Hierarchical resource management in grid computing

Korkhov, V.V.

Publication date
2009
Document Version
Final published version

Chapter 2. Resource management in Grid computing

This chapter builds the background for the research presented in this thesis giving an overview of the state of the art, issues, and related work in the field of resource management, scheduling, workload balancing, and workflow management on the Grid.

2.1 Issues of Grid resource management

The main issues addressed by Grid computing are coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations. Grid computing typically involves many resources (compute, data, I/O, instruments etc.) to solve a single, large problem that could not be performed on a single resource. Generally Grid computing requires the use of specialized middleware to reduce the complexity of integrating of distributed resources within an enterprise or a public collaboration [23, 33, 89].

As a starting point we present the definitions of the most important terms used further.

Grid resource management is defined as the process of identifying requirements, matching resources to applications, allocation those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Grid applications compete for resources that are very different in nature, including processors, data, scientific instruments, networks and other services. Complicating this situation is the general lack of data available about the current system and the competing needs of users, resource owners, and administrators of the system [89].

Grid scheduling (or metascheduling) is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. Job is defined to be anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep). We use the term resource to mean anything that can be scheduled: a machine, disk space, a QoS network, and so forth [99].

This chapter is partially based on: V. Korkhov, A. Bogdanov, L.O. Hertzberger. On Issues of Resource Management in Grid Environment, Proceedings of International Conference on Distributed Computing and Grid Technologies in Science and Education, Dubna, Russia, 2004.

Load balancing is a technique to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, and minimize response time [141].

Grid workflow is the automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service [37].

While Grids are becoming commonplace, the use of Grid resource management tools is still complicated by many open issues in the field. As stated in [89] they are:

- Multiple layers of schedulers. Grid resource management involves many players and possibly several different layers of schedulers. At the highest layer are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources. Other layers may be in between these, for example one to handle a set of resources specific to a project.

- Lack of control over resources. Grid schedulers are not local resource management systems; a Grid-level scheduler usually does not have ownership or control over the resources. Most of the time, jobs will be submitted from a higher-level Grid scheduler to a local set of resources with no more permissions that the user would have.

- Shared resources and variance. Related to the lack of control is the lack of dedicated access to the resources. Most resources in a Grid environment are shared among many users and projects. Such sharing results in a high degree of variance and unpredictability in the capacity of the resources available for use. The heterogeneous nature of the resources involved also plays a role in varied capacity.

- Conflicting performance goals. Grid resources are used to improve the performance of an application. Often, however, resource owners and users have different performance goals: from optimizing the performance of a single application for a specified cost goal to getting the best system throughput or minimizing response time. In addition, most resources have local policies that must be taken into account.

In large-scale Grid systems traditional approaches to resource management are not suitable as they attempt to optimize system-wide performance. Traditional approaches use centralized policies that need complete state information and a common fabric management policy. Due to the complexity in constructing successful Grid environments, it is impossible to define an acceptable system-wide performance metric and common management policy. Centralized management cannot support scalability of the Grid environment either. Therefore other methods are needed, and hierarchical and decentralized approaches are promising for Grid resource and operational management [22].

The main challenging problems of resource management in Grid environment are:

- Absence of centralized management: In Grid environment its impossible to register all submitted jobs, coordinate the jobs centrally etc. Consequently, traditional multiprocessor approaches to resource and task management appear to be not applicable. There exists neither global queue of jobs nor static resource pool, all the entities are distributed and dynamic.
- Dynamic heterogeneous resources with multi-domain distribution: The resources are dynamically changing availability, load and status. Permanent monitoring is necessary to ensure up-to-date information is supplied to schedulers. Resources are distributed across multiple administrative domains that should be considered by access policies to enable proper functioning of the Grid.
- Communication with local resource managers: Grid environment consists of resources under control of local resource managers. The local resource manager is a resource provider to the metacomputing system, it acts as a middle layer between Grid environment and the resource itself.
- Scalability: The number of resources can increase dramatically in global Grid environment, so applications and middleware must be able to deal with it. Centralized approach to management cannot be used, only hierarchical and decentralized can be used in globally distributed system.
- Resource reservation: Resources should support the possibility of advanced resource reservation which can guarantee the availability of the resource for the application during a specified time-frame.
- Co-allocation: This issue is related to the resource reservation and implies the possibility of simultaneous reservation of a number of resources for a Grid application.
- Data movement/relocation: A large number of Grid applications manipulate big data sets stored in data repositories. Effective data processing might need that the initial data sets might have to be relocated to a better accessible location as the application might be scheduled to a host that has low performance connection to the initial data location. A number of experiments on the selection of effective data location have been performed in [95].
- Metascheduling: One of the most important issues is distributing a Grid application across the available resources in the Grid in order to optimize application performance. Here we discuss high performance scheduling (also called application scheduling) which aims at the performance of individual applications by optimizing performance measures such as execution time. This may conflict with goals of high-throughput scheduling (also called job scheduling) which promotes the performance of the system by optimizing throughput e.g. measured by number of jobs executed by the system. The issues of metascheduling will be overviewed in detail below.

The dynamic character of the Grid has made the management of the resources more difficult. Following is a list of issues related to the dynamic character of the Grid [12, 100]:

- Resource discovery, selection and location; These activities are needed to select a set of resources to schedule the tasks of the application on. Resource discovery and location determine which resources are available to the application, resource selection operates the process of selecting candidate resources from a pool.

- Dynamic monitoring; As resources change the state, load, and availability in time, the process of monitoring is required. The monitoring data may be used in rescheduling process.

- Migration during runtime; If an application suffers from performance degradation caused by the overload of the resource used then it may be migrated to another resource with better performance after checkpointing the state of the application.

- Prediction of resource state; To be able to compose schedules of applications running on Grid resources, it is necessary to estimate the resources state at certain time-frames. So it is necessary to employ special forecasting algorithms, which can predict resources load and availability. An example of such a predicting tool is the Network Weather Service (NWS) [117], which is widely used in applications requiring future resource state information.

- Network resources; Grid applications use not only processor cycles and data storages but also network resources to communicate between distributed application components. Thus network connections performance should be also considered while composing an application schedule and performance prediction mechanisms should be also used to forecast network efficiency at certain time-frames.

One of the most common approaches for large-scale Grid systems is to introduce the Grid resource broker which mediates between producers (the resource owners) and consumers (the resource users) [62]. The resources become Grid-enabled after deploying low-level middleware. The core middleware deployed on producer's Grid resources support the ability to handle resource access authorization and permits only authorized users to access them. The user-level and core middleware on consumer's resources support the ability to create Grid enabled applications or necessary tools to support the execution of legacy applications on the Grid. Upon authenticating to the Grid, consumers interact with resource brokers for executing their applications on remote resources. The resource broker takes care of resource discovery, selection, aggregation, data and program transportation, initiating execution on remote resources and gathering results. For the operation of a computational Grid, the broker discovers properties of the resources that the user can access through the Grid information servers, negotiates with Grid-enabled resources or their agents using middleware services, maps tasks to resources (scheduling), stages the application and

data for processing (deployment) and finally gathers results. It is also responsible for monitoring application execution progress along with managing changes in the Grid infrastructure and resource failures.

The main difference between a metascheduler and a common multiprocessor scheduler is that a metascheduler does not own the resources and therefore does not have total control over them. It has to deal with a number of local subschedulers performing individual machine management. Furthermore, the metascheduler does not have a full control over the entire set of jobs on a system. The metascheduling issues include [12, 99]:

- Candidate schedule performance modeling; To run an application in Grid environment a metascheduler generates a schedule of execution, which is based on the information about available resources obtained using resource discovery, application performance model, and resources load forecasts. A number of candidate schedules are generated, and the one with the best performance (which is estimated using a specified performance metric) is selected.

- Application models; This model is used to predict application performance depending on a number of specific characteristics, resource load and a given time-frame.

- Resource state predictions; Using special forecasting techniques it is possible to predict resource characteristics as the available amount of memory, processor or network load etc. in a given time-frame. These predictions are used during candidate schedule performance modelling to evaluate the efficiency of executing applications and the schedule process as a whole.

- Scheduling policy (and performance metric); The policy is to choose the best resources from the set of candidate resources according to performance metric.

- Execution monitoring; The performance of executing application and load of resources is permanently monitored. If a task with higher priority assumes a significant part of the resources and leads to performance degradation of other tasks (including the monitored application) a decision to migrate the application may be taken.

- Application migration during runtime (ref. to dynamic resources); This feature allows preserving performance of the application in expected range even if some resources experience much higher load as predicted. Application may be checkpointed and migrated to another available resource.

- Coordinating multiple schedulers and applications (using advanced reservation mechanism); There always exist tasks that have been scheduled on the resource by different independent scheduling software in Grid environments. As centralized control in Grid is missing a collision may occur, the amount of available resource will be less than expected because all independent schedules were composed in consideration that no competing tasks would appear. To eliminate

this problem, advanced reservation of resources should be used. Thus scheduling process is composed of two parts: modeling of schedule performance with information about reservations of available resources by other tasks; reserving the resources for the optimal schedule. This mechanism requires that all jobs being allocated on resource pass the stage of reservation.

- Rescheduling; The state of resources is dynamically changing in Grid. To reflect these changes and to improve (at least preserve expected) application performance the rescheduling process is required from time to time. While rescheduling the application state and current state of the available Grid resources is estimated, which may lead to a new schedule. If the new schedule appears to be more effective than the current one (taking in account the overhead which takes place during process migration) then the tasks may be migrated according to this new schedule.

A metascheduler may also take care of minimizing the influence that new applications can create on already running applications trying to select unused resources. On the other hand, a metascheduler may include a feature, which allows to facilitate new applications to execute faster by stopping certain competing applications. This is particularly useful in case of short-run application which gain more resources by pausing some long-running and resource consuming applications for a short time which will not affect much the overall performance of these applications in long-term scale. The typical process of Grid scheduling is shown in Figure 2.1.

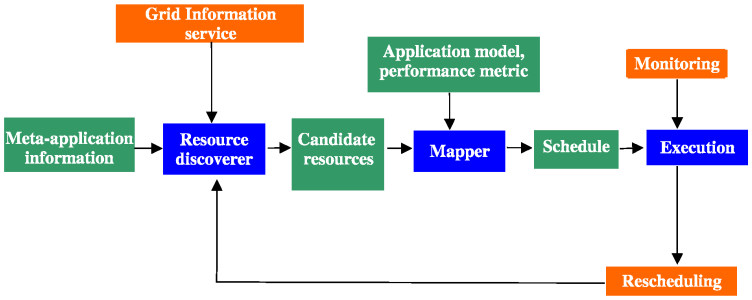

Figure 2.1: Grid scheduling: the meta-application requirements are processed by the resource discoverer that gets available resource specifications from the Grid information service; the set of suitable candidate resources is generated; mapping of meta-application to resources is performed based on the application model and the performance metric; an execution schedule is generated and executed; monitoring is performed during the execution, and in case performance drops significantly rescheduling can be performed.

2.2 Dynamic and transparent workload balancing

An immanent feature of the Grid is the heterogeneity of the resources that shall be bound together to solve a single computational problem. To achieve higher efficiency of the execution in a heterogeneous environment, a proper workload balancing strategy is necessary. Two important features allow the classification of different strategies in load balancing: dynamicity and transparency. The first load-balancing tools proposed static methods in which task allocation is fixed a priori. This technique assumes that the progress of the computation can be monitored in advance and that the processing time can be estimated. In this kind of model, the necessary resources can be anticipated and reserved. In contrast to these static strategies, dynamic load-balancing techniques address irregularities that occur at runtime. Such irregularities can include: the varying computational cost of each task, dynamic task creation or the variation in the availability of computational resources due to factors that are external to the application. In order to make different decisions, profiling techniques are used to get information on the load.

An example of a static method is used by graph partitioning. The graph partitioning model [16, 47] tries to find a partition of a graph (which is the representation of interactions between different parts of the application) into a number of disjoint subdomains. Each of them has necessarily a roughly equal number of vertices. The number of edges that are cut by the partitioning should be minimal. This technique is used by METIS [98] to provide initial load balance. To adapt the evolution of the application to load variations, iterative static approaches using repeated partitionings are applied. This approach is not easily scalable since the overhead increases with the number of processors or with the problem size increase.

Static partitioning of dynamic parallel applications (e.g. adaptive mesh computations) can cause a load imbalance between processors at runtime: the problem size and resource needs can change dynamically. Thus a dynamic scheduling strategy is required for such types of applications.

As parallel architectures evolve and adaptive computations are more widely applied, efficient scheduling strategies sensitive to application classes and parallel architectures become more required. Ideally, the architecture of a parallel system should be selected optimally for a given computation class and problem size. In current practice, the number of processors used to solve a problem is fixed for the entire lifetime of the computation. This implies a known a priori upper bound on the problem size. Another possible approach is to initially start with a fixed number of processors determined according to an initial problem size, and then change the number of processors working on the computation dynamically at runtime as the mesh is progressively refined or coarsened. The selection of the number of processors for a simulation may be based on problem size, processor speed, latency, memory availability and other factors.

To efficiently execute parallel computations on parallel distributed systems two key issues must be resolved. First, a number of processors should be selected ap-

propriately according to the problem size and expected communication overhead. Second, the workload should be balanced among the selected processors proportional to their computational power [54]. Using the maximum number of available processors in parallel is generally not optimal for all problem sizes. In addition, the problem size may vary during a simulation. In some classes of applications the problem size changes unpredictably during the solution process. In such computations it is impossible to predict the computational load and communication requirements in advance. Hence, it is difficult to estimate the optimal number of processors until immediately prior to execution of a simulation "stage" (assuming the simulation needs multiple stages). In some cases, using a larger number of processors can cause redundant communication and data movement. This is a wasteful use of computing resources and may even significantly increase the total execution time because of latency and other overheads.

The problem of assigning multiple communicating modules of a parallel program onto processing nodes of a distributed memory multiprocessor has been extensively studied in the literature. Bokhari [15] has shown that the associated mapping problem is NP-hard. Hence due to the intractable nature of the mapping problem most load balancing approaches proposed in the literature are based on heuristics. In modern adaptive computations the problem size can change dynamically with time. In such computations dynamic load balancing is critical to achieve scalability and efficient resource utilization. The extremely rich design space of dynamic load balancing allows designing a variety of mapping heuristics. Some studies of important aspects of these heuristics are presented in [3, 7, 10, 18, 24, 46, 58, 79, 97, 113].

A major class of dynamic load balancing strategies are application specific [80, 91]. These partitioning and dynamic load-balancing strategies fit more closely to application specific needs. For example, an effort by Zaki et al. [119] proposes a customized, dynamic load-balancing strategy for data-mining applications and examines the behavior of global versus local and centralized versus distributed load balancing strategies. They show that different schemes are preferable for different applications under varying program and system parameters.

On the other hand, substantial effort has been invested to develop general purpose dynamic load balancing frameworks that are applicable to a wide range of applications. The idea is to abstract load balancing details from the user by providing simple-to-use interfaces and library functions. Typically, these libraries contain efficient implementations of various parallel partitioning algorithms and data movement functions. Early attempts to develop such frameworks were based on the fact that in a network personal workstations typically have low utilization and might be dedicated to parallel computations during these periods of low utilization. An early successful project of this type is Condor [83]. Several such distributed computing frameworks allow changing the number of compute nodes dynamically according to the availability of compute nodes [120]. However, when parallel computations are executed on these frameworks the number of compute nodes is fixed during the lifetime of the program. These frameworks can be extended, or new middleware developed [59], to allow changing the number of compute nodes during the lifetime of parallel computations.

2.3 User-level scheduling

One of the inherent features of Grid resources is their dynamics, both in terms of resource parameters varying over time and in resource reliability, e.g.:

- multiple jobs from different users may run concurrently on the same worker node creating dynamically changing load,
- Computing Elements are connected by unreliable wide-area networks,
- infrastructure is asynchronously upgraded and modified, etc.

Moreover a typical Grid infrastructure is built of classical batch farms and the access to the resources is optimized for high-throughput computing what may incur arbitrary delays in job execution. Additionally Grid scheduling which involves a hierarchy of intermediate services (Resource Brokers, Computing Elements, Batch Queues) offers only very coarse, application-unspecific mechanisms to handle failures and resubmit jobs. Therefore many efforts in the Grid research have been focused on customization layers which would overcome the deficiencies of the generic infrastructure and middleware.

User-level (or application-level) scheduling is a virtualization layer on the application side. Instead of being executed directly, the application is executed via an overlay scheduling layer that runs as a set of regular user jobs and therefore it operates entirely inside user space. These overlay jobs are processed in standard queues of the resources and after being executed they provide immediate access to the actual applications. This approach is especially beneficial for the jobs with short execution times (the latency of waiting in the queue can exceed the execution time a lot) or for series of jobs, particularly in the case of jobs with input parameters that depend on the execution of the previous job in the series (which means that all the jobs can not be queued at the same time, and in the standard situation each job repeats the period of waiting in the queue on the resource) [88].

Numerous software packages have been developed as hard-wired solutions to specific applications. gPTM3D [45] is a Grid-enabled implementation of medical image reconstruction program in a master/worker model with self-scheduling. MPI-BLAST [26] is a parallel implementation of the genomic alignment search tool and may be run in-conjunction with Grid-enabled MPI implementations such as MPICH-G2 [60]. Client-server architecture has been exploited as ad-hoc solution for parallel earthquake source determination in EGEE Grid [114]. While being useful for their respective applications such implementations may not be easily reused in other contexts and outside of their application domains.

Central coordination of the Grid activities in data production, simulation and analysis in large experiments in High Energy Physics have driven projects such as Alien [96] and DIRAC [110]. These systems are implemented as end-to-end solutions deployed on the Grid at the level of Virtual Organizations (VOs). Job scheduling is based on pilot agents executing on the ordinary Grid resources and pulling tasks from central VO task queue. Resource negotiation sometimes involves additional services deployed directly at the Grid sites. Alien and DIRAC systems sup-

port thousands of concurrent jobs from hundreds of users with higher efficiency than the generic Grid middleware. However central and site services require systematic deployment and maintenance which may only be afforded by very large collaborations. Additionally such systems also tend to be application-specific and difficult to reuse.

Finally a number of reusable application-level scheduling systems have been developed in recent years. Nimrod [136] is a top-down solution for parameter-sweep applications which is able to negotiate resources using standard Globus protocols. Nimrod handles the file transfer and automatic partitioning of the computation based on the user-supplied declarations in a special-purpose language. Based on the declarations job wrapper scripts are automatically created for black-box application executables. Nimrod then controls the execution of jobs and resubmits the failed ones if necessary. The resource performance monitoring uses the Network Weather Service [117].

AppLES [13] provide generic templates for creating application-level schedulers and APST is an AppLES-based execution environment for parameter-sweep applications. XML-based specification is used for the description of the execution workflows and customization of task scheduling priorities. In this respect APST supports more flexible application execution models than Nimrod. Both approaches aim at the "farming" applications and lack sufficient support for resource-adaptive "smart" scheduling of parallel applications.

2.4 Workflow management

Within the e-Science and Grid communities, Workflow Management Systems (WMS) are the subject of intensive research since they provide an appropriate abstraction level to allow any scientist to take advantage of the capabilities of geographically distributed resources [20, 85]. Different approaches have been proposed to formulate the workflow concept and to promote the development of diverse trends in terms of the functionality and features of workflow systems.

Grid-enabled WMS(s), especially in e-Science, often have to manage geographically distributed concurrent computational processes that exchange data in a peer to peer fashion across different security domains. To achieve this goal, different approaches have been developed by various research groups. For example, the P-GRADE portal [57] operates defining dependencies between different components with respect to their execution order which is appointed by output/input file transfers between jobs. The management of workflow in P-GRADE is performed by a WMS based on Condor DAGMan [128] with extensions to provide necessary file transfers.

The recent trend towards Service Oriented Architectures (SOA) stimulated the development of another category of WMS(s) targeting the composition of services, often implemented as Web services (e.g. Taverna [92], Triana [108], or Kepler [6]). In the case of the Triana project, the proposed WMS supports the composition of workflows where components can be executed locally, as Web services, or accessed via the GAT-interface job submissions [5]. Taverna is another WMS popular

within the bio-informatics community; it is mostly oriented to work with a set of biological web services, it has a number of additional components which allow semantic annotation of workflow components and data provenance. Kepler inherits a powerful and matured framework from Ptolemy II [52]; it provides a rich library of actors including the actors for Web services composition and Grid job submission.

In e-Science, there is a large set of libraries and applications that is difficult to re-implement to fit the new Grid-computing paradigm; This software is considered as legacy code that cannot be modified. The manipulation of the legacy code is an important feature for both WMS supporting SOA based components and the ones utilizing job submissions to grid-enabled resources.

Another important feature of some e-Science applications is the demand for direct data streaming between distributed processes on the grid, especially for application domains that rely upon semi-realtime data processing. A number of workflow management systems focus on the development of robust and efficient data streaming over the Grid. Some of these systems focus on enabling data streaming capabilities, such as SCiFLo [116] where the data streaming is provided by binary channels, GriddLeS [64] where streaming functionality is supported over pipes, and Narada Brokering [93], which supports the interesting concept of hybrid streams where multiple "simple streams" are intrinsically linked. Some systems offer more complete frameworks enabling more control and steering of the streams. The UniGrids Streaming Framework [11] provides steering capabilities as well as data streaming. A UniGrids Web Service is used to control a set of available stream types, to create streams, and to manage already created ones. The Styx Grid Service (SGS) [14] is another example of a streaming framework which uses a remote service type that allows data to be streamed directly between service instances. The Styx clients can monitor progress and status through persistent connections. SGS can interoperate with other service types such as Web Services in a workflow. Other systems such as ASSIST [4] propose high-level structured parallel programming capabilities which includes a skeleton-based language, and a set of compiling tools and runtime libraries for supporting the data streaming.

To describe application workflows, a variety of workflow description languages are used. Most of the existing WMS use their own languages with different levels of complexity. For example, in Taverna data models can be represented using SCUFL (Simple Conceptual Unified Flow Language), they consist of inputs, outputs, processors, dataflow, and control flow. In addition to specifying execution order, the control flow can also be triggered by state transitions during the execution of parent processors. Askalon [32] employs AGWL (Abstract Grid Workflow Language) which provides a set of constructs to express sequence, parallelism, choice and iteration workflow structures. Teuta, another Askalon component, supports graphical specification of Grid workflow applications based on the UML activity diagram which is a graphical interface to AGWL. For a complete survey and taxonomy on existing workflow systems we refer the reader to [118].

2.5 Conclusion and research motivation

A lot of research projects address resource management and load balancing issues in distributed and Grid computing, but a single view throughout the whole hierarchy of solutions for different application layers seems to be missing. The foremost motivation of the research presented in this thesis is to study a hierarchy of resource management layers that span from simple parallel applications running on heterogeneous resources up to workflows on the Grid that include and combine a variety of lower-level resource management solutions. The concern is to embrace all different application and resource management layers within a single structure: to go through all the levels, study the peculiarities of each one, propose and examine methods for resource management and workload balancing that could be generic enough to be applied on the different layers, and finally build a prototype of an integrated solution for a complex multi-layered application on the Grid.