Decentralized Dynamic Resource Allocation for Workflows in Grid Environments

Florin Pop Ciprian Dobre Valentin Cristea

Faculty of Automatics and Computer Science, University "*Politehnica*" of Bucharest, Romania
Emails: {florin.pop, ciprian.dobre, valentin.cristea}@cs.pub.ro

Abstract

In this paper we present an innovative decentralized dynamic resource scheduling solution of large scale application workflows onto distributed, heterogeneous Grid environments. The presented resource allocation solution aims to optimize large-scale scientific application workflows by efficiently and effectively mapping and scheduling them onto Grid resources. The proposed solution is part of a framework that aims to help scientists easily deploy large-scale scientific workflow applications from a wide-range of research fields. The decentralized architecture, together with the scheduling policies and fault management being proposed in this paper, are specifically designed to optimize the scheduling management for workflows in case of a wide range of Grid environment. In order to demonstrate the validity and performance of the presented scheduling solution we propose using modeling and simulation techniques. For that we present the characteristics of a generic simulator that includes all the necessary components to help in evaluating a wide-range of scheduling procedures, such as the ones presented in this, in the context of large-scale heterogeneous Grid environment.

Keywords

Resource Allocation, Grid Scheduling, Workflows Management, Decentralized Architecture, Simulation.

1 Introduction

Grid computing was long seen as the de-facto solutions for enabling scientists to seamlessly solve large-scale problems using the resources being shared in Grid environments. The original vision was that the complexity of the Grid is hidden from the scientist. However, the current solutions being proposed by most Grid environments are far from this vision. They lack the capability of allowing the scientist deploy large-scale scientific workflow applications without him worrying about the existing resources needed to successfully complete the work. Also many of the existing Grid workflow frameworks have limited scheduling capabilities [1]. They are limited to using random, on-line or FCFS scheduling policies, resulting in poor turnaround times.

In this paper we propose a solution to scheduling large-scale scientific workflows in Grid environments. We present an innovative architecture based on decentralized dynamic resource allocation techniques that is most suitable for optimal and accurate scheduling of workflows in Grid environments. The proposed solution is part of a framework designed to help scientists to easily deploy large-scale scientific workflows.

Our proposed solution considers the dependencies between the jobs composing the scientific application being deployed in a Grid environment. For that we present several adequate scheduling policies that aim to optimizing the turn-around times of jobs having dependencies as specified by upper-level scientific workflow applications. We also consider that Grids are far from being error-free environments. For that we propose the use of several approaches to account for the case when resource fail and the workflows need to be adequately re-scheduled without compromising the entire process of the executed application. We state that our proposed solution introduces a series of novelty procedures that are specific to workflow scheduling case.

The demonstration of the superiority of the proposed solution is conducted using modeling techniques, using an adequate simulation instrument also presented in this. The simulator includes all the necessary components that allow the evaluation of scheduling procedures and related procedures in the context of large-scale heterogeneous Grid environments.

The rest of this paper is organized as follows. We first present related work in scheduling workflows in Grid environments. Section 3 presents the workflow scheduling solution being proposed in this paper. We continue, in Section 4, with presenting a proposed decentralized architec-

ture for dynamic resource allocation in Grids, together with the scheduling policies and fault management techniques being considered. The decentralized architecture uses a series of dedicated policies aiming to optimize the turnaround times by imposing restrictions on resource allocations. We then present a series of solutions for managing workflows in the presence of faults occurring in various components of the underlying Grid environments. In Section 5 we present the means to evaluate the correctness and performance of the proposed decentralized dynamic resource allocation solution using modeling and simulation. We present the architecture of a modeling framework that is appropriate to be used to evaluate the scheduling procedures in the presence of scientific workflows and large-scale Grid environments. In Section 6 we give the conclusions and present future work.

2 Related Work

A decentralized resource allocation policy in minigrid is presented in [2]. It is shown that the traditional resource allocating policies statically assign resources to the jobs according to the distribution schema computed by the job scheduling policy. Those policies cannot handle the DAGs with large jobs that could be represented by services. Decentralized resource allocating policy is proposed to solve this problem by sharing the computing burden on several processors. However the existing decentralized policies cannot dynamically compute the dependent relationships of a given DAG or even schedule the data driven jobs. In [2] is proposes a cluster-based resource allocating policy, by which the large-scale distributed systems can precisely schedule the data driven jobs and the computing cost of jobs' scheduling can be controlled in an acceptable range as well.

In [3] is proposed a decentralized and cooperative workflow scheduling algorithm. The proposed approach utilizes a Peer-to-Peer (P2P) coordination space with respect to coordinating the application schedules among the Grid wide distributed workflow brokers. The algorithm presented in [3] is completely decentralized in the sense that there is no central point of contact in the system, the responsibility of the key functionalities such as resource discovery and scheduling coordination are delegated to the P2P coordination space. For this proposed solution the feasibility through an extensive trace driven simulation study.

In [4] is describes the approach for the optimized execution of computational tasks in Grid environments. Tasks are represented as workflows that define interactions between different services. Functional service descriptions written in OWL-S are extended with non-functional properties, allowing to specify the resource requirements of services depending on given inputs. An optimization algorithm is presented in [4]. It distributes the execution of a workflow in a Grid, supporting the dynamic deployment of software components on demand, in order to fulfill user requirements, such as a limit on the total workflow execution time. Using this solution, workflows are executed in a fully decentralized way.

3 Workflow Scheduling in Grid

Scheduling algorithm are very different for Grid Systems than for symmetric multiple processor machines or parallel processor computers. These differences are mainly derived from the dynamic nature of the Grid. Also, in Grid environments the scheduler cannot have accurate information about the background load of the machines, or about every resource available.

These limitations can lead to the need for a different approach to task scheduling for Grid Systems. This aspect of scheduled tasks is very important because most complex applications submitted by users have jobs that depend on the termination of others. In this case the scheduler must know how to efficiently allocate resources and optimize overall performance. In this case the application is usually described using a directed acyclic graph (DAG) with node and edge weights. Each node represents an individual job that needs to be executed. The node weights represent the resources needed to execute the job and the edges describe the precedence relations between jobs.

Scheduling a set of jobs described using a directed acyclic graph is an NP-complete problem in its general form. In order to solve the scheduling problem, three approaches have been suggested: state-space search, dynamic programming and heuristics. The first two techniques give optimal solutions under relaxed constraints, but they are not useful since most of them work under restricted environments and most importantly they lead to an exponential time in the worst case.

Heuristics are the usual approach to solve the scheduling problem. They achieve good efficiency, but, on the other side, they usually cannot generate optimal solutions and there is no guarantee about their performance in general, since the average, worst, and best case performances of these algorithms are not known.

Because in Grid systems communication delays are significant all algorithms must achieve maximum parallelism while minimizing data communication. Scheduling algorithms fall into one of the following categories:

List Heuristics. Each node is assigned a priority and the scheduling algorithm attempt to execute the higher priority nodes first. Most scheduling algorithms for Grid systems are based on this approach. Calculating rank values involves matching a specific component to a set

of resources and then ranking the resources. At first it is checked whether the Grid resource meets certain hard requirements (like required OS, required memory, storage, required minimum CPU speed etc.) for the component. If the resource doesn't meet the hard requirements, it is given a rank value of infinity [5]. If the resources meet the requirements, the node is evaluated.

Duplication based algorithms. The transfer of results from a predecessor to a successor can be avoided by duplicating some tasks, thus reducing the communication cost. For example the algorithm TDS (Task Duplication-based Scheduling Algorithm) can solve this problem with a complexity of $O(v^2)$.

Clustering heuristics. Clustering is another way of reducing the communication cost by grouping the heavily communicating tasks to the same clusters and then assigning tasks in a cluster to the same resources. Using this approach the number of slow communication speeds between tasks is reduced. Task clustering algorithms can have two phases: the initial task clustering and a post-clustering phase which can refine the clusters produced in the previous phase.

In Grid computing minimizing communication times is very important due to its heterogeneous nature and the fact that communication speeds can greatly affect performance. Other problems for Grid scheduling can appear due to the dynamism of the system, which can lead to conflicts when the scheduling decision is made.

One solution to the problem of minimizing communication times in Grid systems is to try to schedule as many tasks as possible to a single cluster to try to reduce the inter-cluster communication.

The dynamism of the Grid system is another factor that must be considered when implementing the algorithms presented in this chapter. If we consider the case of a meta-scheduler like GridWay, we have a multi-cluster system, each cluster with its own local scheduler and background workload. In this case the global scheduler is responsible for mapping tasks to different clusters according to their latest finish time in order to minimize the excess over the length of critical path.

3.1 Grid Workflow Scheduling Systems

DAGMan. The Directed Acyclic Graph Manager (DAGMan) is a service provided by Condor for executing multiple jobs with dependencies. DAGMan does not support automatic intermediate data movement, so users have to specify data movement transfer through pre-processing and post-processing commands associated with processing job. The DAGMan meta-scheduler processes the DAG dynamically, by sending to the condor scheduler the jobs as soon as their dependencies are satisfied and they become ready to execute. DAGMan can also help with the resubmission of uncompleted portions of a DAG when one or more nodes resulted in failure. If any node in the DAG fails, the remainder of the DAG is continued until no more forward progress can be made based on the DAG's dependencies [6].

Askalon. ASKALON is a Grid environment for composition and execution of scientific workflow applications. The scheduler optimizes for performance using the execution time as the most important goal function. The scheduler interacts with the enactment engine which is a service that supervises the reliable and fault tolerant execution of the tasks and transfer of the files. Scientific workflows executed in the ASKALON environment are based on the model described in the AGWL specification language. Different scheduling strategies can be applied, considering the trade-off between dinamicity and look-ahead range in workflow processing. Just-in-time strategy consists of mapping the tasks to the resources, always choosing the most appropriate solution for the current step. This approach benefits from using the most up-to-date performance data, which is important for the Grid, but on the other hand it neglects the graph structure, and the whole workflow may not be scheduled optimally [7].

Askalon uses an implementation of the HEFT algorithm which is based on the List Heuristics approach [8]. The HEFT algorithm that was applied consists of 3 phases:

1. Weighting assigns the weights to the nodes and edges in the workow;
2. Ranking creates a sorted list of tasks, organized in the order how they should be executed;
3. Mapping assigns the tasks to the resources.

Pegasus. Pegasus, which stands for Planning for Execution in Grids, was developed at ISI as part of the GriPhyN and SCEC/IT projects. Pegasus is a configurable system that can map and execute complex workflows on the Grid. Currently, Pegasus relies on a full-ahead-planning to map the workflows. Pegasus was first integrated with the GriPhyN Chimera system. In that configuration, Pegasus receives an abstract workflow (AW) description from Chimera, produces a concrete workflow (CW), and submits it to DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). Pegasus contains a "Concrete planner" which consults a Transformation Catalog to determine a location where the job can be executed. If there is more than one possible location, a location is chosen randomly. Pegasus also contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations. In this configura-

tion, Pegasus similarly generates the necessary submit files and sends the concrete workflow to DAGMan for execution. Once the resources are identified for each task, Pegasus generate s the submit file for Condor-G. The resulting concrete DAG is sent to DAGMan for execution [9].

GridWay. The GridWay Metascheduler Project has also added support for job dependency with version 4.9. Job dependency capability allows the execution of a submitted job depending on the completion of other jobs submitted in the grid. Latest versions support branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the workflow of certain type of applications. The bash script flow control structures and the GridWay commands allow the development of workflows described in an abstract form without referring to specific resources for task execution [10].

4 Dynamic Resource Allocation in Grids

The resource allocation problem in Grid Environment in a complex one. When talking about the design of a schedule for a given system and implicitly of a scheduling algorithm must consider all aspects of the system and applications that will run in it. First of all it is necessary to draw a distinction between local and global organizational level.

4.1 Decentralized Architecture

The decentralized organization is a good solution for Grid meta-scheduling. It naturally addresses several important issues such as fault-tolerance, scalability, site-autonomy, and multi-policy scheduling. If a distributed scheduling algorithm is adopted, the next problem that needs to be resolved is whether the nodes involved in job scheduling are working cooperatively or independently (non-cooperatively). In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of the scheduling system. When we are dealing with cooperative schedulers, each of the involved Grid schedulers has the responsibility to carry out its own portion of the scheduling task, but all schedulers are working toward an overall goal. This involves the implementation by the scheduling algorithm of communication protocols between the agents which must take the scheduling decision.

When talking about dynamic scheduling, the basic idea is to perform resource allocation on the fly while other applications are in execution. This is useful in the case where jobs arrive in a real-time mode. Dynamic scheduling is usually applied when it is difficult to estimate the cost of applications, or jobs are coming on-line dynamically (in this case, it is also called on-line scheduling). A good example of these scenarios is the job queue management in some meta-computing systems like Condor and Legion. Dynamic task scheduling has two major components: system state estimation (other than cost estimation in static scheduling) and decision making.

System state estimation involves collecting state information throughout the Grid and constructing an estimate. On the basis of the estimate, decisions are made to assign a task to a selected resource. Since the cost for an assignment is not available, a natural way to keep the whole system healthy is by balancing the loads of all resources. The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behaviour of the application before execution. It is particularly useful in a system where the primary performance goal is maximizing resource utilization, rather than minimizing runtime for individual jobs. If a resource is assigned too many tasks, it may invoke a balancing policy to decide whether to transfer some tasks to other resources, and which tasks to transfer. According to who will initiate the balancing process, there are two different approaches: sender-initiated where a node that receives a new task but doesn't want to run the task initiates the task transfer, and receiver-initiated where a node that is willing to receive a new task initiates the process.

According to how the dynamic load balancing is achieved, there are four basic approaches: unconstrained First-In-First-Out (FIFO, also known as First-Come-First-Served), balance-constrained techniques, cost-constrained techniques, hybrids of static and dynamic techniques.

4.2 Scheduling Policies

We could use for the dynamic resource allocation component the GridWay model for supporting policies. The Last release for GridWay provides state-of-the-art scheduling policies, comprising job priority policies (fixed priority, urgency, share, deadline and waiting-time) and resource priority policies (fixed priority, usage, failure and rank). The policies based scheduling could uses some heuristics methods to obtains schedule decision.

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum make-span and maximum resource utilization. But due to the NP-Complete nature of scheduling algorithms and the difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories: approximate and heuristic algorithms.

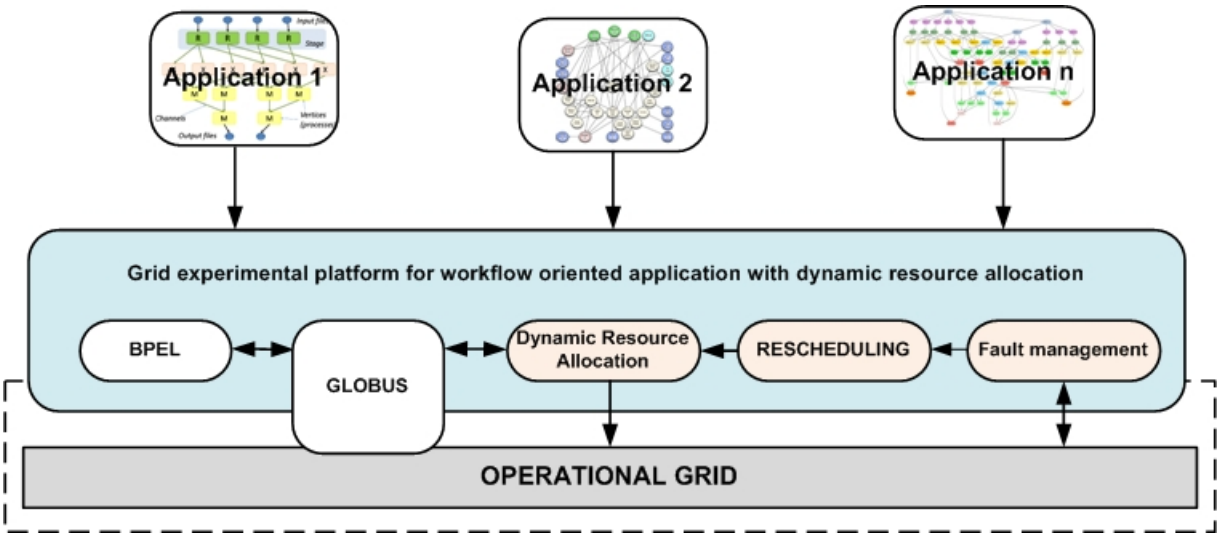

Figure 1. Solution for Dynamic Resource Allocation in Grids

4.3 Faults Management for Workflow

Faults management for workflow has to components: execution context and the application code itself.

At the execution context, the analysis boils down to the resource scheduler and its capabilities to support fault tolerance. As far as the application code is concerned, we distinguish two cases: when the applications invoke one Grid services and when the application is a composition of several different grid services.

The fault tolerance level for a simple application can be achieved by using grid check-pointing strategies [16]. The role of Fault management component in the proposed solution (see figure 1) is to provide transparent check-pointing support for Grid services, by capturing and restoring local and global states through the use of a execution coordinator.

If the services are developed in Java, there are more particular approaches which aim to offer a certain degree of fault tolerance for Java, which consist basically of a fault tolerant RMI layer. Such layer replaces remote references automatically in case a reference is in the impossibility to execute an operation over a connection.

In case of a workflow applications containing more collaborative or/and independent tasks which are running in parallel, the Condor job scheduler (DagMAN) mentioned in the previous section is being able to launch only one job.

Techniques based on GridRPC to obtain a fault tolerant service have main target applications that are running uninterruptible computations, where stopping the calculus for a limited duration is not critical. One the other pole, Grid services aim to provide the result immediately, which reduces the applicability of these techniques for our project. Of course, in case of some grid operations which have to be completed without any timing constraints, such techniques could be applied.

5 Validation model using Simulation

The use of discrete-event simulators in the design and development of large scale distributed systems is appealing due to their efficiency and scalability. For this reason we propose to evaluate the proposed decentralized dynamic resource allocation approach using simulation. The simulation based validation approach is suitable for evaluating different scheduling procedures considering for example a wide range of Grid system architectures.

Our proposed solution is based on MONARC. The generic simulator is specifically designed for modeling large scale distributed systems, allowing the realistic simulation of a wide-range of distributed system technologies, with respect to their specific components and characteristics. Its simulation model in particular includes the necessary components to describe various actual distributed system technologies, and provides the mechanisms to describe concurrent network traffic, evaluate different strategies in data replication, and analyze job scheduling procedures.

MONARC is built based on a process oriented approach for discrete event simulation, which is well suited to describe concurrent running programs, network traffic as well as all the stochastic arrival patterns. Threaded objects or ”Active Objects” (having an execution thread, program counter, stack, etc.) allow a natural way to map the specific behavior of distributed data processing into the simulation program.

In order to provide a realistic simulation, all the components of the system and their interactions were abstracted.

The chosen model is equivalent to the simulated system in all the important aspects. A first set of components was created for describing the physical resources of the distributed system under simulation. The largest one is the regional center (see Figure 2), which contains a farm of processing nodes (CPU units), database servers and mass storage units, as well as one or more local and wide area networks. Another set of components model the behavior of the applications and their interaction with users. Such components are the "Users" or "Activity" objects which are used to generate data processing jobs based on different scenarios. The job is another basic component, simulated with the aid of an active object, and scheduled for execution on a CPU unit by a "Job Scheduler" object. Any regional center can dynamically instantiate a set of users or activity objects, which are used to generate data processing jobs based on different simulation scenarios. Inside a regional center different job scheduling policies may be used to distribute jobs to corresponding processing nodes.

Using the MONARC simulator it is possible to build a wide range of models, from the very centralized to the distributed system models, with an almost arbitrary level of complexity (multiple regional centers, each with different hardware configuration and possibly different sets of replicated data).

One of the strengths of MONARC is that it can be easily extended, even by users, and this is made possible by its layered structure. The first two layers contain the core of the simulator (which we call "simulation engine") and models for the basic components of a distributed system (CPU units, jobs, databases, networks, job schedulers etc.); these are the fixed parts on top of which some particular components (specific for the simulated systems) can be built. The particular components can be different types of jobs, job schedulers with specific scheduling algorithms or database servers that support data replication.

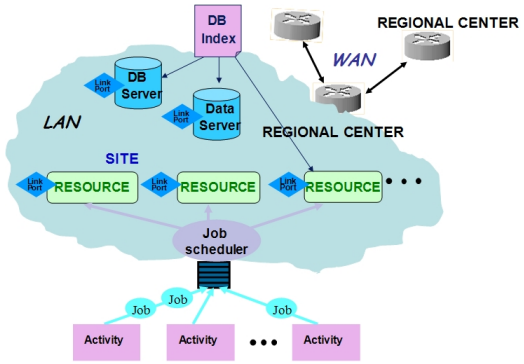

Figure 2. The Regional center model

The simulation model was extensively validated using queuing methodologies and testbed measurements [11]. The evaluation of MONARC concerned several aspects: sharing of CPU and I/O resources, queuing mechanism, performance of the database and network models. Other validation experiments [12] demonstrated that the simulation model includes the necessary components to correctly describe various actual distributed system technologies in respect to their specific components and characteristic. The model provides the mechanisms to describe concurrent network traffic, evaluate different strategies in data replication, and analyze job scheduling procedures. MONARC can be used to simulate a large number of possible scenarios, ranging from high-granularity network protocols to large-scale distributed systems comprising of many type of resources, to the modeling and simulation of the behavior of various applications making it an ideal instrument to accurately evaluating the performances of scheduling procedures such as the ones presented in this paper.

The superior performance of the simulator was demonstrated in terms of experimenting with a large number of concurrent threads, evaluation of the interrupt model and in terms of time needed to conduct simulation experiments of large scale [12]. For example, Figure 3 presents the performance results of the simulator in terms of concurrent threads. The series represents the different architectures on which the evaluation experiment was conducted. In MONARC all the jobs being concurrently processed by the same processing node are handled by one thread, and all the messages being concurrently generated from the same host are also managed by one thread. The obtained results demonstrate the capability of the simulator to cope with experiments describing distributed systems consisting of thousands of processing nodes and concurrent network links.

The maturity of the simulation model was demonstrated in previous work. For example, a number of data replications experiments were conducted in [13], presenting important results for the future LHC experiments, which will produce more than 1 PB of data per experiment and year, data that needs to be then processed. A series of scheduling simulation experiments were presented in [13], [14] and [15]. In [15] we successfully evaluated a simple distributed scheduler algorithm, proving the optimal values for the network bandwidth or for the number of CPUs per regional center. In [14] the simulation model was used to conduct a series of simulation experiments to compare a number of different scheduling algorithms. Probably the most extensive simulation scenario is the one described in [13]. The experiment tested the behavior of the Tier architecture envisioned by the two largest LHC experiments, CMS and ATLAS. The obtained results indicated the role of using a data replication agent for the intelligent transferring of the produced data.

Being a generic simulator, MONARC is a very good vehicle for performance evaluation of a wide range of schedul-

ing procedures and technologies such as the ones presented in this paper, taking into consideration a wide range of large scale distributed system architectures and related technologies. With the MONARC simulation model, users can define various types of jobs to model common types of actions that can occur in any distributed systems: processing, data transfer (both pull and push models are supported), database handling, etc. The model is not limited to specific activities, the user having the possibility to easily incorporate new advanced job behaviors, as specified in the simulation experiment being executed.

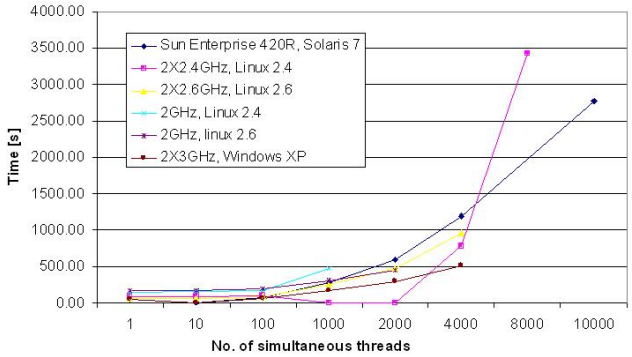

Figure 3. Performance of the MONARC simulator in terms of concurrent threads

A simulated regional center uses the services of a job scheduler. In order to schedule a job for execution the scheduler executes an appropriate scheduling algorithm, which can be specified by the user. For resource scheduling, the simulator is able to model the scheduling policies used by resource brokers to determine which resource should process an arriving task. The MONARC simulation model provides a meta-scheduling functionality, allowing the execution the simulated jobs in a distributed manner. The meta-scheduler can incorporate a wide-range of user-defined scheduling algorithms. Local scheduling algorithms can also be added in any user-defined experiments.

The modeled scheduled job object contains a number of parameters that are used to estimate the time needed for execution. The time needed by a job to complete a CPU-intensive operation is estimated based on a number of attributes such as the CPU power, memory and the processing time needed to complete. For the data processing jobs, these attributes depend on the type of data that the job works with. The concurrent job modeling is accomplished with the help of an interrupt mechanism: when a concurrent job starts executing on a processing unit, an interrupt mechanism is used to handle the re-estimation of the time needed for the jobs already running on that processing unit. Within the job model the user can define new jobs starting from the basic behavior provided. It can even combine several behaviors in one single composite job type. This aspect can be used to simulate a job that transfer some data, then processes it and in the end transfers back the obtained results. This behavior represents a composition between the processing and data transfer types of jobs and can be modeled using only five lines of codes. But, in the same type, the user can easily construct a job which does all the data processing according to some advanced algorithm, extending in this case one method provided by the processing data type of job.

One interesting aspect is the job decompositions being offered by the job model. The user can specify a situation where a job requests some data, and then split itself in several parallel tasks, each one processing a particular chunk of data, and in the end the obtained results are reassembled and sent back. This fork-join programming paradigm can be modeled with the dependence mechanisms being offered. So the model considers that a job is allowed to instantiate new jobs. This means that, for example, one job receives the data, splits it into chunks and instantiate processing jobs, each one supplied with one specific chunk of data. It then specify the dependence, meaning it specifies what jobs must be executed after it finishes its own execution. The dependency between jobs can be specified in the form of a DAG structure.

The simulation model also allows the evaluation of advanced scheduling algorithms such as DAG Grid strategies and decentralized scheduling solutions. A recent added extension to the default scheduler behavior for example considers the use of realistic scheduling decisions, the scheduler now considering both the underlying topology, as well as the full topology of task dependencies.

Due to simulator's openness and modularity we were able to develop simulation scenarios using custom user-provided scheduling algorithms and procedures to assigns tasks for execution on existing simulated resources. We plan to use the scheduling model and related components being provided by MONARC to evaluate through simulation the proposed decentralized dynamic resource allocation solution being proposed in this paper in the context of large scale Grid environments.

6 Conclusions

This paper presents an innovative decentralized dynamic resource scheduling solution of large scale application workflows onto distributed, heterogeneous Grid environments.

We presents the related work and demonstrate that the paper subject is a newly one and many projects consider the problem of dynamic resource allocation. According with releated work, our solution aims to optimize large-scale scientific application workflows by efficiently and effectively mapping and scheduling them onto Grid resources.

We describe the proposed solution component as a part of a framework that aims to help scientists easily deploy large-scale scientific workflow applications from a wide-range of research fields.

The novelty of this paper is represented by the decentralized architecture, together with the scheduling policies and fault management, that are specifically designed to optimize the scheduling management for workflows in case of a wide range of Grid environment.

In order to demonstrate the validity and performance of the presented scheduling solution we propose using modeling and simulation techniques.

For future work, we will test the integration between existing Grid middleware offered by gLite and a workflow engine (BPEL).

References

[1] Deelman, E., J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping abstract complex workflows onto grid environments, Journal of Grid Computing, 1(1):25-39, 2003.

[2] Yang, J., Bai, Y., and Qiu, Y. 2007. A decentralized resource allocation policy in minigrid. Future Gener. Comput. Syst. 23, 3 (Mar. 2007), 359-366.

[3] Ranjan Rajiv, Rahman Mustafizur, Buyya Rajkumar, A Decentralized and Cooperative Workflow Scheduling Algorithm, Cluster Computing and the Grid, 2008. CCGRID 8th IEEE International Symposium on Volume 1, 19-22 May 2008 Page(s):1 - 8.

[4] Walter Binder, Ion Constantinescu2, Boi Faltings, Nadine Heterd, Optimized, decentralized workflow execution in grid environments, Jurnal on Multiagent and Grid Systems, Volume 3, Number 3 / 2007, Pages: 259-279, ISSN: 1574-1702.

[5] Anirban Mandal, Ken Kennedy, Charles Koelbel, Gabriel Marin, John Mellor-Crummey: Scheduling Strategies for Mapping Application Workflows onto the Grid, 2005.

[6] Malewicz, G., Foster, I., Rosenberg, A.L., Wilde, M.A, Tool for Prioritizing DAGMan Jobs and Its Evaluation, High Performance Distributed Computing, 2006, 15th IEEE International Symposium, ISBN: 1-4244-0307-3, Page(s): 156-168.

[7] Marek Wieczorek, Andreas Hoheisel, Radu Prodan, Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem, CoreGRID Workshop on Grid Middleware, June 25-26, 2007, Dresden, Germany, Springer-Verlag.

[8] Marek Wieczorek, Mumtaz Siddiqui, Alex Villazon, Radu Prodan, Thomas Fahringer, Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid, In Proceedings of 2nd IEEE International Conference on e-Science and Grid Computing, December 4-6, 2006, Amsterdam, Netherlands.

[9] Singh, G., Deelman, E., Mehta, G., Vahi, K., Su, M., Berriman, G. B., Good, J., Jacob, J. C., Katz, D. S., Lazzarini, A., Blackburn, K., and Koranda, S. 2005. The Pegasus portal: web based grid computing. In Proceedings of the 2005 ACM Symposium on Applied Computing (Santa Fe, New Mexico, March 13 - 17, 2005). L. M. Liebrock, Ed. SAC '05. ACM, New York, NY, 680-686.

[10] Vazquez-Poletti, J. L.; Huedo, E.; Montero, R. S.; Llorente, I. M., Workflow Management in a Protein Clustering Application, Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium, May 2007, Page(s):679 - 684.

[11] Dobre, C. M., C. Stratan, MONARC 2 - distributed systems simulation, in Proc. of the 14th International Conference on Control Systems and Computer Science, Bucharest, Romania, 2003, pp. 145-149.

[12] Dobre, C.M., V. Cristea, A Simulation Model for Large Scale Distributed Systems, Proc. of the 4th International Conference on Innovations in Information Technology, Dubai, United Arab Emirates, November 2007.

[13] Legrand, I. C., H. Newman, C. M. Dobre, C. Stratan, MONARC Simulation Framework, in Proc. of the International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Tsukuba, Japan, 2003.

[14] Pop, F., C. M. Dobre, G. Godza, V. Cristea, A Simulation Model for Grid Scheduling Analysis and Optimization, in Proc. of PARELEC Conference, Bialzstok, Poland, Septe,ber 2006, pp. 133-138.

[15] Dobre, C., C. Stratan, MONARC Simulation Framework, in Proc. of the RoEduNet International Conference, Timisoara, Romania, 2004.

[16] Nathan Stone, Derek Simmel, and Thilo 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/draftggf-gridcpr-Architecture-2.0.pdf, May 2004.