimal finished time

After a number of studies, we found that λ=20 brings good results and we used this value for all experiments. k_{ij} is the penalty of candidate RMS r_j of sub-job s_i. $I_{ij}(a)$ is an indication of whether $s_i : r_j$ exists in m. $I_{ij}(a) = 1$ if $s_i : r_j$ exists in a; 0 otherwise

Function Util(a,r_j,k_{ij}) is determined as follow.

$$Util(a, r_j, k_{ij}) = I_{ij}(a) * \frac{1}{1 + k_{ij}} \tag{4.2}$$

The GLS algorithm starts with an initial configuration. It calls the local search to improve the configuration as far as possible according to the modified finished time function. The more the r_j of s_i is selected before, the less it will become attractive for the next selection and the other candidate RMS has chance to be selected. During the search process, the original finished time of the candidate configuration is also computed and the best one is recorded.

```
Repeat  {
    Pick randomly an initial configuration a
    Compute finished_time p of a
} until p meets Criteria 1
Compute cost c of a
Initialize penalty kij=0   //  each candidate RMS rj of sub-job i has a kij
a"=a  // a" is the final configuration, c" is the cost of a"
while(num_mv<max) {
   Compute cost c of a
   f'=c + f(a)
   LocalSearch(a) finds local optima a according to f' function and
   records local optima a' according to c
   Compute cost c' of a'
   if(c'<c"){
      a"=a'
      c"=c' }
   Compute Util(a,rj,kij)
   kij++ if Util(a,rj,kij) max
}
```

Figure 4.17: GLS algorithm to find the minimal cost

Finding the minimal cost for workflow with GLS

The GLS algorithm to find minimal cost solution is presented in Figure 4.17.

The GLS algorithm for finding minimal cost of the heavy workflow is similar to the algorithm presented in Figure 4.16 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. Cost value is computed as described in Part 3.1.3. The second difference is that after having the new configuration a' by choosing random configuration or selecting from the neighborhood set, the module computing timetable is called on to check if the finished time of a' is within the user's limitation. If the finished time of a' exceeds the user's limitation, a new configuration will be selected.

4.2.6 Genetic Algorithm (GA)

GA [65, 66] is a part of Evolutionary Computing, which is inspired by Darwin's theory of evolution. GA begins with a set of solutions called population. Solutions from one population are selected according to their fitness and used to form a new population. This is motivated by a hope that the new population will be better than the old one. This process is repeated to find out the best solution.

Finding the minimal finished time with GA

The GA algorithm to find the minimal finished time of a workflow within SLA context is presented in Figure 4.18.

```
Generate random population includes n configurations
while(num_mv<max) {
  Evaluate the finished_time of each configuration
  a"= best configuration
  Add a" to the new population
  while the new population is not enough {
    Select two parent configuration according to their finished_time
    Crossover the parent with a probability to form new configuration
    Mutate the new configuration with a probability
    Put the new configuration to the new population }
}
return a"
```

Figure 4.18: GA algorithm to find the minimal finished time

Parents are selected according to the roulette wheel method. The fitness of each configuration = 1/makespan. Firstly, the sum L of all configuration fitness is calculated. Then, a random number l from the interval (0,L) is generated. Finally, we go through the population to sum the fitness p. When p is greater than l, we stop and return to the configuration where we are.

The crossover point is chosen randomly. The child is formed by copying from two part of the configurations. The mutation point is chosen randomly. At mutation

```
Repeat{
   Repeat  {
      Pick randomly an initial configuration a
      Compute finished_time p of a
   } until p meets Criteria 1
   put a to the population set
}until the population includes n configurations
while(num_mv<max) {
  Evaluate the cost of each configuration
  a"= best configuration
  Add a" to the new population
  while the new population is not enough {
     Repeat{
        Select two parent configuration according to their cost
        Crossover the parent with a probability to form new configuration
        Mutate the new configuration with a probability
        Compute finished_time p of the new configuration
     } until p meets Criteria 1
     Put the new configuration to the new population }
}
return a"
```

Figure 4.19: GA algorithm to find the minimal cost

point, r_j of s_i is replaced by another RMS in the candidate RMS set. It is noted that the probability to have mutation with a child is low, ranging approximately from 0.5% to 1%.

Finding the minimal cost for workflow with GA

The GA algorithm to find the minimal cost solution is presented in Figure 4.19.

The GA algorithm for finding the minimal cost of the heavy workflow is similar to the algorithm presented in Figure 4.18 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. Cost value is computed as described in Part 3.1.3. The second difference is that after having

```
Generate random population includes n configurations
Call LocalSearch() to improve the population
while(num_mv<max) {
  Select the best half of the population
  a"=the best configuration
  Estimate joint probability distribution of the selected configurations
  using UMDA with Laplace correction
  Put a" to the new population
  Generate (n-1) individuals by sampling joint probability distribution
  Call LocalSearch() to improve the new population
  Replace the old population with the new one
}
return a"
```

Figure 4.20: EDA algorithm to find the minimal finished time

the new configuration a' by choosing random configuration, doing crossover or doing mutation, the module computing timetable is called on to check if the finished time of a' is within the user's limitation. If the finished time of a' exceeds the user's limitation, a new configuration will be selected.

4.2.7 Estimation of Distribution Algorithm (EDA)

EDA [94, 95] is a new area of Evolutionary Computation. In EDAs, there is neither a crossover nor a mutation operator. New population is generated by sampling the probability distribution which is estimated from a database containing selected individuals of the previous generation.

Finding the minimal finished time with EDA

The EDA algorithm to find the minimal finished time of a workflow within SLA context is presented in Figure 4.20.

The neighborhood structure, the assigning sequence, and the procedure to determine timetable are the same as those described in the Tabu Search section. The local

search module used in EDA algorithm is similar to the one used in the ILS algorithm. We choose Univariate Marginal Distribution Algorithm (UMDA) [96] because the probability k_{ij} of assigning r_j to s_i does not depend on other RMSs in the candidate set. If the best half of the population includes N configurations, k_{ij} increases with the increasing of number appearing s_i:r_j in the selected individuals:

$$k_{ij} = \frac{\Sigma(r_j -> s_i) + 1}{N + t_i} \tag{4.3}$$

where t_i is the size of the RMS candidate set of sub-job s_i.

The sampling process to create a new generation is based on probability k_{ij}. With each sub-job s_i, the RMS r_j is selected based on the roulette wheel mechanism as described in part 4.4.4. RMS having a big probability will have more chance to be selected. Thus, the EDA algorithm tends to choose the RMS that usually appears in high quality configurations.

Finding the minimal cost for workflow with EDA

The EDA algorithm to find the minimal cost solution is presented in Figure 4.21.

The EDA algorithm for finding the minimal cost of the workflow is similar to the algorithm presented in Figure 4.20 with two minor differences. The first difference is that the finished time value is replaced by the cost of the configuration. Cost value is computed as described in Part 3.1.3. The second difference is that after having the new configuration a' by choosing random configuration or selecting from the neighborhood set, the module computing timetable is called on to check if the finished time of a' is within the user's limitation. If the finished time of a' exceeds the user's limitation, a new configuration will be selected.

4.2.8 Evaluation of metaheuristics

We have done some experiments to evaluate the performance of standard metaheuristics when applying to the problem of mapping Grid-based workflow within the SLA context. Detail about the methods and results of the experiment are presented in

```
Repeat{
   Repeat  {
      Pick randomly an initial configuration a
      Compute finished_time p of a
   } until p meets Criteria 1
   put a to the population set
}until the population includes n configurations
Call LocalSearch() to improve the population
while(num_mv<max) {
  Select the best half of the population
  a"=the best configuration
  Estimate joint probability distribution of the selected configurations
  using UMDA with Laplace correction
  Put a" to the new population
  Repeat{
     Repeat  {
        Generate an individual a by sampling joint probability
        distribution
        Compute finished_time p of a
     } until p meets Criteria 1
     put a to the new population set
  }until the new population includes n configurations
  Call LocalSearch() to improve the new population
  Replace the old population with the new one
}
return a"
```

Figure 4.21: EDA algorithm to find the minimal cost

Section 4.4. Through the experiment results, especially the results in sections 4.4.2 and 4.4.3, we have some observations as follow:

- When the size of the problem is relatively small, all metaheuristics found nearly quality-equal results. We can deduce that the found results convergence to a value that is very near the optimal solution.
- When the size of the problem is huge, the result from each metaheuristic is different from each other. This means that the found results do not converge and may still be far from the optimal solution.
- When the size of the problem is huge, metaheuristics using local search such as GLS, ILS, EDA, found better results than other metaheuristics, which do not use local search.
- When the size of the problem is huge, the runtime of higher performance metaheuristics such as GLS, ILS, EDA is usually exponent. This factor is a drawback when applying in the real system.
- Except for Tabu Search, most standard metaheuristics use a random mechanism to rapidly explore many different areas in the search space. This mechanism may bring good results but it is very difficult to control the searched area and also the search process. In contrast, the standard Tabu Search searches very carefully the area around the initial configuration and moves only to a nearby search area.

Through these observations, directly applying standard metaheuristics to our problem faces many disadvantages, especially the long runtime. Therefore, a new mapping mechanism must be built.

4.3 Proposed mapping mechanism

We propose a mapping mechanism which can eliminate the disadvantages of standard metaheuristics. As using local search could find better result than not using local

search, we also make use of local search as a fundamental component in our mapping mechanism. The proposed mapping mechanism includes three sub-algorithms. L-Tabu finds cost optimal mapping solution for light workflow in which amount of data to be transferred among sub-jobs is little (L stands for light). H-Map finds cost optimal mapping solution for heavy workflows in which amount of data to be transferred among sub-jobs is large (H stands for heavy). w-Tabu finds the runtime optimal solution for both cases of workflows (w stands for workflow).

```
1. Determine candidate RMSs for each sub-job.
   If resource in the Grid free {
2.              if workflow has little data transfer
                    Call L-Tabu algorithm
3.              if workflow has a lot of  data transfer
                    Call H-Map algorithm
   }else {
4.              Call w-Tabu algorithm
                then call L-Tabu or H-Map
   }
```

Figure 4.22: Mapping mechanism overview

Figure 4.22 presents the basic principle of the proposed mapping mechanism. If there are a lot of Grid resources free at a specific time period, L-Tabu or H-Map algorithm is called to find the cost-optimal solution. If there are few Grid resources free, w-Tabu is called to find a feasible solution. Starting from this feasible solution, L-Tabu or H-Map will find the optimal solution. In fact, the signature of having many or few Grid resources free and the method to call on the w-Tabu algorithm are integrated in the L-Tabu and H-Map algorithms. The following sections will describe each algorithm in detail.

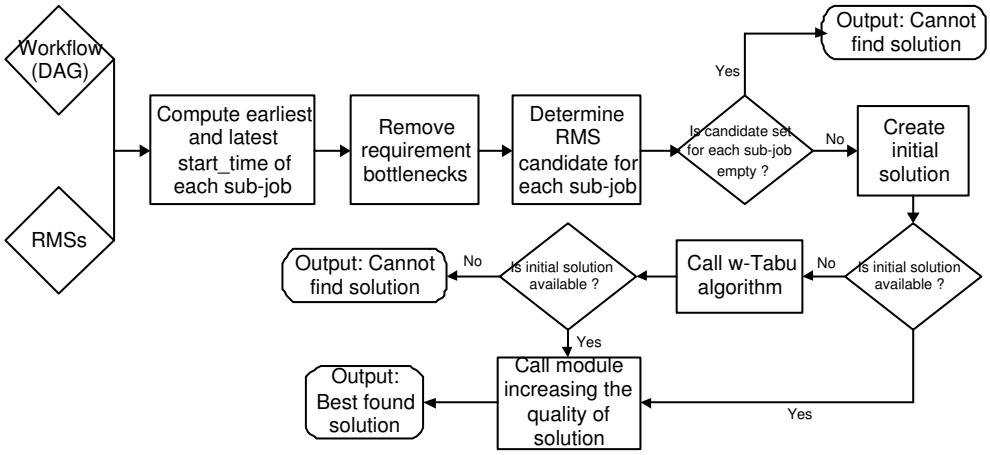

Figure 4.23: The frame work of L-Tabu algorithm

4.3.1 L-Tabu algorithm

High Performance Computing centers usually connect with the network through a link with broad bandwidth which is greater than 100Mbps. In addition, one time slot in our system is usually from 2 to 5 minutes. Thus, data transferred through the link in one time slot can be from 1.2GB to 3GB. Data transferred on the link in one second, which is approximately 10MB, is very small compared to the data transferred in one time slot. If the data transferred between each related couple sub-jobs in the workflow is smaller or equal to 10MB, then we can consider the workflow as light workflow. With that small amount of data, the data transfer can happen within any time slot in the bandwidth reservation profile without affecting other reserved data transfer task. With such kind of workflow, it is not necessary to reserve bandwidth on the link connecting two RMSs, which execute two co-related sub-jobs. In the planning table for workflow, the data transfer task takes one time slot.

L-Tabu algorithm is applied for light workflow with the purpose of finding a feasible solution with optimal cost. Figure 3.1 presents a sample light workflow with the number above each link representing the amount of data transfer. Its resource requirement is described in table 3.1. The frame work of the proposed algorithm is depicted in Figure 4.23.

- **Step 1:** Computation of the earliest start time and the latest start time for each sub-job by analyzing the input workflow with traditional graph techniques.
- **Step 2:** Removing requirement bottlenecks. This task aims at detecting bottlenecks with a large number of resource requirements which can reduce the possible start/end times of a sub-job. Based on this information, sub-jobs are moved to other sites with more available resources in order to gain a longer time span for the positioning of the sub-jobs in the workflow.
- **Step 3:** Definition of the solution space by grouping all RMS candidates which have enough available resources to start the sub-jobs within the determined slack time and to run the sub-job until it is completed. This task is performed by analyzing the resource reservation profiles of the RMSs.
- **Step 4:** The gained search space is evaluated with respect to the contained number of feasible solutions (large or small number of possible solutions). If the number of candidate RMS for each sub-job is large, the number of possible solutions is also large and vice versa. If a sub-job does not have any RMS candidate, there is no possible solution and the algorithm stops. Subsequently, an initial solution is created. If an initial solution is not found, we can deduce that the Grid resource is busy and the w-Tabu algorithm is invoked. If w-Tabu cannot find an initial solution, the algorithm will stop.
- **Step 5:** Starting with the initial solution, a specific module is called on in order to increase the quality of the solution as far as possible. This is the function of local search. However, instead of using the described local search procedure, we use Tabu Search because it can also play the role of local search but with wider search area.

In following sections, the individual algorithm steps are described in detail.

Resolving requirement bottlenecks

The generated requirement and the resource profiles show the time slots where a large number of resources are required but not available. A sample of such situation

is shown in Figure 4.24 with an example of required and available CPU instances. At each possible time slot, the total number of available CPUs over all involved RMS and the total number of required CPU as sum of the requirements of all sub-jobs possibly running in this time slot are computed. The contribution of each sub-job in the profile is computed from the earliest start time to latest possible deadline. Figure 4.24 shows that at period 28-55 or 72-84 the number of required CPUs is larger than those in other profile periods. This leads to a significantly reduced number of feasible start times and thus reduces the probability of finding an optimal solution or even a good solution. Therefore, the peak requirements have to be reduced by moving selected sub-jobs to other time slots. This is possible because the actual start and stop time of the sub-job can be shifted within a range. We can change the value of that range to move a sub-job out of the peak period. Furthermore, by reducing the number of parallel running sub-jobs on the same site, the probability for cost-effective execution of two subsequent sub-jobs on the same site with a low communication effort increases.

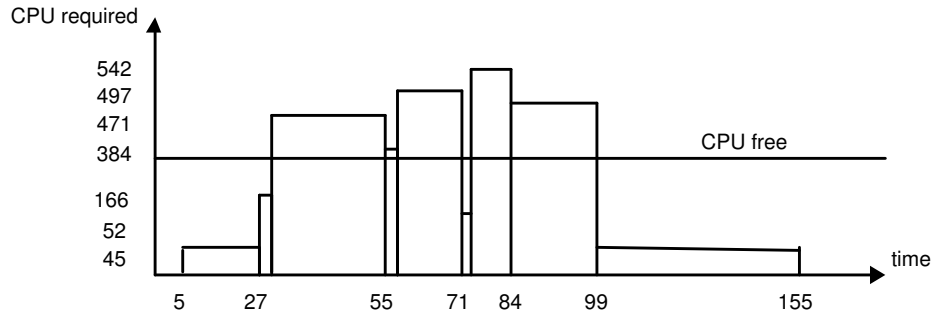

Figure 4.24: Relation between available and required CPUs

For resolving the requirement bottleneck, the profiles of the required resources and the available resources are compared as shown in Figure 4.25. At each time slot, we define J as the set of m sub-jobs running at that time slot and R as the set of n possible resource candidates for J. Subsequently, the following measures can be computed.

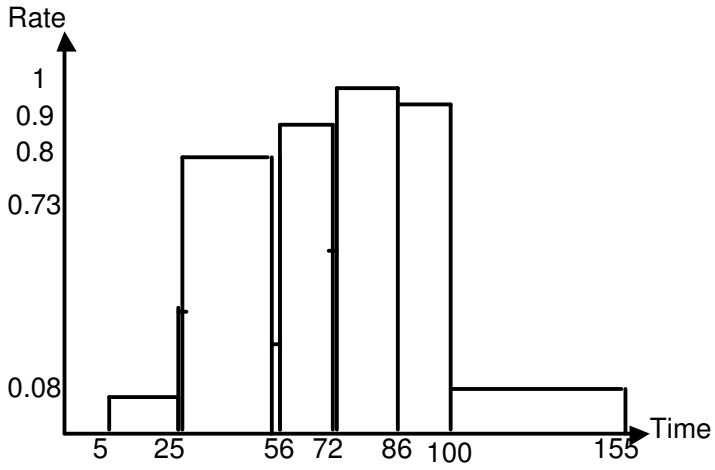

Figure 4.25: Rate profile of the sample

$$TotalCPU_{require} := \sum_{i=1,m} J_i.CPU_{req} \quad \text{with} \quad J_i \in J \tag{4.4}$$

$$TotalMEM_{require} := \sum_{i=1,m} J_i.mem_{req} \quad \text{with} \quad J_i \in J \tag{4.5}$$

$$TotalEXP_{require} := \sum_{i=1,m} J_i.EXP_{req} \quad \text{with} \quad J_i \in J \tag{4.6}$$

$$TotalCPU_{avail} := \sum_{j=1,n} R_j.CPU_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{4.7}$$

$$Totalmem_{avail} := \sum_{j=1,n} R_j.mem_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{4.8}$$

$$TotalEXP_{avail} := \sum_{j=1,n} R_j.exp_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{4.9}$$

Parameter $J_i.CPU_{req}$, $J_i.mem_{req}$, and $J_i.EXP_{req}$ represent the number of required CPUs, the size of needed memory (in MB), and the required experts for supervision of the of subjob J_i, respectively. Finally, m_j is the number of subjobs which r_j can run simultaneously.

$$rate := \frac{\frac{TotalCPU_{require}}{TotalCPU_{avail}} + \frac{Totalmem_{require}}{Totalmem_{avail}} + \frac{TotalEXP_{require}}{TotalEXP_{avail}}}{3} \tag{4.10}$$

The removal of the requirement peak is performed by adjusting the start time slot or the end time slot of the sub-jobs and thus the sub-job is moved out of the bottleneck area. One possible method is shown in Figure 4.26, where either the latest

finish time of sub-job 1 is set to t_1 or the earliest start time of sub-job 2 is set to t_2. The second way is to adjust both sub-jobs simultaneously as depicted in Figure 4.27. A necessary prerequisite here is that, after adjusting, the latest completion time - earliest start time is larger than the total runtime.

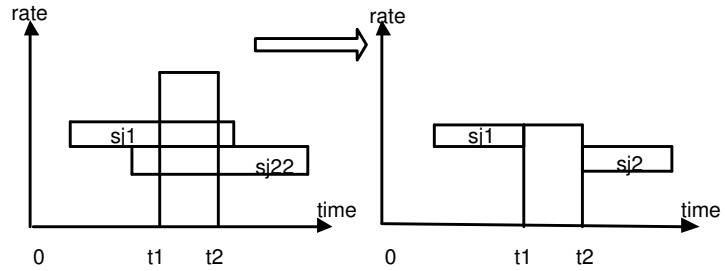

Figure 4.26: Moving subjobs

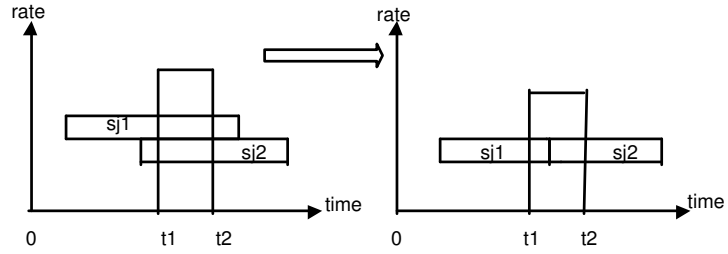

Figure 4.27: Adjusting subjobs

After doing the resolving task, we have to re-compute the earliest start time and latest finish time of each sub-job which relates to the adjusted sub-job to ensure the dependency property. The consequence is that the distribution of resource requirement has changed and we have to re-compute the new relation profile and then do the next resolving task. These steps are repeated until sub-jobs cannot be adjusted to resolve the peak. The result of applying this procedure to our example case is a compact profile as depicted in Figure 4.28.

After this phase, the variety in time factor is decreased a lot and contributes little to the final result. Thus, we fix the *start_time* of each sub-job to the earliest start time and the *end_time*=*start_time*+execution time.

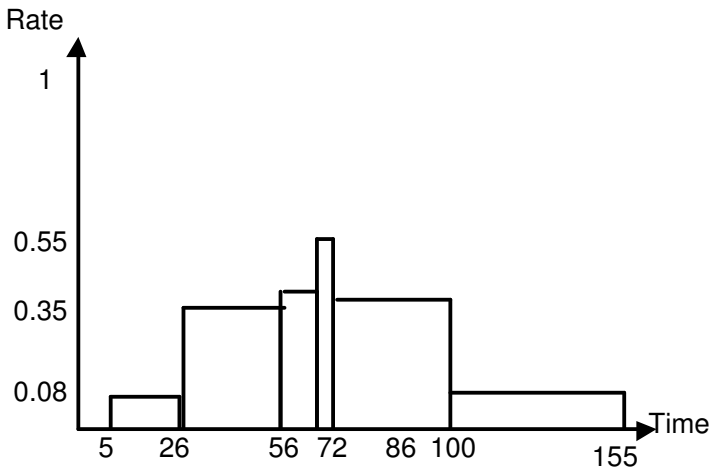

Figure 4.28: Rate profile of the sample after doing adjustments

Determining the solution space

After determining the start time for each sub-job, we need to refine the solution candidate for each sub-job. Because of the resource reservation character, it is necessary to check if an RMS candidate has enough free resource during the specific runtime period of the sub-job. Once again, this work is done by executing the SQL statement. With each RMS candidate, we query to find a period overlapping with the sub-job's runtime period that has a free resource less than the resource requirement. If a result is found then remove the RMS out of the candidate list.

Creating the initial solution

The algorithm for determining the initial solution is based on a fail-first heuristic and the forward checking algorithm [74]. According to the Greedy strategy for each sub-job under investigation, an RMS with the minimal cost is selected and assigned as described in the following four steps:

- Determine the sub-job in the workflow with the smallest number of RMS candidates, which can execute the sub-job according to the provided specification.

- If the set with RMS candidates for this sub-job is empty, then assign one randomly selected RMS and mark the assignment as conflict. Otherwise, assign the RMS with the minimal cost to that sub-job.
- Repeat the process with the next sub-job until all sub-jobs are assigned.
- Resolve the marked conflicts.

In case of conflicts, we can deduce that there is little free Grid resource and the w-Tabu algorithm is invoked as the last effort to fulfill the user requirement. If this is possible, the found feasible solution is declared as the initial solution and the mapping process proceeds with the specific procedure to improve the quality of the solution as far as possible. Otherwise, the owner of the workflow is notified that the workflow cannot be executed according to the given conditions and will be rejected until a new specification is submitted.

Improving solution quality algorithm

To improve the quality of a solution, we use a specific procedure based on Tabu search for this problem. By this time, all the start times and stop times of each sub-job in the workflow have been determined. We have to find the mapping sub-job:RMS in a way that optimizes the cost. In a normal Tabu search, in each move iteration, we will try assigning each sub-job $sj_i \in S$ with each RMS r_j in the candidate set K_i and compute the cost and then check for an overall improvement and pick the best one. This method is not so efficient as it requires searching among all the candidates. This will lead to long time running when the number of candidates is large. We will improve that method by proposing a new neighborhood for the Tabu search.

As we are working with light workflow, the data to be transferred among sub-jobs are very small and the total cost for transferring data is also small. If we have a price 0.01USD/MB for data transmission and the workflow has total 100 MB data to be transferred among sub-jobs, the total cost for this task is only 1USD. For that reason, we can bypass the impact of data transfer cost in the overall cost of the running workflow. Thus, in each moving iteration, we do not need to check for all

```
While not num_loop < max_loop{
  For each sub-job in the workflow {
    For each cheaper RMS in the candidate set {
        Compute cost
        Store tuple (sub-job, RMS, cost) to a list
  } }
  Sort the list
  Pick the best solution which is cheaper or or not
  affect tabu rule
  Assign tabu_number for the selected RMS
  If  cheaper then store the solution
  num_loop++
}
```

Figure 4.29: Improving solution quality procedure for light workflow

RMS in the candidate list but concentrate on RMS having a cheaper running cost than the current RMS. Tracing through cheaper RMS can be made easier by using a sorted list. With this new neighborhood, the algorithm has very short runtime while still finding high quality solutions. The pseudo-code of the procedure is found in Figure 4.29.

4.3.2 H-Map algorithm

H-Map algorithm maps heavy workflow to the Grid RMSs. As the data to be transferred among sub-jobs in the workflow are huge, to ensure the deadline of the workflow it is necessary to make bandwidth reservation. In this case, the time to do a data transmission task becomes unpredictable as it depends on the bandwidth and the reservation profile of the link, which varies from link to link. The variety in the completion time of the data transmission task makes the total runtime of the workflow also flexible. The goal of H-Map algorithm is finding out a solution, which ensures Criteria 1, and is as cheap as possible. The overall H-Map algorithm is presented in Figure 4.30.

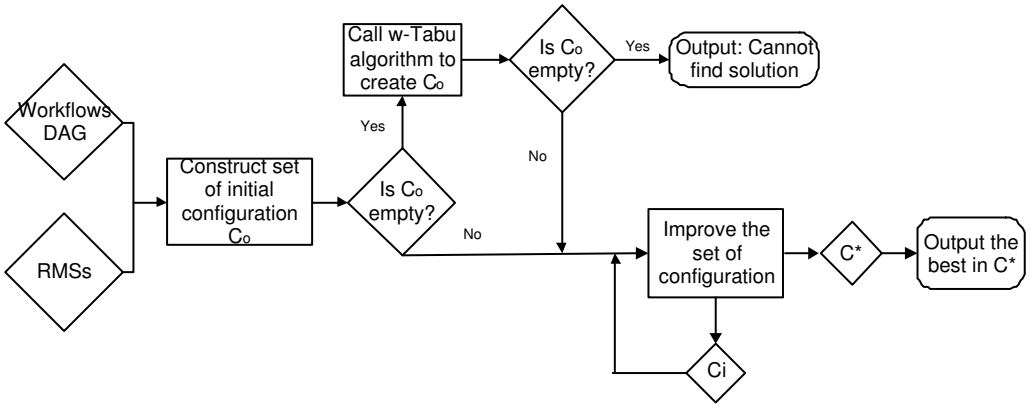

Figure 4.30: H-Map algorithm overview

Firstly, a set of initial configurations C_o is created. The configurations in C_o should be distributed widely over the search space and must satisfy Criteria 1. If $|C_o| = \emptyset$, we can deduce that there is little resource free on the Grid and the w-Tabu algorithm is invoked. If w-Tabu also cannot find out a feasible solution, the algorithm stops. If $|C_o| \neq \emptyset$, the set will gradually be refined to have better quality solutions. The refining process stops when the solutions in the set cannot be improved more and we have the final set C*. The best solution in C* will be output as the result of the algorithm. The following sections will describe detail each procedure in the algorithm.

Constructing the set of initial configurations

The purpose of this algorithm is to create a set of initial configurations which will be distributed widely over the search space.

Step 0: With each sub-job s_i, we sort the RMSs in the candidate set K_i according to the cost they need to run s_i. The cost is computed according to Formula 4.3. The configuration space of the sample now can be presented in Figure 4.31 and Table 4.3. In Figure 4.31, the RMSs lying along the axis of each sub-job have cost increasing in the direction from inside to outside. The line connecting each point in every sub-job axis will form a configuration. Figure 4.31 presents 3 configurations with an increasing index in the direction from inside to outside. Figure 4.31 also presents the cost

distribution of the configuration space according to Formula 3.3. The configuration in outer layers has a greater cost than the inner layers. The cost of the configuration lying between two layers is greater than the cost of the inner layer and smaller than the cost of the outer layer.

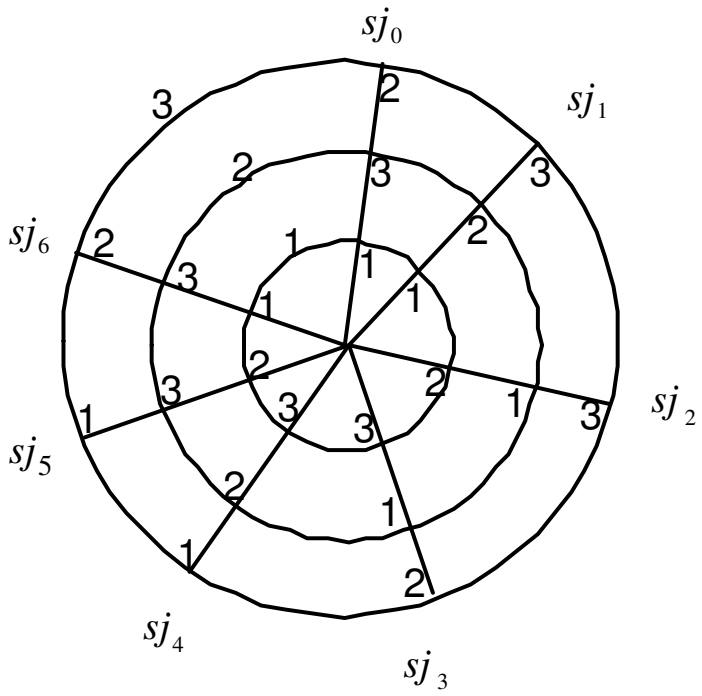

Figure 4.31: The configuration space according to cost distribution

Step 1: We pick the first configuration as the first layer in the configuration space. The determined configuration can be presented as a vector. The index of the vector represents the sub-job, and the value of the element represents the RMS. The first configuration in our example is presented in Figure 4.32. Although the first configuration has minimal cost according to Formula 4.3, we cannot be sure that this is the optimal solution. The real cost of a configuration must consider the neglected cost of data transmission when two sequential sub-jobs are in the same RMS.

Step 2: We construct the other configurations by doing a process similar to the one described in Figure 4.33. The second solution is the second layer of the configuration space. Then we create a solution having cost located between layer

Sj_ID	RMS	RMS	RMS
sj0	R1	R3	R2
sj1	R1	R2	R3
sj2	R2	R1	R3
sj3	R3	R1	R2
sj4	R3	R2	R1
sj5	R2	R3	R1
sj6	R1	R3	R2

Table 4.3: RMSs candidate for each sub-job in cost order

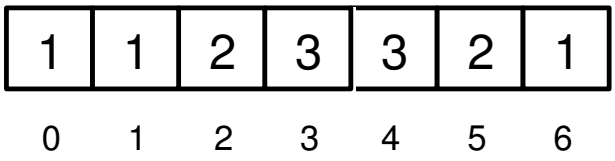

Figure 4.32: The first selection configuration of the sample

1 and layer 2 by combining the first and the second configuration. To do this, we take the p first elements from the first vector configuration and then the p second elements from the second vector configuration and repeat until having n elements to form the third one. Thus, we get (n/2) elements from the first vector configuration and (n/2) other elements from the second one. Combining in this way will ensure the target configuration of having a greater difference in cost according to Formula 3.3 compared to the source configurations. The process continues until reaching the final layer. Thus, we have in total 2*(m-1) configurations. With this method, we can ensure that the set of initial configurations is distributed over the search space according to cost criteria.

Step 3: We check Criteria 2 of all 2*m-1 configurations. To verify Criteria 2, we have to determine the timetable for all sub-jobs of the workflow. The procedure to determine the timetable of the workflow is similar to the one described in Figure 4.10. If some of them do not satisfy the Criteria 2 requirement, we construct more to have enough 2*m-1 configurations. To do the construction, we change the value of p parameter in the range from 1 to (n/2) in step 2 to create the new configuration.

After this phase we have set C_0 including maximum (2m-1) valid configurations.

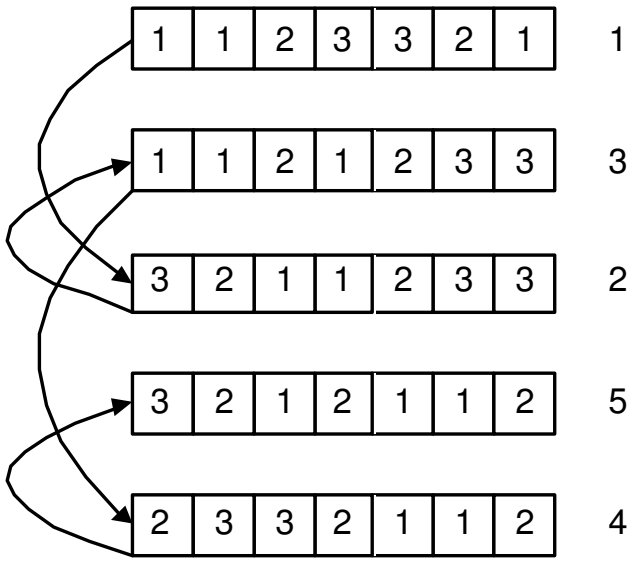

Figure 4.33: Procedure to create the set of initial configurations

Improving solution quality algorithm

To improve the quality of the solutions, we use the neighborhood structure as described in Section 4.3.2. Call A the set of neighborhood of a configuration. The procedure to find the highest quality solution includes the following steps.

Step 1: $\forall a \in A$, calculate cost(a) and timetable(a), pick $a*$ with the smallest cost($a*$) and satisfy Criteria 2, put $a*$ to set C_1. The detailed technique of this step is described in Figure 4.34.

We consider only the configuration having a smaller cost than the present configuration. Therefore, instead of computing the cost and the timetable of all configuration in the neighborhood set, we compute only the cost of them. All the cheaper configurations are stored in a sorted list. And then we compute the timetable of cheaper configurations along the list to find the first feasible configuration. This technique helps to decrease a lot of the algorithm's runtime.

Step 2: Repeat step 1 with all $a \in C_0$ to form C_1.

Step 3: Repeat step 1 to 2 until C_t=C_{t-1}.

Step 4: $Ct \equiv C*$. Pick the best configuration of $C*$.

```
For each subjob in the workflow {
   For each RMS in the candidate list {
      If cheaper then put (sjid, RMS id, improve_value)
      to a list }}
Sort the list according to improve_value
From the begin of the list{
   Compute time table to get the finished time
   If finished time < limit
      break
}
Store the result
```

Figure 4.34: Algorithm to improve the solution quality

4.3.3 w-Tabu algorithm

The main purpose of the w-Tabu algorithm is finding out a feasible solution when there are few free Grid resources. This destination is equal to finding a solution with the minimal finished time. Within the SLA context as defined in section 4.1, the finished time of the workflow depends on the reservation state of resources in RMSs, bandwidth among RMSs, and bandwidth reservation state. It is easy to show that this task is an NP hard problem. Although the problem has the same destination as most of existing algorithm mapping a DAG to resources [83, 10, 21, 13], the defined context is different from all other context appearing in the literature. Thus, the problem needs a new approach to be solved. We propose a mapping strategy as depicted in Figure 4.35.

Firstly, a set of referent configurations is created. Then we use a specific module to improve the quality of each configuration as far as possible. The best configuration will be selected. This strategy looks similar to an abstract of a long term local search such as Tabu search, Grasp, SA, etc. However, detailed description makes our algorithm distinguishable from them.

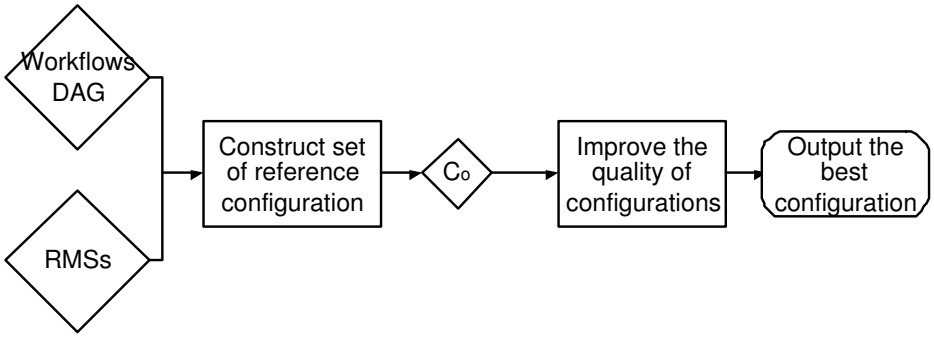

Figure 4.35: w-Tabu algorithm overview

Generating reference solution set

Each configuration from the reference configurations set can be thought of as the starting point for a local search so it should be spread as widely as possible in the searching space. To satisfy the space spreading requirement, the number of the same map sub-job:RMS between two configurations must be as small as possible. The number of the member in the reference set depends on the number of available RMSs and the number of sub-jobs. During the process of generating a reference solution set, each candidate RMS of a sub-job has a co-relative *assign_number* to count the times that RMS is assigned to the sub-job. During the process of building a reference configuration, we use a similar set to store all defined configurations having at least a map sub-job:RMS similar to one in the creating configuration. The algorithm is defined in Figure 4.36.

While building a configuration with each sub-job in the workflow, we select the RMS in the set of candidate RMSs, which create a minimal number of similar sub-job:RMS with other configurations in the similar set. After that, we increase the *assign_number* of the selected RMS. If this value is larger than 1, which means that the RMS were assigned to the sub-job more than one time, there must exist configurations that contain the same sub-job:RMS and thus satisfy the similar condition. We search these configurations in the reference set which have not been in the similar set, and then add them to the similar set. When finished, the configuration is put to the reference set. After all reference configurations are defined, we use a specific

```
assign_number of each candidate RMS =0
While m_size < max_size {
  Clear similar set
  For each sub-job in the workflow {
     For each RMS in the candidate list {
        For each solution in similar set {
                If solution contains sub-job:RMS
                  num_sim++
                Store tuple (sub-job, RMS, num_sim) in
                a list }}
     Sort the list
     Pick the best result
     assign_number++
     If assign_number > 1
        Find defined solution having the same
        sub-job:RMS and put to similar set
}}
```

Figure 4.36: Generating reference set algorithm

procedure to refine each of the configuration as far as possible.

Solution improvement algorithm

To improve the quality of a configuration, we use a specific procedure based on short term Tabu search for this problem. Just likes the L-Tabu algorithm, we use Tabu Search because it can also play the role of a local search but with a wider search area. Before improving the quality of the configuration, we have to determine the specific runtime period for each sub-job as well as the finished time of the present configuration. This task is done with the algorithm in Figure 4.37. This algorithm is similar to the algorithm in Figure 4.10 with only a minor difference in determining the end time of data transfer parameter. As the w-Tabu algorithm applies both for light workflow and heavy workflow, determining the parameter for each case cannot

```
With each sub-job k following the assign sequence {
   Determine set of assigned sub-jobs Q, which having output
   data transfer to the sub-job k
   With each sub-job i in Q {
      min_st_tran=end_time of sub-job i +1
      If heavy weight workflow {
         Search in reservation profile of link between RMS running
         sub-job k and RMS running sub-job i to determine start and
         end time of data transfer task with the start time >
         min_st_tran } else {
            end time data transfer = min_st_tran }
   }
   min_st_sj=max end time of all above data transfer +1
   Search in reservation profile of RMS running
   sub-job k to determine its start and end time with
   the start time > min_st_sj
}
```

Figure 4.37: Determining timetable algorithm for workflow in w-Tabu

be the same. With light workflow, the end time of data transfer equals the time slot after the end of the correlative source sub-job. With a heavy workflow, the end time of data transfer is determined by searching the bandwidth reservation profile.

In normal Tabu search, in each move iteration, we will try assigning each sub-job $sj_i \in S$ with each RMS r_j in the candidate set K_i and use the procedure in Figure 4.37 to compute the runtime and then check for overall improvement and pick the best one. This method is not efficient as it requires a lot of time for computing the runtime of the workflow which is not a simple procedure. We will improve the method by proposing a new neighborhood with two comments.

Comment 1: The runtime of the workflow depends mainly on the execution time of the critical path. In one iteration, we can move only one sub-job to one RMS. If the sub-job does not belong to the critical path, after the movement, the old critical path will have a very low probability of being shortened and the finished time of the workflow has a low probability of improvement. Thus, we concentrate only on sub-jobs in the critical path. With a defined solution and runtime table, the critical path of a workflow is defined with the algorithm in Figure 4.38.

```
Let C is the set of sub-jobs in the critical path
Put last sub-job into C
next_subjob=last sub-job
do{
   prev_subjob is determined as the sub-job having
   latest finished data output transfer to  next_subjob
   Put prev_subjob into C
   next_sj=prev_subjob
} until prev_sj= first sub-job
```

Figure 4.38: Determining critical path algorithm

We start with the last sub-job determined. The next sub-job of the critical path will have latest finish data transfer to the previously determined sub-job. The process continues until next sub-job is equal to first sub-job. Figure 4.39 depicts a sample critical path of the workflow in Figure 3.1.

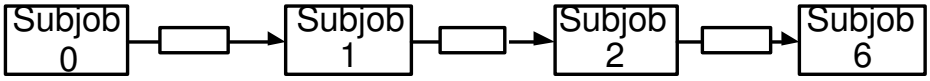

Figure 4.39: Sample critical path of the workflow in Figure 3.1

Comment 2: In one move iteration, with only one change of one sub-job to one RMS, if the finish time of the data transfer from this sub-job to the next sub-job in the critical path is not decreased, the critical path cannot be shortened. For this reason, we only consider the change which shortens the finish time of consequent data transfer. It is easy to see that checking if we can improve the data transfer time is much shorter than computing the runtime table for the whole workflow.

With two comments and other remaining procedures similar to the standard Tabu search, we build the overall improvement procedure as presented in Figure 4.40.

```
while (num_loop<max_loop){
   Determine critical path
   For each sub-job in the critical path {
     For each RMS in the candidate set {
       If can improve the finished time of the
       sequence data transfer {
         Compute timetable for new solution
         Store tuple (sub-job, RMS, makespan) to
         candidate list
   } } }
   Pick the solution having smaller makespan
   or not affect tabu rule
   Assign tabu_number for the selected RMS
   If  smaller makespan  then store the solution
   num_loop++
}
```

Figure 4.40: Configuration improvement algorithm in w-Tabu

4.4 Performance experiment

The performance experiment is done with simulation to check for the quality of the mapping algorithms. The hardware and software used in the experiments is rather standard and simple (Pentium 4 2,8Ghz, 2GB RAM, Linux Redhat 9.0, MySQL). The whole simulation program is implemented in C/C++. We generated several scenarios with different workflow configurations and different RMS configurations to suit with the ability of the comparing algorithms. The goal of the experiment is to measure the feasibility, the quality of the solution and the time needed for the computation.

Table 4.4 describes the resource configuration used in all experiments. In reality, the number of sites in the Grid can be large but they are also heterogeneous in static resource configurations for example CPU_speed, CPU_type, OS and so on. Therefore, the number of sites having the resource configuration greater than the requirement

RMS	CPU_type	CPU_speed	Num_CPU	Storage	Expert	Cost_cpu	Cost_stor	Cost_exp	Cost_tran
1	0	1700	256	300000	4	0.0501	0.00701	0.15	0.06
2	0	1700	128	200000	3	0.0503	0.00802	0.133333	0.05
3	0	1700	256	300000	4	0.0502	0.00703	0.15	0.06
4	0	1700	256	300000	4	0.0501	0.00701	0.15	0.06
5	0	1700	256	300000	4	0.0504	0.00702	0.15	0.06
6	0	1700	64	100000	2	0.0502	0.00801	0.166667	0.06
7	0	1700	128	200000	3	0.0501	0.00802	0.133333	0.05
8	0	1700	256	300000	4	0.0503	0.00703	0.15	0.06
9	0	1700	64	100000	2	0.0502	0.00801	0.166667	0.06
10	0	1700	128	200000	3	0.0501	0.00801	0.133333	0.05
11	0	1700	128	200000	3	0.0502	0.00802	0.133333	0.05
12	0	1700	128	200000	3	0.0504	0.00803	0.133333	0.05
13	0	1700	128	200000	3	0.0502	0.00801	0.133333	0.05
14	0	1700	128	200000	3	0.0502	0.00803	0.133333	0.05
15	0	1700	256	300000	4	0.0501	0.00703	0.15	0.06
16	0	1700	256	300000	4	0.0503	0.00701	0.15	0.06
17	0	1700	128	200000	3	0.0501	0.00802	0.133333	0.05
18	0	1700	256	300000	4	0.0502	0.00704	0.15	0.06
19	0	1700	64	100000	2	0.0501	0.00801	0.166667	0.06
20	0	1700	128	200000	3	0.0503	0.00803	0.133333	0.05

Table 4.4: Resource configuration used in the experiment

and having the same static resource configuration is not so big. Beside that, the static resource configuration has little impact on the efficiency of the algorithms as it can be filtered easily with SQL command. The difference in the number of resources and the price of resources in each RMS has a great influence on the quality of the solution and the execution time of the algorithms. Thus, 20 RMSs used in the experiments having the same static configuration but different in total resource and price policy can simulate well the real Grid scenario.

The workflow configurations and the resource reservation information in each RMS are changed to suite with the scenario of each algorithm in the experiments.

4.4.1 Optimizing cost for light communication workflows experiment

In this experiment, we use the comparing algorithms to map the light communication workflows to the Grid resources. To do the experiment, 18 workflows with different configurations were generated. The workflows are different in:

- Topology.
- Number of sub-jobs. Number of sub-jobs is in the range 7-35.
- Sub-job specifications.
- Amount of data transfer. Amount of data transfer is in the range 1MB - 10MB.

	L-Tabu		Tabu		SA		ILS		GLS		GA		EDA	
Wf	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost
1	2	629.04	67	632.69	36	633.34	27	632.37	14	632.69	23	632.37	78	632.37
2	1	753.13	89	756.77	17	756.77	41	756.76	25	757.46	27	756.77	129	756.76
3	2	776.10	130	780.13	31	780.13	71	780.12	33	780.62	30	780.12	198	780.12
4	3	831.13	167	835.97	23	835.97	85	835.97	46	836.39	32	835.97	223	835.97
5	3	886.12	207	891.81	25	891.81	117	891.81	63	892.85	39	891.81	296	891.81
6	3	941.12	286	947.61	65	947.61	174	947.61	87	948.34	36	947.61	473	947.61
7	4	996.10	334	1003.47	96	1003.47	204	1003.47	107	1004.26	39	1003.47	497	1003.47
8	5	1051.19	432	1059.36	22	1059.36	257	1059.36	150	1059.98	44	1059.36	700	1059.36
9	5	1175.24	503	1183.45	34	1183.76	322	1183.76	187	1184.44	53	1183.77	628	1183.76
10	7	1299.20	594	1308.02	32	1308.03	367	1308.03	199	1308.81	62	1308.03	758	1308.02
11	7	1354.20	689	1363.86	146	1363.86	459	1363.86	243	1363.97	49	1363.87	1107	1363.86
12	8	1478.19	836	1482.25	49	1488.20	532	1488.20	272	1489.08	54	1488.21	1088	1488.20
13	9	1501.23	989	1511.62	58	1511.62	698	1511.62	339	1512.64	57	1511.64	1384	1511.62
14	9	1556.25	1306	1567.97	56	1567.50	805	1567.48	386	1567.84	57	1567.50	1491	1567.48
15	12	1579.29	1292	1590.69	94	1590.93	890	1590.91	445	1591.52	55	1590.92	2000	1590.90
16	15	1772.36	2480	1785.82	107	1785.36	1682	1785.35	779	1786.11	66	1785.38	3056	1785.35
17	18	1873.23	2914	1887.24	55	1887.92	2562	1887.91	1045	1888.50	76	1887.92	3788	1887.91
18	25	2199.35	4495	2215.56	80	2215.89	3276	2215.88	1572	2216.90	88	2215.92	6038	2215.89

Table 4.5: Experiment results of the L-Tabu algorithm

In the algorithms, number of sub-job is the most important factor to the execution time of the algorithm. We stop at 35 sub-jobs for a workflow because as far as we know, with our model of parallel task sub-job, most existing Grid-based workflows [82, 11, 105] just include 10-20 sub-jobs. Thus, we believe that our workload configuration can simulate accurately the requirement of the real problems. Those workflows will be mapped to 20 RMSs with different resource configurations and different resource reservation contexts. The reserved resource in each RMS is not much and there are a lot of free resources over the time axis. This character ensures of having a great number of feasible solutions. The workflows are mapped by 7 algorithms L-Tabu, TS, SA, GLS, ILS, GA, and EDA. The cost and the runtime of solutions generated by each algorithm correlative with each workflow are recorded. The result of the experiment for this case is presented in table 4.5. Column Wf (Workflow) presents the id of workflows used in the experiment. The number of sub-jobs in the workflow increases with the increasing of id. The cost and runtime of solutions generated by each algorithm correlative with each workflow is recorded in column Rt (Runtime) and Cost respectively.

From the results, we can see that the L-Tabu algorithm created higher quality solutions than all comparing algorithms. Compares to other algorithms, faster computation and slightly lower cost of the solution created by the proposed algorithm was observed. The difference in the cost and runtime of the founded solutions between L-Tabu and other algorithms increases when the size of the workflow increases. With large-scale problems, algorithms such as ILS, GLS, and EDA, which use a local search module, have an exponential runtime and find out just equal results with other ones, except L-Tabu.

4.4.2 Optimizing cost for heavy communication workflows experiment

In this experiment, we use the comparing algorithms to map the heavy communication workflows to the Grid resources. To do the experiment, 18 workflows with different configurations were generated. The workflows are different in:

- Topology.
- Number of sub-jobs. Number of sub-jobs is in the range 7-35.
- Sub-job specifications.
- Amount of data transfer. Amount of data transfer is in the range 1GB - 6GB.

In the algorithms, the number of the sub-job is the most important factor to the execution time of the algorithm. We also stop at 35 sub-jobs for a workflow because as far as we know, with our model of parallel task sub-job, most existing Grid-based workflows include only 10-20 sub-jobs. Thus, we believe that our workload configuration can simulate accurately the requirement of the real problems. Those workflows will be mapped to 20 RMSs with different resource configurations and different resource reservation contexts. The workflows are mapped by 7 algorithms H-Map, TS, SA, GLS, ILS, GA, and EDA. The cost and the runtime of solutions generated by each algorithm correlative with each workflow are recorded. The final result of the experiment is presented in table 4.6. Column Wf (Workflow) presents the id of workflows used in the experiment. The number of sub-jobs in the workflow increases with the increasing of id. The cost and runtime of solutions generated by each algorithm correlative with each workflow is recorded in column Rt (Runtime) and Cost respectively.

The experiment results show that the H-Map algorithm finds out equal or higher quality solutions with a much shorter runtime than other algorithms in most cases. With small problems, some metaheuristics using local search such as ILS, GLS, and EDA find out equal results with H-Map and better than SA or GA. But with large-scale problems, they have an exponential runtime and find out unsatisfactory results.

4.4.3 Optimizing the finished time for workflows experiment

Optimizing the finished time of the Grid-based workflow is a well known problem in Grid Computing. Many attempts at this problem have resulted in a number of mapping methods such as [83, 10, 21, 13]. Although most of the previous work has been

	H-Map		Tabu		SA		ILS		GLS		GA		EDA	
Wf	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost
1	1	632.58	58	632.98	105	632.58	21	632.58	14	632.58	25	632.58	62	632.58
2	0.5	756.96	83	757.42	84	756.90	37	756.96	19	756.90	22	756.96	88	756.90
3	1	780.42	157	780.60	114	780.42	54	780.42	29	780.34	27	780.34	126	780.34
4	1	836.18	212	836.63	78	836.18	81	836.18	46	836.18	28	836.18	178	836.18
5	1	892.02	259	926.00	105	892.02	114	892.02	59	892.02	29	892.21	241	892.02
6	2	948.10	235	948.64	90	948.10	147	948.10	80	948.10	36	1005.27	390	947.86
7	1	1003.7	279	1059.77	78	1003.99	201	1003.7	98	1003.99	36	1075.19	462	1003.7
8	2	1059.89	669	1114.5	121	1059.89	250	1059.89	127	1059.89	32	1059.89	558	1059.89
9	2	1184.21	453	1247.35	130	1184.21	307	1184.21	167	1184.21	44	1248.86	659	1183.92
10	2	1308.53	607	1352.28	146	1332.53	398	1308.53	187	1308.53	47	1383.53	680	1308.53
11	3	1364.14	696	1433.89	124	1376.42	502	1364.14	222	1377.63	52	1440.81	956	1364.42
12	2	1488.12	744	1575.41	184	1551.74	462	1521.74	303	1501.95	51	1569.39	854	1536.74
13	7	1512.26	1005	1512.04	174	1512.26	620	1512.26	354	1566.09	56	1620.17	1136	1512.26
14	3	1567.74	1306	1683.23	162	1631.15	815	1567.74	392	1568.15	56	1663.81	1255	1601.15
15	6	1591.12	1225	1713.00	161	1675.67	876	1591.12	524	1621.67	70	1764.18	1663	1621.67
16	5	1786.56	1912	1846.81	180	1871.81	1394	1840.31	763	1843.55	85	1914.91	2845	1830.87
17	7	1889.78	2567	2030.13	197	1960.87	1695	1892.27	1258	1936.83	93	2028.06	4170	1961.30
18	10	2217.34	4545	2350.36	272	2276.33	2046	2283.67	1623	2256.53	1953	2406.97	10976	2276.33

Table 4.6: Experiment results of the H-Map algorithm

for workflow without resource reservation, the principle ideas can also apply to our problem. For that reason, beside applying and doing experiments with metaheuristic algorithms, we also apply and do experiments with other proposed mechanisms.

Comparing with metaheuristic algorithms

To do the experiment, we used the workflow configurations and resource configurations like in the case of H-Map experiment. The makespan and the runtime of solutions generated by each algorithm correlative with each workflow are recorded. The final result of the experiment is presented in table 4.7. Column Wf (Workflow) presents the id of workflows used in the experiment. The number of sub-jobs in the workflow increases with the increasing of id. The runtime and the value makespan (makespan equals to the finished time subtracts the pre-determined start time) of the solutions generated by each algorithm correlative with each workflow is recorded in

column Rt (Runtime) and Mksp respectively.

	w-Tabu		Tabu		SA		ILS		GLS		GA		EDA	
Wf	Rt	Mksp	Rt	Mksp	Rt	Mksp	Rt	Mksp	Rt	Mksp	Rt	Mksp	Rt	Mksp
1	1	69	55	69	9	69	8	69	28	69	28	121	57	69
2	1	85	80	87	13	93	28	85	50	129	32	114	89	85
3	3	100	124	103	27	107	15	161	89	119	41	129	134	96
4	4	97	138	97	26	97	72	96	108	117	38	100	153	101
5	2	109	175	114	38	115	56	120	143	190	47	204	209	109
6	4	99	232	99	45	120	102	105	212	181	45	136	274	120
7	4	105	271	129	44	135	111	112	265	166	49	192	324	99
8	5	124	344	120	67	144	373	116	357	164	54	177	395	141
9	5	116	410	148	60	158	98	125	339	196	53	176	462	161
10	5	133	660	139	78	142	119	142	413	195	51	202	580	148
11	6	127	557	134	84	154	165	157	488	204	57	239	637	160
12	6	162	668	165	104	160	141	175	622	212	64	215	838	179
13	10	192	814	168	168	168	305	163	859	216	66	256	1047	173
14	11	160	941	161	136	184	172	256	990	197	63	197	966	200
15	9	164	1037	183	107	245	315	233	909	247	67	292	1122	186
16	13	172	1695	242	187	196	1678	172	1695	217	80	276	1786	208
17	16	173	2411	187	233	212	3056	179	2281	260	84	279	2258	212
18	16	216	2626	234	312	253	907	404	3323	281	94	330	2846	247

Table 4.7: Experiment results of the w-Tabu algorithm

The experiment results show that the w-Tabu algorithm finds out an equal or a higher quality solution with much shorter runtime than other algorithms in most cases. With small problems, some metaheuristics using local search such as ILS, GLS, and EDA find out equal results with the w-Tabu and better than the SA and the GA. But with large-scale problems, they have an exponential runtime and find out unsatisfactory results.

Comparing with other existed mapping algorithms

We employed all the ideas in the recently appeared paper [83, 10, 21, 13] related to mapping workflow to Grid resource with the same destination to minimize makespan and adapt them to our problem. Those algorithms include w-DCP, Grasp, minmin,

maxmin, and suffer. To compare the quality of all the described algorithms above, we generated 90 different workflows which:

- Have different topologies.
- Have different number of sub-jobs. The number of sub-jobs is in the range 7-35.
- Have different sub-job specifications.
- Have different amount of data transfer. The amount of data transfer is in the range from several hundred MB to several GB.

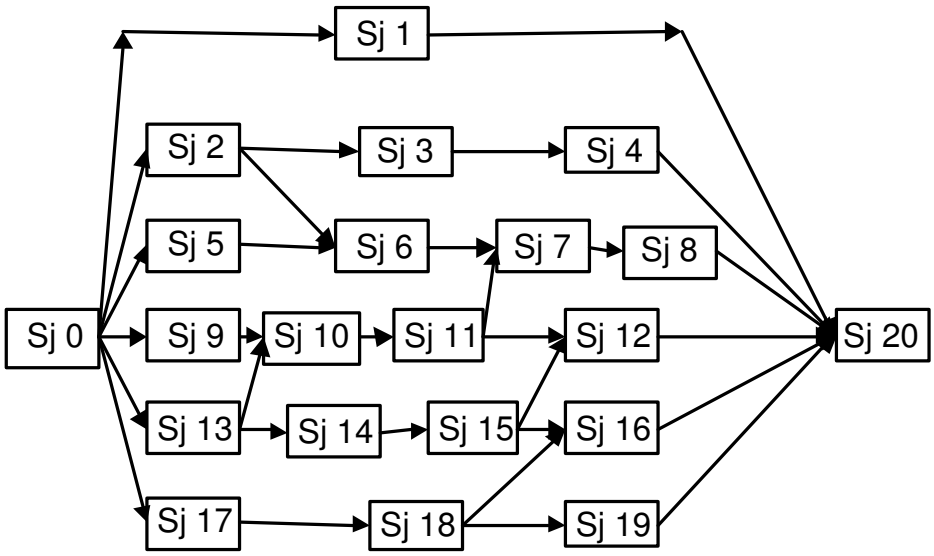

Figure 4.41: Work flow with 21 sub-jobs

With each workflow, we do mapping to 20 RMSs with different resource configurations in 9 different start times that means with 9 different resource scenarios. The total of running instances for each algorithm is 810. The quality of the solution is determined by the makespan of the workflow. The experiment results show that our algorithm needs a maximum of 13 seconds to map a workflow to the resource. Presenting all experiment data in this dissertation is impossible because of the space limitations, so we will describe here only some of the whole data. Figure 4.41 and

St	w-tb	w-dcp	grasp	minmin	maxmin	suffer
10	127	132	145	132	132	132
15	132	136	150	139	137	137
20	137	137	155	144	142	143
25	142	146	162	149	146	142
30	147	155	161	151	151	151
35	152	152	170	156	156	156
40	157	159	172	161	161	161
45	162	169	173	172	172	162
50	167	175	181	177	177	167

Table 4.8: Experiment results with workload 21 sub-jobs

table 4.8 present a sample detailed experiment data when mapping workflow with 21 sub-jobs to 20 RMSs. Each row of the table 4.8 presents the start time slot and the end time slot of each solution found by each algorithm respectively. Table 4.9 views some of the experiment result in average relative value of the solution makespan from other algorithms compared to our algorithm. Figure 4.42 depicts the overall comparison among algorithms.

From the experiment data, it can be seen that our algorithm outperforms all other algorithms in most cases. With runtime is just only few seconds, the algorithm is effective enough to be used in real systems.

4.5 Summary

This chapter has presented a method which performs an efficient and precise assignment of workflow sub-jobs to Grid resources with respect to SLAs defined deadlines and sub-job dependencies. With each workflow characteristic and Grid resource state, a different specific algorithm is used. The performance evaluation has shown that the proposed algorithm creates solution of equal or better quality than existing methods and needs an acceptable computation time. The latter is a decisive factor for the applicability of the method in real environments because large-scale workflows can be planned and assigned efficiently.

Id	w-Tb	w-dcp	grasp	minmin	maxmin	suffer
1	1.00	1.49	1.48	1.48	1.47	1.55
6	1.00	1.47	1.46	1.45	1.46	1.56
10	1.00	1.51	1.50	1.50	1.49	1.59
15	1.00	1.47	1.48	1.47	1.47	1.61
20	1.00	1.44	1.45	1.46	1.47	1.57
25	1.00	1.42	1.41	1.42	1.41	1.51
30	1.00	1.45	1.43	1.43	1.42	1.51
35	1.00	1.45	1.44	1.44	1.46	1.60
40	1.00	1.40	1.38	1.39	1.38	1.50
45	1.00	1.40	1.38	1.39	1.39	1.52
50	1.00	1.33	1.32	1.33	1.33	1.44
55	1.00	1.32	1.31	1.31	1.31	1.44
60	1.00	1.34	1.33	1.32	1.32	1.45
65	1.00	1.33	1.32	1.31	1.34	1.50
70	1.00	1.30	1.31	1.30	1.30	1.47
75	1.00	1.30	1.30	1.31	1.32	1.47
80	1.00	1.30	1.29	1.29	1.30	1.44
85	1.00	1.26	1.25	1.26	1.24	1.37
90	1.00	1.21	1.20	1.19	1.19	1.34

Table 4.9: Some experiment results in relative value

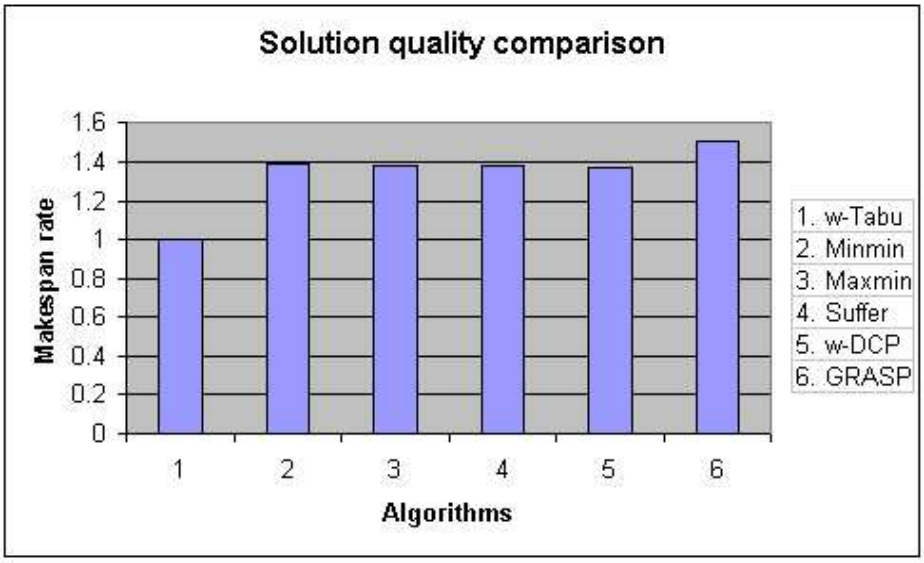

Figure 4.42: Overall quality comparison

Chapter 5

SLA negotiation protocol for Grid-based workflow

An SLA negotiation protocol includes the SLA language, the components attending to negotiation, and the negotiation procedure. This chapter will describe all these issues.

5.1 Related works

The process of forming an SLA between consumer and provider is called SLA negotiation protocol. Up to now, although with some variations, most proposed SLA negotiation protocols [27, 18, 26, 89] have the same model. To start an SLA negotiation for a job, a consumer forms an SLA template including software, hardware guarantees, computing task description as well as the expected runtime period. The filled template is sent as an offer to the provider. The provider decides whether to accept or reject the requested job. The decision depends on the ability of the provider which can satisfy the requirement such as the number of resources free at the expected period. The service provider answers the offer with a confirmation or a rejection. In current proposals, Grid jobs are defined as monolithic entities, where the user sends the input data to a service, computes the data - without dependencies – on this site and receives the results.

In [26], Czajkowski et al introduces a general negotiation model called Service Negotiation and Acquisition Protocol (SNAP), which proposes a resource management model for negotiating resources in distributed systems such as Grids. SNAP defines three types of SLAs to manage resources across different administrative domains, which are Task SLA (TSLA), Resource SLA (RSLA), and Bind SLA (BSLA). The TSLA describes the task that needs to be executed, and the RSLA describes the resources needed to accomplish this task. The BSLA provides an association between the resources from the RSLA and the application 'task' in the TSLA. These types of SLA can be used together to describe a complex service requirement in a distributed environment. The SNAP protocol is general and true for most cases. When dealing with a specific problem, a further extension must be done to realize these theories. Besides that, the SNAP protocol only focuses on a weaker form of agreement and no cost model or punish model is associated.

An effort to express detailed SLA negotiation and implementation is presented in [27]. A group of authors at the GGF Grid Resource Agreement and Allocation Protocol Working Group proposed a draft Agreement-based Grid Service Management (OGSI-Agreement) model. This document defines a set of OGSI-compatible portTypes through which management applications and services can negotiate. This agreement defines the required behavior (QoS) of a delivered service with reference to a particular service consumer. Further, the document contains an abstract management protocol to manage the agreement stages of the QoS lifecycle - from service creation until termination.

Burchard et al also propose an SLA aware architecture in which SLA negotiation for execution parameters is done among resource managers [18]. The SLA management is achieved via a Virtual Resource Manager (VRM) - that enables interaction among a number of local resource management on different clusters. The VRM acts as a coordinator to aggregate SLAs negotiated with different sub-systems. This architecture is intended to work with a group of co-located clusters, not grid wide and does not support workflow execution.

In [89], Nassif et al present an Agent-based SLA Negotiation in Grid. In the

negotiation module there are different forms of negotiation called bilateral, multi-issue and chaining negotiation. A bilateral negotiation occurs when there are only two parts involved. In a multi-issue negotiation, different aspects are negotiated such as price, quality and time schedule. The negotiation between user_agent and network_agent is also called chaining negotiation. A chaining negotiation occurs after the negotiation between user_agent and server_agent has finished.

The initial work to design and implement a system supporting SLA for Grid job is described in [4, 37]. However, all the above works only support a single Grid job, and cost model as well as SLOs (Service Level Objective) were not sufficiently discussed.

5.2 SLA language

The SLA-aware workflow in Grid environments is defined on an abstract layer by specifying the data and the control flow among software and data resources. The definition of the Grid-based workflow is independent of the hardware infrastructure, and it is up to the Grid architecture to map this workflow onto adequate Grid resources using additional metadata and to ensure its integrity in execution. The proposed language is based on the Common Job Description Markup Language [85] and can describe most aspects required for an SLA-aware workflow such as business description, computational task description, SLO description and data transfer description.

5.2.1 Business description

A typical SLA starts with a business description, which describes general information about the title of the SLA, the main aim, valid time period, customers, responsible provider, costs, penalties, etc.

5.2.2 Computational task description

The computational task gives the requirements for software, hardware, input/output, executables and all other resources needed for a successful running. The computation task description can be divided into three categories.

1. Required software components including specification of the operating system, the database system, the message passing library, and so on.

2. Required hardware components including specifications of the CPU architecture, memory/storage capacity, CPU speed, network speed and other special devices such as scanner, DVD Writer, etc.

3. Task description with specification of the input/output data, executable, environment, start parameters, etc.

An example for modeling these aspects is given in Figure 5.1. For the implementation of such an example with other languages, e.g. language proposed in [6], the integration of at least two additional language components is mandatory.

5.2.3 SLO description

SLOs focus on deadlines and the number/capacity of required resources. Traditional SLOs solely impose the responsibility of failure to provider. However, within an SLA context in the Grid environment, both provider and consumer can cause failures of a Grid job. For example, if the time to finish the computation task passes the deadline, the reason can be the failure of the resources in the provider site or it can be the underestimated processing duration of the customer. Another scenario is related to the number of requested and really used resources. If during the runtime period more resources are used than allocated, other reservations can be affected and thus the job has to be stopped. None of the available SLA languages states these issues. A sample SLO extracted from the completed SLA example in the GGF document [6] is presented in Figure 5.2. It is obvious that no reason, no responsible site, and no action information are specified.

The proposed SLO states all these problems by implementing the following structures:

- Condition: What type of problem occurred, e.g. runtime exceeded the deadline or over used resources.

```
<SectionEquation attribute="COMPUTE_TASK">
    <BooleanEquation attribute="SOFTWARE_REQUEST">
        <StringGreaterThanOrEquals>
            <StringLHS>
                <StringVariable name="Soft:Opsys"/>
            </StringLHS>
            <StringRHS>
                <StringValue>Linux1.2.4</StringValue>
            </StringRHS>
        </StringGreaterThanOrEquals>
    </BooleanEquation>
    <BooleanEquation attribute="RESOURCE_REQUEST">
        <IntegerEqual>
           <IntegerLHS>
                <IntegerVariable name="Resouce:numnode"/>
           </IntegerLHS>
           <IntegerRHS>
               <IntegerValue>51</IntegerValue>
           </IntegerRHS>
        </IntegerEqual>
    </BooleanEquation>
    <SectionEquation attribute="TASK_DESCRIPTION">
        <StringListEquation attribute="Arguments">
            <StringListValue>
                <StringValue>-a</StringValue>
                <StringValue>1024</StringValue>
                <StringValue>-p</StringValue>
                <StringValue>55</StringValue>
            </StringListValue>
        </StringListEquation>
        <StringEquation attribute="Executable">
            <StringValue>file1</StringValue>
        </StringEquation>
    </SectionEquation>
</SectionEquation>
```

Figure 5.1: SLA specifications of HW/SW resources as well as the task description

- Reason: Why the problem occurred, e.g. system failure or wrong estimation.
- Responsible site: Who is responsible for the problem, the provider or the consumer.
- Action: How the system will treat the task, e.g. cancel or continue.
- Penalty: Which penalty is to be paid by the site responsible for the problem.
- Monitor information: Which job information has to be monitored so that the consumer can track the progress or observe the circumstances of failure.

```
< wsag:GuaranteeTerm
    wsag:Name = "MaxEndTime"
    wsag:ServiceScope ="ComputerJob1 ComputerJob2" >
    < wsag:Variables >
        < wsag:Variable
        wsag:Name = "endTime" wsag:Metric = "job:endTime" >
        <wsag:Location > /wsag:AgreementOffer/
            wsag:Terms /wsag: ALL>
        </wsag:Location >
        </wsag:Variable>
    </wsag:Variables >
    <wsag:ServiceLevelObject> endTime
        IS_BEFORE 2004-05- 16T00.00.00
    </wsag:ServiceLevelObject>
    <wsag:BusinessValueList>
    < wsag:Penalty>
    <wsag:Assessmentinterval>
    <wsag:ValueExpression>5
</wsag:ValueExpression>
</wsag:Penalty> </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Figure 5.2: A GGF sample SLO in SLA language

An example for the statement "If runtime exceeds 3h because of system failure, the provider will be fined 1000 EUR per hour overdue. The provider sends an exit code to the customer when the job is finished." is given in Figure 5.3.

In fact, not every SLO needs all the above information. The number of the data fields depends on the current scenario under investigation.

5.2.4 Data transmission description

In the next step the data transmission between sub-jobs is described. The data transmission between related sub-jobs also defines the rank order between them. Therefore, each data transmission presents a directed arc in the workflow graph. The arc is described in a pair (source sub-job ID, destination sub-job ID). Additional information about data to be transferred is presented by a list of files. The corresponding example is presented in Figure 5.4. As the workflow is under DAG format, the data transmissions also describe the structure of the workflow.

The full specifications of the SLA language for Grid-based workflow are described in Appendix B.

```
<SectionEquation atribute="SLO_1">
    <StringEquation attribute="SLO_CONDITION">
        <StringValue>Runtime exceed</StringValue>
    </StringEquation>
    <StringEquation attribute="SLO_ REASON">
        <StringValue>System failure</StringValue>
    </StringEquation>
    <StringEquation attribute="SLO_ RESPON_SITE">
        <StringValue>Provider</StringValue>
    </StringEquation>
    <SectionEquation atribute="SLO_PUNISH">
        <SectionEquation attribute="MONEY">
            <RealEquation attribute="AMOUNT">
                <RealValue>1000</RealValue>
            </RealEquation>
            <StringEquation attribute="ENTITY">
                <StringValue>USD</StringValue>
            </StringEquation>
        </SectionEquation>
        <StringEquation attribute="ENTITY">
            <StringValue>Hour</StringValue>
        </StringEquation>
    </SectionEquation>
    <SectionEquation atribute="SLO_MOINITOR">
        <StringEquation attribute="WHAT">
            <StringValue>ExitCode</StringValue>
        </StringEquation>
        <StringEquation attribute="ENTITY">
            <StringValue>MB</StringValue>
        </StringEquation>
        <StringEquation attribute="WHEN">
            <StringValue>Job terminate</StringValue>
        </StringEquation>
    </SectionEquation>
</SectionEquation>
```

Figure 5.3: SLO specification

5.3 SLA negotiation for workflow

As presented in Figure 3.4, there are three main components participating to perform an SLA for the workflow. They are consumer, broker and providers. These components also join the SLA workflow negotiation process. The interaction among them in the negotiation process is depicted in Figure 5.5.

As can be seen in Figure 5.5, three different types of sub SLA negotiations using three different types of SLA text exist.

- User - Broker. This negotiation takes place by using SLA for workflow text.
- Broker - Provider. This negotiation process uses SLA for sub-job text.

```
<SectionEquation attribute="DATA_TRANSFER">
    <StringEquationattribute="ID">
        <StringValue>subjob0-subjob1</StringValue>
    </StringEquation>
    <SectionEquation attribute="DEADLINE">
        <RealEquation attribute="AMOUNT">
            <RealValue>20</RealValue>
        </RealEquation>
        <StringEquation attribute="ENTITY">
            <StringValue>min</StringValue>
        </StringEquation>
    </SectionEquation>
    <SectionEquation attribute="SOURCE_SUBJOB_ID">
        <StringEquation attribute="ID">
            <StringValue>PDB-PHYS-THEORY-0</StringValue>
        </StringEquation>
    </SectionEquation>
    <SectionEquation attribute="DEST_SUBJOB_ID">
        <StringEquation attribute="ID">
            <StringValue>PDB-PHYS-THEORY-1</StringValue>
        </StringEquation>
    </SectionEquation>
    <SectionEquation attribute="DATA_SIZE">
        <RealEquation attribute="AMOUNT">
            <RealValue>100</RealValue>
        </RealEquation>
        <StringEquation attribute="ENTITY">
            <StringValue>MB</StringValue>
        </StringEquation>
    </SectionEquation>
    <SectionEquation attribute="files">
        <SectionEquation attribute="file1.txt">
        <SectionEquation attribute="file2.txt">
    </SectionEquation>
</SectionEquation> </SectionEquation>
```

Figure 5.4: Specification of data to be transferred

- Provider - Provider. This negotiation negotiates the data transmission between sub-jobs (and also between the providers), thus the SLA for data transmission is used.

Although there are three types of SLA negotiations in the process of SLA workflow negotiation, all negotiation procedures are similar to each other. Only the service attribute is different. Figure 5.6 describes the basic procedure in client - server model.

- In the first step, the client creates a template SLA with some preliminary service attribute values and sends it to the server.

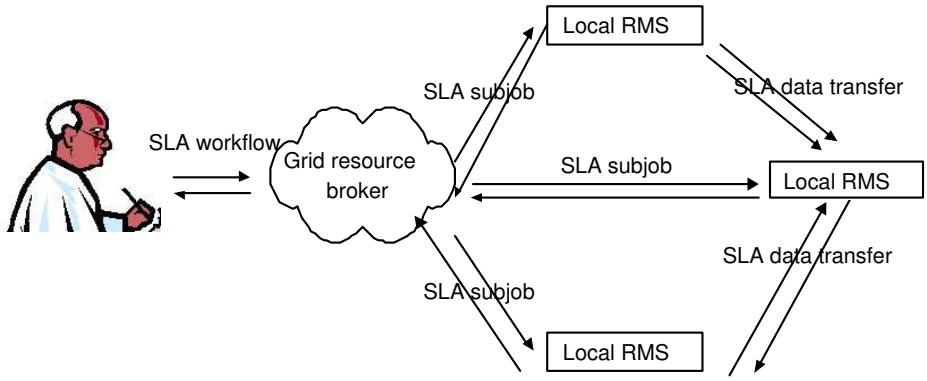

Figure 5.5: Interaction among participants in SLA negotiation process for workflow

Figure 5.6: Basic SLA negotiation procedure

- The server parses the text to get information and checks all client requirements. If there is something inappropriate, the server reforms the SLA and sends it back to the client. The difference of the new version compared to the old one lies in two aspects: modified information and additional information. Modified information can be start, stop time, cost, etc. Additional information can be SLO, FTP address, path, etc. The client checks the new SLA version and sends it back to the server. The process is repeated until there is nothing to be changed or the SLA is cancelled. That is the negotiation phase.

- When both client and server accept the content of SLA, the signing phase begins. The server signs the SLA and then sends it to the client. The client signs it

and sends it back to the server. After signing, the SLA is valid and cannot be changed.

The SLA negotiation for workflow is a complex process with the participation of many components in the system and happens in many phases. Figure 5.7 depicts the process in a time sequence.

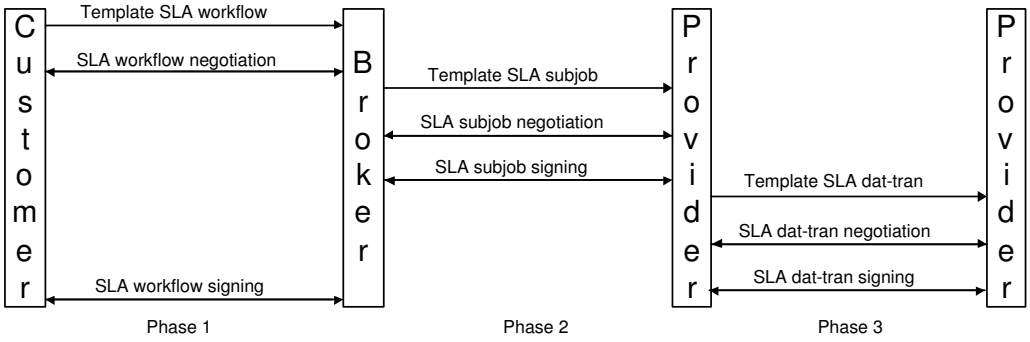

Figure 5.7: SLA negotiation process for workflow

The process starts with the customer sending a template SLA workflow to the broker. The broker finds a solution and negotiates with customer to have an acceptance SLA. After that, broker will make SLA sub-job negotiations with the providers for all sub-jobs in the workflow. The SLA data transfer negotiations can be done only when all related SLA sub-job negotiations finish. If the SLA sub-job negotiation phase and SLA data transmission negotiation phase are finished without problem, the SLA workflow will be signed. Otherwise, the broker will find another solution and phase 2 and phase 3 are repeated. If a solution cannot be found, the broker will negotiate with the client to change something in the SLA workflow content. If the client agrees, the whole process is started again. Otherwise, the SLA will be canceled.

In the three phases, creating a template SLA step and a signing SLA step are quite straightforward. The following parts will describe detail each SLA text used in the three phases and the negotiation with strong focus on the operation of each participant and the modified and additional information in the SLA text.

5.3.1 Customer - Broker negotiation

SLA text for workflow

SLA workflow is used in the negotiation between the user and the broker. It includes five main parts as depicted in Figure 5.8.

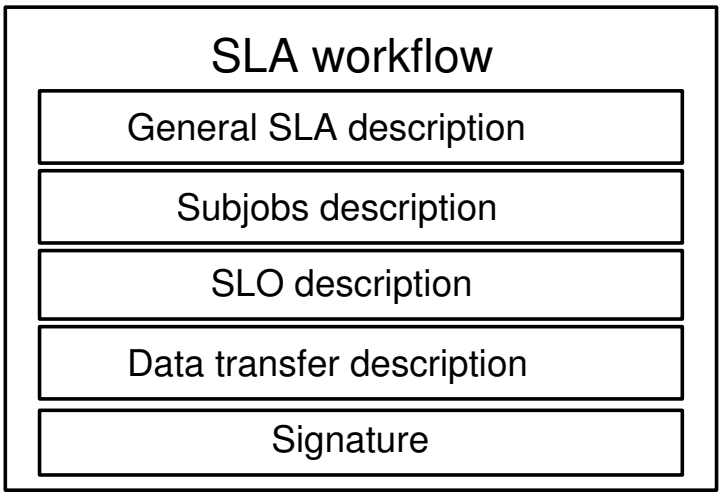

Figure 5.8: SLA workflow structure

- The general SLA part describes the start, stop time of the workflow, information about consumer and broker, cost of the SLA, etc.
- The sub-jobs part describes specification of all sub-jobs in the workflow. A more detailed description about the sub-job will be presented in the SLA sub-job section.
- The SLOs part expresses complex conditions determining boundaries over many service attributes. The SLOs focus on deadlines and the number/capacity of required resources.
- The data transmission part describes data to be transferred between sub-jobs. This information also describes the dependency structure of the workflow.
- The signature part describes the signature of two sites to ensure the legality of the SLA.

It can be said that an SLA workflow is the template for SLA subjobs as well as SLA data transfer.

SLA negotiation

When receiving an SLA from customer, the broker parses it to get all information about the general SLA, sub-jobs, SLO, data transmission, the dependency among sub-jobs and the structure of the workflow. From the information of sub-jobs and the structure of workflow, the broker does mapping to determine the appropriate provider and the time period to run each sub-job. Within the SLA context, if a user wants to use a resource he must pay the cost. The essence is that every user wants to find a feasible solution but as cheaply as possible. Therefore the global optimal destination of the algorithm is finding a feasible solution with the lowest or near lowest cost. The detail of mapping algorithm has been described in chapter 4.

The modified information can be:

- Cost of SLA. The cost of executing a Grid workflow is the sum of 4 primary factors: cost of using nodes, cost of using expert, cost of using storage, cost of transferring data. If two sequential sub-jobs run on the same RMS, the cost of transferring data from the previous sub-job to the later sub-job is zero.

- Start, stop time of the workflow. Depending on the state of the Grid, a mapping module can find a feasible solution in the expected time period or not. If not, it will find the earliest solution and ask for the consumer's approval.

The additional information can be:

- Start, stop time of each sub-job. When submitting to the broker, the consumer determines only the runtime period of each sub-job but not exactly when. After mapping, the starting time as well as the stop time are determined and announced to the consumer.

- The case of the start and stop time of each data transfer SLA is also similar to above.

- SLO for workflow. In our system, there are five common SLOs: SLO indicating that runtime is exceeding because of system failure, SLO indicating that workflow cannot run because of system failure, SLO indicating that runtime is exceeding because of a wrong estimation, SLO indicating that storage is exceeding because of a wrong estimation, SLO indicating that memory is exceeding because of a wrong estimation.

5.3.2 Broker - Provider negotiation

The SLA sub-job is used in the negotiation process between the broker and the provider to execute a sub-job in the workflow. The structure of the SLA sub-job also includes five main parts like the SLA workflow, which are general SLAs, compute task description, SLO, sub-job related data transmission and signature. Each sub-job is identified by a unique ID. The compute task description includes software request, hardware request and the task itself, which lists the execution file name, parameter, stdin, stdout, etc.

When receiving the SLA for a sub-job, the provider parses the document and checks for following information.

- The availability of its software and hardware in the required time period.
- The ability to fulfill SLOs and monitor data requirement.
- The cost to execute the sub-job.

If every thing is OK, the provider will add information about the FTP address and storage path to the part of data transmission SLA. This information can be applied only for the SLA data transmission user - RMS, and RMS - user.

5.3.3 Provider - Provider negotiation

SLA for data transmission negotiation

Returning to Figure 3.4, there are three types of data transmission activities, which are user to provider, provider to provider and provider to user. Because of this, there

are also three types of SLA data transmission. To distinguish among those SLA data transmissions, we use a field in the SLA text as control data. The structure of an SLA data transmission includes four main parts as depicted in Figure 5.9.

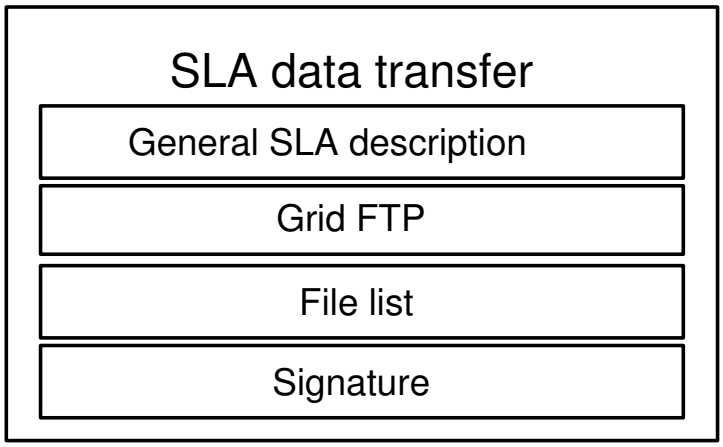

Figure 5.9: SLA data transmission structure

- Besides similar parameters in the SLA workflow, the general SLA of data transfer describes the amount of data to be transferred.
- The Grid FTP part describes the address of the FTP server and the path to the location where the data files are actually stored.
- The file list part describes a list of file names to be transferred. Those files could have a relative path and a default directory where sub-job is executed.

SLA negotiation

The provider - provider negotiation process is a little more complex. It can be done only when the two related SLA sub-job negotiations are finished. If not, the destination provider will find that the submitted SLA data transmission belongs to none of the jobs it manages and discards it.

To solve this problem, the broker will play the role of service point to synchronize the negotiation process. As the broker can control the state of all sub-job negotiations,

it will transfer appropriate data transmission SLA text to the customer and the provider. If every thing is OK, the destination provider will add information about the FTP address and storage path to the part of data transmission SLA.

5.4 Summary

The SLA negotiation process for workflows in the Grid environment is more sophisticated than the case of a single job with the participation of more components, the appearance of many SLA forms, and the more complex SLA structure. This chapter has intensively presented this process. The negotiation starts with workflow and then with each sub-job and finally with data transmission between sub-jobs. In each phase, there are different negotiating procedures, SLA text structures and negotiable data.

Chapter 6

Error recovery mechanism for Grid-based workflow

Error recovery is an important issue with a system supporting SLA for Grid-based workflow. In a large and complex system like the Grid, errors can happen at any time and in any part of the system with high frequency. The source of errors varies with network cable breakage, scratched software, hardware failure and so on. This fact makes a big risk of breaking the negotiated SLA to a system supporting SLA for Grid-based workflow. Therefore, the system must have some preparation for the bad situation in order to avoid or eliminate the bad effect. In this chapter, we concentrate on the case of catastrophic failure, when one or several RMSs are detached out of the Grid system at a particular time.

6.1 Related works

The importance of fault tolerance in Grid computing has already been recognized by the establishment of the Grid Checkpoint Recovery Working Group [115]. Its purpose is to define user-level mechanisms and grid services for fault tolerance.

Up to now, we have not noticed other works exploiting the error recovery problem for workflow, especially in the SLA context. There is, however, a considerable amount of work in related areas, especially in finding recovery methods for single Grid job.

Garbacki et al present a transparent fault tolerance for the Grid application based on Java RMI [42]. They use a globally consistent checkpoint to avoid having to restart long-running computations from scratch after a system crash.

In [61], Hwang et al present a flexible handling failure framework for the Grid. Central to the framework is the flexibility in handling failure, which is achieved by using the workflow structure as a high-level recovery policy specification.

Heine et al describe an SLA-aware job Migration in Grid environment in [55]. The checkpoint of the running job is migrated to the same cluster or another cluster running HPC4U software [56]. An architecture called VRM (Virtual Resource Management) manages and responds for the process to happen fluently.

6.2 Error recovery mechanism

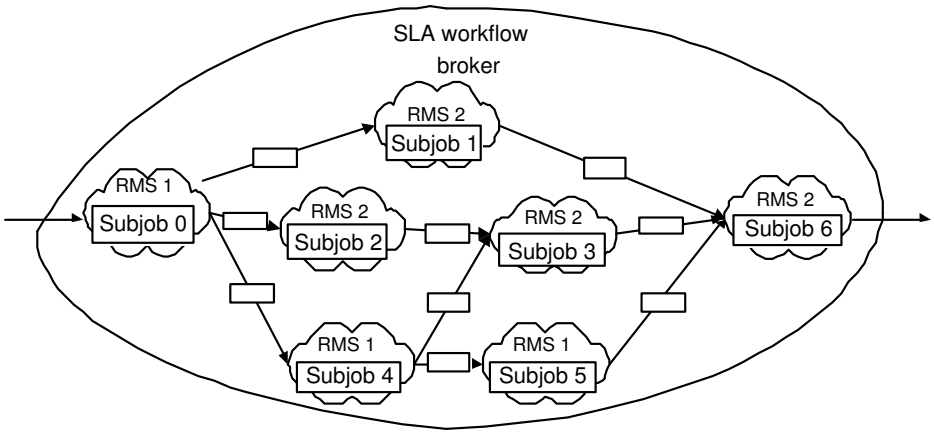

Figure 6.1: Sample running workflow scenario

In the SLA context, each sub-job of the workflow is planned to run on reserved resources within a specific time period to ensure the QoS while still preserving the integrity of the workflow. An example of such a scenario applied for the example workflow in Figure 6.1 is presented in Table 6.1 and Table 6.2. In Table 6.2, we have a note that with two dependent sub-jobs running in the same RMS, the data transfer time between them is equal to 0.

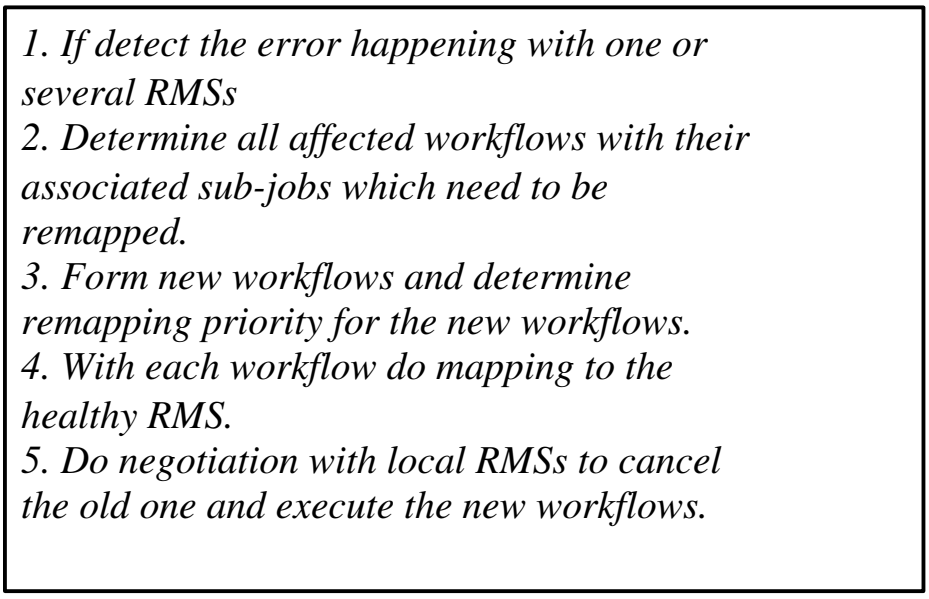

Figure 6.2: Abstract error recovery mechanism

During the running process of the workflow, one or several RMSs can be detached out of the system at a time. We propose a mechanism as described in Figure 6.2 to detect and recover the error when it happens. Steps 1 and 5, which detect the happening of the error and realizes the error recovery procedure, are described in chapter 7. In this part, we concentrate on describe the kernel of the error recovery mechanism, which includes step 2, 3 and 4.

ID	RMS	Duration
0	RMS1	0-5
1	RMS2	7-14
2	RMS2	17-21
3	RMS2	23-30
4	RMS1	7-15
5	RMS1	17-30
6	RMS2	32-35

Table 6.1: Running timetable of the sample workflow in Figure 6.1

Sj link	RMS linh	Duration
0-1	1-2	6-6
0-2	1-2	13-16
0-4	1-1	0
1-6	2-2	0
2-3	2-2	0
4-3	1-2	20-22
4-5	1-1	0
3-6	2-2	0
5-6	1-2	31-31

Table 6.2: Data transfer time table of the sample workflow in Figure 6.1

6.2.1 Determining sub-jobs to be replanned

First of all, we determine the set of affected workflows, which includes workflow having its sub-jobs running/waiting or finished but the output data still not transferred in the failed RMSs. After that, determining all affected sub-jobs in a workflow is done with the three following observations.

- Sub-jobs, which are running in the failed RMS, are directly affected. Remapping those sub-jobs will lead to the necessity of remapping all other sequential sub-jobs to ensure the integrity character. When the remapping proceeds, there is no guarantee that it will ensure the execution order for other waiting sub-jobs. Besides that, re-planning all the waiting sub-jobs in an healthy RMSs does not affect the final destination of minimizing the workflow execution makespan. Thus, we can say that all directly affected sub-jobs and all waiting sub-jobs in the affected workflow belong to the set of affected sub-jobs and need to be re-planned.

- With determined affected sub-jobs in the fail RMS, they will not have input data to run if their directly finished previous sub-jobs are also in the failed RMS. Thus, it is necessary to rerun those finished sub-jobs.

- With determined affected sub-jobs in the healthy RMSs, they will not have input data to run if their directly finished previous sub-jobs are in the failed

RMS and the related data transfer task is not finished. Those finished sub-jobs must also be rerun.

From the above observations we define an algorithm as presented in Figure 6.3.

```
foreach one in the set of affected workflow{
   foreach sub-job of the workflow {
      if have end_time > fail_slot
         put to set of affected sub-job
      if have end_time < fail_slot and next related sub-
      job waiting/running in the failing RMS
         put to set of affected sub-job
      if have end_time < fail_slot and executed in
      failing RMS and output data still not transferred
      to waiting sub-jobs in the healthy RMSs
         put to set of affected sub-job
}
```

Figure 6.3: Determining affected sub-jobs algorithm

For illustration purposes, we use the workflow sample in Figure 6.1 with the runtime parameter as in Table 6.1 and Table 6.2. Supposing that the RMS 1 fails at time slot 10, sub-job 4 is directly affected leading to sub-job 3, 5, and 6 will also be affected. When sub-job 3 is re-planned, there is no guarantee that it will be run after the finished slot of the waiting sub-job 2. Thus, we determined that all sub-jobs 2, 3, 4, 5, 6 in the affected workflow belong to the set of affected sub-jobs and need be re-planned. Because of the failure of RMS1, the output from sub-job 0 is lost and there is no data input for rerunning sub-job 4. It is necessary to add sub-job 0 to the list of affected sub-jobs.

With our example, after this phase, all affected sub-jobs of the affected workflow form a new workflow as depicted in Figure 6.4. These sub-jobs inherit all characteristics about resource and runtime requirement of the original sub-jobs.

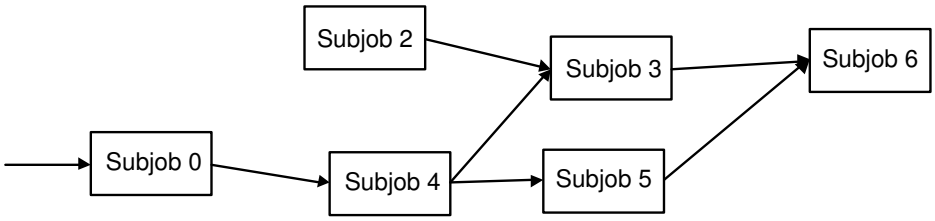

Figure 6.4: All determined affected sub-jobs in the sample workflow

6.2.2 Determining re-mapping priority

When an error happens, many workflows can be simultaneously affected and we have to re-plan many new workflows which are formed from sets of determined affected sub-jobs. One raised problem is to determine which in those newly formed workflows should be re-planned earlier. It is important because the workflow which is mapped earlier will have a lower latency. Here, we use the policy Earliest Deadline First (EDF), which is used broadly in a real time system. The workflow having an earlier deadline will be given higher priority as it occupies resources shorter and the other workflow need shorter time to wait for available resource. Thus, the total latency is reduced and the fine amount is also reduced.

To clarify the problem, we look at an example as presented in Figure 6.5. Suppose that we have 2 workflows that need to be re-mapped because of the error and the Grid system can execute one workflow at a time. Suppose that the penalty for each hour late is P. If workflow 2 is mapped first, workflow 1 has to wait until the workflow 2 is finished. Thus the minimal fine will be $P * (t2 - fail\_slot)$. If workflow 1 is mapped first, the minimal fine will be $P * (t1 - fail\_slot)$. Thus mapping workflow 1 first is better than workflow 2. In real complex situations, mapping workflow 1 first gives more chance to finish workflow 1 earlier, to release resources earlier and give more chance for workflow 2 to be mapped with smaller latency.

Based on this priority, each workflow will be mapped in sequence to the healthy RMSs. To do the mapping, we refine the new workflow under Directed Acyclic Graph (DAG) format and then use the mapping module to map this new DAG workflow to RMSs. When forming DAG for a workflow, it is necessary to consider the dependency of affected sub-jobs with running sub-jobs in healthy RMS to ensure the integrity of

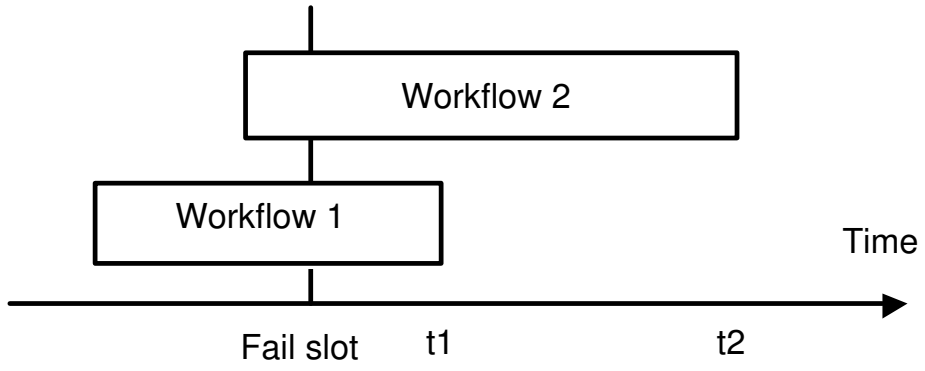

Figure 6.5: Scenario of two workflows need to be re-mapped

the workflow. To present that dependency, in the new workflow, with each running sub-job in the healthy RMSs, we create a pseudo corresponding sub-job, which is:

- Runtime equal to Deadline - fail slot - time overhead
- number of required CPU equal to 0
- number of required storage equal to 0
- number of required expert equal to 0

where time overhead value is the period to do the recovery process. Moreover, we also need a new pseudo source sub-job for the workflow with a runtime and resource requirement equal to 0.

In our example scenario, sub-job 6 must run after the finish of sub-job 1 but sub-job 1 does not appear in the set of affected sub-jobs. Thus we add a pseudo sub-job with a runtime equal to 4 and resource requirement equal to 0 to the workflow. After this step the new formed example workflow is as in Figure 6.6.

6.2.3 Mapping algorithm

As the goal of the mapping algorithm in this phase is finding out a solution, which has the makespan as small as possible. This task can be handled by the w-Tabu algorithm without changing. The detail description of w-Tabu algorithm is in chapter 4.

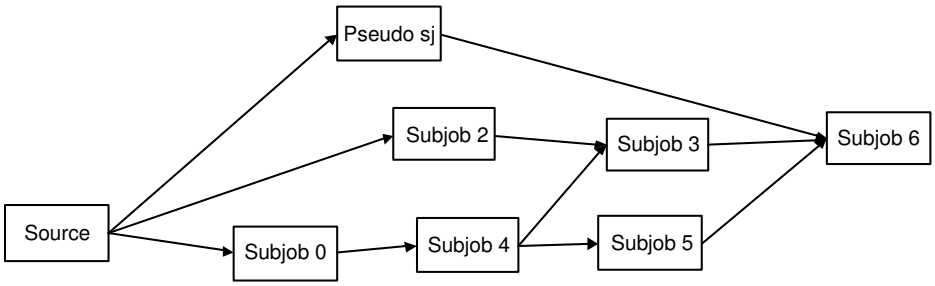

Figure 6.6: DAG form of the new workflow

6.3 Performance experiment

The performance experiment is done with simulation to check for the quality of the algorithm in part 6.3 and the overall performance of the mechanism. The hardware and software used in the experiments are rather standard and simple (Pentium 4 2,8Ghz, 1GB RAM, Linux Redhat 9.0, MySQL).

The goal of the experiment is to measure the total reaction time of the recovery mechanism in absolute value when the error happens. Determining total reaction time is important because it helps defining the earliest start time of the remap workflow, which is a necessary parameter for mapping algorithm. To do the experiment, we use 20 RMSs with different resource configuration and then we fill all the RMSs with randomly generated workflows having start time slot = 20. The number of failing RMS increases from 1 to 3 and which RMS fails is selected randomly. With each number of failing RMS, the fail slot is increased along the reservation axis. The reason for this activity is that the error can happen at any random time slot along the reservation axis. Therefore, the broader the range of experiment time is, the more correctly the reaction time value is determined. At each time, we used the described recovery mechanism to remap all affected workflow as well as all affected sub-jobs and measured runtime. The runtime is computed in seconds. The experiment results are described in Table 6.3, 6.4, 6.5.

From the result tables, it can be seen that the speed of the algorithm depends on the number of the affected workflow, the number of the affected sub-jobs in the workflow and the number of healthy RMSs. With the same number of RMSs, the

Fail-slot	Total-wf	Total-sj	runtime
50	10	156	90
60	10	143	74
70	10	138	66
80	10	135	55
90	10	117	44
100	10	109	45
110	10	98	36
120	10	89	29

Table 6.3: Experiment results with 1 failing RMS

Fail-slot	Total-wf	Total-sj	runtime
50	13	199	80
60	13	180	63
70	13	170	56
80	13	152	48
90	13	140	47
100	13	129	46
110	13	115	35
120	13	103	29

Table 6.4: Experiment results with 2 failing RMSs

total reaction time increases following the increase of total number of affected sub-jobs. In the real world situation, the number of RMSs can be increased to be a very large number but also to be very heterogeneous. Thus, the number of RMSs which have the same resource parameter is not so large. When the number of failing RMSs increases, the number of affected sub-jobs increases but the number of remaining healthy RMSs also decreases. This leads to the fact that the final result does not have considerable difference compared to the case of having 1 failing RMS. Furthermore, the probability of more than 2 failing RMSs simultaneously at a time is very rare. For these reasons, the simulation data can be dependable. With the total reaction time only just less than 2 minute compared to the hourly running workflow, the performance of the algorithm is well accepted in real situations. In the mapping algorithm, time is computed in slot, which can have resolution from 2 to 5 minutes. The reaction time of the mechanism will occupy 1 time slot and the time for system

Fail-slot	Total-wf	Total-sj	runtime
50	14	213	79
60	14	194	71
70	14	183	61
80	14	164	54
90	14	152	51
100	14	140	46
110	14	127	41
120	14	115	34

Table 6.5: Experiment results with 3 failing RMSs

to do negotiation takes about 1 time slot. Thus the start time slot of the re-mapping workflow can be assigned to the value of present time slot plus 2.

6.4 Summary

This chapter has described an error recovery method for the workflow within SLA context. We do not cover all cases of errors, which can happen to the system supporting SLA for workflow but have concentrated on a catastrophic scenario where one or several RMSs detached out of the system at a time. The main contribution of the chapter locates in the newly stated problem and the proposed mechanism to solve it. We have proposed the algorithm to detect all affected sub-jobs when the error happen and apply w-Tabu algorithm to re-map those sub-jobs to the remaining healthy RMSs with makespan optimization.

Chapter 7

System implementation

To provide the service of executing the workflow within SLA context, many described functional modules must work together within a system. As the workload and the resource in our problem have distinguishing characters, the core execution engine of the system also requires a specific working mechanism. This chapter will describe the architecture, the execution engine, and the initial deployment of the system.

7.1 Introduction

The theory will have little impact if it cannot be realized to solve a problem. All described theory topics above will have little meaning if they cannot work in a real system. The constraint in realizing the system is that all modules must be combined in a way that they can co-operate with each other to perform the ultimate function: executing the workflow within SLA context.

In the system, the execution module does the important work, which is executing the workflow. A prerequisite for a planning-based scheduling needed for the SLA-aware execution is that the maximal runtime of the job is estimated and a-priori known. In case of an underestimated runtime, the sub-job will be stopped before completion and thus will affect the entire workflow. Furthermore, the mapping algorithm considers all RMSs with resource configuration at least equal to or better than the requirements. If the computing task runs on more powerful resources, it will need

shorter time to run than estimated. For example, the task requires 15 time slots to run on 8 CPU 1GHz. If it was run on 8 CPU 1,7 GHz, it would require 13 time slots to finished. This fact will make inefficient usage of resources in RMS. We can look more concretely at a scenario with a workflow being mapped to 2 RMSs as in table 7.1. If the real runtime of each sub-job = 0.8 estimate time, the resource usage in RMS 2 is fragmented as presented in Figure 7.1. With just several narrow slots, it is difficult to insert a job and those slots are wasted.

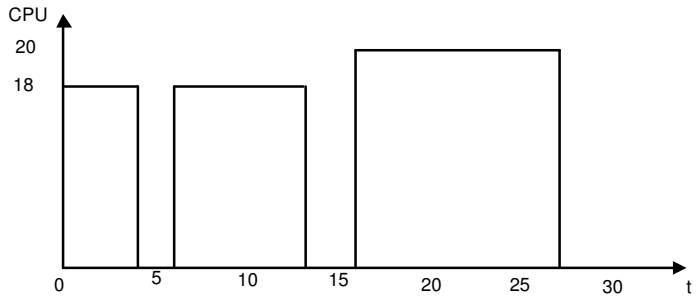

Figure 7.1: CPU usage profile in RMS 2 if have no adjustment

ID	Nr CPU	RMS	Duration
0	18	RMS1	0-5
1	16	RMS2	7-14
2	20	RMS2	17-21
3	22	RMS2	23-30
4	18	RMS1	7-15
5	20	RMS1	17-30
6	16	RMS2	32-35

Table 7.1: Initial running time table of the sample workflow

Moreover, a distributed engine for handling complex workflow models within the SLA context imposes several constraints compared to execution of single jobs as well as compared to workflows not bound to SLAs. In particular the communication and cooperation as well as the synchronization between the components of the system have to be engineered in a suitable way in order to ensure the integrity of the exchanged data and the workflow itself.

The following parts will describe the implementation structure of the system in which the interaction among many modules will be presented. The detail about adaptive execution mechanism will be more concentrated.

7.2 Related works

In several research projects related to Grid-based workflow execution [30, 87], the execution engine is based on Condor's DAGman [79] or [36]. In those systems, the user defines the sub-job as well as the dependency among them and then submits the job to the execution engine. The engine executes the workflow step-by-step and returns the results to the user. In the system architecture, Condor's DAGman and Unicore act as a meta-scheduler. However, both systems are based on a queuing model of resource allocation so they do not support the reservation to ensure the desired QoS levels and the fulfillment of the SLAs.

Wesner et al. [124] describe the system GRASP, which supports certain QoS for the workflow execution at a broker level. The execution engine in GRASP is based on the BizTalk Server 2004 working primarily with Web Services. Therefore, it lacks the ability of supporting Grid services. To overcome that problem, GRASP used the support of reliable Web Services which acts as wrapper for the real service implementation. After mapping the so-called pre-SLA to the composite services with the SLA of the single Grid services involved in the orchestration, the execution engine discovers at runtime the best hosting environment for the single services with the expected SLA. There is no resource reservation so the system is not able to predict the QoS of the composite service.

Afzal et al. present a distributed execution architecture which can handle the execution of a workflow [2]. The Launching Service is a framework service providing a generic interface to different DRM systems. The Launching Service may represent a single resource (in the case of a ShellScript-Launcher) or an entire cluster of computers (SGE launcher). Jobs are submitted to the Launching Service as a JDML document and a set of user credentials (through the launchJob port). The Launching Service and the attached launcher process the JDML document and deploy the work as

appropriate. When a Launching Service is started it advertises its existence to the Grid. However, in this work the execution engine work primarily with the non-reservation model and the support for resource reservation is still work in progress.

Recent works [86, 130, 16] also built systems to support QoS features for Grid-based workflow. In their work, a workflow includes many sub-jobs, which are sequential programs, and a Grid service has ability to handle one sub-job at a time. The resource is also reserved to ensure the QoS. However, in those works, an adaptive execution engine does not appear.

7.3 System implementation

An implemented prototype fulfils the above constraints and provides basic features which allow any client to negotiate SLAs, monitor running workflows and to receive the result. Based on standard components such as Globus Toolkit 3.2, MySQL database, Maui ME for a local RMS, the system is compatible with the existing Grid infrastructure. The overall architecture schema is depicted in Figure 7.2.

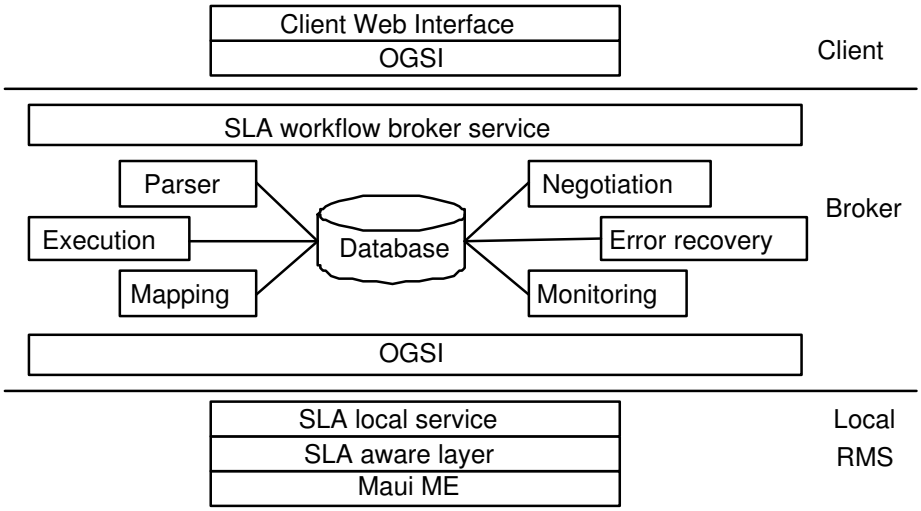

Figure 7.2: System implementation layers

The system includes three main components: the client plays the consumer role, the SLA-aware workflow broker, and the SLA-aware local RMS act as provider.

7.3.1 SLA-aware RMS

The local RMS provides SLA services for each individual site on the Grid. This service follows the OGSA principles and is based on the latest version of Globus Toolkit. When deployed, this service is one among many services that are provided by the Globus Toolkit. The local RMS must be SLA aware, as resource reservations are necessary. Furthermore, standard features such as resource monitoring and fault recovery are mandatory. However, most of the existing RMSs do not support these features, therefore we provide with MauiME an SLA-aware layer on top of the RMS. The structure of the local RMS is depicted in Figure 7.3.

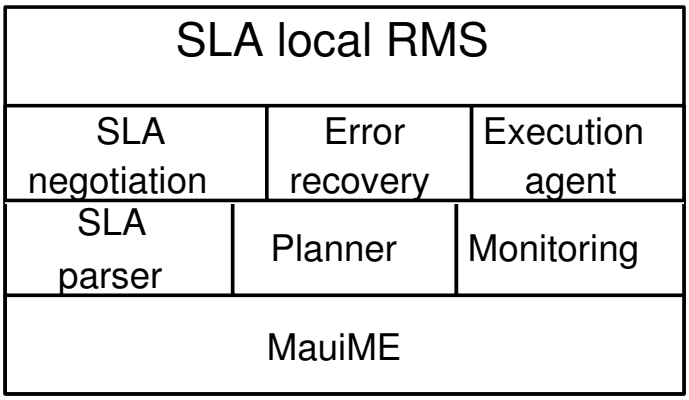

Figure 7.3: SLA-aware local RMS architecture

- Planner. MauiME supports solely node reservations. So we did not use this feature of MauiME but developed a planner which can support reservations for nodes, storage, and experts.
- Monitoring. This module monitors the state of jobs/nodes/resources. This information is used to detect failures or QoS violations.
- SLA parser. This module is written in Java to parse or create SLA text using the SLA language for workflows.
- SLA negotiation. This module uses the SLA parser and the module planner to check the feasibility of the SLA. The planner output is used to negotiate the

SLA with other participants.

- Error recovery. This module is responsible for running jobs even if some nodes have failed. The monitoring information delivers reports on possible failures. The error recovery module cancels the specific job, uses the planner to allocate new resources and re-executes the job from the checkpoint image.

- Execution agent. This module is responsible for executing sub-jobs and is described in depth in section 7.4.

7.3.2 SLA workflow broker

The SLA workflow broker is the central system unit and responsible for processing the client requirements and for dispatching sub-jobs as well as for the SLA negotiation between clients and RMS. The SLA workflow broker provides services to clients and uses services from the local RMS. The communication (client – broker and broker – local RMS) is done by the OGSI platform. As can be seen in Figure 7.3, the SLA workflow broker includes many modules, some of which were described in previous chapters. For co-operation among these modules, the SLA broker uses a database to manage all aspect of operation. The co-operation among modules are described as follows.

Performing SLA negotiation

The SLA broker receives an SLA requirement from the client and parses it to get all sub-jobs information. The information is stored in three files: General SLA description, sub-jobs description, and arcs of the workflow description. Based on these files and the data of the local RMS (resource description, reservation in database), the SLA broker invokes the mapping algorithm. If a feasible solution is found it returns the solution which includes the sub-job ID with its associate RMS ID and the starting time slot. The module SLA negotiation uses this information. If the client accepts the solution from the previous step, the SLA broker will invoke an insertion module to enter all necessary information into the database. Subsequently, the next module

the SLA local service client module negotiates with the local RMS, collects all GFTP handle services and returns the SLA flow ID to the client together with all GFTP handle services.

Performing error recovery

Error detection is done with the monitoring module. The monitoring module collects information about the RMS state, the RMS resource, the RMS reservation, the sub-jobs state and so on from all RMSs. This information will be analyzed and stored in the central database to ensure that the broker module could have an overall image of the system. With the push model, after a period of time, local RMSs will connect with the broker and push monitor information. If the broker does not receive information from a local RMS it will consider that RMS as failed and activate the error recovery module. When error recovery module is activated, it will do the following actions in a strict sequence:

- Accessing database to retrieve information about failure RMSs and determine affected workflow as well as necessary sub-jobs of the workflow to be remapped.
- Based on determined information about affected workflows and sub-jobs, activating the negotiation module to cancel all SLA sub-jobs with local RMSs related to specific sub-jobs. All negotiation activities are done with the help of the SLA text as the mean of communication.
- Activating the monitoring module to update the newest information about the RMS, especially information about resource reservation.
- Calling the mapping module to determine where and when sub-jobs in the affected workflow will be run.
- Based on mapping information, activating the negotiation module to sign a new SLA for each sub-job with the specific local RSM.
- Updating workflow control information and sub-jobs information in the central database.

7.3.3 Client

The client is implemented in Java and provides a Grid service interface. The SLA text is compiled in a file and transferred to the SLA broker through the OGSI infrastructure. In the negotiation period, the difference between submitted and received SLA text is detected and presented to the user. The client component allows the user to supervise the performance of the system, periodically or randomly.

7.4 Adaptive execution engine

The adaptive SLA execution engine contains the adaptive runtime control mechanism and is implemented/integrated in the framework for SLA-aware workflows.

7.4.1 Adaptive runtime control mechanism

The main idea is based on shifting forward the sequential sub-jobs of the workflow a period equal or near equal to the spare time of the finished sub-job. The procedure start when the finished state of a sub-job in the workflow is received.

In the first step a GO/No-GO decision is required. For this purpose the description of each sub-job in the Grid workflow including the main components such as input data, computing task and output data as well as the source, and destination for the data transfer are considered. Thus, there is only one computing task, but many data transfer tasks for each sub-job. Based on the estimated runtime, a start/stop time of the workflow is computed serving as a basis for resource allocation. A typical time sequence of sub-job tasks is depicted in Figure 7.4.

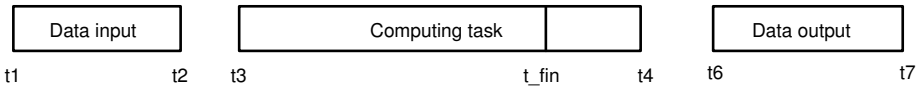

Figure 7.4: Time sequence of a sub-job

The shifting occurs only when the finished time t_fin of the sub-job is earlier than the estimated ending time $t4$. In this case, the set of sub-jobs to be shifted is

determined. The sub-job candidate for shifting depends directly on the finished job. For example, when sub-job 4 finishes early, sub-job 3 and 5 will be considered to be shifted. However, not all of them can be moved. Only a sub-job without dependencies to any running/waiting sub-job, is able to shift while preserving the workflow integrity. In our example, at time slot 13, sub-job 3 still depends on the running of sub-job 2 and it cannot be adjusted at this point. Thus, the shifting candidate set contains only sub-job 5. A further shifting of other indirect depended sub-job such as sub-job 6 will not be performed because the problem complexity increases and the success chances drop rapidly.

After determining the set of sub-job to be adjusted, the shifting time-period has to be computed for each job. Let sj_s be the sub-job to be shifted and sj_f the finished sub-job. In Figure 3 we can see the maximum shifting period is $sj\_f.t4 - sj\_f.t\_fin$. The new start time of sj_s as well as the shifting period is determined by checking the bandwidth reservation profile and the resource reservation profile of the RMS. If the new start time is earlier than the old value, the shifting procedure is started. After determining the adjusted sub-job and time period, the modified runtime requirements are sent to the RMS in order to adapt the planned runtime of the task. Subsequently, the modified transfer time is sent to the RMS which finished the sub-job earlier. From that control information, the RMS does the rest to finish the task. After applying the adjustment mechanism, the resource usage profile is as in Figure 7.5.

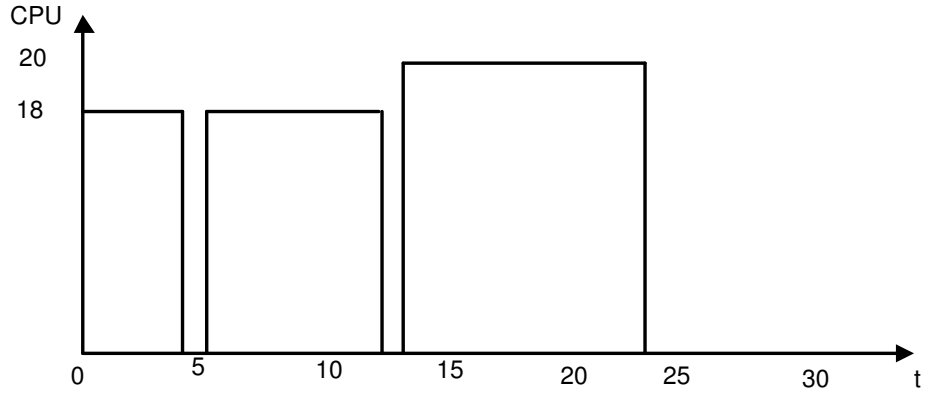

Figure 7.5: CPU usage profile of RMS 2 after doing the adjustment

This procedure allows more efficient use of the RMS resources and increases the reliability of the system in executing Grid workflows within an SLA context. In case of resource failure, the finish time of computing task cannot meet the deadline and the service provider will violate the SLA. Shifting sub-jobs to run earlier will increase the spare time, which can be used for implementation fault tolerance measures such as checkpointing and migration to other resources.

7.4.2 Adaptive SLA execution engine

The SLA execution engine handles the execution task. Because sub-jobs of the workflow are distributed over multiple RMSs, ensuring the complete execution of the workflow requires a distributed model. The execution engine includes a management module and several agents modules located in the involved RMSs. The module execution manager controls the work of all execution agents to ensure the integrity and the completeness of the workflow. A module execution agent supervises the execution of the sub-jobs directly. The following sections describe the functionality of those modules in detail.

Execution agent

The execution agent is responsible for computing the task. All tasks are stored in a queue which is analyzed by the execution agent in regular intervals in order to find the task which can be run next and execute it. There are two types of tasks, each with different execution procedures. The data transfer task moves the data output from the completed computing task of this sub-job to another sub-job. The execution engine checks for the completeness of the associated computing task and the existence of necessary data file. Then it uses the FTP protocol to transfer the file to destination.

To execute a computing task, the execution agent checks if all necessary input files are already in the appropriate directory and then defines the submission file and submits the task to the RMS. After submission, the execution agent investigates the state of the computing task until it is finished. During the runtime, the state of the computing task is periodically sent to the execution manager. The computing task

can be canceled by the local RMS if it uses more time or more resource than allowed. In this case, the execution agent will remove all related data transfer tasks from the queue and inform the execution manager. Otherwise the computing task might be finished before the estimated deadline because of overestimated runtime or because of increased computing power compared to the specified resources in the SLA. In this situation, the execution agent waits for instruction from execution manager.

Workflow execution management

Both states – canceled and finished earlier – affect the entire workflow. The work of the execution management module reacts to the state in a suitable way to ensure the integrity of the workflow. A canceled sub-job leads to the termination of the whole workflow as the data dependencies cannot be resolved for the remaining workflow part. In this case, the execution manager announces a cancel request to all running and waiting sub-jobs.

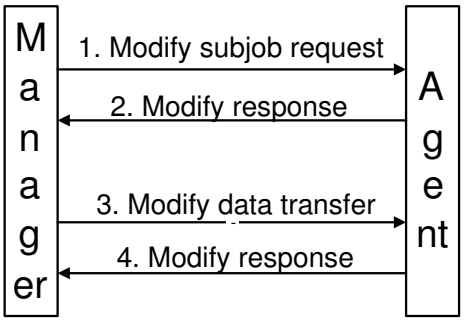

Figure 7.6: Negotiation procedure to shift sub-job

When one computing task finishes earlier than estimated, the execution agent uses the shifting procedure as described above. The communication procedure to realize the task is presented in Figure 7.6. The execution agent parses the request to get necessary information about the sub-job and then backs up and removes all related data of the sub-job. Subsequently, the agent plans the sub-job scheduling with the new information; if the planning is successful, it inserts the sub-job into the queue. The result is sent to the execution manager which computes the new data transfer

time of the sub-job which finished earlier. Finally, it sends the modified data transfer request to the execution agent.

7.4.3 Performance measurements

The performance measurements are based on extensive simulations of Grid-based workflows and aim at evaluating the quality and efficiency of the adaptive runtime adjustment. The simulation is done in several Grid resource configurations involving varying number of RMSs and different resource configurations. The Grid-based workflows are generated randomly and mapped by the mapping algorithm. Thereafter, at each time slot in each RMS, we generate randomly a sub-job and compute a scheduling for the RMS with and without activation of the runtime adjustment method. The amount of successfully mapped sub-jobs and the achieved workload is depicted in Table 7.2.

	Total nr sub-jobs mapped		
Nr RMS	Adjust	Non-adjust	Workload inc rate
5	483	468	2.3%
8	695	672	4.6%
10	934	896	3.7%
15	1420	1253	5.1%
20	1876	1578	4.3%

Table 7.2: Simulation results

Within the SLA context, increasing workload rate 5% means that the income of the RMS also increases 5%, a worth of considerable value in business.

7.5 System deployment

The prototype is deployed in three cluster systems running Linux, LAM-MPI 7.1, MauiME with different resource configurations as described in Table 7.3. On the front-node of each cluster, Globus Toolkit 3.2 and the module SLA-aware local RMS service are installed. One separate machine is used to run the SLA workflow broker.

On this system, we have experimented with several state of the art workflows. The common experiment scenario is presented in Figure 7.7 with a workflow is executed over many RMSs.

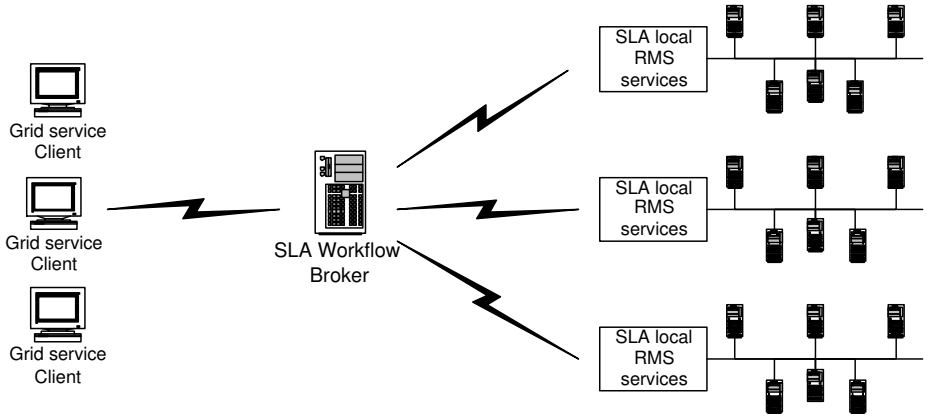

Figure 7.7: Experiment system deployment

ID	Nodes	Storage	Expert
RMS1	7	200	2
RMS2	14	100	1
RMS3	9	300	2

Table 7.3: Resource configuration of RMSs

The online demonstration of the running process, which includes the mapping, negotiation process, state monitoring as well as the execution, of a workflow consisting of seven sub-jobs in cooperation of three local RMSs, can be found at http://pc-kao3.upb.de:9035/manual/test.html. Figure 7.8 depicts the Web-based user interface in the running process.

7.6 Summary

This chapter has presented an architect for definition and implementation of SLA-aware workflows in Grid environments consisting of several building blocks. The global architecture includes components for SLA definition and negotiation, task

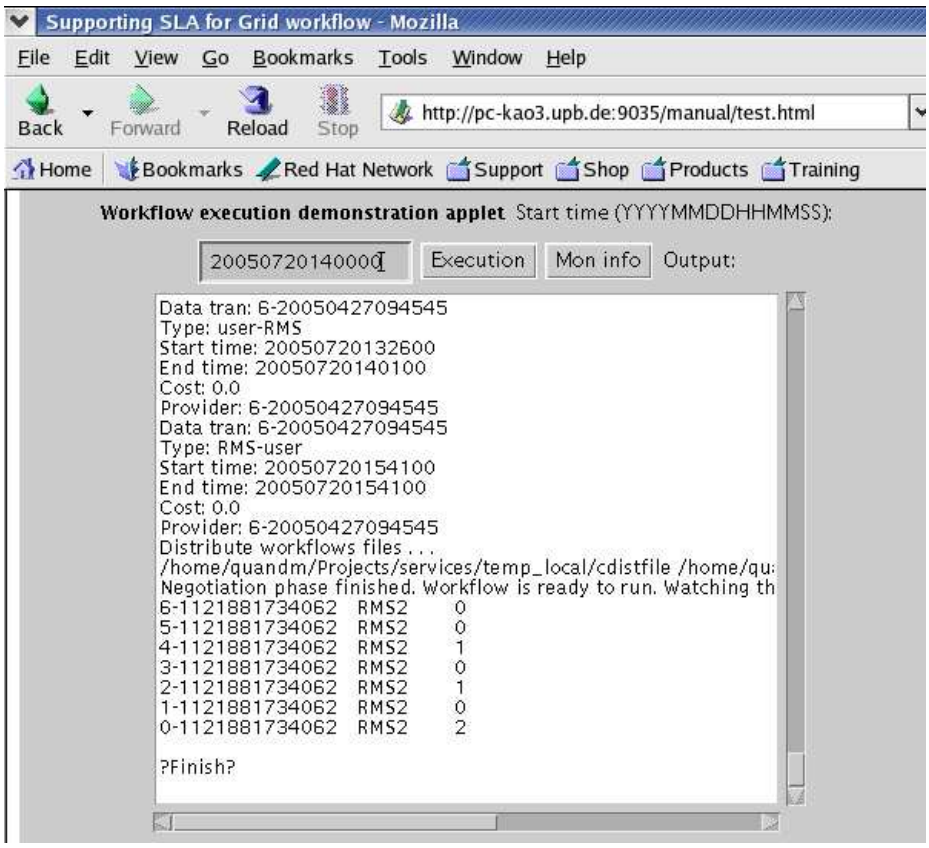

Figure 7.8: Initial web based client

mapping, monitoring, and fault reaction. The adaptive runtime control mechanism is included in a full-operational engine within a framework for support of SLA-aware Grid workflows. With adaptive runtime control mechanisms, Grid resources are used more efficiently and the broker itself has more time to activate and use fault tolerance methods in case of failure. Performance measurements with the implemented prototype and simulated workflows showed a performance increase of between 2% and 5%. All components are implemented using standard Grid and RMS components and provided as online demonstration.

Chapter 8

Conclusions and future work

8.1 Conclusions

The problem stated in Chapter 3 has been solved: as shown in Chapter from 4 to 7, a basic system support SLA for scientific workflow has been developed. The heart of the system is the mapping mechanism, which includes three sub-algorithms, to map sub-jobs of the workflow to Grid resources in an efficient way. The SLA negotiation protocol provides an SLA language and negotiation procedure to help many components in the system to achieve common agreement in providing service. The error recovery mechanism concentrates on the catastrophic scenario where one or several RMSs are detached out of the system at a time. Finally, a prototype system, which combines all theory mechanisms in a unified organization, provides primary functions to execute scientific workflow within SLA context. The work in this thesis contributes to the literature knowledge in several aspects as we have:

- Raised a new issue in Grid computing, supporting SLA for workflow on the Grid environment. The workflow with several dependent parallel sub-jobs running on reserved Grid resources in the scope of a business contract defines a new problem which requires new techniques to be solved properly.
- Developed a new mapping mechanism to map sub-jobs of the workflow to Grid resources. The mapping mechanism includes three effective sub-algorithms to

cope with three circumstance of the mapping scenario. The L-Tabu algorithm finds the low cost solution for workflows which have little data to be transferred among sub-jobs. The H-Map finds the low cost solution for workflows having a lot of data to be transferred among sub-jobs. The W-Tabu algorithm finds the low makespan solution for workflows. The effectiveness of those algorithms was proved by several simulation experiments.

- Proposed a new SLA negotiation protocol for workflow. The SLA language clarifies the Service Level Objective and the workflow structure. The negotiation procedure set off the role as well as the information to be negotiated of participants in the system.

- Designed a new error recovery mechanism. The mechanism concentrates on solving the case of unstated catastrophic scenario where one or several RMSs are detached out of the system at a time.

- Demonstrated the first real system supporting SLA for workflow.

8.2 Future work

Future work can extend the results of this thesis in two main ways.

- The first is called deep extension approach, which adds more features to the existing skeleton system. For example, as running a workflow requires the complex co-operation of many sites on the Grid, ensuring the truthful and safe communication among them is an important unconsidered issue. Another problem is related to financial aspect of an SLA system. The relation between user and provider in the system is business relation, which concerns with money payment. Thus, having a mechanism supporting this activity to happen fluently is a considerable requirement.

- The second is called wide extension approach. As can be seen in the introduction, supporting SLA for workflow in the scavenging Grid and data Grid

are still open issues. With different resource characteristics compared to the computation Grid, realizing the destination in scavenging Grid and data Grid requires many unexploited adaptations and methods.

Appendix A

List of Acronyms

AEFT	Absolute Earliest Finished Time
ALFT	Absolute Latest Finished Time
APN	Arbitrary Processor Network
BNP	Bounded Number of Processors
BPEL	Business Process Execution Language
BSLA	Bind Service Level Agreement
CCS	Computer Center System
CP	Critical Path
DAG	Directed Acyclic Graph
DRM	Distributed Resource Managers
EDA	Estimation of Distribution Algorithm
EP	Execution Plan
FJSSP	Flexible Job Shop Scheduling Problem
FTP	File Transfer Protocol
GA	Genetic Algorithm
GB	Giga Byte
GEMSS	Grid-Enabled Medical Simulation Services
GGF	Global Grid Forum
GLS	Guided Local Search
GRASP	Grid-based Application Service Provision
HPC4U	High Predictable Clusters for Internet Grids

HPCC	High Performance Computing Center
ICENI	Imperial College e-Science Network Infrastructure
ILS	Iterated Local Search
IP	Integer Programming
JDML	Job Description Markup Language
JSSP	Job Shop Scheduling Problem
KB	Kilobyte
LAM-MPI	Local Area Multicomputer - Message Passing Interface
MB	Mega Byte
Mbps	Megabit per second
MCT	Minimum Completion Time
OGSA	Open Grid Services Architecture
OGSI	Open Grid System Interconnection
PBS	Portable Batch System
PDDL	Planning Data Description Language
QoS	Quality of Service
QoWL	QoS-aware Grid Workflow Language
QWE	QoS-aware Grid Workflow Engine
RAM	Random Access Memory
RMI	Remote Method Invocation
RMS	Resource Management System
RSLA	Resource Service Level Agreement
SA	Simulated Annealing
SGE	Sun Grid Engine
SLA	Service Level Agreement
SLO	Service Level Objective
SNAP	Service Negotiation and Acquisition Protocol
SQL	Structure Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TDB	Task Duplication Based

TSLA	Task Service Level Agreement
WfMC	Workflow Management Coalition
WSDL	Web Service Definition Language
XML	Extensible Markup Language
UDDI	Universal Description, Discovery and Integration
UMDA	Univariate Marginal Distribution Algorithm
UNC	Unbounded Number of Cluster
VGE	Vienna Grid Environment
VO	Virtual Organization
VRM	Virtual Resource Manager

Appendix B

SLA language for Grid-based workflow specification

B.1 Common tag

B.1.1 Identification

It describes the specific ID string for each SLA workflow, SLA subjob, SLA data transferring, etc.

Element Name : ID

Element Type : StringEquation

Example :

```
<StringEquation attribute="ID">
    <StringValue>28-6-2004-PC2-PADERBORN-PHYSIC-THEORY</StringValue>
</StringEquation>
```

B.1.2 Title SLA

This tag describes the name of the SLA.

Element Name : TITLE_SLA

Element Type : StringEquation

Example :

```
<StringEquation attribute=" TITLE_SLA">
    <StringValue>SLA - Running CFD simulation job</StringValue>
</StringEquation>
```

B.1.3 Description

This tag describes details about SLA, computation task, etc.

Element Name : DESC

Element Type : StringEquation

Example :

```
<StringEquation attribute=" DESC">
    <StringValue> Running CFD to find some information about the affection
    of fluid to auto running condition </StringValue>
</StringEquation>
```

B.1.4 Amount

This tag describes the amount number.

Element Name : AMOUNT

Element Type : RealEquation

Example :

```
<RealEquation attribute="AMOUNT">
    <RealValue>10000</RealValue>
</RealEquation>
```

B.1.5 Entity

This tag describes the measurement entity.

Element Name : ENTITY

Element Type : StringEquation

Example :

```
<StringEquation attribute="ENTITY">
    <StringValue>USD</StringValue>
</StringEquation>
```

B.1.6 Cost SLA

This tag describes the cost.

Element Name : COST_SLA
Element Type : SectionEquation
Required Element : Amount,Entity
Example :

```
<SectionEquation attribute="COST_SLA">
    <RealEquation attribute="AMOUNT">
        <RealValue>10000</RealValue>
    </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>Euro</StringValue>
    </StringEquation>
</SectionEquation>
```

B.2 General SLA description

This tag contains a general SLA description.

Element Name : GENERAL_SLA
Element Type : SectionEquation
Required Element : Title, Description, Start time, End time
: Provider, Consumer, Cost

B.2.1 Start time

This tag describes the starting time of an SLA. The time is described under format YYYYMMDDHHMMSS.

Element Name : STARTTIME_SLA

Element Type : StringEquation

Example :

```
<StringEquation attribute="STARTTIME_SLA">
    <StringValue>20050425083000</StringValue>
</StringEquation>
```

B.2.2 End time

This tag describes the ending time of an SLA. The time is described under format YYYYMMDDHHMMSS.

Element Name : ENDTIME_SLA

Element Type : StringEquation

Example :

```
<StringEquation attribute="ENDTIME_SLA">
    <StringValue>20050428083000</StringValue>
</StringEquation>
```

B.2.3 Provider

This tag describes the name of the service provider.

Element Name : PROVIDER

Element Type : StringEquation

Example :

```
<StringEquation attribute="PROVIDER">
    <StringValue>PC2 -Uni Paderborn</StringValue>
</StringEquation>
```

B.2.4 Consumer

This tag describes the name of the consumer.

Element Name : CONSUMMER_SLA
Element Type : StringEquation
Example :

```
<StringEquation attribute="CONSUMMER_SLA">
    <StringValue>Physic department - Uni Paderborn</StringValue>
</StringEquation>
```

B.3 SLA for data transmission description

This tag contains a SLA description for data transmission.
Element Name : GENERAL_SLA_TRAN
Element Type : SectionEquation
Required Element : Title, Description, Start time, End time
: Provider, Consumer, Data, Cost

B.3.1 Data

This tag describes the amount of data to be transferred in SLA data transmission.
Element Name : DATA
Element Type : SectionEquation
Example :

```
<SectionEquation attribute="DATA">
    <RealEquation attribute="AMOUNT">
        <RealValue>100</RealValue>
    </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>MB</StringValue>
    </StringEquation>
</SectionEquation>
```

B.4 Computing task description

This tag contains the computation task description of a sub-job.

Element Name : COMPUTE_TASK
Element Type : SectionEquation
Required Element : Description, Software request, Resource request, Job description
Example :

B.4.1 Software request

This tag describes the software requirement.

Element Name : SOFTWARE_REQUEST
Element Type : SectionEquation
Required Element : os, database, message lib, other
Example :

```
<SectionEquation attribute="SOFTWARE_REQUEST">
    <StringGreaterThanOrEquals attribute="os">
        <StringValue>Linux</StringValue>
    </StringGreaterThanOrEquals>
    <StringGreaterThanOrEquals attribute="database">
        <StringValue>MySQL</StringValue>
    </StringGreaterThanOrEquals>
    <StringGreaterThanOrEquals attribute="meslib">
        <StringValue>LAM</StringValue>
    </StringGreaterThanOrEquals>
</SectionEquation>
```

B.4.2 Resource request

This tag describe the resource requirement.

Element Name : RESOURCE_REQUEST
Element Type : SectionEquation
Required Element : Number of cpu, CPU speed, Architecture, Memory, Storage, Expert
Example :

Number of cpu

```
<IntegerEquation attribute="numnode">
    <IntegerValue>4</IntegerValue>
</IntegerEquation>
```

CPU speed

```
SectionEquation attribute="CPUspeed">
    <RealEquation attribute="AMOUNT">
        <RealValue>1000</RealValue>
    </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>Mhz</StringValue>
    </StringEquation>
</SectionEquation>
```

Architecture

```
<StringEquation attribute="arch">
    <StringValue>x86</StringValue>
</StringEquation>
```

Memory

```
<SectionEquation attribute="mem">
    <RealEquation attribute="AMOUNT">
        <RealValue>256</RealValue>
    </RealEquation>
```

```
    <StringEquation attribute="ENTITY">
        <StringValue>MB</StringValue>
    </StringEquation>
</SectionEquation>
```

Storage

```
<SectionEquation attribute="storage">
    <RealEquation attribute="AMOUNT">
        <RealValue>10</RealValue>
    </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>MB</StringValue>
    </StringEquation>
</SectionEquation>
```

Expert

```
<SectionEquation attribute="expert">
    <RealEquation attribute="AMOUNT">
        <RealValue>1</RealValue> </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>Person</StringValue>
    </StringEquation>
</SectionEquation>
```

B.4.3 Job description

This tag contains the computing job description.

Element Name : TASK_DESCRIPTION

Element Type : SectionEquation

Required Element : Worker, Arguments, Executable, StdinFile, StdoutFile
: StdlogFile, CheckpointDir

Worker

This tag describes which worker will execute the job, for example MPI, PVM, shell.

Element Name : Worker

Element Type : StringEquation

Arguments

This tag describes the arguments which need to be passed to the executable file when it is started.

Element Name : Arguments

Element Type : StringListEquation

Example :

```
<StringListEquation attribute="Arguments">
    <StringListValue>
        <StringValue>-a</StringValue>
        <StringValue>1024</StringValue>
        <StringValue>-p</StringValue>
        <StringValue>55</StringValue>
    </StringListValue>
</StringListEquation>
```

Executable

This tag describes the executable file name.

Element Name : Executable

Element Type : StringEquation

StdinFile

This tag describes the Stdin file name.

Element Name : StdinFile

Element Type : StringEquation

StdoutFile

This tag describes the Stdout file name.
Element Name : StdoutFile
Element Type : StringEquation

StdlogFile

This tag describes the Stdlog file name.
Element Name : StdlogFile
Element Type : StringEquation

CheckpointDir

This tag describes the checkpoint directory.
Element Name : CheckpointDir
Element Type : StringEquation

B.5 SLO description

This tag compounds information about SLOs.
Element Name : SLO_SLA
Element Type : SectionEquation
Required Element : Condition, Reason, Respond site, Punish, Action
: Monitor

B.5.1 Condition

This tag describes the problem, which can happen within the SLA valid period.
Element Name : SLO_CONDITION
Element Type : StringEquation
Example :

```
<StringEquation attribute="SLO_CONDITION">
    <StringValue>Runtime exceed</StringValue>
</StringEquation>
```

B.5.2 Reason

This tag describes the reason of the problem.

Element Name : SLO_REASON

Element Type : StringEquation

Example :

```
<StringEquation attribute="SLO_REASON">
    <StringValue>System failure</StringValue>
</StringEquation>
```

B.5.3 Respond site

This tag describes the site, which responds for the problem.

Element Name : SLO_RESPON_SITE

Element Type : StringEquation

Example :

```
<StringEquation attribute="SLO_RESPON_SITE">
    <StringValue>Provider</StringValue>
</StringEquation>
```

B.5.4 Punish

This tag describes the punishing activity to the responsible site.

Element Name : SLO_PUNISH

Element Type : StringEquation

Example :

```
<SectionEquation attribute="SLO_PUNISH">
    <RealEquation attribute="AMOUNT">
        <RealValue>1000</RealValue>
    </RealEquation>
    <StringEquation attribute="ENTITY">
        <StringValue>Euro</StringValue>
```

```
    </StringEquation>
</SectionEquation>
```

B.5.5 Action

This tag describes the activity when problem happens.

Element Name : SLO_ACTION

Element Type : StringEquation

Example :

```
<StringEquation attribute="SLO_ACTION">
    <StringValue>Terminate</StringValue>
</StringEquation>
```

B.5.6 Monitor

This tag describes the information that user wants to know during the process of handling SLA.

Element Name : SLO_MONITOR

Element Type : SectionEquation

Example :

```
<SectionEquation attribute="SLO_MONITOR">
    <StringEquation attribute="WHAT">
        <StringValue>Runtime</StringValue>
    </StringEquation>
    <StringEquation attribute="WHEN">
        <StringValue>Each 5 minuten</StringValue>
    </StringEquation>
</SectionEquation>
```

B.6 Data transmission description

This tag contains description about data transmission.

Element Name : DATA_TRANSMISSIC
Element Type : SectionEquation
Required Element : File list, gFTP

B.6.1 File list

This tag describes the list of file to be transferred.

Element Name : FILE_NAMES
Element Type : StringListEquation
Example :

```
<StringListEquation attribute="FILE_NAMES">
    <StringListValue>
        <StringValue>al.B.1</StringValue>
        <StringValue>lu.A.2</StringValue>
        <StringValue>bp.C.2</StringValue>
        <StringValue>dg.C.8</StringValue>
    </StringListValue>
</StringListEquation>
```

B.6.2 gFTP

This tag describes the FTP server address and the path to the directory, which stores the file.

Element Name : gFTP
Element Type : SectionEquation
Example :

```
<SectionEquation attribute="gridFTP">
    <StringEquation attribute="gFTPserver">
        <StringValue>gsiftp://server2.icenigrid.org</StringValue>
    </StringEquation>
```

```
    <StringEquation attribute="path">
        <StringValue>not/so/deep</StringValue>
    </StringEquation>
</SectionEquation>
```

B.7 SLA workflow description

Element Name : SLA_JOBFLOW
Element Type : SectionEquation
Required Element : Identification, General SLA, SLA sub-jobs,
: SLA data transmissions, SLOs, Signature

B.7.1 SLA sub-job

Element Name : SLA_SUBJOB
Element Type : SectionEquation
Required Element : Identification, SLA data transmissions,General SLA
: job description, SLA data transmissions, SLOs, Signature

B.7.2 SLA data transmissions

Element Name : DATA_TRAN_SLA
Element Type : SectionEquation
Required Element : Identification, General data transmission SLA,
: data transmissions description, Signature

B.7.3 Signature

This tag describes the signature of provider and customer.

Element Name : Signature
Element Type : SectionEquation
Example :

```
<SectionEquation attribute="Signature">
    <StringEquation attribute="Provider">
```

```
        <StringValue>oieshfio23874sodr</StringValue>
    </StringEquation>
    <StringEquation attribute="Customer">
        <StringValue>cxgdf4577xkldflee</StringValue>
    </StringEquation>
</SectionEquation>
```

Bibliography

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list scheduling for parallel processing systems", Communication ACM, v. 17 n. 12, pp. 685–690, 1974.

[2] A. Afzal, A. Mayer and L. Young, "Predictable workow deployment services", Proceedings of Grid Services Workshop, GGF11. IEEE, June 2004.

[3] I. Ahmad , Y. K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling", IEEE Transactions on Parallel and Distributed Systems, v.9 n.9, pp. 872–892, September 1998.

[4] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail, "G-QoSM: Grid Service Discovery using QoS Properties", Computing and Informatics Journal, Special Issue on Grid Computing, v. 21 n. 4, pp. 363–382, 2002.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock, "Kepler: Towards a Grid-Enabled System for Scientific Workflows", Proceedings of the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany, March, 2004.

[6] A. Andrieux, "Web services agreement specification (wsagreement)", Global Grid Forum, http://www.ggf.org, 2003.

[7] J. Baxter, and J. H. Patel, "The LAST algorithm: A heuristic-based static task allocation algorithm", Proceedings of the International Conference on Parallel Processing (ICPP '89, Aug.), Pennsylvania State University, University Park, PA, pp. 217–222, 1989.

[8] S. Benkner, I. Brandic, G. Engelbrecht and R. Schmidt, VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing, Proceedings of the Fifth IEEE/ACM InternationalWorkshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.

[9] S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. E. Middleton and R. Schmidt, GEMSS: Gridinfrastructure for Medical Service Provision, Methods of Information in Medicine 2005; 44: 177-181, Schattauer Publishers, 2005.

[10] F. Berman, H. Casanova, A. Chien, K. Cooper , H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel , D. Reed, Z. Shi, O. Sievert, H. Xia and A. YarKhan, "New Grid Scheduling and Rescheduling Methods in the GrADS Project", International Journal of Parallel Programming, v. 33, pp. 209–229, 2005.

[11] G. B. Berriman, J. C. Good, A. C. Laity, "Montage: a Grid Enabled Image Mosaic Service for the National Virtual Observatory" , ADASS, v. 13, 2003.

[12] P. E. Black, "Algorithms and Theory of Computation Handbook", CRC Press LLC, 1999.

[13] J. Blythe, J. Jain, E. Deelman, Y. Vahi, A. Mandal, K. Kennedy, "Task Scheduling Strategies for Workflow-based Applications in Grids", Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), IEEE Press, 2005.

[14] A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis", Artificial Intelligence, v 90, pp. 281–300, 1997.

[15] BPEL Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ 2003.

[16] I. Brandic and S. Benkner and G. Engelbrecht and R. Schmidt, QoS Support for Time-Critical Grid Workflow Applications, Proceedings of e-Science 2005.

[17] P. Brucker, "Scheduling Algorithm", Third edition, Springer Verlag, 2001.

[18] L. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The Virtual Resource Manager: An Architecture for SLA-aware Resource Management", Proceedings of the IEEE CCGrid 2004, IEEE Press, pp. 126–133, 2004.

[19] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing", Proceedings of 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 12-15, pp. 198, 2003.

[20] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid Load Balancing Using Intelligent Agents", Future Generation Computer Systems special issue on Intelligent Grid Environments: Principles and Applications, v. 21 n. 1, pp. 135–149, 2005.

[21] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep applications in Grid environments", Proceedings of the 9th HeterogeneousComputing workshop (HCW'2000), 2000.

[22] H. Chen, B. Shirazi, and J. Marquis, "Performance evaluation of a novel scheduling method: Linear clustering with task duplication", Proceedings of the 2nd International Conference on Parallel and Distributed Systerns, pp. 270–275, 1993.

[23] Y.-C. Chung , S. Ranka, "Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors", Proceedings of the 1992 ACM/IEEE conference on Supercomputing, pp. 512–521, November 16-20, 1992.

[24] E. G. Coffman, "Computer and Job-Shop Scheduling Theory", John Wiley and Sons, Inc., New York, NY, 1976.

[25] J. Y. Colin, and P. Chretienne, "Scheduling with small computation delays and task duplication". Operation Research, v. 39 n. 4, pp. 680-684, 1991.

[26] K. Czajkowski and I. Foster and C. Kesselman and V. Sander and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating

Resource Management in Distributed Systems", Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing, 2002.

[27] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. "Agreement- based Grid service management (WS-Agreement)", Global Grid Forum, 2003.

[28] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke, "From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution", http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf , March 5, 2004.

[29] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, M. Livny, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, v. 1 n. 1, pp. 25–39, 2003.

[30] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny, "Pegasus : Mapping Scientific Workflows onto the Grid", Proceedings of the 2nd European Across Grids Conference, Nicosia, Cyprus, January 28-30, 2004.

[31] M. Dorigo and G. Di Caro, The Ant Colony Optimization Meta-Heuristic, New Ideas in Optimization, McGraw-Hill, pp.11-32, 1999.

[32] K.A. Dowsland, "Simulated Annealing", in C.R. Reeves (ed.) ModernHeuristic Techniques for Combinatorial Problems , John Wiley & Sons, 1993.

[33] e-Science Workflow Services Workshop. http://www.nesc.ac.uk/action/esi/ contribution.cfm?Title=303, December 2003.

[34] C. A. Ellis, "Information Control Nets: A Mathematical Model of Office Information Flow", Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, 1979.

[35] A.K. Elmagarmid, "Database Transaction Models for Advanced Applications", Morgan Kaufmann, 1992.

[36] D. W. Erwin, and D. F. Snelling, "UNICORE: A Grid Computing Environment", Lecture Notes in Computer Science, v. 2150: pp. 825–834, 2001.

[37] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservation and coallocation", Proceedings of the International Workshop on Quality of Service, pp. 27–36, 1999.

[38] I. Foster, "What is the Grid? A Three Point Checklist", Daily News and Information for the Global Grid Community, July 22, 2002, [Last accessed May 31st, 2004], http://www.gridtoday.com/02/0722/100136.html

[39] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, v. 15 n. 3, 2001.

[40] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[41] M. Fox, D. Long, "PDDL2.1: An extension of PDDL for expressing temporal planning domains", Journal of AI Research, v. 20, pp. 61–124, 2003.

[42] P. Garbacki, B. Biskupski, and H. Bal, "Transparent Fault Tolerance for Grid Application", Proceedings of the European Grid Conference,(EGC 2005), LNCS 3470, pp. 671–680, 2005.

[43] M. R. Gary and D. S. Johnson, "Computers and Intractability: A Guide to the theory of NP-Completeness", W. H. Freeman and Co, 1979.

[44] The GEMSS Project: Grid-Enabled Medical Simulation Services, EU IST Project, IST-2001-37153, http://www.gemss.de/

[45] I.P. Gent, E. Macintyre, P. Prosser, B.M. Smith, T. Walsh, "An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem",

Proceedings of Principles and Practices of Constraint Programming - CP96, LNCS 1118.

[46] D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, v. 3 n. 2, April 1995.

[47] A. Gerasoulis, and T. Yang, "A comparison of clustering heuristics for scheduling DAG's on multiprocessors", J. Parallel Distributed Computing, v. 16, pp. 276–291, 1992.

[48] "Globus Toolkit 4.0 Documentation Overview", http://www.globus.org/toolkit/docs/4.0/doc_overview.html

[49] F. Glover, "Tabu search Part I", ORSA Journal on Computing, pp. 190–206, 1989.

[50] F. Glover, "Tabu search Part II". ORSAJournal on Computing, v.2, pp. 4–32, 1990.

[51] A. Grimshaw, "WHAT IS A GRID?", Grid today / DECEMBER 9, v. 1 n. 26, 2002.

[52] J.K. Hao, J. Pannier, "Simulated annealing and Tabu search for constraint solving", Artificial Intelligence and Mathematics IV, 1998

[53] P. Hansen, N. Mladenovic, "Variable Neighborhood Search: Principles and Applications", European Journal of Operational Research, v.130, 2001.

[54] D. Harel and A. Naamad, The STATEMATE Semantics of Statecharts, ACM Transactions on Software Engineering and Methodology, 5(4):293.333, 1996

[55] F. Heine, M. Hovestadt, O. Kao, A. Keller, "Provision of Fault Tolerance with Grid-enabled and SLA-aware Resource Management Systems", Proceedings of the Parallel Computing Conference 2005, 2005.

[56] F. Heine, M. Hovestadt, O. Kao, A. Keller, "SLA-aware Job Migration in Grid Environments", in: Grandinetti, (Eds.), Grid Computing: New Frontiers of High Performance Computing, Elsevier press, 2005.

[57] J. Hoffmann, "Extending FF to Numerical State Variables", Proceedings of the 15th European Conference on Artificial Intelligence, 2002.

[58] M. Hovestadt, "Scheduling in HPC Resource Management Systems:Queuing vs. Planning", Proceedings of the 9th Workshop on JSSPP at GGF8, LNCS, pp. 1–20, 2003.

[59] E. S. H. Hou , N. Ansari , H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Transactions on Parallel and Distributed Systems, v.5 n.2, pp. 113–120, February 1994.

[60] M. Hsu (ed.), "Special Issue on Workflow and Extended Transaction Systems", IEEE Data Engineering, v. 16 n. 2, June 1993.

[61] S. Hwang, C. Kesselman, "GridWorkflow: A flexible failure handling framework for the Grid", Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, IEEE Press, pp. 126–131, 2003.

[62] D. Jackson, Q. Snell and M. Clement, "Core Algorithms of the Maui Scheduler", Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2221, pp. 87–102, 2001.

[63] B. Jacob, "Grid computing: What are the key components?", http://www-128.ibm.com/developerworks/grid/library/gr-overview/

[64] K. Jansen K, M. Mastrolilli, R. Solis-Oba, "Approximation Algorithms for Flexible Job Shop Problems", Proceedings of Latin American Theoretical Informatics (LATIN'2000), LNCS 1776, pp. 68–77, 2000.

[65] I. Kacem, S. Hammadi, P. Borne, "Pareto-optimality Approach for Flexible Jobshop Scheduling Problems: Hybridization of Evolutionary Algorithms and Fuzzy Logic", Journal of Mathematics and Computers in Simulation, Elsevier, 2002.

[66] I. Kacem, S. Hammadi, P. Borne, "Approach by Localization and Multi-objective Evolutionary Optimization for Flexible Job-Shop Scheduling Problems", IEEE Transactions on Systems, Man, and Cybernetics. Part C, v. 32 n. 1, pp. 1-13, 2002.

[67] H. Karloff. Linear Programming. Birkhauser, 1991.

[68] Keahey and K. Motawi, "The Taming of the Grid: Virtual Application Service", Argonne National Laboratory Technical Memorandum, No. 262, 2003.

[69] S. J. Kim, and J. C. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures", Proceedings of International Conference on Parallel Processing, pp. 1–8, 1988.

[70] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing", Science, v. 220, pp. 671–680, 1983.

[71] W. H. Kohler and K. Steiglitz, "Characterization and Theoretical Comparison of Brand-and-Bound Algorithms for Permutation Problems", Journal of ACM, v. 21 n. 1, pp. 140–156, 1974.

[72] B. Kruatrachue, and T. G. Lewis, "Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems", Oregon State University, Corvallis, OR, 1987.

[73] B. Kruatrachue , T. Lewis, "Grain Size Determination for Parallel Processing", IEEE Software, v.5 n.1, pp. 23–32, January 1988.

[74] V. Kumar, "Algorithms for constraint-satisfaction problems: a survey", AI Magazine, v.13 n.1, pp. 32–44, Spring 1992.

[75] Y. K. Kwok, I. Ahmad, "Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures", Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing, pp. 36, October 25-28, 1995.

[76] Y. K. Kwok , I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", IEEE Transactions on Parallel and Distributed Systems, v.7 n.5, pp. 506–521, 1996.

[77] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors", ACM Computing Surveys (CSUR), v. 31 n. 4, pp. 406–471, 1999.

[78] G. Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, "GridAnt: A Client-Controllable Grid Workflow System", Proceedings of the 37th Hawaii International Conference on System Science, Island of Hawaii, Big Island, Jan. 5-8, 2004.

[79] M. Litzkow and M. Mutka, "Condor a hunter of idle workstations", Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS 88), pp. 104–111, 1988.

[80] M. Lorch, and D. Kafura, "Symphony - A Java-based Composition and Manipulation Framework for Computational Grids", Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany, May 21-24, 2002.

[81] R. Lovas, G. Dzsa, P. Kacsuk, N. Podhorszki, D. Drtos, "Workflow Support for Complex Grid Applications: Integrated and Portal Solutions", Proceedings of 2nd European Across Grids Conference, Nicosia, Cyprus, 2004.

[82] S. Ludtke, P. Baldwin, and W. Chiu, "EMAN: Semiautomated Software for High-Resolution Single-Particle Reconstructio" , Journal of Structure Biology, v. 128, 1999.

[83] T. Ma and R. Buyya, "Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids", Proc. 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005), IEEE CS Press, 2005.

[84] M. Mastrolilli, L.M. Gambardella, "Effective Neighborhood Functions for the Flexible Job Shop Problem", Journal of Scheduling, v. 3 n. 1, pp. 3–20, 2000.

[85] S. McGough, "A common job description markup language written in xml", Global Grid Forum, http://www.ggf.org, 2003.

[86] S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young ,Making the Grid Predictable through Reservations and Performance Modelling, The Computer Journal, v.48 n.3, pp. 358–368, 2005.

[87] A. Morajko, E. Fernandez, "Workfow management in the crossgrid project", Proceedings of the European Grid Conference,(EGC 2005), pp. 424–433. LNCS. Springer Verlag, February 2005.

[88] N. Mehdiratta, and K. Ghose, "A bottom-up approach to task scheduling on distributed memory multiprocessor", Proceedings of the 1994 International Conference on Parallel Processing, CRC Press, Inc, pp. 151–154, 1994.

[89] L. Nassif, J. M. Nogueira, M. Ahmed, R. Impey, A. Karmouch, "Agent-based Negotiation for Resource Allocation in Grid", Proceedings of the 3rd Workshop on computational Grids and applications, Summer program LNCC, 2005.

[90] E. Nowicki, C. Smutnicki, "A fast taboo search algorithm for the job shop problem", Management Science, v. 42, pp. 797–813, 1996.

[91] E. Nowicki, C. Smutnicki, "An advanced taboo search algorithm for the job shop problem", Journal of Scheduling, pp. 145–159, 2005.

[92] C. H. Papadimitriou , M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms", SIAM Journal on Computing, v.19 n.2, pp. 322–328, April 1990.

[93] "OpenPBS Technical Overview", http://www.openpbs.org/overview.html.

[94] M. Pelikan, D. Goldberg, and E. Cant-Paz, BOA: The Bayesian optimization algorithm, Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99 (Orlando, Fla.), pp. 525-532, 1999.

[95] M. Pelikan, D. Goldberg and F. Lobo, A survey of optimization by building and using probabilistic models, Tech. Rep. No. 99018, IlliGAL, University of Illinois, 1999.

[96] M. Pelikan, and H. Mhlenbein, The bivariate marginal distribution algorithm, Advances in Soft Computing-Engineering Design and Manufacturing, pp. 521-535, 1999.

[97] D.M. Quan, O. Kao, "On Architecture for an SLA-aware Job Flows in Grid Environments", Journal of Interconnection Networks, World scientific computing, pp. 245–264, 2005.

[98] D.M. Quan, O. Kao, "SLA negotiation protocol for Grid-based workflows", Proceedings of the International Conference on High Performance Computing and Communications (HPPC-05), LNCS 3726, pp. 505–510, 2005.

[99] D.M. Quan, O. Kao, "On Architecture for an SLA-aware Job Flows in Grid Environments", Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA 2005) , IEEE Press , pp. 287–292, 2005.

[100] D.M. Quan, O. Kao, "Mapping Grid job flows to Grid resources within SLA context", Proceedings of the European Grid Conference,(EGC 2005), LNCS 3470, pp. 1107–1116, 2005.

[101] D.M. Quan, "Error recovery mechanism for Grid-based workflow within SLA context", To be published by International Journal of High Performance Computing and Networking (IJHPCN), 2006.

[102] D.M. Quan, "Mapping heavy communication Workflows onto Grid Resources within SLA context", To be published in proceedings of the International Conference of High Performance Computing and Communication (HPCC06), 2006.

[103] D.M. Quan, "Adaptive SLA execution engine for Grid-based workflow within SLA context", Submitted to Middleware 2006 Conference, 2006.

[104] H. E. Rewini , T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines", Journal of Parallel and Distributed Computing, v.9 n.2, pp. 138–153, June 1990.

[105] L. Richter, "Workflow Support for Complex Grid Applications: Integrated and Portal Solutions", Proceedings of the 2nd European Across Grids Conference, 2004.

[106] N. Sadeh and M.S. Fox, "Variable and Value Ordering Heuristics for the Job Shop Constraint Satisfaction Problem", Artificial Intelligence, v. 86, pp. 1–41, 1996.

[107] V. Sarkar, "Partitioning and Scheduling Parallel Programs for Multiprocessors", MIT Press, Cambridge, MA, 1989.

[108] A. Shahid , S. T. Muhammed , M. Sadiq , "GSA: scheduling and allocation using genetic algorithm", Proceedings of the conference on European design automation, pp. 84–89, September 19-23, 1994.

[109] M. Shields, and I. Taylor, "Programming Scientific and Distributed Workflow with Triana Services", Proceedings of the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany, March, 2004.

[110] G. C. Sih , E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", IEEE Transactions on Parallel and Distributed Systems, v.4 n.2, pp. 175–187, 1993.

[111] G. C. Sih , E. A. Lee, "Declustering: A New Multiprocessor Scheduling Technique", IEEE Transactions on Parallel and Distributed Systems, v.4 n.6, pp. 625–637, June 1993.

[112] M. P. Singh and M. A. Vouk , "Scientific Workflows: Scientific Computing Meets Transactional Workflows", http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html

[113] "What is Service Level Agreement?", http://www.webopedia.com/TERM/S/Service_Level_Ag

[114] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini and G. R. Nudd, "Local Grid Scheduling Techniques Using Performance Prediction", IEEE Proceedings - Computers and Digital Techniques, 150(2), pp. 87–96, 2003.

[115] Na. Stone, D. Simmel, and T. Kielmann, "GWD-I: An architecture for grid checkpoint recovery services and a GridCPR API", Grid Checkpoint Recovery Working Group Draft 3.0, Global Grid Forum, http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-Architecture-2.0.pdf. 2004.

[116] T. Stuetzle, Iterated local search for the quadratic assignment problem, Tech. rep. aida-99-03, FG Intellektik, TU Darmstadt, 1999.

[117] T. Stuetzle, Local Search Algorithms for Combinatorial Problems—Analysis, Algorithms and New Applications, DISKI—Dissertationen zur Knstliken Intelligenz. infix, Sankt Augustin, Germany, 1999.

[118] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0", Global Grid Forum Draft Recommendation, 6/27/2003.

[119] C. Voudouris, Guided local search for combinatorial optimization problems, PhD dissertation, Department of Computer Science, University of Essex, 1997.

[120] C. Voudouris, and E. Tsang, Guided local search, Europe Journal of Operation Research, v. 113 n.2, pp. 469–499, 1999.

[121] R. Wolski, "Experiences with Predicting Resource Performance On-line in Computational Grid Settings", ACM SIGMETRICS Performance Evaluation Review, v. 30 n. 4, pp. 41–49, 2003.

[122] R. Wolski, N. Spring and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Journal of Future Generation Computer System, v. 15 n. 5-6, pp. 757–768, 1999.

[123] http://www.webopedia.com/TERM/S/Service_Level_Agreement.html

[124] S. Wesner, B. Serhan, "Overview of an architecture enabling grid based application service provision", Proceedings of the European Across Grids Conference 2004, pp. 113–118, 2004.

[125] L. Fischer, "Workflow Handbook 2004 ", Future Strategies Inc., Lighthouse Point, FL, USA, 2004.

[126] M. Y. Wu , D. D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems", IEEE Transactions on Parallel and Distributed Systems, v.1 n.3, pp. 330–343, 1990.

[127] T. Yang , A. Gerasoulis, "List scheduling with and without communication delays", Parallel Computing, v.19 n.12, pp. 1321–1344, 1993.

[128] L. Young, S. McGough, S. Newhouse and J. Darlington, Scheduling architecture and algorithms within ICENI, Proceeding of UK e-Science All Hands Meeting, EPSRC, 2003

[129] J. Yu, R. Buyya, "A taxonomy of scientific workflow systems for grid computing", SIGMOD Record 34(3), pp. 44–49, 2005.

[130] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-Aware Middleware for Web Services Composition, IEEE Transactions on Software Engineering, v.30 n.5, pp. 311–327, may 2004.