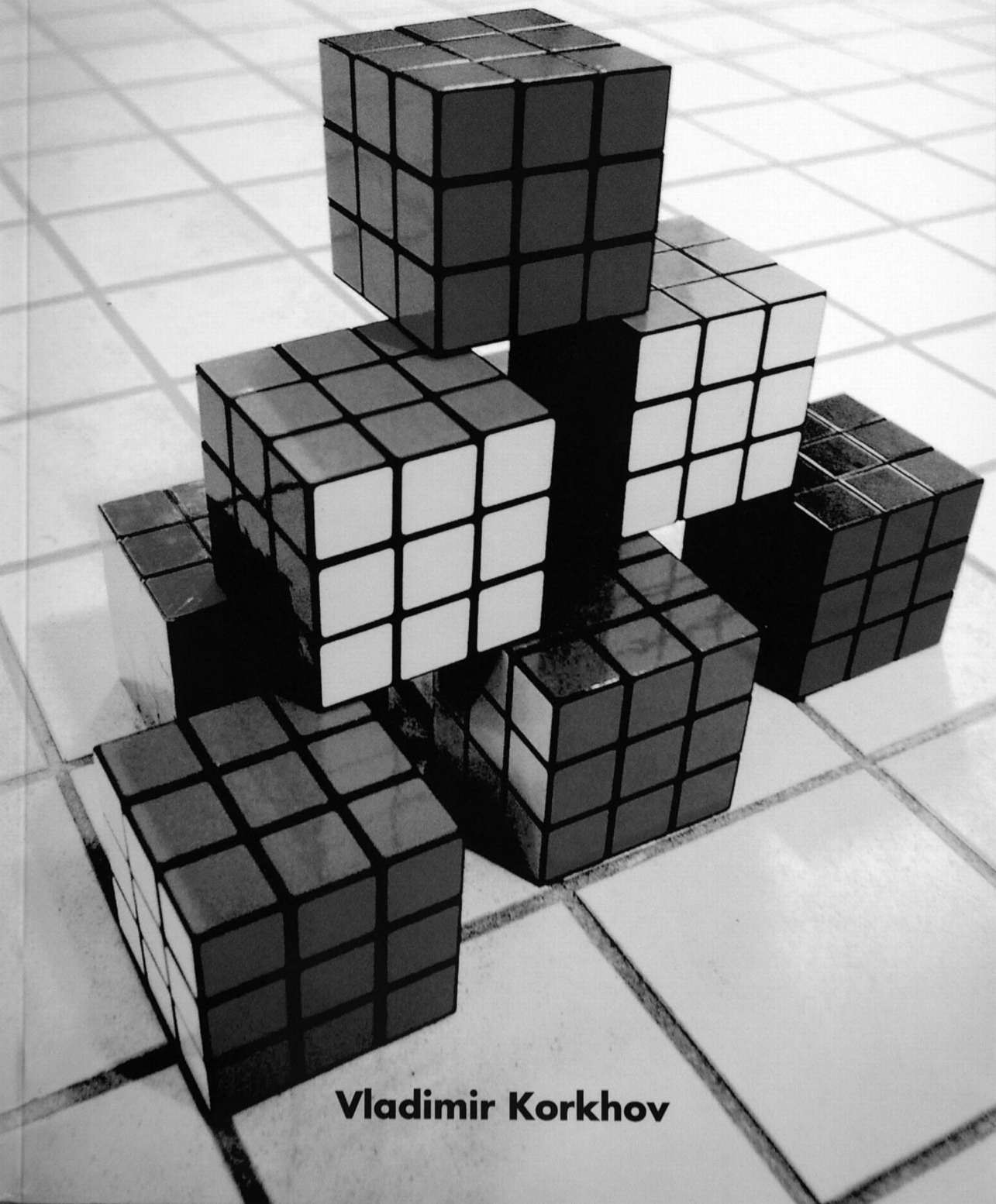
Hierarchical resource management in grid computing

Korkhov, V.V.

Publication date
2009
Document Version
Final published version

HIERARCHICAL RESOURCE MANAGEMENT IN GRID COMPUTING



Vladimir Korkhov

Hierarchical Resource Management in Grid Computing

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C. van den Boom
ten overstaan van een door het college voor promoties
ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op dinsdag 7 april 2009, te 10:00 uur

door

Vladimir Vladislavovich Korkhov

geboren te St.Petersburg, Rusland.

Promotiecommissie

Promotor:	Prof. Dr. P.M.A.Sloot
Co-promotor:	Dr. A.S.Z. Belloum
Overige leden:	Prof. Dr. P.W. Adriaans Prof. Dr. A.V. Bogdanov Prof. Dr. M.T. Bubak Prof. Dr. L.O. Hertzberger Dr. A.G. Hoekstra

Faculteit der Natuurwetenschappen Wiskunde en Informatica

The research described in this thesis was supported by the Virtual Laboratory for e-Science Bsik project and the University of Amsterdam.

Universiteit van Amsterdam

ISBN 978-90-5776-189-8
Printed by Ipskamp Drukkers B.V., Enschede

Contents

List of Abbreviations

AGWL	Abstract Grid Workflow Language.
AMR	Adaptive Mesh Refinement.
API	Application Programming Interface.
ASCI	Advanced School of Computing and Imaging.
AWLB	Adaptive WorkLoad Balancing.
BG	Basic Greedy algorithm.
BGM	Basic Greedy Modified algorithm.
CE	Computing Element.
CNP	Computation-Network Prioritized algorithm.
CORBA	Common Object Request Broker Architecture.
DAS	Distributed ASCI Supercomputer.
DOP	Degree of Parallelism.
EGEE	Enabling Grids for E-sciencE.
GASS	Global Access to Secondary Storage.
GEMLCA	Grid Execution Management for Legacy Code Applications.
GSI	Grid Security Infrastructure.
GUI	Graphical User Interface.
HCG	Homogeneous Computational Grid.
IIOP	Internet Inter-Orb Protocol.
JNI	Java Native Interface.
LA	Legacy Application.
LBE	Lattice Boltzmann Equation.
LBM	Lattice Boltzmann Method.
MD	Migrating Desktop.
MDS	Monitoring and Discovery Service.
MPI	Message Passing Interface.
NFS	Network File System.
NWS	Network Weather Service.
ORB	Object Request Broker.
PECVD	Plasma Enhanced Chemical Vapour Deposition.
PKI	Public Key Infrastructure.
PSE	Problem Solving Environment.
QoS	Quality of Service.
RDG	Russian-Dutch computational Grid.
RM	Resource Manager.
RPC	Remote Procedure Call.

RSL	Resource Specification Language.
RTS	Run-Time System.
RTSM	Run-Time System Manager.
SA	Simulated Annealing.
SOA	Service Oriented Architecture.
SPMD	Single Process Multiple Data.
SWMS	Scientific Workflow Management System.
ULS	User-Level Scheduling.
VLAM-G	Virtual Laboratory AMsterdam on the Grid.
VNC	Virtual Network Computing.
VO	Virtual Organization.
VR	Virtual Reactor.
WMS	Workflow Management System.
WSDL	Web Service Description Language.
XDR	eXternal Data Representation.

Chapter 1. Introduction

1.1 Multi-layered applications on the Grid

The rapid development of computing and networking technologies brings more power of computational resources to potential users every year. To be able to make a good use and get the most of the variety of accessible computing systems and networks a thoughtful approach is needed. On the other hand, the complexity of scientific applications, often with multi-component and hierarchical structure, grows as well. The consequence of both trends is the need for resource management strategies and methods that enable proper distribution and control of the applications on heterogeneous and dynamic resources.

This thesis addresses the issues of multi-layer resource management and workload balancing for parallel and distributed applications in heterogeneous distributed environment.

Lots of complex scientific applications operate in multi-scale, multi-time, multi-physics domains which affects the dynamics of the application requirements. The architecture of modern distributed systems features multiple resource layers that show their dynamics in performance and availability. Thus the core issue of executing scientific applications on contemporary distributed systems is in proper mapping of the applications to the environment. This thesis presents a view on the approaches to identify the layers of application's hierarchy and the ways of mapping each layer to the computing environment.

The main questions we pose and try to find answers to are:

- What are the technology invariant approaches to efficient resource management for complex parallel applications in a distributed heterogeneous environment?
- How to map different application layers to the computing infrastructure and manage these layers?
- When is a distribution of parallel applications across a set of Grid resources beneficial?

We start from exploring the basics: the case of a single parallel application running on a set of heterogeneous resources. Usually it is the lowest layer in the complex application hierarchy. A countless number of parallel applications have been developed for traditional (i.e. static homogeneous) parallel systems. The real problem in porting such applications to Grid environments is to keep up a high level of parallel

efficiency. To assure efficient utilization of Grid resources, special methods for workload distribution control should be applied. Proper workload optimization methods should take into account two aspects: (1) the application characteristics (e.g. the amount of data transferred between the processes, amount of floating point operations and memory consumption) and (2) the resource characteristics (e.g. processors, network and memory capacities, as well as the level of heterogeneity of the dynamically assigned resources). The method should be computationally inexpensive not to impose too high overheads.

We propose and evaluate the method of adaptive workload balancing that depends on dynamic characteristics of both application and resources. Traditionally there are different approaches to the issue of workload balancing: a) carefully calculate the distribution of the workload taking into account all the properties of environment and application - the task that might be time and resource consuming by itself; b) distribute the workload in a straight forward way, considering only processing power of the worker nodes at most - the fast but not very efficient way in terms of resulting performance of workload distribution. We study an intermediate approach that combines the fast calculation of initial approximate workload distribution together with the precision of getting this distribution close to the optimal one during several refining iterations.

Moving to a larger scale, more questions appear. Shall we use several parallel computers together for a single parallel application? When and why will it speedup the parallel program? To check that we evaluate the theory of parallel applications speedup in a multi-cluster environment. Here a single parallel application spans over a set of computational clusters acquiring a number available nodes at each of them. Knowing the parameters of computational environment and parallel application model it is possible to predict if such distribution is efficient. This is the extension of a simple single parallel application, a horizontal layer of expansion in the application hierarchy.

Data processing is often organized not within a single parallel application but in a set of separate components that perform information exchange only at the start and finish of the execution. We call this type of software a multi-job application, and each job can be a separate parallel application in turn. This forms another layer in a vertical application hierarchy rising over a single parallel application. To examine the efficiency of multi-job applications processing divisible workload we bring the concepts developed for adaptive workload balancing of a single parallel application to the multi-job environment and compare it to standard self-scheduling mechanism. The multi-job application execution environment combined with user-level scheduling enables such features as transparent workload balancing, resource pooling and dynamic resource selection during application run-time.

While multi-job applications described above typically perform similar type of processing on a divisible workload, another type of applications in distributed environments is the Grid workflow: a scheduled sequence of functionally decomposed communicating processes coordinated by control or data flow. This feature of multi-functionality together with the variety of execution schemes (sequenced or pipelined, scattered in time and location) builds another layer of complex application hierarchy. Our research in this field is focused on data-driven workflows with direct data stream-

ing between the workflow components. We study a model of a data-driven workflow, evaluate different methods of workflow components scheduling, describe and validate the environment developed for data-driven workflows enactment and resource management.

1.2 Hierarchical structure of large-scale distributed applications

Distributed e-Science applications can be composed of components that have different functionalities and employ different types of processing: high-performance or high-throughput computing, batch or interactive processing, computations or communications intensive tasks, distributed or local execution of application components etc. The combination of functionalities and technologies, within a single large application, forms a multi-layer structure. Typically the lowest layer represents simple computing jobs on distributed and parallel resources. The next layer is formed by a set of jobs often required to perform the exploration of large parameter space or divide working area between multitude of worker processes performing similar task. The highest layer combines all the diverse functionalities together forming a distributed workflow.

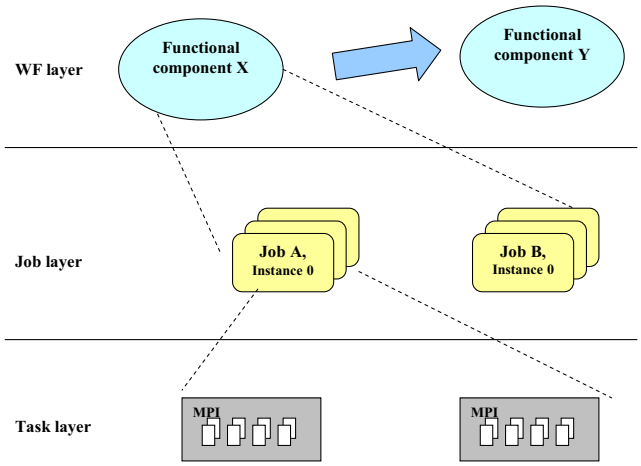


Figure 1.1: Hierarchical structure of multi-layered applications. The lowest layer represents simple computing jobs on distributed and parallel resources. The next layer is formed by a set of jobs to perform the exploration of large parameter space or divide workload between multitude of worker processes performing similar task. The highest layer combines all the diverse functionalities forming a distributed workflow.

We consider a three-layer structure of distribution and parallelism (Figure 1.1): task (e.g. single parallel application), job (e.g. parameter sweep and task farm-

ing) and workflow (functional decomposition of the application). Each layer of this structure has its own requirements to maximize the performance and enable good scalability. Thus appropriate resource management and workload balancing has to be addressed while building the distributed multi-layer and multi-component application as a whole:

- Task layer: adaptive workload balancing of parallel applications on dynamic heterogeneous resources
- Job layer: workload distribution management for job farming and parameter sweeps
- Workflow layer: execution of functionally decomposed communicating components coordinated by control or data flow

1.3 Grid architecture hierarchy

The hierarchical structure of complex applications is supplemented with the hierarchy of the Grid environment architecture. The classical architecture layering is introduced in [34]: fabric, connectivity, resource, collective, application layers. Components within each layer share common characteristics but can build on capabilities and behavior provided by any lower layer. In this thesis we refer to this classical definition of the Grid architecture but acknowledge the existence of other flavors oriented to Web services concepts and technologies.

Foster and Kesselman describe the following functionality of the classical Grid architecture layers [34]:

- Fabric layer: The Grid Fabric layer provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogs, network resources, and sensors. A "resource" may be a logical entity, such as a distributed file system, computer cluster, or distributed computer pool; in such cases, a resource implementation may involve internal protocols (e.g., the NFS storage access protocol or a cluster resource management systems process management protocol), but these are not the concern of Grid architecture. Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of sharing operations at higher levels.
- Connectivity: The Connectivity layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources.
- Resource: The Resource layer builds on Connectivity layer communication and authentication protocols to define protocols for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on

individual resources. Resource layer implementations of these protocols call Fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the Collective layer discussed next.

- Collective: While the Resource layer is focused on interactions with a single resource, the Collective layer contains protocols and services that are not associated with any specific resource but rather are global in nature and capture interactions across collections of resources. Because Collective components build on the narrow Resource and Connectivity layer "neck" in the protocol hourglass, they can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared. For example, collective layer provides:
  - Directory services that allow Virtual Organization (VO) participants to discover the existence and/or properties of resources;
  - Co-allocation, scheduling, and brokering services that allow requesting the allocation of one or more resources for a specific purpose and the scheduling of tasks on the appropriate resources;
  - Monitoring and diagnostics services to support the monitoring of VO resources for failure, overload, etc.;
  - Data replication services to support the management of VO storage resources to maximize data access performance with respect to metrics such as response time, reliability, and cost.
  - Workload management systems and collaboration frameworks also known as problem solving environments ("PSEs") that provide for the description, use, and management of multi-step, asynchronous, multi-component workflows.
- Application: The final layer in our Grid architecture comprises the user applications that operate within a VO environment.

The key problem of complex Grid computing applications is to properly correlate all their components with the Grid architecture layers. Most of these aspects can be taken care of by middleware, libraries and integrated environments for composition and execution of complex distributed applications on the Grid. The latter are usually referred as Problem Solving Environments (PSE).

1.4 Problem solving environments

A majority of modern computational engineering applications are in essence multidisciplinary problems challenging researchers from diverse knowledge domains and institutions to share their experience, methodologies, software, data, computational and instrumental resources. One of the most prominent features of such projects is that many components essential for solving one problem are distributed over multiple

organizations. Gathering all the required pieces of a problem solving puzzle in one site is often not possible, nor is it desirable.

To solve the interdisciplinary problems, several aspects shall be addressed: First, organizing a collaborative environment that would facilitate the exchange of ideas, models and codes. Second, proving the participating groups with a unified transparent and secure way to share and use computational power and other resources of the consortium. Third, developing a user-friendly environment that would guide end-users through the multiple stages of solving the problem under investigation. These stages include: properly defining a physical engineering problem, finding appropriate simulation components, choosing optimal numerical methods, accessing data bases, initiating simulations, discovering computational resources, submitting and monitoring the jobs, analyzing and archiving the results. All these tasks can be semi-automated within generic problem solving environments or virtual laboratories for e-Science [72].

A problem solving environment (PSE) provides a complete integrated computing environment for composing, compiling, running, controlling and visualizing applications. It incorporates many features of an expert system and provides extensive assistance to users in formulating problems and integrating program codes, processes, data, and systems in distributed computer environments [41]. Information sources and codes may be widely distributed, and the data processing requirements can be highly variable, both in the type of resources required and the processing demands put upon these systems. Grid technology as integrating middleware is a major cornerstone of today's PSEs [35, 50]. By offering a unified means of access to different and distant computational and instrumental resources, unprecedented possibilities and benefits can be expected. Connectivity between distant locations, inter-operability between different kinds of systems and resources, and high levels of computational performance are some of the most promising characteristics of the Grid. Grid technology provides dedicated support such as strong security, distributed storage capacity, and high throughput over long distance networks. Besides these immediate benefits, the computational resources of the Grid provide the required performance for large-scale simulations and complex visualization in collaborative environments, which are expected to become of major importance to the modern hi-tech society. Scientific communities have been investing great efforts into development of Grid-aware problem solving environments for complex applications [75, 112, 122–124, 127, 132, 134, 139].

An important flavor of PSE is a virtual laboratory which is defined as a heterogeneous distributed problem solving environment that enables a group of researchers located around the world to work together on a common set of projects [53]. A virtual laboratory allows scientists in a number of different physical locations, each with unique expertise, computing resources, and/or data to collaborate efficiently not simply at a meeting but in an ongoing way. Such environment extends and pools resources while engendering orderly communication and progress toward shared goals.

This thesis begins from outlining the challenges posed to a complex application within a PSE deployed in the Grid environment. The original PSE we use to make a start from was initially developed for traditional parallel architectures, and we immediately dive into the problems of executing parallel applications in heterogeneous distributed environment. Step by step we examine the methods and approaches that

can be used by a Grid-based PSE for execution of parallel and distributed applications. The growth of distributed application complexity leads to the formation of the concept of a Grid workflow within a Grid-enabled Virtual Laboratory [140]: a scheduled sequence of functionally decomposed distributed processes coordinated by control or data flow. Grid workflows and workflow management systems make up the most convenient, advanced and generic form of PSE on the Grid. Thus we close the cycle which started from the PSE created for traditional architectures, followed the layers of parallel processing in distributed heterogeneous environment and came to the Virtual Laboratory on the Grid.

1.5 Thesis outline

The outline of the thesis is as follows:

Chapter 2 gives the background of our research, presents the related work on resource management, scheduling and workload balancing.

Chapter 3 investigates the problem of running a dynamic parallel application on a set of dynamic heterogeneous resources. The dynamics means that the requirements of the application can vary during the runtime, and the resources can vary the performance. We propose the adaptive workload balancing algorithm (AWLB) for parallel applications with divisible workload and evaluate it on a case study of a parallel solver from the Virtual Reactor (VR) for Plasma Enhanced Chemical Vapour Deposition (PECVD).

Chapter 4 takes further steps in evaluating the possibilities of running parallel applications in distributed environment. Here a multi-cluster system is used as a resource, and a single parallel application spans over several sub-clusters. We outline the concept of Grid speedup and the theoretical approach for its evaluation introduced in [49] and perform its experimental validation. The Lattice Boltzmann Method (LBM) solver is used as a case study application; a simple model of this application is used for prediction of possible performance gain from multi-cluster distribution.

Chapter 5 applies the Adaptive WorkLoad Balancing (AWLB) method developed in Chapter 3 to multi-job applications with divisible workload. We present a hybrid resource management environment, operating on both application and system levels, developed for minimizing the execution time of parallel applications with divisible workload on heterogeneous Grid resources. The system is based on the AWLB algorithm incorporated into the DIANE User-Level Scheduling (ULS) environment.

Chapter 6 presents the VLAM-G workflow management system and its core component: the Run-Time System (RTS). The RTS is a dataflow driven workflow engine which utilizes Grid resources, hiding the complexity of the Grid from end users. Special attention is paid to the concept of dataflow and direct data

streaming between distributed workflow components. We present and evaluate the resource management algorithms and solutions for data-driven workflows used in VLAM-G.

Chapter 7 brings a summary and conclusions of the research.

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.

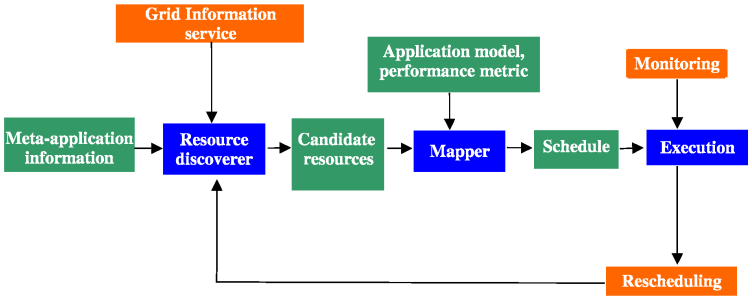


Figure 2.1: Grid scheduling: the meta-application requirements are processed by the resource discoverer that gets available resource specifications from the Grid information service; the set of suitable candidate resources is generated; mapping of meta-application to resources is performed based on the 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.

Chapter 3. Multi-layered applications on the Grid: Virtual Reactor Case Study

3.1 Introduction

The issue of running parallel applications in a heterogeneous environment became more evident after the Grid technology was introduced. Large number of computational applications and problem solving environments (PSE) developed for traditional parallel systems required modifications in order to enable efficient execution on distributed and heterogeneous environment such as the Grid.

This chapter outlines the challenges posed by the Grid to the multi-layered complex applications. A lot of applications of this kind were initially developed for traditional parallel architectures and later required to be ported to the Grid environment. We study these challenges using the Virtual Reactor (VR) for Plasma Enhanced Chemical Vapor Deposition (PECVD) as a driving application. We discuss the dependencies and functional decomposition of the VR components and finally focus on efficient execution of parallel solvers in the heterogeneous environment.

The key point is to investigate the problem of running a dynamic parallel application on a set of dynamic heterogeneous resources: the application requirements and the resource performance can vary during the runtime. We propose an adaptive workload balancing algorithm (AWLB) for parallel applications with divisible workload and evaluate it on a case study of a parallel solver from the Virtual Reactor (VR).

The question of decomposing complex applications into a multi-layered hierarchy and mapping these layers on the Grid is one of the core research questions we posed in Chapter 1. Here we present the solution for a typical example of such applications, the Virtual Reactor, and discuss the issues, theoretical approaches and practical solutions for the transfer of parallel applications from traditional homogeneous to heterogeneous dynamic resources.

This chapter is based on: V.V. Korkhov, V.V. Krzhizhanovskaya and P.M.A. Sloot. A Grid Based Virtual Reactor: Parallel Performance and Adaptive Load Balancing. Journal of Parallel and Distributed Computing, Vol 68/5, pp 596-608, DOI: 10.1016/j.jpdc.2007.08.010, Elsevier, 2008.

3.2 Virtual Reactor problem solving environment

3.2.1 Introducing Virtual Reactor

Deploying complex distributed applications on the Grid poses a challenge to computer and computational sciences, mostly due to the dynamic and decentralized nature of the Grid. The consideration of parallel computational solvers further complicates the problem because of a severe heterogeneity of Grid resources characterized by a wide range of processing power and network bandwidth. The scientific community has been investing a lot of efforts into development of Grid-aware problem solving environments for complex applications [122–124, 127, 132, 134, 139].

As a test-case of a complex multi-layer application, we selected the Virtual Reactor developed for simulation of plasma enhanced chemical vapour deposition (PECVD), a multiphysics process spanning a wide range of spatial and temporal scales [70, 75, 76]. Simulation of three-dimensional flow with chemical reactions and plasma discharge in complex geometries is one of the most resource-demanding problems in computational science, requiring both high-performance and high-throughput computing.

Grid computing technology opens up new opportunities to access virtually unlimited computational resources, and inspired many researchers to develop new methodologies and algorithms for parallel distributed applications on the Grid. The PECVD Virtual Reactor discussed in this chapter serves as a test-case driving and validating the development of the Russian-Dutch computational Grid (RDG) for distributed high performance simulation [103, 112]. The Virtual Reactor is especially suitable for execution on the Grid since it can be decomposed into a number of functional components. In addition to that, this application requires large parameter space exploration, which can be efficiently organized on the Grid [136].

The work on deployment of the Virtual Reactor on the Grid is carried out within the framework of the CrossGrid EU project [127] and the Virtual Laboratory for e-Science [140]. Some results of these efforts were reported in [75]. The RDG Grid is the successor of the CrossGrid in a sense that it uses many of the CrossGrid infrastructure services and operates as a testbed for the Virtual Reactor application. The final Grid-based Virtual Reactor problem solving environment aims at being a collaborative system, a distributed scientific workbench with advanced interaction and visualization facilities.

In this chapter we address the issue of porting an existing complex problem-solving environment (PSE) from homogeneous cluster environment to dynamic heterogeneous Grid resources. The Russian-Dutch Grid provides a strong infrastructure background for this research as it contains sites with both homogeneous and heterogeneous computing and networking resources. To build a Grid-enabled PSE based on a modular application, a proper functional decomposition of application components is required. To assure that the components - especially computational modules - are distributed efficiently, it is necessary to evaluate their performance and behaviour, tracking the dependencies on the input data and computational parameters, pinpointing the scalability and evaluating the influence of the infrastructure parameters.

A countless number of parallel applications have been developed for traditional

(i.e. static homogeneous) parallel systems. The real problem in porting such applications to Grid environments is to keep up a high level of parallel efficiency. To assure efficient utilization of Grid resources, special methods for workload distribution control should be applied. Proper workload optimization methods should take into account two aspects: (1) the application characteristics (e.g. the amount of data transferred between the processes, amount of floating point operations and memory consumption) and (2) the resource characteristics (e.g. processors, network and memory capacities, as well as the level of heterogeneity of the dynamically assigned resources). The method should be computationally inexpensive not to impose too high overheads. In this chapter, we present such a method and validate it using one of the parallel solvers of the Virtual Reactor.

3.2.2 Virtual Reactor application architecture

A complex problem-solving environment usually has a modular architecture and consists of a number of loosely or tightly coupled components [112]. Our test case, the Virtual Reactor, includes the basic components for reactor geometry design; computational mesh generation; plasma, flow and chemistry simulation; editors of chemical processes and gas properties connected to the corresponding databases; pre- and postprocessors, visualization and archiving modules [70].

The application components perform the following functions: problem description, simulation, visualization and interaction. This is schematically shown in Figure 3.1, where the simulation component encompasses two interacting parallel solvers.

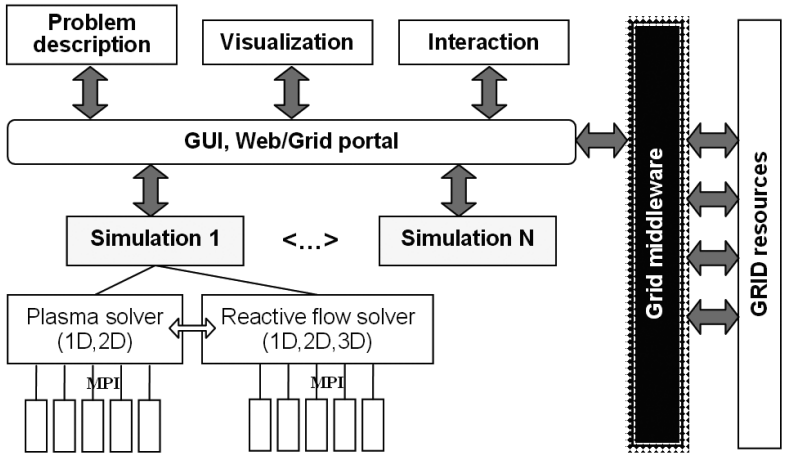


Figure 3.1: Functional scheme of the Virtual Reactor application. Interface components (problem description, visualization, interaction) are mediated by the Grid portal layer that provides access to the resources and controls the simulation execution.

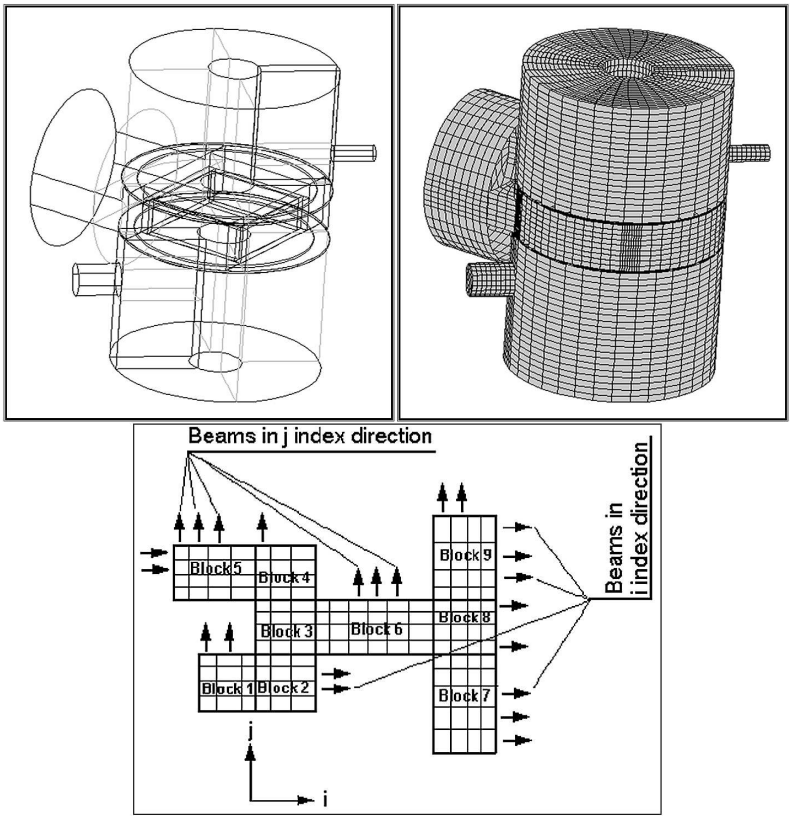


Figure 3.2: Virtual reactor geometry (meshed) and sample beams of computational cells

The core components are modules simulating plasma discharge, gas flow, chemical reactions and film deposition processes occurring in a PECVD reactor. The details on numerical methods and parallel algorithms employed in the solvers are described in [71]. The most important features relevant to the Grid implementation are as follows: for stability reasons, implicit finite volume schemes were applied, thus forcing us to use a sweep-type algorithm for solving equations in every "beam" of computational cells in each spatial direction of the Cartesian mesh (Figure 3.2).

A special parallel algorithm was developed with beams distributed among the processors. Communications are organized exploiting a Master-Slave model, where at each simulated time step the Master prepares instructions for the Slaves, sends them the data to be processed, receives the results, and processes them before proceeding to the next step. The algorithm was implemented in an SPMD (Single Process Multiple Data) model [25], using the MPI message passing interface with MPI Barrier points for synchronization. Data exchange between the Master and the Slaves is repeated every

time step, and simulation proceeds for thousands to millions of steps. In the testbed we use generic MPICH-P4 built binaries that can be executed on all the testbed machines using the Globus job submission service. To study the influence of various parameters on the simulated processes we run a number of simulations in parallel (shown in Figure 3.1 as "Simulation 1" "Simulation N" blocks) using Nimrod-G [136] as a backend for parameter sweep.

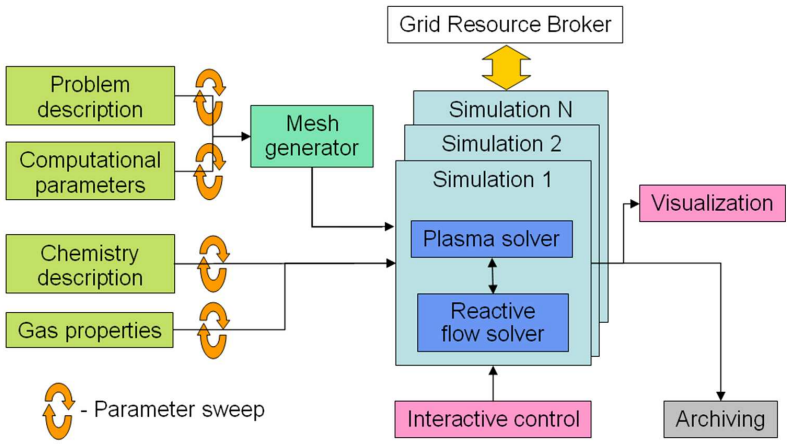


Figure 3.3: Scheme of the Virtual Reactor workflow: the arrows indicate some of the interdependencies of the components e.g, plasma and reactive flow solvers as input data use: description of the physical and chemical properties created by the editor components; parameters of the plasma chemical deposition processes; generated computational mesh; and computational parameters

In order to make the system user-friendly we need to hide the complexity of the underlying components. For that we have developed an advanced graphical user interface that seamlessly integrates the disparate distributed modules into one transparent user environment, which is presented to the user via a Web-interface and a Grid portal. The PSE architecture supports interaction on various levels: interaction with the workflow through the user interface, interactive visualization, control over the simulation processes and interactive job control on the middleware level. The basic Virtual Reactor functional components and a scheme of a typical workflow cycle are sketched in Figure 3.3. The arrows indicate some of the interdependencies of the components. For instance, the solvers for plasma and reactive flow simulations use as input data: description of the physical and chemical properties created by the editor components; parameters of the plasma chemical deposition processes; generated computational mesh; and computational parameters (e.g. Courant number, numerical scheme parameters, number of processors for parallel computing, etc.)

One of the key components of the CrossGrid architecture used for the Virtual Reactor PSE is the Migrating Desktop (MD) Grid portal, built on advanced middle-

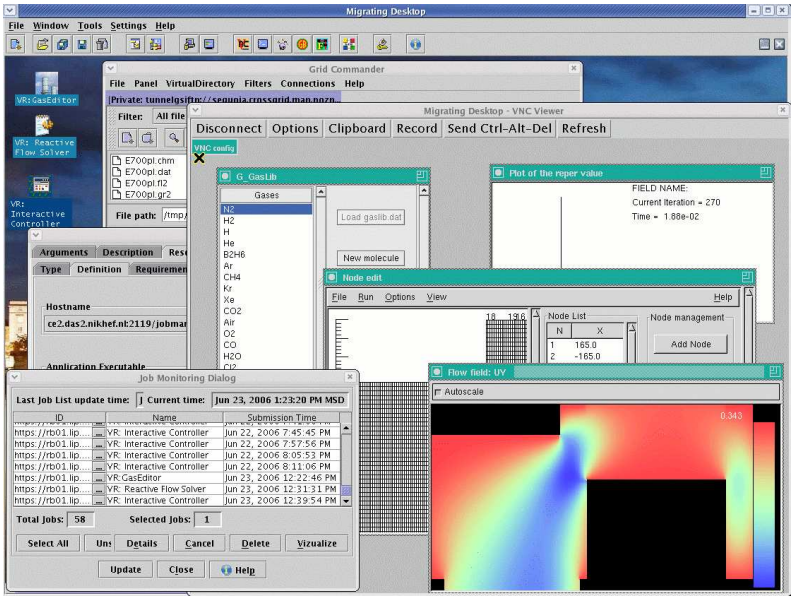


Figure 3.4: A running experiment of the Virtual Reactor on the Grid. The distributed components of the application, including the interactive editors and visualizers, are accessible via the virtual folder of the Migrating Desktop portal. Separate PSE components and simulations are run on distributed Grid resources, but the graphical output is displayed on the same virtual screen of an individual user.

ware programming and Web technologies. The MD is a user-friendly interface to the Roaming Access Server backend. It provides a transparent user working environment, independent of the system version and hardware, allowing the user to access Grid resources and local resources from remote computers, via a back-end access service. The MD portal allows the use of Public Key Infrastructure (PKI) security, based on Globus GSI and other components such as Virtual Organization servers. It allows the user to run applications, manage data files, and store personal settings, independently of the location or the terminal type. With the use of MD we achieved secure Grid access, site discovery and registration, Grid data transfer, application initialization, interactive control and visualization.

Among other services, the MD provides plug-in support for simple workflows and visualization which enables distributed execution of the Virtual Reactor PSE components within a workflow model. Figure 3.4 shows a running experiment consisting of a number of components: The Gas Editor component is executed on one of the Grid sites, which provides a chemical database. Generated by the Editor output data files are automatically transferred to a private storage or a virtual directory space operated by the MD, according to the specifications of the workflow planned (see also Figure 3.3). Next, an interactive problem description user interface is called, where

one can also define a reactor geometry and computational parameters. This user interface initiates the parallel simulation solvers, using the data files from the private storage. The solvers are submitted for execution using the Grid Resource Broker that takes care of the computational resource discovery, registration, selection and allocation. The two solvers within one simulation block in Figure 3.3 regularly exchange the variable fields in order to track the mutual influence of plasma, chemical and flow transport processes.

The interactive controller allows monitoring the execution by visualizing intermediate calculation results. This feature is enabled by periodic retrieval (refresh) of the simulation output files provided by the MD. When the simulation is complete, the result files can be retrieved from the private storage or moved to the experiments archive. All the interactive graphical components use common virtual display, so geographically distributed tasks appear on the same screen. One of the MD utilities gives the possibility to monitor the execution of the submitted tasks using the special monitoring tool shown in the left lower corner of Figure 3.4. A list of submitted jobs is shown, each provided with detailed status information. The output of the simulations goes through the post-processing module to optimization, visualization and results archiving components.

Collaborative work within the Virtual Reactor PSE is provided in two ways: First, different users can start several instances of the PSE, create different numerical experiments and run them independently on the testbed computational resources, replicating the simulation components and sharing access to databases, archives and other resources. Second, different users can connect to one common virtual display, thus having the same graphical output and interactive steering capabilities. This case sets some additional requirements to organizing the fault tolerance of the PSE components. Basic sharing features are incorporated into the VNC which is used to provide the common virtual display (i.e. blocking the pointer while simultaneously used by another user). Sharing on the program logic level requires additional modifications: blocking commands and semaphores should be incorporated into the shared interactive modules.

To provide efficient execution of a parallel application on heterogeneous resources, it is needed to clearly understand the application performance dependencies on homogeneous resources first. This gives an insight into the application scalability, induced fractional overhead, dependencies of the amount of the communications and calculations on the number of processors used, etc. The results of such tests can help estimating and predicting the behaviour of the application on heterogeneous resources, thus simplifying the adaptation process.

3.2.3 Resource infrastructure: Russian-Dutch Grid testbed

Generally the infrastructure of a site within a Grid testbed can be of one of the following types depending on the underlying resources:

I. Traditional homogeneous computer cluster architecture: homogeneous worker nodes and uniform interconnection links;

II. Homogeneous worker nodes with heterogeneous interconnections;

III. Heterogeneous worker nodes with uniform interconnections;
IV. Heterogeneous nodes with heterogeneous interconnections.

A complete Grid infrastructure is always of the Type IV, characterized by severe heterogeneity with a wide range of processor and network communication parameters. As we show later in this chapter, the type of resources allocated to a parallel application significantly influences its performance, and different load balancing techniques shall be applied to different combinations of the resources. Currently the Russian-Dutch Grid (RDG) testbed consists of six sites with different infrastructures: Amsterdam-1 (contains 3 nodes, 4 processors) - Type IV; Amsterdam-2 (32 nodes, 64 processors) - Type I; St. Petersburg (4 nodes, 6 processors) - Type IV; Novosibirsk (4 processors) - Type II; Moscow-1 (13 nodes, 26 processors) - Type I; Moscow-2 (12 nodes, 24 processors) - Type I. The RDG testbed is built with the CrossGrid middleware [76] based on the LCG-2 distributions and sustains the interoperability with the CrossGrid testbed. More detailed information on the RDG testbed can be found in [74]. The RDG Virtual Organization (VO) is included into the CrossGrid VO, thus allowing the RDG certificate holders to access some of the CrossGrid resources and services. The CrossGrid testbed consists of 16 sites with the infrastructures of all 4 types.

3.3 Adaptive workload balancing on heterogeneous resources: theoretical approach

3.3.1 Resource and application parameters

One of the factors that determine the performance of parallel applications on heterogeneous resources is the quality of the workload distribution, e.g. through functional decomposition or domain decomposition. Optimal load distribution is characterized by two things:

- all processors have a workload proportional to their computational capacity;
- communications between the processors are minimized.

These goals are conflicting since the communication is minimized when all the workload is processed by a single processor and no communication takes place, and distributing the workload inevitably incurs communication overheads. Thus it is needed to find a trade-off and define a metric that characterizes the quality of workload distribution for a parallel problem. One of the existing methods to measure this quality is to introduce a cost function reflecting the application execution time. Minimization of this function corresponds to minimization of the application runtime. The function should be simple and independent of the details of the code. The generic form of such a cost function is [28, 29, 38]:

$$H = H_{calc} + \psi H_{comm} \tag{3.1}$$

where H_{calc} is minimized when the workload distribution among the processors is proportional to the processors capacity (or equal in case of homogeneous processors); H_{comm} is minimized when the communication time is minimal; and ψ is a

parameter that can be varied in order to tune the balance between the calculation and communication terms. This parameter is dependent on the characteristics of both the application requirements and the resources capabilities. The main generic parameters that define a parallel application performance are:

- An application parameter $f_c \sim N_{comm}/N_{calc}$ (N_{comm} and N_{calc} are the amounts of application communications and computations per processor respectively);
- A resource parameter $\mu \sim t_{comm}/t_{calc}$ (t_{comm} is a typical time taken to communicate a single byte between the processors, t_{calc} - typical time required to perform a generic floating point calculation). The product of these two parameters $f_c\mu$ is often called the fractional communication overhead [38].

The goal of load balancing is to minimize the cost function (3.1). The parameter ψ in this expression is an aggregated value based on the application and resource specific parameters f_c and μ . The knowledge of these application and resource properties allows constructing an appropriate form of parameter ψ and performing suboptimal load distribution [28]. However in most real-life complex simulation problems, it is not possible to theoretically calculate the application specific parameter f_c with a reasonable precision. Even a detailed analysis of the algorithms and codes can fail in many practical cases when the code has multiple logical switches and completely different algorithms and computational schemes are used while solving a problem, depending on the initial conditions and computational parameters. Estimation of the resource-specific parameter μ also poses a challenge on heterogeneous Grid resources, since there is a multitude of processors with the ratio of communication to computation performance spanning a few orders of magnitude. Moreover, the Grid exhibits dynamic network and processor performance, therefore static domain decomposition fails to provide realistic estimations and consequently the optimal load distribution, To ensure efficient load balancing of a parallel application on the Grid, it is necessary to estimate the ψ parameter experimentally. There are two possible approaches to that: (1) directly measure the lumped value of ψ for the application on the allocated resources and (2) separately benchmark the resources, estimate μ and then find out the application-specific parameter f_c that would provide an optimal workload distribution on a given set of resources. The first approach requires serious intrusion into the application code. This is certainly not desirable, especially when targeting to build a generic load balancing system which tries to abstract from the application specific issues. Thus we have chosen the second approach which is more generic and requires minimal modifications in the application code.

3.3.2 Adaptive workload balancing algorithm

We have developed a meta-algorithm for adaptive load balancing on heterogeneous resources based on benchmarking the available resources capacity (defined as a set of individual resource parameters $\vec{\mu} = \{\mu_i\}$) and experimental estimation of the application parameter f_c. The algorithm ensures efficient load distribution, thus minimizing the application execution time. The cost function in our case is the experimentally

measured execution time, which depends on the distribution of the workload between the participating processors. The target is to experimentally determine the value of f_c that provides the best workload distribution, i.e. minimal runtime of the application mapped to the resources characterized by a parameter set $\vec{\mu}$. The outline of the load balancing meta-algorithm is as follows:

1. Benchmark the resources dynamically assigned to a parallel application; measure the resource characteristics that constitute the set of resource parameters $\vec{\mu}$ (available processors power, memory and links bandwidth).

2. Estimate the range of possible values of the application parameter f_c. The minimal value is $f_c^{min} = 0$, which corresponds to the case when no communications occur between the parallel processes of the application. The maximal value can be calculated based on the following reasoning: For the parallel processing to be efficient, that is to ensure that running a parallel program on several processors is faster than sequential execution i.e. $T_p < T_1$ where T_p is the execution time on p processors. For homogeneous resources this condition leads to:

$$f_c^{max} = \frac{(p-1)}{p\mu} \tag{3.2}$$

   Analogously, for heterogeneous resources the upper limit can be found as:

$$f_c^{max} = \frac{(p-1)max(t_{calc}^i)}{min(t_{comm}^i)p} \tag{3.3}$$

3. Search through the range of possible values of f_c in $[0..f_c^{max}]$ to find the optimal value f_c^* minimizing the application execution time. For each value of f_c calculate the corresponding load distribution based on the resource parameters $\vec{\mu}$ determined in step 1 (details on calculating the load distribution weights will follow this algorithm). With this distribution perform one time step (iteration), and measure the execution time – the target optimization function. Selection of the next value of f_c can be done by any optimization method for unimodal smooth functions; for instance a simple line-search method can be used.

4. Execute further calculations using the discovered f_c^*.

5. In case of dynamic resources where performance is influenced by other factors (which is generally the case on the Grid), a periodic re-estimation of resource parameters $\vec{\mu}$ and load re-distribution is performed during the run-time of the application. Re-balancing is invoked if the application performance over the last step drops below a certain user-defined threshold (expressed as a relative change in the execution time).

6. If the application is dynamically changing (for instance due to adaptive meshes, moving interfaces or different combinations of physical processes modelled at different simulation stages) then f_c^* must be periodically re-estimated on the same set of resources.

Periodic re-estimations in steps 5 and 6 can be easily organized for iterative, time-stepping or discrete-event simulations. After each step (iteration) the resource characteristics are automatically updated and in case of significant application performance drop (below the user-defined threshold), the next step starts with an adapted load distribution. For other types of applications (continuous and not divided into logical steps), load re-balancing can be organized via check-pointing, which is a necessary capability for efficient fault-tolerant computing on the Grid.

The combination of $\vec{\mu}$ and f_c determines the distribution of the workload between the processors. To calculate the amount of the workload per processor, we assign a weight-factor to each processor according to its processing power, memory and network connection. A similar approach was applied in [109] and in [104] for heterogeneous computer clusters, but the mechanism for adaptive calculation of the weights and application requirements was not developed there. Moreover, the tools developed for cluster systems can not be used in Grid environments without modifications since static resource benchmarking is not suitable for dynamic Grid resources, where the weights shall be calculated every time the solver is started on a new set of dynamically assigned processors.

Let us assume that for the i^{th} processor: p_i is the available processor performance, m_i is the available memory and n_i - available network bandwidth to the processor. An individual resource parameter μ_i then can be represented using the values of p_i, m_i, n_i. In a simple case when memory is considered only a constraining factor (and not driving the load balancing process) it is $\mu_i = p_i/n_i$. This resource parameter is widely used in scientific applications where the most important factor is the ratio of the computational power to the network bandwidth. In a more general case, two parameters shall be considered, μ_i and m_i. And for the memory-intensive applications, the ratio of the available memory to the network capacity of that processor m_i/n_i should play the major role in resource evaluation.

3.3.3 Weighting factors and workload distribution

To reflect the processor capacity, we introduce a weighting factor w_i for each processor. It determines the final workload for a processor given by: $W_i = w_i W$, where W is the total workload. To determine the weighting factors we introduce parameters c_p, c_m, and c_n that reflect computational, memory and communication requirements of the application. Then the weight of each processor is estimated using the following expression:

$$w_i = c_p p_i + c_m m_i + c_n n_i, \sum w_i = 1. \tag{3.4}$$

This weighting factor w_i reflects a relative capacity of the resources according to the estimated infrastructure parameter $\mu_i = \mu(p_i, m_i, n_i)$ and the application parameter f_c. The infrastructure parameters μ_i can be determined by a set of benchmark runs before the actual calculations start (but after the resources have been assigned to the application). Searching through f_c with fixed values of μ_i gives us the optimal value f_c^* which corresponds to the optimal mapping of the workload to the resources.

The parameters c_p, c_m, and c_n depend not only on the application characteristics

but also on the heterogeneity of the resources. Let us analyse how these parameters and weighting factors w_i are related to f_c and μ_i. Consider a traditional situation when memory is only a constraining factor ($c_m = 0$). Then parameters c_p and c_n should be proportional to the amount of application communications (computations) and the heterogeneity factors:

$$c_p = N_{calc}\phi_{proc}; c_n = N_{comm}\phi_{net} \tag{3.5}$$

Here ϕ_{proc} and ϕ_{net} are heterogeneity metrics of processors and network links that help additionally balance weighting according to the level of resource heterogeneity. Considering a natural homogeneous case when the workload should be divided equally between the processors as a staring point and involving the standard deviation of the set of normalized dimensionless resource parameters as the correction for heterogeneous case the following expressions are derived:

$$\phi_{proc} = \frac{p_{avg}^2 + \sum_{i=1}^{N}(p_i - p_{avg})^2}{Np_{avg}^2}, \phi_{net} = \frac{\sum_{i=1}^{N}(n_i - n_{avg})^2}{Nn_{avg}^2} \tag{3.6}$$

where N is the number of participating processors. Substituting expressions (3.5) for c_p and c_n in 3.4, the weights can be re-written as:

$$w_i = N_{calc}\phi_{proc}p_i + N_{comm}\phi_{net}n_i = N_{calc}\phi_{proc}(p_i + n_i f_c \phi_{net}/\phi_{proc}) \tag{3.7}$$

Defining $\phi = \phi_{net}/\phi_{proc}$ as an aggregate heterogeneity metric of resources, keeping in mind that $\mu_i = p_i/n_i$, and omitting the constant multiplier $N_{calc}\phi_{proc}$ before the brackets (which will be cancelled while calculating the normalized dimensionless weights), yields:

$$w_i = p_i(1 + f_c\phi/\mu_i), \tag{3.8}$$

Normalized dimensionless weights will be: $\acute{w}_i = w_i/\sum w_i$

Knowing the fractional overhead of the application and the heterogeneity level of the resources, we can optimize the workload distribution using this fast weighting technique. To evaluate the efficiency of the workload distribution we introduce the load balancing speedup Θ:

$$\Theta = \frac{T_{non-balanced}}{T_{balanced}}100\% \tag{3.9}$$

where $T_{non-balanced}$ is the execution time of the parallel application without the load balancing, and $T_{balanced}$ is the execution time using load balancing on the same set of resources. This metric is used to estimate the f_c^* that provides the best performance on given resources, i.e. the largest value of Θ in a given range of f_c. In a non-trivial case we expect to find a maximum of Θ and thus an optimal f_c^* for some workload distribution. Finite and non-zero value of f_c^* means that the application requirements best fit the resources in this particular workload distribution, which minimizes the total run-time of the application. The case of $f_c^* = 0$ while $\phi \neq 0$ means

that the application is totally computation dominated i.e. there is no communication between different processes, and the optimal workload distribution will be proportional only to the computational power of the processors. The case of $\phi_{net} = 0$ means that we consider the resource infrastructure of heterogeneous processors connected by homogeneous network links and the value of f_c does not influence the distribution, which shall be proportional only to the processing power.

In the discussion presented above while deriving eq. 3.4 , we considered a simple case when memory requirements only put a Boolean constraint to the allocation of processes on the resources: either there is enough memory to run the application or not. But it can play a role in the load balancing process being one of the determining factors of application performance. This is the case for applications that are able to control memory requirements according to the available resources. In this case there will be additional parameters analogous to f_c and μ_i (or these functions will be more complex), but the idea and the load balancing mechanism remain the same.

3.4 Performance of the Virtual Reactor on the Grid

3.4.1 Definitions

Speedup $S(p)$ - is defined as the ratio of the execution time of the best possible sequential algorithm on a single processor to the parallel execution time of the selected algorithm on a p-processor parallel system under the assumption that both algorithms solve the same problem:

$$S(p) = \frac{T(1)}{T(p)} \tag{3.10}$$

Relative speedup $S(p)$: Eq. 3.10 is often called absolute speedup. Since it is difficult to find the "best" sequential algorithm to solve a problem, the concept of relative speedup is introduced. The execution time of the parallel program is equal to $T(1)$ if the parallel program runs on a single processor. Because $T(1)$ and $T(p)$ are both based on the same parallel program, the ratio of $T(1)$ to $T(p)$ is called the relative speedup. In the following discussions, speedup is assumed to have the meaning of the relative speedup by default.

Efficiency ε is defined as the ratio of the speedup to the number of processors used at the same time for the parallel computations:

$$\varepsilon(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)} \tag{3.11}$$

3.4.2 Speedup of the chemistry-disabled and chemistry-enabled simulations

The measurements were carried out on all the Grid sites within the RDG testbed. The parallel solver showed a noticeable speedup on the Moscow and Amsterdam sites of Type I (homogeneous cluster with uniform communication links). Figures 3.5 and 3.6

demonstrate the total execution time and speedup of the parallel solver for different types of simulation: A chemistry-disabled "light-weighted" simulation (Figure 3.5) and a chemistry-enabled "heavy" simulation (Figure 3.6).

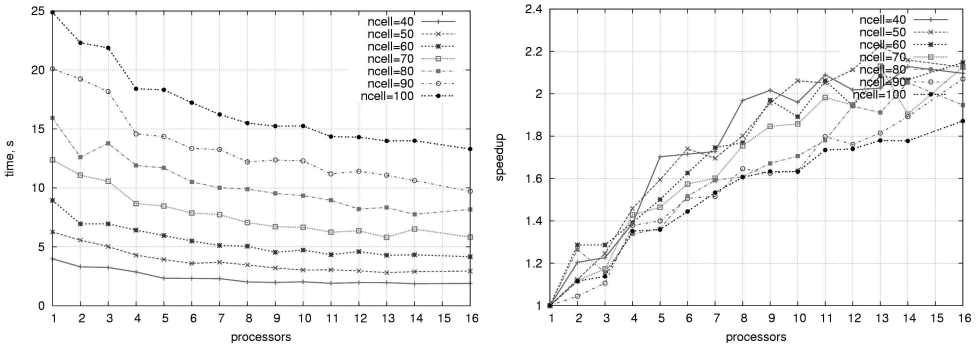


Figure 3.5: Light-weight (no chemistry) simulation: total execution time and speedup for different computational mesh sizes

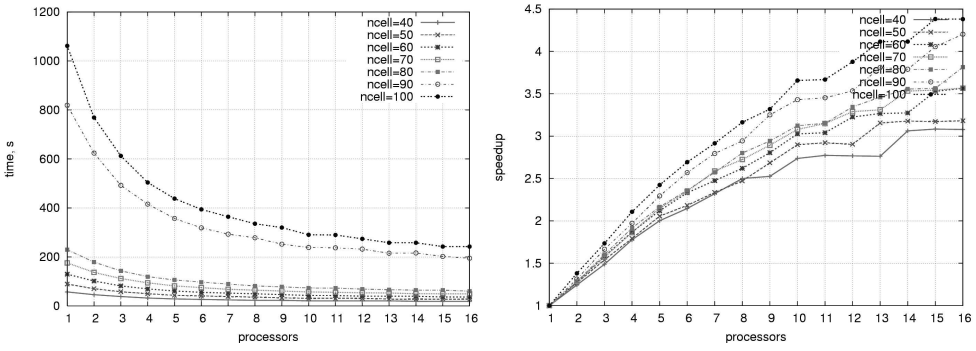


Figure 3.6: Chemistry-enabled simulation: total execution time and speedup for different computational mesh sizes

We observe different trends of the solver performance: for the light-weighted simulation, the speedup decreases with the increase of the mesh size (see the different curves in Figure 3.5, right), while for the chemistry-enabled simulation, the speedup increases with the problem size increase (Figure 3.6). The different absolute values of the speedup in Figures 3.5 and 3.6 are mostly dependent on the resources: The results presented in Figure 3.5 were obtained on the Moscow-1 site with slow interprocessor links, and Figure 3.6 shows the results of the Amsterdam-2 site with fast communications. Different trends in the speedup dependency on the problem size are discussed and explained in detail in Sections 3.4.3 and 3.4.4.

Obviously, the solver has a significant sequential part that can not benefit from parallelization, nevertheless the same parallel solver tested on homogeneous Grid sites with a higher ratio of the inter-process communication bandwidth to the processor performance achieved much higher speedups, for instance on lisa.sara.nl with Infiniband interconnections it was 3 times higher for the large problem size simulations. The type of MPI library also influences the parallel efficiency of a program: a specialized library optimized for the native communication technology (e.g. MPICH-GM for Myrinet communications on das2.nikhef.nl) increases the speedup up to 50 percent compared to the generic MPICH-P4 or MPICH-G2.

3.4.3 Computation to communication ratio

In Figure 3.7 the total execution time is presented along with the contributions of calculation and communication. For a smaller computational mesh (Figure 3.7 left), the communication time makes a relatively small contribution to the total execution time even for a large number of processors involved. For a larger mesh (Figure 3.7 right), communication makes up to 30% of the execution time. This result confirms that the network bandwidth is not sufficient for this type of problem (see also the explanations to Figure 3.6).

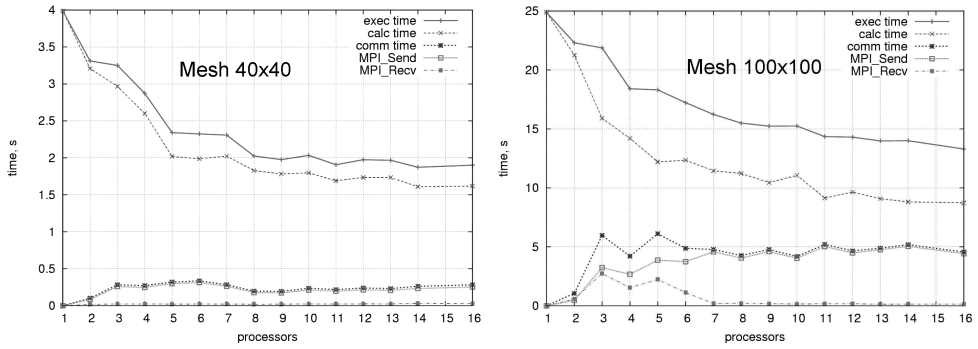


Figure 3.7: Total execution time and contributions of the calculation and communication depending on the number of processors for different computational mesh sizes (light-weighted simulation)

As it was mentioned in the previous Section, the solver can simulate the chemical and plasma processes within the reactor along with the gas flow. Figure 3.8 demonstrates the ratio of computation to communication time for different mesh sizes with different types of the simulation. The higher the ratio is, the less communications are required, which obviously offers a better parallel efficiency and application scalability. The ratios in Figure 3.8 explain the different speedup trends observed in Figures 3.5 and 3.6 for chemistry-enabled and chemistry-disabled (light-weighted) simulations. From the presented graphs we can see that the behaviour of this ratio does not depend on the mesh size for the chemistry-enabled simulations, while this behaviour for the

light-weighted simulations significantly differs for small and large mesh sizes. For a small mesh size, the ratio stays decently high, and for 6 processors and more it reaches the level of the chemistry-enabled simulations. For a larger mesh, the computation/-communication ratio for the no-chemistry simulations is very low, thus diminishing the overall parallel efficiency.

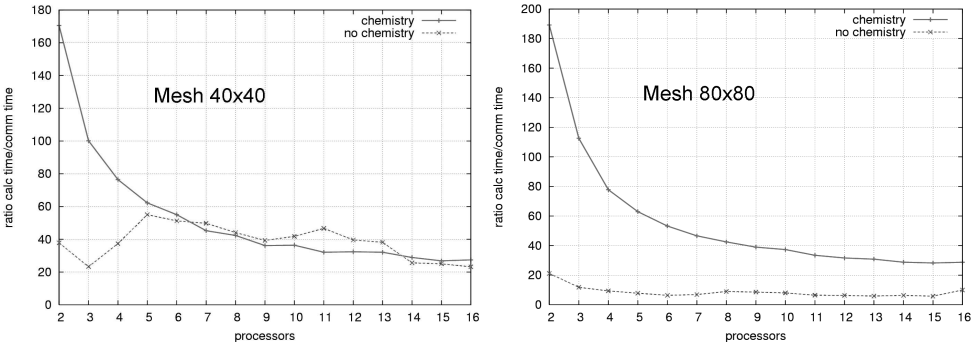


Figure 3.8: The ratio of the computation to communication time for chemistry-enabled and light-weighted simulations

3.4.4 Homogeneous resources: results and discussion

The results presented in Section 3.4.2 show that the parallel speedup is lower for a larger problem size for the simulations with no chemistry (see Fig. 3.5). This fact indicates that the ratio of the inter-process communication bandwidth to the processor performance was not high enough for light-weight problems with relatively small number of operations per computational cell. It means that for an optimal usage of computing power, a large number of processors for one parallel run shall only be used for relatively small computational meshes. Thus the communication technology puts a limit to the scalability of the solver for this problem type. On the other hand, the simulation of the flow with chemical processes shows higher speedup with larger meshes (see Fig. 3.6). Here the amount of computations brought by simulating the chemistry changes the behaviour of the solver qualitatively. This leads us to the conclusion that different resource allocation strategies should be applied for different types of simulation and meshes used.

3.4.5 Heterogeneous resources: results and discussion

To illustrate the approach described in Section 3.3 we present the results obtained for different types of simulation (chemistry-disabled and enabled) of a reactor geometry with 10678 cells on the St. Petersburg Grid site. This site is heterogeneous in both the CPU power and the network connections of the processors (Type IV). There are two 1.8 GHz nodes (nwo1.csa.ru, nwo2.csa.ru) and two dual 450 MHz nodes (crow2.csa.ru,

crow3.csa.ru), all having 512 MB RAM. One of the dual nodes (crow3.csa.ru) is placed in a separate network segment with 10 times lower bandwidth (10 Mbit/s against 100 Mbit/s in the main segment). The load balancing tests were performed with a moderate-size problem which does not pose restrictions on required memory, thus the memory influence parameter c_m was reduced to zero and the exploration was done for the application parameter f_c. The link bandwidth between the Master and Slave processors was estimated by measuring the time of MPI_Send transfers of a predefined data block (with the MPI buffer size equal to 10E+6 of MPI_DOUBLEs) during the solver execution, after the resources have been allocated. In these measurements the same logical network topology was used as employed in the solver. The CPU power and available memory were obtained by a function from the `perfsuite` library [77]. To validate the approach presented in Section 3.3 we applied the workload balancing technique for a single simulation running on different sets of heterogeneous resources. The estimation of performance for different possible values of the parameter f_c (hence different weighting and workload distribution) was carried out. For one simulation type we expect to obtain approximately the same value of the parameter f_c^* (that provides the best performance, see Section 3.3) on different sets of resources. Figure 3.9 (left) illustrates the load-balancing speedup Θ achieved by applying the workload balancing technique for different values of the parameter f_c on several fixed sets of heterogeneous resources for a light-weighted (chemistry-disabled) simulation. In Table 1 we summarize the combinations of processors dynamically allocated in 4 tests (different sets of resources) and the weights assigned to each processor for the values of f_c^* providing the best execution time, thus the maximal balancing speedup (see Figure 3.9 left).

Resource sets	Weights								
	nwo1 1.8GHz 100Mb/s	crow2/1 450MHz 100Mb/s	crow3/1 450MHz 10Mb/s	crow2/2 450MHz 100Mb/s	nwo2 1.8GHz 100Mb/s	crow3/2 450MHz 10Mb/s	ϕ_{proc}	ϕ_{net}	$\Theta, \%$
set I (3 proc)	0.58	0.27	0.15	-	-	-	0.62	0.61	196
set II (4 proc)	0.45	0.22	0.11	0.22	-	-	0.64	0.50	182
set III (5 proc)	0.31	0.15	0.08	0.15	0.31	-	0.59	0.44	201
set IV (6 proc)	0.28	0.16	0.06	0.16	0.28	0.06	0.62	0.61	207

Table 3.1: Balancing weights providing the best load balancing speedup for different sets of resources.

Figure 3.9 (left) shows that for a given simulation the best performance is delivered by weighting the resources with the value of $f_c \approx 0.3 - 0.4$. Noticeably, this corresponds to the value obtained for this simulation during the preliminary analysis on homogeneous resources (compare to results for similar simulations in Section 3.4.3, Figure 3.8). The results show that the algorithm gives the increase of the balancing speedup Θ up to 207 percent compared to the initial non-balanced version of the code

(with homogeneous workload distribution) on the tested resource sets. We can see that the distribution of the workload proportional only to the processor performance ($f_c = 0$) also gives a significant increase of the performance, but the introduction of the dependency on application specific communication/computation ratio f_c and the resource infrastructure parameters μ_i adds another 40 percent to the balancing speedup Θ.

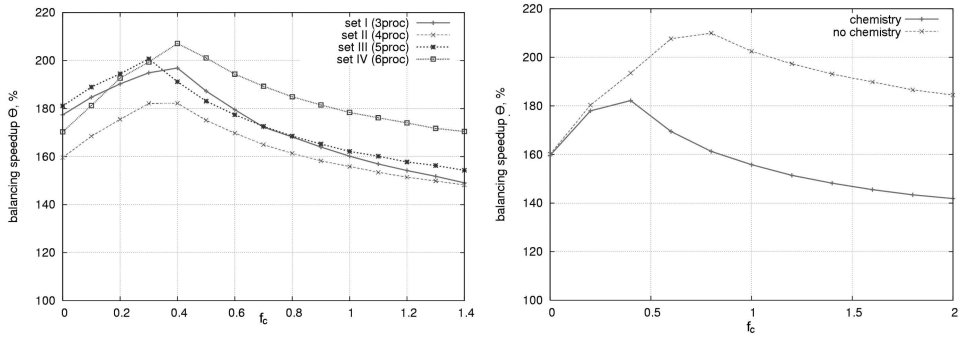


Figure 3.9: Dependency of the balancing speedup Θ on the parameter f_c. Left: single simulation on different sets of resources. Right: different types of simulation on the same set of resources.

Figure 3.9 (right) shows the dependency of the balancing speedup Θ for different types of simulation (chemistry enabled or disabled) on the same set of resources (set III from Table 3.1). The chemistry-disabled simulation has a higher communication/-computation ratio (as was shown also in Section 3.4.3, Figure 3.8). This is clearly seen in the experimental results where chemistry-disabled simulation obtains the highest balancing speedup Θ at higher values of f_c. Moreover, the gain in the balancing speedup (maximal value of Θ) is higher for the simulation with a larger fraction of communications. These results illustrate that the introduced algorithm for resource adaptive workload balancing can bring a valuable increase in the performance for communication-intensive parallel programs running on heterogeneous resources.

3.5 Synthetic application and experimental setup

To evaluate the performance of the proposed load balancing technique for generic cases, we developed a "synthetic" application modeling different types of parallel applications mapped to the resources of various capacity and levels of heterogeneity. From a technical point of view, this synthetic application is an MPI program running on a homogeneous computer cluster system. Flexible configuration capabilities allow tuning the communication-computation ratio f_c within the application, and designing the communication logical topology (i.e. the patterns of interconnections between the processes). The latter gives the possibility to model different connectivity schemes,

e.g. Master-Worker, Mesh, Ring, Hypercube etc. The value of the application parameter fc is controlled by changing the total amount of calculations to be performed and the total amount of data to be sent between the nodes. The underlying heterogeneous resources are modeled by imposing extra load on the selected processors or links, thus reducing the capacity available for the application.

The load balancing algorithm was implemented as an external library using the MPI message passing interface, and the synthetic application (also an MPI program) has been instrumented with this library. We use this experimental setup to examine how a specific parallel application defined by a combination of communication/computation ratio fc and communication logical topology will behave on different types of heterogeneous resources, and what types of applications can show the best performance on a given set of resources. To validate the synthetic simulator, we modeled and analyzed the performance of the Virtual Reactor solvers on sets of resources similar to those used in our previous experiments on the RIDgrid [66, 68]. The experiments were carried out on the DAS-2 computer cluster [130], using MPICH-P4 implementation of MPI.

3.5.1 Load balancing speedup for different applications

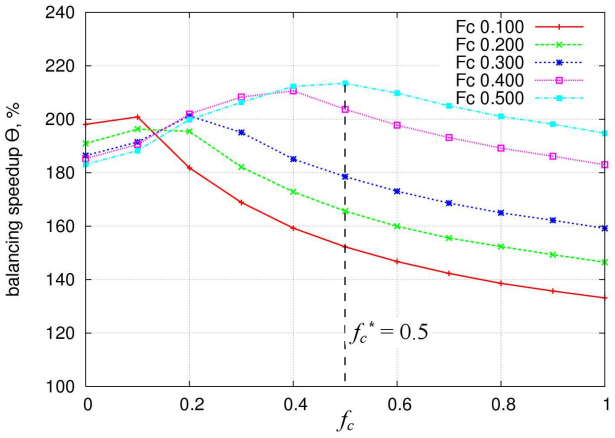


Figure 3.10: Dependency of the load balancing speedup Θ on the "guessed" application parameter f_c for 5 synthetic applications with different values of F_c.

In this section we illustrate the idea of searching through the space of possible values of the application parameter f_c in order to find the actual application requirement F_c (see Step 5 of the meta-algorithm and the detailed description of the procedure in Section 3.3.2). Figure 3.10 presents the results of load balancing of our synthetic application with the Master-Worker non-lockstep asynchronous communication logical topology (where a Worker node can immediately start calculation while the Master continues sending data to the other Workers). We show a load balancing speedup

for 5 applications with different pre-defined values of F_c (0.1 - 0.5) on the same set of heterogeneous resources. The value of f_c^* corresponding to the maximal speedup assures the best application performance. We can see that the best speedup in all cases is achieved with f_c^* close to the real application F_c, thus proving the validity of our approach. Another observation is that the applications characterized by a higher communication to computation ratio F_c, achieve a higher balancing speedup, which means that the communication-intensive applications benefit more from the proposed load balancing technique. It is also worth noticing that the distribution of the workload proportional only to the processor performance ($f_c = 0$) also gives a significant increase of the performance (180 % in case of $F_c = 0.5$), but introduction of the dependency on application and resource parameters adds another 35 % to the balancing speedup in this case (up to 217 %). In experiments with a higher level of resource heterogeneity, this additional speedup contributed up to 150 %.

3.5.2 Load balancing for master-worker model: heuristic vs. analytical load distribution

To test our load balancing algorithm, we analytically derived the best workload distribution parameters for some specific communication logical topologies of parallel applications, and compared the speedup achieved with our heuristic algorithm with that provided by the theoretical method. Here we present the analytically derived weights and the performance comparison for a widely used Master-Worker non-lockstep asynchronous communication model. The values of the weighting factors defining the best (most optimal) load distribution have been derived from the principle of equalizing the time spent by each processor working on the application, following the same idea used for derivation of eq. 3.8. Omitting the mathematical details, we present the final recurrence relation for calculating the weights:

$$q_N = \Big(1 + \sum_{i=2}^{N} \prod_{k=i}^{N} \frac{\tau_k + T_k}{T_{k-1}}\Big)^{-1}; q_{i-1} = q_i \frac{\tau_i + T_i}{T_{i-1}}, i = N..2; w_i = \frac{q_i}{\sum_{j=1}^{N} q_j} \qquad (3.12)$$

where $\tau_i = N_{comm}/n_i$ is the time for sending the total amount of application communications N_{comm} from the Master to the i-th Worker node over the network link with the measured bandwidth n_i; and $T_i = N_{calc}/p_i$ is the time for performing the total amount of application's calculations N_{calc} by the i-th processor with the processing power of p_i.

We have tested our synthetic applications with different communication to computation ratios F_c on different sets of resources, with the two different load distributions: theoretical and heuristic. In Figure 3.11 we present an example of comparison of the execution times achieved with these load balancing strategies on a set of highly heterogeneous resources. We can see that the heuristic time is only about 5-15 percent higher than the best possible for these applications (the larger difference attributed to the very communication-intensive test). Considering that our approach is generic and suits any type of communication topology, this overhead is a relatively small impediment.

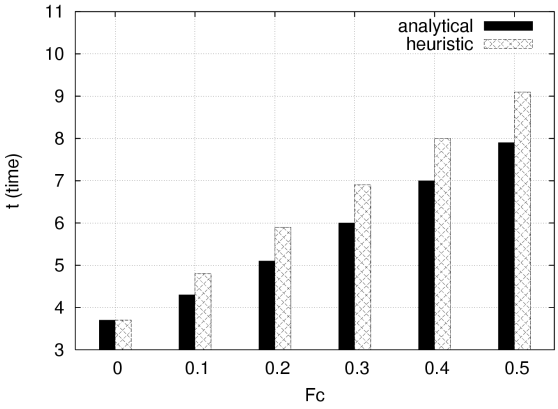


Figure 3.11: Comparison of the execution times for different weighting: the best theoretical distribution versus the generic heuristic load balancing.

3.6 Conclusions

One of the most challenging problems in porting parallel applications from homogeneous cluster environments to heterogeneous resources is to keep up a high level of parallel efficiency of the computational components. To tackle this problem, we developed a theoretical approach and a generic workload balancing technique that takes into account specific parameters of the resources dynamically assigned to a parallel job, as well as the application requirements. We validated the proposed algorithm by applying it to the Virtual Reactor parallel solvers running on the Russian-Dutch Grid testbed. It is worth noting that the load balancing speedup goes through a maximum at $f_c = f_c^*$ as shown in Fig. 3.9. This indicates that the load balancing strategy does find an optimum in the complex parameter space of the heterogeneous application/architecture combination. The clear maximum gives an unbiased guide towards automatic load balancing. The developed approach is well suited for either static or dynamic load balancing, and can be combined with the Grid performance prediction models or application-level scheduling systems [13, 106].

Traditional approaches to workload balancing can be divided to the following classes: a) carefully calculate the distribution of the workload taking into account all the properties of environment and application - the task that might be time and resource consuming by itself; b) distribute the workload in a straight forward way, considering only processing power of the worker nodes at most - the fast but not very efficient way in terms of resulting performance of workload distribution. The approach proposed here takes the intermediate place that combines the fast calculation of initial approximate workload distribution together with the precision of getting this distribution close to the optimal one during several refining iterations.

In order to optimize the resource management strategy for the driving application, the Virtual Reactor, we benchmarked the individual components on a set of

diverse Russian-Dutch Grid resources, and extensively studied the behaviour of the parallel solvers with various problem types and input data on different resource infrastructures. The results clearly show that even within one solver different trends can exist in the application requirements and parallel efficiency depending on the problem type and computational parameters, therefore distinct resource management and optimization strategies shall be applied, and automated procedures for load balancing are needed to successfully solve complex simulation problems on the Grid.

To further test the load balancing algorithm, we have developed a synthetic application with tuneable characteristics. It allows to model applications with different computation and communication requirements and logical network topologies. Some results of that work have been published in [73]. In [66, 67] we compared the theoretically derived optimization parameters for some specific topologies of parallel applications with those predicted by our heuristic algorithm.

From a middleware point of view the proposed approach forms the extension of the middleware resource management functions to the application level. The method uses the resource information that is typically processed by the environment but in this case the application controls the data distribution itself. This application-centric responsibility for the management of resources enables more fine-tuned distribution of the workload, and combination of application and middleware control on the selection, acquirement and load of resources allows to reach both application independent allocation of suitable resources on middleware layer and precise workload distribution between these resources on the application level. This strategy is further discussed in Chapter 5.

In the next chapter we continue studying the possibilities of running parallel applications in a distributed environment. The focus of the attention is transferred from workload balancing to speedup and efficiency issues: a model parallel application behaviour is examined on a set of homogeneous clusters, and the estimation of the expected pay off of such distribution is given.

Chapter 4. Parallel applications in multi-cluster environment: speedup and efficiency on the Grid

4.1 Introduction

In this chapter we continue to study the multi-layer hierarchy described in Chapter 1. We evaluate the possibilities of running parallel applications in a loosely coupled distributed environment: here we evaluate the 'horizontal' expansion of the parallel applications on the Task layer (see Figure 1.1). A multi-cluster system is used as a resource, and a single parallel application spans over several sub-clusters. We outline the concept of Grid speedup and the theoretical approach for its evaluation introduced in [49] and perform its experimental validation. The Lattice Boltzmann Method (LBM) solver is used as a case study application; a simple model of this application is used for prediction of possible performance gain from the multi-cluster distribution. Compared to the solution for parallel applications spanning over a set of heterogeneous resources proposed in Chapter 3, the approach presented in this chapter is valid for homogeneous Grids, and gives rather high accuracy in predicting the application speedup using a simple application model and a set of basic environment characteristics.

4.2 Speedup and efficiency

In this section we refer to the definitions of speedup and efficiency given in Section 3.4.1 and present more detailed analysis of these metrics.

Degree Of Parallelism (DOP) is defined as the number of processors participating in executing a program at a particular instant over time. That means that when k processors are executing the program, DOP is equal to k. The plot of the DOP over the execution time is usually called Parallelism Profile. The average DOP is defined as the average number of processors used to execute a program.

Let's define the amount of parallel workload (expressed e.g. in Mflop) with a degree of parallelism (DOP) i as W_i [51, 78, 107], the computing capacity of a processor (expressed in e.g. Mflop/s) as Δ. Then the amount of workload executed while running a part of the program with DOP $= i$ is

$$W_i = \Delta i t_i \tag{4.1}$$

where t_i is the total amount of time during which DOP $= i$. The total amount of workload is

$$W = \sum_{i=1}^{m} W_i \tag{4.2}$$

where m is the maximal DOP in the application.

Assuming that the workload W is executed on p processors, the execution time for the portion of work with DOP $= i$ is

$$t_i(p) = \frac{W_i}{i\Delta} \left\lceil \frac{i}{p} \right\rceil \tag{4.3}$$

A possible load imbalance is implicitly taken into account in the formulation of eq. 4.3. The total execution time of workload W on p processors, $T_p(W)$, equals

$$T_p(W) = \sum_{i=1}^{m} t_i(p) = \sum_{i=1}^{m} \frac{W_i}{i\Delta} \left\lceil \frac{i}{p} \right\rceil + Q(W,p) \tag{4.4}$$

with the (communication) overhead for a p processor system for the completion of the workload W defined as $Q(W,p)$, and understanding that $Q(W,1) = 0$.

The *Obtained Speedup* for the workload W on a p processor system is defined as

$$S_p(W) = \frac{T_1(W)}{T_p(W)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i} \left\lceil \frac{i}{p} \right\rceil + \Delta Q(W,p)} \tag{4.5}$$

Amdahl's law and the scaled speedup laws of Gustafson-Baris and Sun-Ni [51] can be derived from these equations.

We will consider first a few simple cases:

1. Assume $W_i = 0$ if $i \neq p$. This means that the application has the only DOP equal to p, thus the application is purely parallel and contains no sequential part. In this case:

$$T_1(W) = T_1(W_p) = \frac{W_p}{\Delta}; T_p(W) = T_p(W_p) = \frac{W_p}{p\Delta} + Q(W_p,p) \tag{4.6}$$

and the expressions for relative speedup and efficiency are:

$$S_p = \frac{T_1(W)}{T_p(W) = \frac{p}{1+f(W_p,p)}}; \varepsilon_p = \frac{T_1(W)}{pT_p(W)} = \frac{1}{1+f(w_p,p)} \tag{4.7}$$

where $f(W,p)$ is the fractional overhead function:

$$f(W_p,p) = p\Delta\frac{Q(W,p)}{W} \tag{4.8}$$

2. Assume $W_i = 0$ if $i \neq 1, i \neq p$. This means that the application consists of the parts with DOP equal to p and to 1, thus the application contains a sequential part and a parallel part with fixed number of processors participating in the parallel execution. In this case:

$$T_1(W) = T_1(W_1 + W_p) = \frac{W_1}{\Delta} + \frac{W_p}{\Delta}; T_p(W_1 + W_p) = \frac{W_1}{\Delta} + \frac{W_p}{p\Delta} + Q(W, p) \quad (4.9)$$

The expression for the speedup now becomes

$$S_p = \frac{T_1(W)}{T_p(W)} = \frac{1}{W_1/W + W_p/pW + f(W,p)/p} = \frac{1}{\alpha + (1-\alpha)/p + f(W,p)/p} \quad (4.10)$$

where $\alpha = W_1/W$, the fraction of sequential workload. Considering $f(W, p) = 0$, equation 4.10 is reduced to Amdahl's law, with the Amdahl bottleneck $1/\alpha$ in the limit of infinite number of processors.

4.3 Parallel applications on a multi-cluster

The core of a theoretical development for the Grid speedup and scaling is given in [49]. In the following sections we will summarize this approach and present an experimental validation.

4.3.1 Hierarchical decomposition of parallel applications

We consider the case of a parallel application with a workload W running on a multi-cluster which is also referred as a Homogeneous Computational Grid (HCG) computing environment: a set of identical parallel systems bound together as a single distributed computing resource. In this case the workload is decomposed among C Computing Elements (CE's), and each CE is a parallel computing system with p nodes that have the same computing capacity Δ. Thus a two-level hierarchical decomposition is introduced: first, the workload is decomposed between C CE's, and then for each CE the portion of workload is again decomposed between the processors of this CE (see Fig. 4.1). Examples of Grid-enabled applications of this kind can be found in literature, e.g. [61].

In Figure 4.1, illustrating the hierarchical decomposition, three levels can be identified. The first level is the full non-decomposed workload W - the level of sequential computing. The next *level 1* introduces the division of the workload between the CE's. This decomposition induces a *level 1* overhead $Q_1(W, C)$, that reflects communication between CE's or load imbalance among CE's. Next, within each CE the

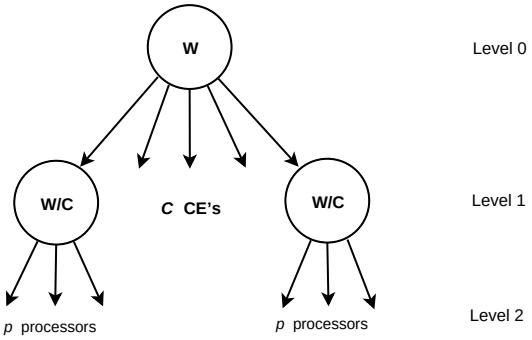


Figure 4.1: Hierarchical decomposition of a parallel application on a homogeneous computational Grid: the total workload W is distributed between C computing elements with p processors each.

workload W/C is again decomposed between the p processors of the CE. On this level another overhead $Q_2(W/C, p)$ is encountered. In case of a single CE (i.e. $C = 1$ and $Q_1(W, 1) = 0$) the standard parallel case is observed where $Q_2(W, p)$ plays the role of $Q(W, p)$ mentioned in the previous section. In homogeneous Grid environment each CE receives an equal share of the total workload. Moreover, the same overhead function $Q_2(W, p)$ applies within each CE.

In the further analysis we consider the case of $W_i = 0$ if $i \neq p$. Let us denote the execution time on the HCG with C CE's and p processors per CE as $T_{C,p}(W)$, and with the definitions introduced earlier we find:

$$T_{C,p}(W) = T_{C,p}(W_p) = \frac{W_p/C}{p\Delta} + Q_2(W_p/C, p) + Q_1(W_p, C) = \frac{W_p}{pC\Delta} + Q_2(W_p/C, p) + Q_1(W_p, C) \quad (4.11)$$

Equation 4.11 points to several facts:

- Running a tightly coupled parallel application decomposed over several CE's in a computational Grid might be of no use because of the large overheads that are be induced by the communication between the CE's (see e.g. [81]). Indeed, this overhead is expressed by $Q_1(W_p, C)$

- On the other hand, the execution of the application on more than one CE attracts more processors for processing (with the assumptions taken, a factor C more processors). Moreover, the overhead per CE is changed, from $Q_2(W_p, p)$ in the case of running on one CE to $Q2(W_p/C, p)$ in the case of running on C CE's.

Adapting the definition of the obtained speedup (eq. 4.5) to the execution in homogeneous multi-cluster environment, we get:

$$S_p^C = \frac{T_{1,1}(W)}{T_{C,p}(W)} \tag{4.12}$$

where a superscript C is added to denote the decomposition over C CE's. The derived expression for the obtained speedup is:

$$S_p^C = \frac{pC}{1 + f_2(W_p/C, p) + f_1(W_p, p, C)} \tag{4.13}$$

where

$$f_1(W, p, C) = pC\Delta\frac{Q_1(W, C)}{W} \tag{4.14}$$

$$f_2(W, p) = p\Delta\frac{Q_2(W, p)}{W} \tag{4.15}$$

Note that $f_2(W, p) = f(W, p)$ (compare eq.4.8 with eq.4.15) and that $f_1(W, p, 1) = 0$. Eq.4.13 reduces to eq. 4.7 for $C = 1$. This shows that the hierarchical decomposition introduces a second fractional overhead f_1 in the expression for the speedup.

4.3.2 Grid speedup

To understand if the obtained speedup is a good metric to asses the added value of decomposing an application over CE's, let us recall the reasons why parallelism was introduced initially:

1. The computational problems were compute bounded: the computing time was unacceptably high on one processor, thus more computing power was needed.

2. The computational problems were memory bounded: the memory consumption of the application was so large that it would not fit in memory of a single processor. Using more computing nodes and distributed memory could increase the amount of available memory for the application.

In order to analyze the added value of parallelism, one compared the execution time of the parallel application with a reference value: the execution time on a sequential computer. In a Grid computing environment the reference value should not be the single processor, but the execution time on one single CE. The reasons to decompose the application over more than one CE are exactly the same as the original reasons to parallelize the application, the computational problem is compute or memory bound in one CE, or a combination of both. So the core question the introduced metric should answer is whether decomposing the application over C CE's gives any added value compared to running it on one CE. This leads us to the concept of Grid speedup, defined as:

$$\Gamma_p^C = \frac{T_{1,p}(W)}{T_{C,p}(W)} \tag{4.16}$$

Grid speedup shows the ratio of the execution time of the application on 1 CE to the execution time on C CE's. Note that Grid speedup depends on two parameters, the number of CE's and the number of processors per CE. In case of $p = 1$ the traditional situation of a parallel computation arises, with C playing the role of the number of processor. Grid efficiency is defined as:

$$\gamma_p^C = \frac{T_{1,p}(W)}{CT_{C,p}(W)} \tag{4.17}$$

Let us compute the Grid speedup for the example of a parallel workload W_p. Substitute eq. 4.11 into eq. 4.13, which after some algebraic transformations results in:

$$\Gamma_p^C = \frac{C}{1 + g_2(W_p, p, C) + g_1(W_p, p, C)} \tag{4.18}$$

Two fractional Grid overhead functions are defined as:

$$g_1(W_p, p, C) = C\frac{Q_1(W_p, C)}{T_{1,p}(W_p)} \tag{4.19}$$

$$g_2(W_p, p, C) = \frac{CQ_2(W_p/C, p) - Q_2(W_p, p)}{T_{1,p}(W_p)} \tag{4.20}$$

The fractional Grid overhead function g_1 plays the same role as the fractional overhead f (Eq. 4.8) in the simple parallel case. Indeed, f can be rewritten as $f = pQ(W_p, p)/T_1(Wp)$ which shows that high Grid speedups can be obtained if the Grid fractional overhead g_1 is small, e.g. the grain size, defined as the portion of work per CE, is large enough such that the amount of work per CE is much larger than the amount of overhead Q_1 induced by the level-1 decomposition.

The hierarchical decomposition introduces another fractional Grid overhead g_2. The peculiarity of this overhead is that in theory it can also be negative, thus improving the Grid speedup due to the subtle changes in the *level 2* overheads when decomposing a parallel application over CE's.

The property of positivity of the Grid overhead g_2 directly depends on the linearity of Q_2 overhead on W_p. Three cases can be distinguished:

1. $aQ_2(W/a, p) = Q_2(W, p)$
2. $aQ_2(W/a, p) > Q_2(W, p), a > 1$
3. $aQ_2(W/a, p) < Q_2(W, p), a > 1$

The first option describes the applications for which the level 2 overheads are proportional to the workload. In this case we see that $g_2 = 0$ and

$$\Gamma_p^C = \frac{C}{1 + g_1(W_p, p, C)} \tag{4.21}$$

For this set of applications good Grid speedups can be obtained if the *level 1* fractional Grid overhead function g_1 is small enough, independent of the quality of the obtained speedup of the parallel application on one CE that is dictated by Q_2. This does not mean that the Grid speedup is independent on p, the number of processors in one CE, but it means that high Grid efficiencies are possible for applications that have a small parallel efficiency on one CE. By keeping the grain size large enough it is theoretically possible to get good benefits by decomposing parallel applications over several CE's.

The second case brings $g_2 > 0$, which has a negative effect on the Grid speedup. The magnitude of g_2 depends on the application details.

The third case shows $g_2 < 0$, which claims a positive effect of a hierarchical decomposition on the Grid speedup. Theoretically this effect can be illustrated by imaging applications for which the overhead of running on one CE is reduced faster then associated reduction of the computational workload by a factor of C. In case such applications exist in practice, they can count on the theoretical possibility for super linear Grid speedups.

4.3.3 Limitations and applicability

Although the theoretical derivations presented in the previous sections can be used to estimate parallel efficiency for real applications, we should mention the practical limitations of the presented approach.

First, the initial assumption about homogeneity of the resources is not often met in existing Grid testbeds. The speed of processors in participating clusters, network bandwidth within and between the clusters can be different which should result in a proper workload distribution between the participating subclusters. Taking a look at figure 4.1 we can see that in case of such heterogeneity of the environment the *level 1* decomposition of the workload between clusters should not be equal any more. Instead, application of workload balancing methods, in particular AWLB discussed in Chapter 3, should result in appropriate sharing of the workload according to resource capacity. In this case the value for $T_{C,p}(W)$ introduced in equation 4.11 changes to reflect unequal workload distribution between the participating clusters. Considering the weight of each cluster, that reflects its capacity, as w_i ($\sum w_i = 1$) the share of the workload for each cluster k is $W^k = w_k W$. The requirement for simultaneous completion of computations on all the clusters results in $T^k = \frac{W^k}{p\Delta^k} = T^{HET}$ where Δ^k represents the computing capacity of a processor in the cluster k, p - number of processors used in each cluster, T^k - execution time in the cluster k which is equal to T^{HET} for all the clusters. Equation 4.11 is transformed to the following:

$$T_{C,p}^{HET}(W) = T^{HET} + Q_2 + Q_1 \tag{4.22}$$

where Q_1 and Q_2 should be also re-calculated according to the workload distribution. The initial idea of the Grid speedup has to be updated in this context, as the single

reference value of the time taken to execute the application on p processors will be different for each cluster because of resource heterogeneity. For example, the shortest time of execution on a single cluster within the cluster set can be used instead.

Second, the assumption about equal number of processors p acquired from each cluster is not often feasible in practice. This leads to the concept of so called "fractional C" that allows to analyze the case in which CE's have different number of processors.

4.4 Case study: Lattice Boltzmann Method solver on DAS-2

4.4.1 Strip wise workload decomposition on a homogeneous multi-cluster

To evaluate the theoretical approach described in the previous section we selected a computational hemodynamic solver HemoSolve [8] that uses Lattice Boltzmann Method (LBM) - a mesoscopic method for fluid flow simulation based on a discretized Boltzmann equation with simplified collision operator [105]. For the experiments we used a basic geometry of the tube with a fluid flow inside (originally simulation of a blood vessel and a blood flow) with circle section (Figure 4.2). The method and solver implementation allow parallelization in such a way that only the exchange of boundary information is needed between the processors. The application can be decomposed between several computational clusters (or Computing Elements, CE), and such decomposition implies that a single processor in one CE exchanges boundary information with another processor in another CE, while at the same time inter CE communication between other processors is executed. In this way we get a form of latency hiding, while intra CE communication takes place, we also perform inter CE communication. We expect that this latency hiding will have a beneficial effect on the overall Grid efficiency.

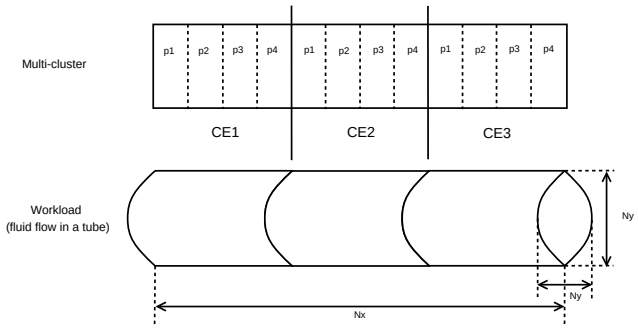


Figure 4.2: Stripwise decomposition of the workload between CE's in the blood flow simulation.

All the experiments were carried out in the DAS-2 multi-cluster environment. It consists of 5 clusters located at different universities in the Netherlands. Each cluster contains at least 32 nodes with the following characteristics: Two 1-GHz Pentium-IIIs; at least 1 GB RAM; A 20 GByte local IDE disk (80 GB for Leiden and UvA); A Myrinet interface card; A Fast Ethernet interface (on-board).

The nodes within a local cluster are connected by a Myrinet-2000 network, which is used as high-speed interconnect, mapped into user-space. In addition, Fast Ethernet is used as OS network (file transport). The five local clusters are connected by the Dutch university Internet backbone [130].

For the experiments a number of tube geometry configurations have been selected:
$N_x = \{36, 64, 128, 256, 512\}$ $N_y = N_z = \{10, 20, 30, 40, 50\}$

where N_x is the length of the tube, $N_y = N_z$ are the diameter of the tube (measured in the points of discretization per dimension).

The theoretical estimation of execution time for LBE code on single processor:

$$T_{C=1}^{p=1} = \frac{N_x N_y^2}{\Delta} \tag{4.23}$$

where Δ is the number of Lattice Updates per second (LUP/s). The execution time on a single CE in this case becomes:

$$T_{C=1}^{p} = \frac{N_x N_y^2}{p\Delta} + T_{comm} \tag{4.24}$$

where T_{comm} is the communication time needed to send the stencil information of a boundary of the processor domain to a neighboring processor.

Let's consider the execution times in a single and multi-cluster environments and derive the value for T_{comm} in eq. 4.24. For a single cluster we get:

$$T_{C=1}^{p} = \frac{N_x N_y^2}{p\Delta} + T_{comm} = \frac{N_x N_y^2}{p\Delta} + 2N_y\tau_{comm} \tag{4.25}$$

where τ_{comm} is the is the communication time needed to send the information of a point on the boundary of the processor domain to a neighboring processor in the cluster. The factor 2 emerges because each processor should communicate with its left and right neighbor. On more than one CE the hierarchical decomposition results in the following execution time:

$$T_{C}^{p} = \frac{N_x N_y^2}{pC\Delta} + N_y^2(\tau_{comm} + \tau_{grid}) \tag{4.26}$$

where τ_{grid} is the time to send the single point information between the border nodes located in different CE's. Note that we assume here a communication pattern where we first communicate between CE's.

4.4.2 Estimation of infrastructure parameters

Based on the experimental data it is possible to evaluate actual values for $\Delta, \tau_{comm}, \tau_{grid}$ in the current experimental environment. From the eq. 4.25 we can derive the expression for Δ:

$$\Delta = \frac{N_x N_y^2}{T_{C=1}^{p=1}} \tag{4.27}$$

Test results with different values of N_x and N_y showed that $\Delta \approx 1.3 \times 10^5 LUP/s$. Further on, we derive and evaluate the value for τ_{comm}:

$$\tau_{comm} = \frac{T_{C=1}^{p} - \frac{N_x N_y^2}{p\Delta}}{2N_y} = \frac{T_{C=1}^{p} p\Delta - N_x N_y^2}{2N_y p\Delta} \tag{4.28}$$

This gives the estimation $\tau_{comm} \approx 5 \times 10^{-6} s$.
The expression for τ_{grid} can be derived in the following way:

$$\tau_{grid} = \frac{T_C^p}{N_y^2} - \frac{N_x}{pC\Delta} - \tau_{comm} \tag{4.29}$$

or

$$\begin{cases} T_{C_1}^{p_1} = \dfrac{N_x N_y^2}{p_1 C_1 \Delta} + N_y^2(\tau_{comm} + \tau_{grid}) \\ T_{C=1}^{p=p_1 C_1} = \dfrac{N_x N_y^2}{p_1 C_1 \Delta} + 2N_y \tau_{comm} \end{cases} \Rightarrow \tau_{grid} - \tau_{comm} = \frac{T_{C_1}^{p_1} - T_{C=1}^{p=p_1 C_1}}{N_y^2} \tag{4.30}$$

The experiments result in the value $\tau_{grid} \approx 3 \times 10^{-5} s$.

4.4.3 Execution time

The core metrics used in the evaluation of application execution in a distributed environment is the execution time. In this subsection we will take a look at typical execution time trends of LBM solver in the DAS-2 testbed depending on application parameters, and resources selected to be used.

Execution time: theoretical and experimental

Now that the estimations for execution environment parameters are available, we can evaluate the theoretical predictions with experimental execution data. Based on the eq. 4.26 we derive the prediction for the execution time and compare it to the results obtained from actual application runs (figure 4.3). We can see the model gives relatively high accuracy in predicting the application runtime: the deviation between the predicted and measured times is less than 5 percent.

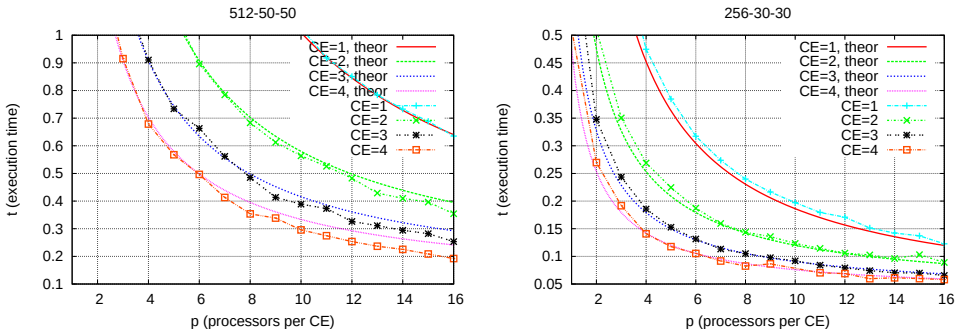


Figure 4.3: Comparison of theoretical prediction vs experimental execution time

Execution time/grain size

One of the important characteristics of the parallel application is the grain size – the size of workload portion per single processor. We define the grain size for the use case application as follows:

$$G = \frac{N_x}{pC} \tag{4.31}$$

Consider the theoretical dependency of the runtime on the grain size. Substituting G into the formula 4.26 we get:

$$T_C^p = G\frac{N_y^2}{\Delta} + N_y^2(\tau_{comm} + \tau_{grid}) = A_1 G + A_2 \tag{4.32}$$

where A_1 and A_2 are constant: $A_1 = \frac{N_y^2}{\Delta}$ and $A_2 = N_y^2(\tau_{comm} + \tau_{grid})$ This theoretical derivation predicts that the dependency of the execution time on the grain side should be linear and result in collapsing curves for all the cases with $CE > 1$. The case of $CE = 1$ should give a parallel line shifted lower by $N_y^2\tau_{grid}$.

Figure 4.4 illustrates the experimental dependency of the execution time on the grain size for different numbers of CEs used. A clear linear dependency is observed, with the curves for the case $CE > 1$ shifted along the Y axis - the influence of the inter-CE communication overhead. The plots for different values of tube width are shown, both keeping the clear linear dependency: the theoretical prediction is confirmed by the experiment.

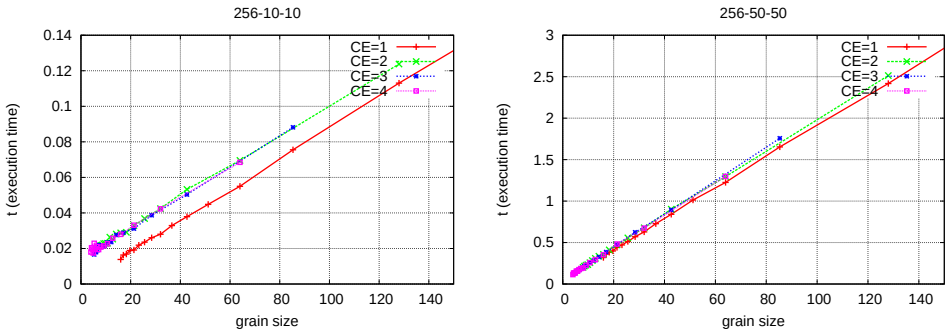


Figure 4.4: Dependency of the execution time on grain size

4.4.4 Grid speedup and efficiency

The main target of this analysis is to check whether the theoretical estimation for the speedup in the distributed multi-cluster environment coincides with the experiment, and if the theoretical expressions can be used to formulate the realistic requirements to the environment to achieve the expected application speedup and efficiency. In this section we examine the behaviour of the Grid speedup and efficiency metric, and compare experimental data with theoretical expectations.

Grid speedup/grain size

The analysis of this dependency gives the insight on the effect introduced by increasing the number of processors to compute the problem of a fixed size, thus decreasing the portion of workload handled by each processor. Naturally, the more processors are involved the less execution time is expected (keeping in mind the limitations put by the Amdahl's law). On the other hand, the larger grain sizes provide higher speedups as the fraction of communications is lower. The application profile of the speedup depending on the grain size gives the instant view on the reasonable size of the workload portions per processor. Figure 4.5 illustrates these dependencies for the tubes of two different sizes. The singularities observed on the left plot are caused by temporary slowdown of the execution for a single measurement in the row of 8 measurements ($\approx$ 5 seconds for the overshoot instead of typical $\approx$ 2 seconds) and do not reflect any application behaviour dependency. This illustrates that even justified assumptions and predictions can not guarantee the performance of a particular execution: distributed environments are typically unpredictable in performance unless special measures are taken towards advance resource and bandwidth reservation, and service-level agreements.

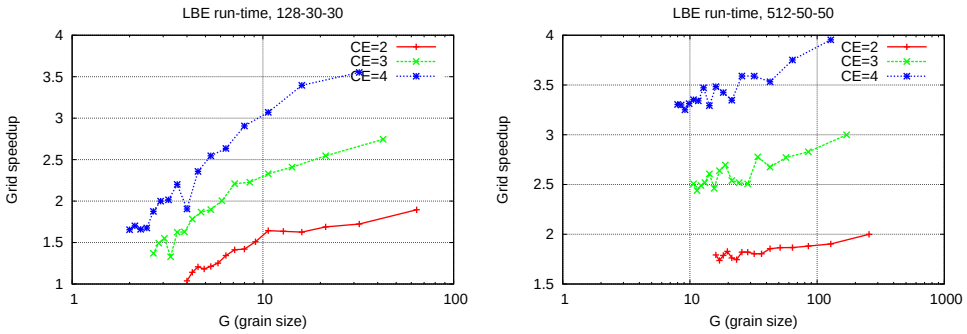


Figure 4.5: Dependency of the Grid speedup on grain size

Grid speedup/processors per Computing Element

In this section we finally address the main metrics that have been introduced to characterize the application execution in the multi-cluster environment. We analyze the theoretical expectations for Grid speedup and efficiency compared to the experimental data, and examine when it pays off to execute the application on more than one CE.

Theoretical expression for the Grid speedup:

$$\begin{cases} \Gamma_C^p = \dfrac{T_{C=1}^p}{T_C^p} = \dfrac{\dfrac{N_x N_y^2}{pC\Delta} + 2N_y\tau_{comm}}{\dfrac{N_x N_y^2}{pC\Delta} + N_y^2(\tau_{comm} + \tau_{grid})} \\ \alpha = \dfrac{\tau_{grid}}{\tau_{comm}} \\ \beta = \dfrac{N_x}{p\Delta\tau_{comm}} \end{cases} \Rightarrow \Gamma_C^p = \frac{C}{1 + \dfrac{C(\alpha+1)-2}{\beta+2}} \quad (4.33)$$

Here the dimensionless parameter α expresses the imbalance in the communication hierarchy between inter- and intra CE communication. The dimensionless parameter β contains the grain size of the application running on one CE, and the balance between computational speed and communication within one CE. Obviously, the Grid speedup increases with larger α and smaller β.

Experimental estimation:

$$\Gamma_C^p = \frac{T_{C=1}^p}{T_C^p} \quad (4.34)$$

The figures 4.6 and 4.7 illustrate the dependencies of the Grid speedup and efficiency on the number of processors used within each CE. We can clearly see that the experimental curves follow the same trends as the theoretical predictions, and the deviation does not exceed 5 percent. These plot can give an instant view on the

reasonable number of processors to use to get a desired value of Grid speedup and efficiency.

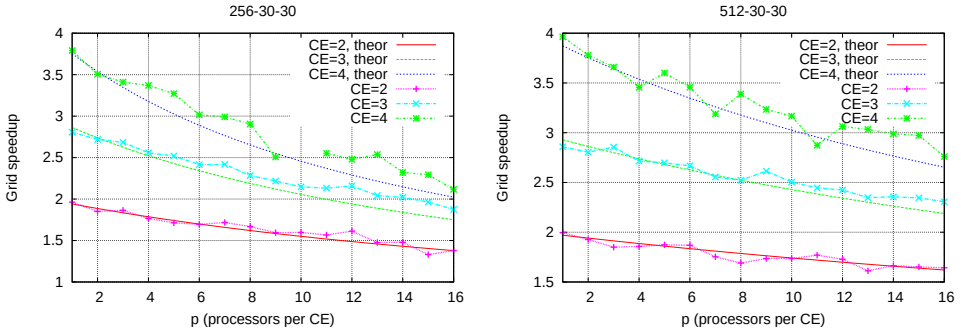


Figure 4.6: Grid speedup

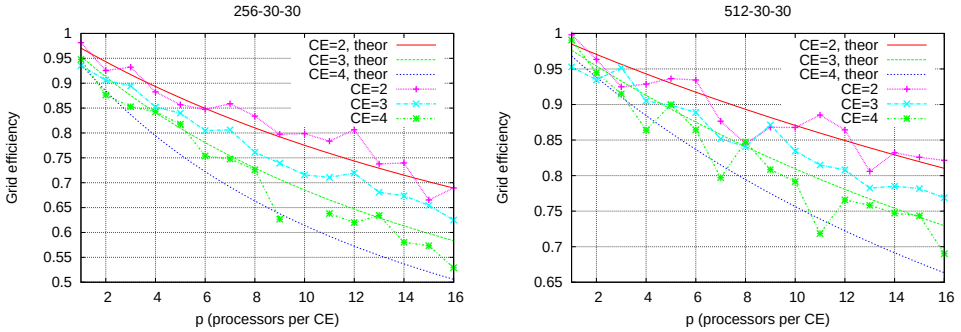


Figure 4.7: Grid efficiency

On some plots a wavy behaviour of the curves is observed. Additional analysis of the experimental data (8 measurements per each point) showed that in some series significant overshoot is observed: one or two measurements of eight in a row differ by up to 30 percent compared to the average value, which results in significant increase of the standard deviation for these series. As the experimental system used for the experiments (DAS-2) is a real Grid-like environment with shared utilization, we explain this deviation by influence of other resource demanding jobs in the system.

Grid speedup/β

A more generic view on the Grid speedup is given while building the dependency on the dimensionless parameter β (see eq. 4.33). In this case a single theoretical profile

fits all the initial data configurations. The figures 4.8 and 4.9 illustrate the Grid speedup and efficiency dependency on β for different tube sizes.

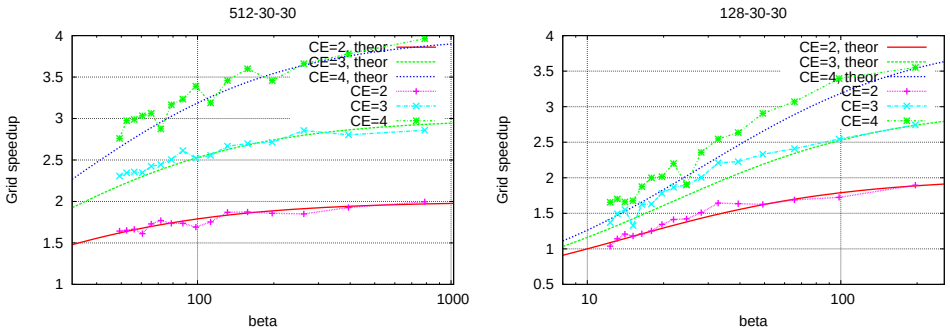


Figure 4.8: Dependency of Grid speedup on β

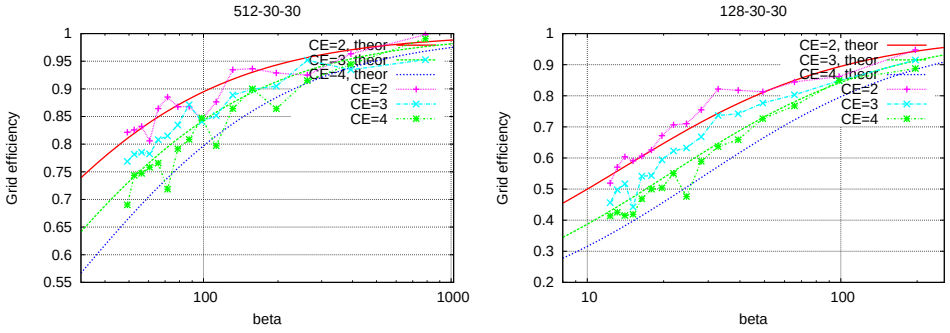


Figure 4.9: Dependency of Grid efficiency on β

The single profile allows building a single dependency of the required β to achieve the desired efficiency.

Consider the requirement $\gamma > \gamma_0$, hence from the equation 4.33 follows:

$$\beta > \frac{\gamma_0(C(\alpha+1)-2)}{1-\gamma_0} - 2$$

Substitution the values obtained in section 4.4.2 gives the value:

$$\alpha = \frac{\tau_{grid}}{\tau_{comm}} = 6.$$

Thus the expression for β becomes:

$$\beta > \frac{\gamma_0(7C-2)}{1-\gamma_0} - 2 \tag{4.35}$$

Consider $\gamma_0 = 0.8$, then

$$\beta^{\gamma>0.8} > 28C - 10 \tag{4.36}$$

Thus $\beta_{CE=2}^{\gamma>0.8} > 46, \beta_{CE=3}^{\gamma>0.8} > 74, \beta_{CE=4}^{\gamma>0.8} > 102$. From the equations 4.33 it follows that $\frac{N_x}{p} = \beta\Delta\tau_{comm}$ or $N_x = \beta\Delta\tau_{comm}p$. Substituting the values we get:

$$CE = 2: \quad N_x = \beta_{CE=2}^{\gamma>0.8}\Delta\tau_{comm}p = 29.9p \tag{4.37}$$

$$CE = 3: \quad N_x = \beta_{CE=3}^{\gamma>0.8}\Delta\tau_{comm}p = 48.1p \tag{4.38}$$

$$CE = 4: \quad N_x = \beta_{CE=4}^{\gamma>0.8}\Delta\tau_{comm}p = 66.3p \tag{4.39}$$

This analysis based on a simple model gives a quick but rather accurate estimation of the pay off to expect from the application running in an environment with known basic characteristics. We can see that the theoretical numbers for reasonable N_x are realistic to be used in practice which is confirmed by the experimental results.

4.5 Conclusions

We have evaluated the Grid speedup metric introduced in [49] that allows analyzing the performance of tightly coupled parallel applications on a Homogeneous Computational Grid. Using the concept of a two level hierarchical decomposition of the workload, a general formalism was introduced that allows computing Grid speedup in terms of two fractional overhead functions. Using this formalism we analyzed a prototypical application, a model for Lattice Bolzmann Method solver and compared the predictions derived from the theoretical approach with the experimental data obtained in a DAS2 multi-cluster environment.

The experiments were used to validate the theory proposed in [49], and they show that the theory can be used indeed for a quick but rather accurate estimation of the pay off to expect from the application running in an environment with known basic characteristics, or what system requirements have to be fulfilled to achieve the required execution efficiency. The limitation of the presented theoretical model is that only homogeneous resources are considered and the number of processors acquired from each cluster is equal for all the participating clusters. These restrictions can be overcome if workload balancing is introduced to ensure proper sharing of the workload in the heterogeneous case, and "fractional C" concept is considered while analyzing the distribution in the environment with different number of available processors per cluster.

In the next chapter we go beyond a single parallel application on a set of distributed resources. Instead of tightly coupled parallel applications discussed in this and previous chapters the attention is focused on a loosely coupled set of tasks assigned to process shared workload. Like in Chapter 3, the main concern is to address the heterogeneity of the environment and study the ways to balance the workload with the help of user-level scheduling.

Chapter 5. User-level scheduling of multi-job applications

5.1 Introduction

This chapter explores the workload and resource management for the layer of complex distributed application hierarchy corresponding to multi-job applications. Here we apply the Adaptive WorkLoad Balancing (AWLB) method presented in Chapter 3 to multi-job applications with divisible workload corroborating the technology independence of this method. We present a hybrid resource management environment, operating on both application and system levels, developed for minimizing the execution time of parallel applications with divisible workload on heterogeneous Grid resources. Compared to the solutions developed for parallel applications in the previous chapters, the approach presented here brings new possibilities and features. Integrated resource management on application and system levels enables more fine-tuned workload distribution, replacement of the workers between processing iterations and intelligent resource selection.

In this chapter we suggest an approach to the integration of a User-Level Scheduling (ULS) environment with the AWLB [66, 68, 73]. The AWLB ensures optimal workload distribution based on the discovered application requirements and measured resource parameters. The ULS controls the user-level resource pool, enables resource selection and controls the execution. The benefits of the integration are twofold: the AWLB is enriched with the capability to select resources most suitable for the application, and the ULS environment is equipped with an advanced strategy to optimize resource usage. The optimization of the workload performed by the AWLB is adaptable to the resource characteristics (CPU power, memory, network bandwidth, I/O speed, etc.) and to the corresponding application requirements. The ULS environment acquires the most appropriate resources to the user-level resource pool, and the AWLB controls the workload distribution. We perform the studies using a model application with tunable characteristics (communication to computation ratio, memory usage, communication topology, etc.) implemented in DIANE ULS environment [43, 88]. We analyze the results of performance comparison of default self-scheduling algorithm used in DIANE with AWLB-based scheduling, evaluate dynamic resource pool and resource selection mechanisms applied to it, and examine dependencies of

This chapter is based on: V.V. Korkhov, J.T. Moscicki, V.V. Krzhizhanovskaya "Dynamic Workload Balancing of Parallel Applications with User-Level Scheduling on the Grid" Future Generation Computer Systems, 25 (2009), pp. 28-34

application performance on aggregate characteristics of selected resources and application profile.

5.2 Integrated adaptive workload balancing and user-level scheduling environment

5.2.1 User-level scheduling features

Large Grid infrastructures, such as EGEE Grid [40], provide access to the computing resources at unprecedented scale. Designed for high-throughput applications, the Grid middleware and infrastructure comes with little support for the high-performance use-cases, especially the ones using a set of heterogeneous resources for a single application. Submitting, scheduling and mapping tasks on the Grid can take up to few orders of magnitude more time than the execution [43]. This is especially true for low-latency and short-deadline scenarios which are pervasive in a number of application domains from medical applications to physics data analysis. User-level scheduling (ULS) is one of the most promising ways to eliminate the difference in scale between short execution times and the large grid middleware latencies. The ULS system contains application-specific knowledge therefore it may provide customized resource selection and control mechanisms - a feature indispensable for enabling high performance applications on the Grid.

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 (user-level scheduler). The overlay scheduling layer 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). User-level scheduling does not require any modification of the Grid middleware and infrastructure nor the deployment of special services in the Grid sites, it provides immediate exploitation of the full range of a Grid sites which are available for a given user [44]. The only constraint is that actual applications have to be instrumented with additional functionality to support scheduling on the user-level.

Parallel data processing is often organized not within a single parallel application but in a set of separate programs that perform information exchange only at the start and finish of the execution or after each iteration in case of iterative execution. We call this type of software a *multi-job application*, and each job can be a separate parallel application in turn. Typically multi-job applications operate on a divisible workload: the amount of data that has to be processed by all the jobs can be shared in an arbitrary way between the jobs. Classical examples of plain multi-job applica-

tions in distributed environment are developed in the projects of distributed.net[131], SETI@home[138] etc. Here an enormous amount of workload is divided by portions between multitude of workers spread all over the world. Each worker processes its portion of the workload independently, reports the results back to the central server and gets a new portion. Usually the portions of the workload have fixed size for all the workers, and the speed and simultaneity of the processing does not play a significant role as all the initial workload should be processed only once. The situation changes when the results of processing generate new data which form new workload that should be processed again. Thus all the workload is generated iteratively, layer by layer. In this case the synchronization between different workers is needed, as the new workload can be formed only when the results of the previous iteration of processing is gathered from all the workers. From the application point of view the efficiency of distributed computations is characterized by minimization of the time taken for the execution. This demand to minimize the execution time of the iterative type of multi-job applications is met when all the workers finish processing of their portion of the workload from the same workload layer at the same time. This requires approaches and methods that ensure the simultaneity of data processing by the control of the data portion sizes assigned to different workers according to their capabilities.

Parameter sweep applications can be also mentioned in this context if the parameter space under question is considered as the shared workload. In this case different parameters are distributed between the workers that report results back to the master to analyze if the search can be optimized and some parameter space regions can be excluded from the processing. An example of parameter sweep environment is Nimrod [136], and one of the core differences of our approach with parameter sweep applications is the difference in the goal. Parameter sweeps analyze the intermediate execution results to select the regions of parameter space that can be excluded from further processing, i.e. the workload is modified during run-time, without explicit attention to load balancing in case of heterogeneous resources. Our main concern is to ensure efficient execution of the existing workload in a heterogeneous environment while the workload is not modified.

5.2.2 Executing applications in the user-level scheduling environment on heterogeneous resources

Efficient execution of parallel applications on heterogeneous and dynamic Grid resources is a challenging problem that requires the development of adaptive workload balancing algorithms that would take into account the application requirements (initially unknown often) and the resource characteristics. Generally studies on load balancing consider distribution of processes to computational resources on the system/library level with no modifications in the application code [9, 55, 56]. Less often, load balancing code is included into the application source-code to improve performance in specific cases [102, 104]. Some research projects concern load balancing techniques that use source code transformations to speedup the execution [27]. In the proposed integrated system, a hybrid approach is employed, where the balancing

decision is taken in interaction of the application with the execution environment.

A number of semi-automatic load balancing methods have been developed (e.g. diffusion self-balancing mechanism, genetic networks load regulation, simulated annealing technique, bidding approaches, multi-parameter optimization, numerous heuristics, etc.), but most of them suffer one or another serious limitation, most noticeably the lack of flexibility, high overheads, or inability to take into consideration the specific features of the application. Moreover, all of them lack the higher-level functionality, such as the resource selection mechanism and job scheduling. By developing a hybrid resource management environment, we make a step forward towards efficient and user-friendly Grid computing.

One of the approaches to create a parallel implementation of a scientific application is to build a single parallel program that is executed on a set of resources and controls the proper distribution of the workload itself. In chapter 3 we discussed an adaptive load balancing algorithm for such type of applications and validated its MPI implementation [66, 68]. Another possibility is to share the responsibility for load balancing between the application itself and the environment that enables its execution on heterogeneous resources. This approach has the advantage of combined resource management on system and application level: the environment selects and acquires the resources according to application requirements while the application controls workload distribution on these resources [73]. The application consists of a set of parallel jobs that process the workload scheduled by the Master tightly connected to the User-level scheduling environment which is an intermediate layer between the system resource manager and the application. It operates the information about available resources combined with application specific information. For iterative simulations, each iteration is performed by a new set of computational jobs; and new more suitable resources can be used to improve the application performance. This is a crucial distinction of this approach compared to traditional parallel programs where resources are allocated once and fixed during the execution, i.e. cannot be replaced during run-time unless special migration libraries are used (e.g. Dynamite [55]). To support the replacement of resources during the application run-time, the concept of user-level adaptive resource pool is employed.

The user-level resource pool contains all the discovered and acquired by ULS environment resources suitable for the application. During the execution of the application the pool is periodically updated, resources can be added or removed. The suitability of resources is determined by the application requirements, and for traditional parallel computing applications it depends on the processing power and network connectivity correlated with the application characteristics.

After resources have been assigned to all the jobs, proper distribution of the workload to each job is necessary to eliminate possible load imbalance. The goal is to finish the execution of all jobs simultaneously. The distribution of the workload depends on resource properties and application characteristics; the method to assign the workload to the worker nodes (adaptive workload balancing algorithm, AWLB) is described in Chapter 3. The computation is performed as an iterative process; after each iteration the distribution of the workload is re-evaluated on the updated resources available, and the AWLB parameters are re-estimated.

To evaluate AWLB in ULS environment we modeled the Virtual Reactor application described in Chapter 3 as a multi-job application. Now the workload is processed not in a traditional parallel environment (MPI implementation) but by a set of independent jobs that receive portions of workload iteratively. Now each job receives a number of beams to process (see Figure 3.2 in Chapter 3), and reports back the updated values of data in the beam cells. New generated beams of computational cells form the new layer of the workload which is divided between the workers for further processing. The size of the data transferred between the workers and the master can be significant, the capabilities of the workers can be different and the workers can be replaced between the iterations - the significant new feature provided by ULS environment compared to the MPI implementation of the Virtual Reactor.

5.2.3 Adaptive load balancing algorithm with resource selection in the user-level scheduling environment

The outline of the integrated solution for adaptive workload balancing in ULS environment is presented in Fig. 5.1 as a meta-algorithm based on the concepts developed in [73]. All the processing is performed in the framework of the ULS environment; two levels are distinguished: User-level resource pool and Application level. ULS environment is responsible for managing the resource pool to supply the application with appropriate resources. In turn, the application updates the environment with changing requirements. The explanation of the steps illustrated in Fig. 5.1 is provided below.

Resource pool level: In parallel to the application execution the resource pool is being monitored and updated by the ULS environment.

- Step R1. Update the pool: Discover available resources using Grid information services, acquire them to the pool if they meet application requirements. Check the resources in the pool for availability, remove no longer available ones.

- Step R2. Benchmark resources: Measuring the computational power and memory available on the worker nodes in the resource pool, network links bandwidth, hard disk capacity and I/O speed. In a more generic sense of "resources", some other metrics can be added characterizing the equipment and tools associated with a particular Grid node.

- Step R3. Rank resources: Update and re-order list of acquired resources (used by the resource selection procedure in Steps A2, A3). The priority of ranking parameters is dependent on the type of application. For traditional parallel computing solvers the first ranking parameter is the computational power (CPU) of the processor, the second parameter being the network bandwidth to this processor. For memory-critical applications, memory is the top-priority metric. For a large emerging class of multimedia streaming applications, the network bandwidth and the disk I/O speed would be the key parameters. In most cases memory ranking is an essential complimentary operation, since available memory can be a constraining factor defining if the resource can be used by the

application or not. The same goes for the free disk space parameter that can constrain the streaming applications that damp data on hard disks.

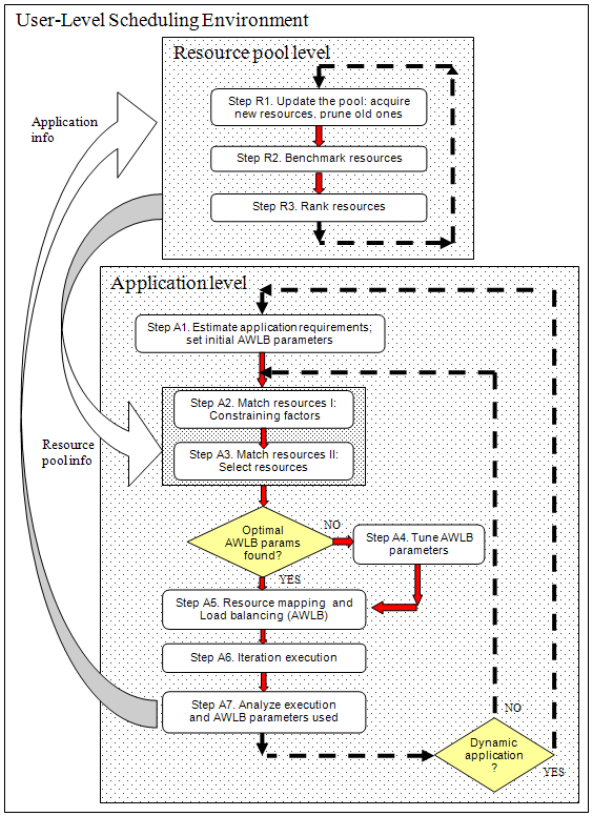


Figure 5.1: Schematic view of iterative execution of parallel application using adaptive workload balancing algorithm in ULS environment with dynamic resource pool. Resource pool level: in parallel to the application execution the resource pool is being monitored and updated by the User-Level Scheduling environment. Application level: the load balancing, resource matching and task mapping is performed on this level, in coordination with the Resource pool level.

Application level: The load balancing, resource matching and task mapping is performed on the application level, in coordination with the Resource pool level.

- Step A1. Estimate application requirements; set initial AWLB parameters: Application requirements are used to set initial values for AWLB parameters (see Section 3.3.2); additional constraints are set (e.g. minimal memory required). The AWLB parameters are automatically tuned during runtime (Step A4).

- Step A2. Matching resources I. Constraining factors: This is the first stage of checking the suitability of the available resources to the given application. It is based on the analysis of the results of Steps R2, R3 and A1 (or A7 in iterative process). In our computational application example, memory can be the constraining factor: in case of insufficient memory on some of the processors, they must be disregarded from the computation. Steps A2,A3 use information from the resource pool and send back the application requirements and requests on chosen resources. The requests are then processed by the ULS environment to book the resources.

- Step A3. Matching resources II. Selecting resources: This step provides the means to select the best-suited resources for each of the computational tasks. It consists of two basic functionalities: finding an optimal number of processors and the actual resource matching. The resource matching procedure (to be distinguished from process mapping) shall take into account the application requirements derived in Step A1. The suitability of a resource is strongly dependent on the application characteristics: the communication-bound applications will achieve a better performance on faster links even with slower processors, and the computation-intensive applications will not care about the network bandwidth. Another question is how many worker nodes shall be assigned to a multi-job application. The answer depends on the application characteristics again: for a majority of embarrassingly parallel applications (employing the resource farming concept), the more processors the better. On the other hand, for a wide class of typical parallel applications (characterized by a speedup saturation with a growing number of parallel processors), an optimal number of processors can be estimated based on the measured resource parameters and the application fractional communication overhead.

  Steps A2, A3 use information from the resource pool and send back the application requirements and the requests on chosen resources. The requests are then processed by the ULS environment to book or release the resources.

- Step A4. Tune AWLB parameters: The AWLB parameters are tuned based on the execution analysis Step A7 to provide better workload distribution. Being an adaptive heuristic, AWLB requires several steps of computations to estimate optimal values of the parameters on a given resource set. If the resources change between the iterations, re-estimation of AWLB parameters is required.

- Step A5. Resource mapping and load balancing: Actual optimization of the work-load distribution within the parallel tasks is performed, i.e. mapping the processes and workload onto the allocated resources. This Step is based on the Adaptive Workload Balancing Algorithm (AWLB) described in Section 4. It includes a method to calculate the weighting factors for each processor depending on the re-source characteristics measured in Steps R3-R5 and application requirements estimated in Step A1, A7.

- Step A6. Iteration execution: Perform an iteration of calculations and data

exchange with the workload distribution defined in Step A4.

- Step A7. Analyze execution and AWLB parameters: Measure the execution time of one iteration (or simulation time step) with current AWLB parameters. The idea is to quantitatively estimate the requirements of the application based on the results of resource benchmarking (Step R2) and measurements of the application response. This Step measures the application performance on a given set of re-sources. The performance information is used to re-estimate AWLB parameters according to the latest iteration performance data (Step A4).

  In case of dynamic resources where performance is influenced by other factors, a periodic re-estimation of resource parameters and load re-distribution shall be performed. This leads to repeating all the meta-algorithm Steps except of Step A1 (see the dashed arrows in Figure 5.1). If the application is dynamically changing (for instance due to adaptive meshes or different combinations of physical processes modeled at different simulation stages) then the application requirements must be periodically re-estimated even on the same set of resources.

5.2.4 Resource pooling and selection

Resource pooling provides for the acquisition, maintenance and refinement of a set of Grid resources. The user-level environment controls the resource pool and maintains a desired amount of resources with certain parameters best fitting the application requirements.

The refinement of resource selection in the pool is based on the basic optimality principle in divisible load scheduling problems which states that to obtain optimal processing time all the participating processors must stop computing at the same instant in time [111]. While the above claim seems to have an intuitive validity, the optimal processing time can be achieved by distributing the load only among the "fast" processor-link pairs. A reduced network can then be obtained after eliminating the slow processor-link pairs and the load is distributed among the remaining processors using the optimality principle.

Based on the above it is reasonable to say that although the optimality principle remains valid for even an arbitrary network topology, the optimal time performance depends crucially on the selection of a proper subset of the available processors. Thus, using a larger set of nodes may yield an inferior performance compared to an optimal subset of nodes among which the load is distributed according to the optimality principle.

Division of resources to fast and slow can be done after introducing a ranking algorithm. Evidently, the meaning of "fast" and "slow" for a resource can depend on the type of application it is supposed to run. Thus we have to introduce a metric for appropriate resource ranking dependent on f_c of a particular application (as defined in Section 3.3).

To rank the resources we selected a metric similar to the one used for processor weighting in AWLB:

$$r_i \sim p_i(1 + f_c/\mu_i) \tag{5.1}$$

where r_i is the rank of processor i.

For the application to run the first M processors with highest rank are selected where M is defined as the number of processors that give a reasonable speedup. M is defined during a procedure of subsequent application iterations with increasing number of processors used, starting from 1 and growing up to the value not giving a significant application speedup growth any more:

$$\begin{array}{l} M = 1 \\ while(t(M+1)/t(M) < 1 - \varepsilon) \\ M = M + 1 \end{array} \tag{5.2}$$

where $t(m)$ - execution time of an iteration on m processors with highest rank, ε - threshold of minimal acceptable speedup growth. The rank r_M is called border rank.

To support dynamic behavior of resources the value M is evaluated each time a resource with rank $r > r_M$ joins or leaves the pool. In case of such resource joining it is inserted into the sorted resource list as: $\{r_0, .., r_{i-1}, r, r_i, .., r_{M-1}, r_M\}$, thus r_M is excluded from the first M processors with highest rank. To evaluate new speedup behavior on the updated resource pool the algorithm 5.2 is applied again (starting from M resources) to determine the current value for M. Similar procedure is performed when a resource i ranked $r_i > r_M$ leaves the resource pool. The sorted resource list looks like $\{r_0, .., r_{i-1}, r_{i+1}, .., r_M\}$, and the new value of M is updated to be equal to $(M - 1)$. The algorithm 5.2 is applied to find out if the speedup dependency has changed, and the value M and the border rank can be updated.

The method to select the threshold ε is typical for the tasks with stepwise approximation: the closer approximation to the optimal value is desired, the smaller the threshold should be. On the other hand, the price of achieving maximal speedup is the occupation of larger number of processors, and the gain of couple of percent in the speedup might not worth acquiring another worker node. In our experiments we set ε equal to 0.1 which was based on the empirical consideration that 10 percent of gained speedup is still worth acquiring another worker. Additional experiments on the influence of ε on the performance of the system can be carried out, but they are not covered in the scope of this thesis.

5.3 DIANE environment for user-level scheduling

To implement and validate the hybrid AWLB+ULS approach described in Section 5.2.3, we chose DIANE - DIstributed ANalysis Environment [86, 129], which is a realization of user-level scheduling environment developed at CERN. The framework provides the execution environment for parametric parallel applications i.e. the applications which are not communication-bound and for which the communication occurs in regular patterns. This covers a broad class of applications including parameter sleep, data-analysis if data locality is assumed, Monte-Carlo simulations

etc. The communication backbone of the environment is based on Master/Worker model however customization allows to achieve more complex task synchronization patterns. DIANE allows to plug-in user-defined scheduling algorithms and failure-recovery strategies. DIANE layer runs as a set of regular user jobs, and therefore it operates entirely inside the user space. User-level scheduling does not require any modification to the Grid middleware and infrastructure, nor the deployment of special services in the Grid sites, thus it provides immediate exploitation of Grid resources available to the user. Lightweight, transient services such as Job Master or Directory Service are run locally on user-controlled computer and may be enabled or disabled at any time.

DIANE has been used with a number of applications, from black-box executables (e.g. image processing [17], telecommunications [84], regression testing and data analysis [121]) to interfacing the applications at the source-code level (e.g. medical physics and bio-informatics [82]). DIANE allows to increase the application performance, minimize the feedback latency [43], and improve certain Quality of Service parameters of the Grid, such as reliability and predictability.

DIANE provides the software plug-in framework and uses Ganga [87] as a job abstraction and management layer. DIANE user-level scheduling exploits the concept of late-binding also known as place-holders or pilot agents similarly to Condor-G glide-ins [39]. The Grid jobs run generic agents which get the workload from a scheduler - a Job Master service. Deferring the mapping of workload until the runtime helps to short-circuit the overhead of hierarchical scheduling and to accurately react to the dynamically changing characteristics of the resources or of the application. The execution of jobs is controlled by the Job Master service running on local user computer.

Typically the Grid worker nodes have the constraints of outbound-only direct connectivity with the outside networks. Certain subsets of Grid sites may provide inbound connectivity on certain ports in order to support the cross-cluster MPI applications [135]. However such support is not ubiquitous and may not be relied upon for an average Grid user in an average VO. The worker node cross-talk is still possible in DIANE however the communication is routed via the Job Master service. In DIANE model the output of one job may trigger execution of another job if decided by the Job Master. This is the only allowed way of inter-worker communication. This model is more efficient for applications with low f_c parameter because the communication even between neighboring workers is routed via a distant point in the wide area network. On the other hand the AWLB methodology may be applied directly because the resource parameter μ is evaluated always against the same worker endpoint (Job Master). Thus the μ is invariant of any communicating worker node pair.

The experiments were carried out on the EGEE grid testbed [40], in Geant4 VO [133]. The model application is implemented as a python-based plug-in for DIANE environment. The master-worker model of execution is employed, where the master selects the amount of workload to be processed by a worker, sends the data to workers and receives the results.

5.4 Simulation results and discussion

5.4.1 Adaptive workload balancing and self-scheduling comparison

To validate the methodology of integrated AWLB+ULS approach, we experimented with a model application with a synthesized workload and tunable communication to computation ratio f_c. The experiments compare performance results achieved using the AWLB algorithm and dynamic resource pool with the results shown by the standard DIANE job dispatching technique - self-scheduling (also called a FIFO scheduling algorithm). In self-scheduling all the workload is divided into jobs of equal size, typically the number of jobs exceeds the number of available worker nodes. As soon as a worker becomes available, the next job from the list is assigned to it. In AWLB all the workload is divided into the number of available workers, and the size of the workload assigned to each job is calculated by the heuristic algorithm, using the resource and application characteristics.

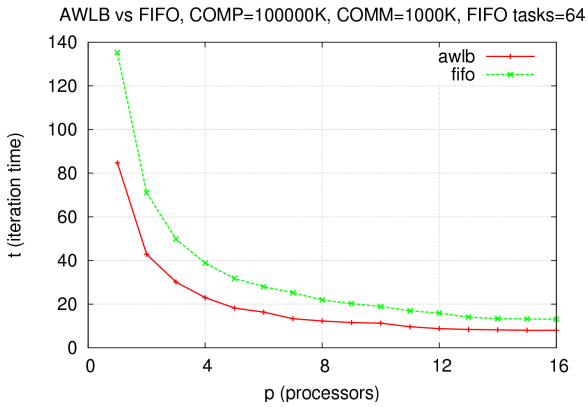


Figure 5.2: Comparison of AWLB and self-scheduling algorithms: Runtime dependency on the number of processors acquired. $f_c = 0.01$

Figure 5.2 presents a comparison of execution times of one iteration with the AWLB and self-scheduling (FIFO) algorithms. In this figure COMP is the total amount of computational operations (in Flops) and COMM is the total amount of communications for each simulation (in bytes transferred). In this experiment the number of processors used is increased by 1 processor for each iteration. In all cases, the AWLB significantly outperforms the self-scheduling almost twice. In some cases the gain can be up to several times.

Table 5.1 illustrates a typical example of AWLB parameters on a set of heterogeneous resources. Each worker is ranked according to the resource parameters and application characteristics (see Section 5.2.4). In the table, PROC and NET are the relative processor and network capacity of a worker (results of benchmarking). The

worker	PROC	NET	R ($\times 10^{-2}$)
1	57.9	4.3	1.1
2	42.4	13.1	1.9
3	52.3	28.8	3.7
4	54.8	28.9	3.8
5	55.5	28.7	3.7
6	96.0	16.4	2.8
7	42.4	28.9	3.6
8	42.4	28.5	3.6
9	41.2	28.4	3.5
10	53.0	8.9	1.5
11	53.2	28.3	3.7
12	33.6	3.2	7.0
13	41.8	28.9	3.6
14	42.1	28.1	3.5
15	49.9	42.6	5.2
16	47.0	27.8	3.5
17	53.8	27.9	3.6
18	22.2	7.4	1.0
19	55.2	28.4	3.7
20	57.8	4.3	1.1
21	39.5	28.0	3.5
22	64.9	11.9	2.0
23	29.9	9.8	1.4
24	41.5	28.9	3.6
25	41.1	28.8	3.6
26	41.1	28.6	3.6
27	41.0	28.5	3.6
28	41.1	19.0	2.5
29	45.9	42.0	5.1
30	41.7	28.6	3.6
31	40.9	29.1	3.6
32	41.4	28.7	3.6

Table 5.1: Sample distribution of processor ranks (R) by AWLB on a set of heterogeneous workers.

application values COMM and COMP are the same as in Fig. 5.2.

Figure 5.3 shows how the application communication to computation ratio f_c influences the execution time for different workload distribution algorithms. During this experiment the computational load (COMP) was kept constant on a fixed set of 16 processors, while the amount of data transferred between the master and the workers (COMM) is varied. For all types of simulations, the AWLB algorithm is significantly faster than the self-scheduling.

Figure 5.4 presents the statistics of an actual run using a dynamically populated resource pool. The core feature of the ULS environment is the ability to change resources during the execution runtime, such that every iteration can run on a different set of resources. Figure 5.4a shows how the resources are gradually added to the resource pool, depending on their availability. We can also notice that some workers are removed from the pool: the number of workers is not growing steadily, but

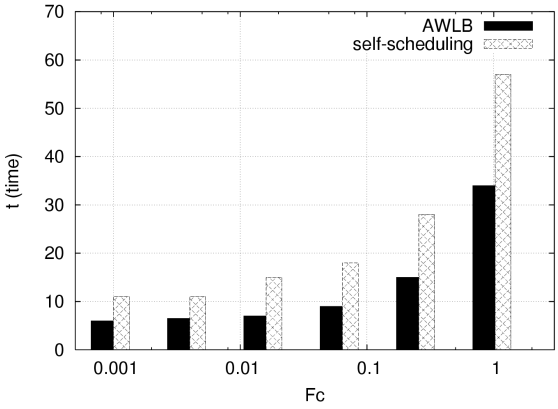


Figure 5.3: Sample dependency of the runtime on communication/computation ratio f_c for 16 workers and 64 FIFO jobs (COMP=30M, variable COMM).

experiences dips every few iterations.

In Figure 5.4b, the basic resource matching and selection mechanism is demonstrated (Step A3). All the workers acquired at each iteration (shown in Figure 5.4a) are used for the execution, and the execution time depends on the number of available workers at the moment. Notice that at iterations number 9,13,16 when some resources left the pool, the execution time increases.

5.4.2 Adaptive resource selection

To illustrate the adaptive resource selection algorithm presented in the Section 5.2.4 we analyze the execution of the model application on the dynamically acquired Grid resources. The distribution of the resources obtained for the experiments is presented in Figure 5.5a. Each point on the plot reflects a single available worker with corresponding processor performance and network connectivity to the master.

To check the behavior of different types of applications we modeled different amount of computations and communications (i.e. different f_c). The resource ranking used for resource selection (Section 5.2.4) is based on application properties, thus the rank of a single resource depends on the application it is used to execute. Figure 5.5b illustrates resource ranking for different application types (i.e. different values of f_c) for the same processor/network parameter distribution shown in Figure 5.5a. Namely, in Figure 5.5a the absciss axis marks the ordered number of a worker in the resource pool, the white triangles represent the processor capacity of the worker (measured on the left ordinate axis), the black triangles represent the network connectivity to the worker (measured on the right ordinate axis). Figure 5.5b and illustrates different values of weights given to the same worker in case of different assumptions about the application parameter f_c. The placement of the figures on the same vertical line and the same scale of the absciss axis allow to easily follow the dependency of weight

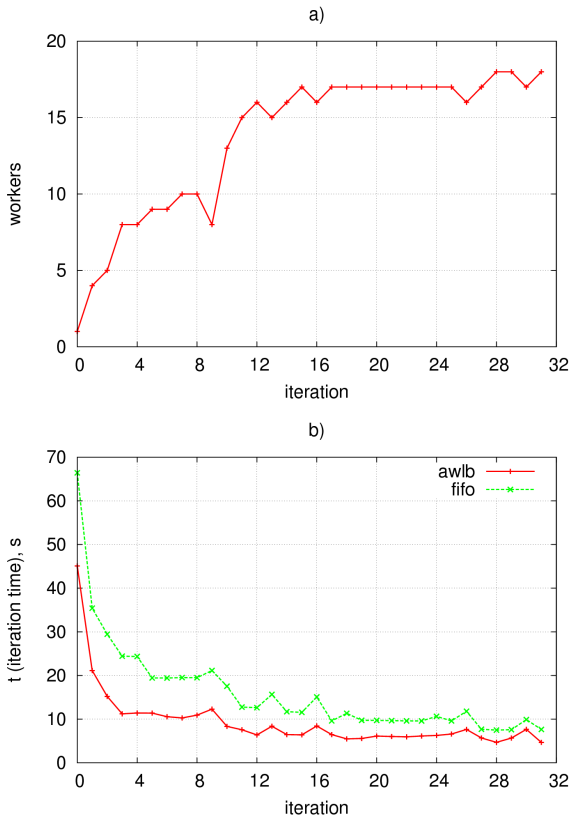


Figure 5.4: a) Dynamic resource pool population; b) Sample execution times using a dynamic pool.

volatility on processor/network pairs for each worker.

The general idea of resource selection is to provide the best resource subset from the set of available resources to ensure the fastest possible execution of the application. To estimate the performance gain from the resource selection procedure, we analyzed execution times on the best, average and worst sets of available resources. The number of workers was fixed and the resources were selected from the top, middle and bottom of the ranked resource list. Thus the performance was estimated for the same number of workers with the highest, average and lowest ranks. Figure 5.6 illustrates the efficiency of the resource selection algorithm by presenting the resulted performance difference.

The number of processors to use for efficient execution of a parallel application with divisible workload depends on both application and resource characteristics, and it is difficult to predict the speedup saturation point (the number of workers with highest

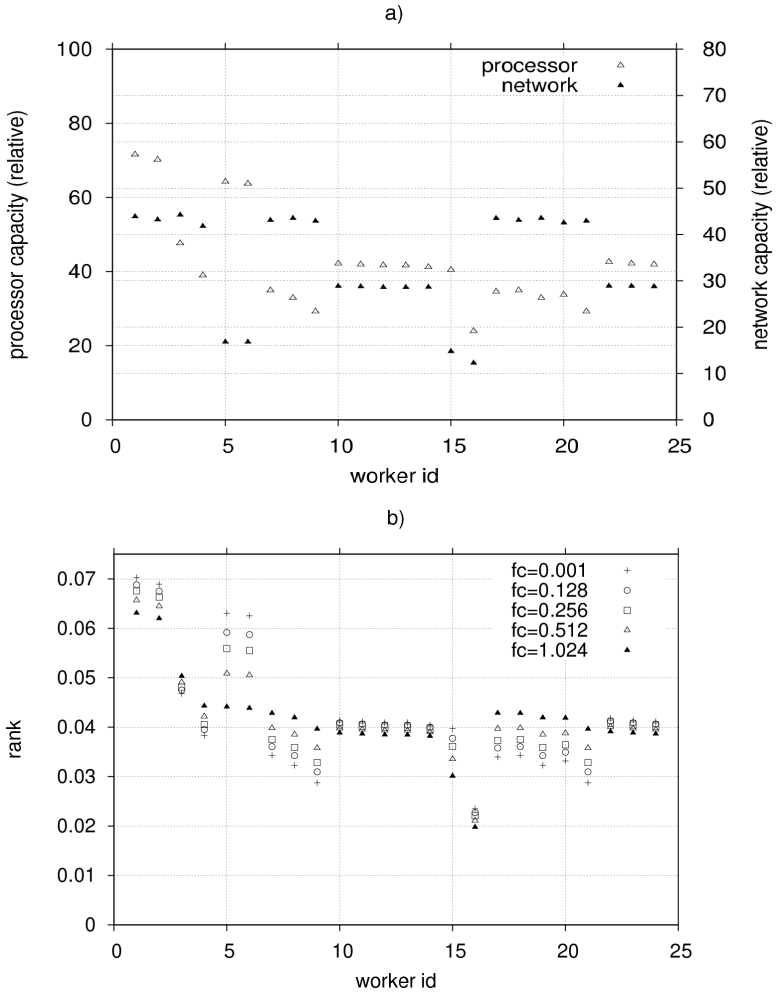


Figure 5.5: a) Sample resource distribution: processor and network capacity for the workers in the resource pool; b) Resource ranks for the workers (dependency on f_c).

ranks used) in advance. The adaptive resource selection (Section 5.2.4) performs the analysis of speedup growth with addition of every new worker to the set of workers used, the threshold ε can be explicitly set by the user. The figure 5.7 presents sample executions of applications with different f_c on the same resource pool and ε equal to 0.1. As it can be predicted, the speedup of the application with higher communication demands saturates on smaller amount of workers, which is tracked by the resource selection algorithm. Thus the resources not acquired from the resource pool to provide the marginal speedup growth can be more useful for another application that accesses

the same resource pool.

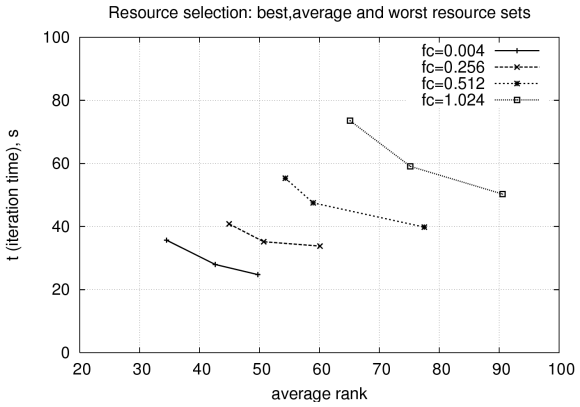


Figure 5.6: Resource selection: comparison of best, average and worst ranked resources performance. On X-axis the average rank of each resource set used for the execution is shown.

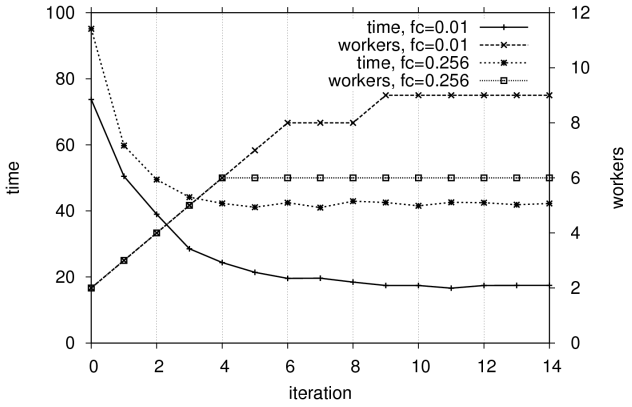


Figure 5.7: Adaptive selection of amount of workers to use, $f_c = \{0.01,0.256\}$

The experimental results presented above mostly address the cases when application parameter f_c is known. But this is not the case for real applications frequently. The AWLB algorithm is designed to be able to figure out the proper value of f_c automatically during several initial iterations of execution. The detailed description of this functionality is given in [68] and chapter 3.

5.5 Conclusions

Evolution of the Grid paradigm to support High Performance Computing is indispensable for a large number of applications which require on-demand access to the Grid resources. In this chapter we proposed an approach to enhance the quality of handling multi-job applications in Grid environment by integrating the Adaptive Workload Balancing Algorithm (AWLB) developed for parallel applications on heterogeneous resources and User-Level Scheduling (ULS) environment. The latter gaps the missing link between applications and Grid resource managers: this user-level middleware is a customizable, application-centric scheduler and application hosting environment. Dynamic benchmarking of resources and estimation of the application characteristics is used to optimize the usage of a dynamic user-level pool of Grid resources maintained by the ULS. We devise a generic recipe on how to solve the workload balancing problem on the Grid for multi-job applications. To prove the concept we performed tests with a synthetic application with configurable requirements using EGEE Grid resources and the AWLB algorithm incorporated into the DIANE user-level scheduler. We present experimental results and discussion on the way to manage the workload of divisible load parallel applications on the Grid, compare different workload distribution methods, illustrate the usage of dynamic resource pool and application performance dependencies with adaptive resource selection.

This chapter builds an intermediate step between parallel applications in Grid environment discussed in chapters 3 and 4 and multi-component distributed applications discussed further. We investigated the management of multi-job applications facilitated by User-Level Scheduling environment that in the aggregate formed an integrated solution for combined resource management on application and system levels. In the next chapter we focus the attention on the multi-component applications that do not share the workload (like it was discussed in all the previous chapters) but process it in a pipeline manner – scientific workflows.

Chapter 6. Data-driven Workflow Management on the Grid

6.1 Introduction

Grid brings the power of many computers to scientists. However, the development of Grid-enabled applications requires knowledge of Grid infrastructure and low-level API to Grid services. Workflow management systems provide a high-level environment for rapid prototyping of experimental computing systems. Coupling Grid and workflow paradigms is important for the scientific community: it makes the power of the Grid easily available to the end user. The paradigm of data driven workflow execution is one of the ways to enable distributed workflow on the Grid.

In this chapter we discuss the highest layer of the complex distributed application hierarchy proposed in Chapter 1: a Grid workflow. Here we concentrate on the workflows controlled by a dataflow: discuss a model of a data-driven workflow and evaluate different strategies of resource management for this type of workflows.

To be executed on the Grid a distributed workflow needs a framework that enables the workflow enactment and execution control: a workflow management system (WMS). In this chapter we present the VLAM-G WMS that was developed to compose, execute and study data-driven workflows in a distributed environment. The core component of VLAM-G is the Run-Time System (RTS). The RTS is a dataflow driven workflow engine which utilizes Grid resources, hiding the complexity of the Grid from a scientist. Special attention is paid to the concept of dataflow and direct data streaming between distributed workflow components. We present the architecture and components of the RTS, describe the features of VLAM-G workflow execution, and evaluate the system by performance measurements.

Being the top layer of the complex application hierarchy, a workflow encompasses the applications of lower layers, introduced in the previous chapters, as its components. The management of these different layers is performed within a single framework, and the execution of each component is coordinated with the others. We illustrate this with a sample workflow for the driving application - the Virtual Reactor.

This chapter is based on: V. Korkhov, D. Vasunin, A. Wibisono, A. Belloum, M. Inda, M. Roos, T. Breit, L.O. Hertzberger. "VLAM-G: Interactive Dataflow Driven Engine for Grid-enabled Resources", Journal of Scientific Programming vol.15(3), pp. 173-188 (2007), ISSN 1058-9244

6.2 Data-driven workflows in a virtual laboratory

Grid environment allows coordinated resource sharing and problem solving among groups of trusted users within Virtual Organizations. Such environments enable global distributed collaborations involving large numbers of people and large scale resources, and make data and computing intensive scientific experiments feasible. Complex scientific experiments often require access to distributed resources such as computational resources, data repositories, third-party applications, scientific instruments etc. The utilization of these resources requires multi-domain expertise which is beyond the common knowledge of a single scientist.

One of the important research topics in e-Science [48] is to develop effective Grid enabled Problem Solving Environments and Virtual Laboratories for different scientific domains. Organizing software utilities (e.g. simulators, visualization and data analysis tools) as a meta experimental environment, a Virtual Laboratory allows a scientist to plan and conduct experiments at high level of abstraction [41] and enable a group of researchers located around the world to work together on a common set of projects. To execute and control experiments within a a Virtual Laboratory that typically consist of a number of interdependent simulations and other activities Scientific Workflow Management Systems (SWMS) are often employed[20, 85]. A SWMS explicitly models the dependencies between experiment processes, and orchestrates the runtime behavior of involved resources according to a flow description.

A wide range of e-Science applications from different scientific domains, namely high energy physics, bio-informatics, bio-diversity, bio-medicine, and tele-science [2] shows that data access and processing play an important role. A typical scenario involves a number of steps: (a) access data on a remote storage system; (b) move the data to one of the available computing nodes; (c) process the data, which results in a new data set (intermediate data) that is further processed on another node and thus needs to be moved again. These steps are repeated until the final data sets are generated and stored in a predefined location.

Frequently, when following this scenario, scientists would prefer to delegate all non-scientific activities such as finding the appropriate available resources and manipulating input, output, and intermediate data sets to a system they trust. Hiding such details of the underlying grid execution from the end user allows them to concentrate on the essential tasks they need to perform in order to achieve their scientific goals, e.g. defining original data sets, defining processes that need to be performed and the order in which these processes must be executed. On the other hand, the in-depth view at underlying processing on the low-levels can be provided as well, and an interested user can check what particular resources are used, examine intermediate results of execution and monitor actual computational processes. The ability to control one or a few parameters of the processes interactively, without having to stop or restart experiments, is also desirable.

In the scenario above, as well as in many other scenarios describing scientific experiments such as weather predictions [42], bio-medical visualization [69], knowledge discovery in databases [63, 90], and other scientific domains [90], data is one of the main drivers of progress in these experiments. In the paper [19] the authors describe

three different use cases from chemistry, cosmology and astronomy, where different patterns of pipelined processes have to be executed concurrently on distributed computing resources. For performance reasons data is streamed between the components composing the different levels of the pipeline. As described in [36], processing data in streams has certain advantages because it can be done on the fly, reducing the need to store intermediate data which is not needed.

The analysis shows that the computational processes in many e-Science experiments correspond to the actual data flow in an experiment; typically from an instrument, through analysis software, to data storage facilities. Consequently, the computation processing can be represented by a data driven workflow.

To efficiently execute large scale workflows in distributed environment, scalability support on different levels is needed. The workflow engine should be scalable enough to enable the execution of workflows consisting of a large number of components, possibly sub-workflows running cross geographically distributed resources. Support for a scalable job management is needed, especially for job farming and parameter sweep jobs. To ensure the efficient execution and utilization of resources a proper resource management is needed on multiple layers: starting from parallel tasks and up to the whole functional decomposition of the application on the workflow level. Finally, the introduction of semantic descriptions enables a semi-automated discovery and matching of workflow components.

Current trends in the research and development of workflow management systems has already been outlined in Section 2.4. Nowadays a number of scientific workflow management systems are developed in various research projects. In particular, some of the most mature frameworks are Triana [108], Taverna [92] and Kepler/Ptolemy [6]. All these workflow management systems have their own features and specifics, e.g. Taverna is totally dedicated to Web service composition highly demanded by contemporary bioinformatics; Triana provides interfaces to invoke web services and use Grid resources (GAP, GAT [5]) and has a rich library of processing modules; Kepler inherits a powerful and matured framework from Ptolemy, it provides a rich library of actors, Nimrod support and several types of workflow execution. An extensive overview of current workflow systems is given in [30].

The workflow management system discussed in this thesis, VLAM-G (Virtual Laboratory AMsterdam on the Grid) developed in the context of the Virtual Laboratory for e-Science project [140], has a set of distinguished features. VLAM-G provides a basic set of capabilities for building workflows by connecting components to each other based on data dependencies. As a core concept VLAM-G uses dataflow between simultaneously executing distributed components as an execution driving activity, while many other systems employ control flow and/or only perform sequential execution of workflow steps according to intermediate data readiness. Intermediate data handling is also performed differently. Some systems like Triana and Taverna use a centralized approach whereby all the data is controlled and collected at the central point where the engine resides. In turn, in VLAM-G data are transferred directly between participating components. Another distinguishing feature of VLAM-G is its ability to control an executing workflow at runtime in several ways. Firstly, the workflow components can be parameterized; the parameters can be controlled during execution.

Secondly, as VLAM-G operates with remote jobs co-allocated at runtime, it enables direct access to the graphical output of a component (i.e. component GUI, if one exists) by providing a shared virtual display. To the best of our knowledge no other system provides similar capabilities.

6.3 Resource management for data-driven workflows

6.3.1 Workflow modeling

Most Grid resource management systems being developed aim at a specific class of applications. In particular they can utilize different program and performance models as well as scheduling policies. The models for current high-performance schedulers generally represent a distributed application in a dataflow style or by a set of application characteristics. A performance model provides an abstraction of the behavior of an application on an underlying system and calculates predictions of application performance within a given timeframe. Current high-performance schedulers employ a wide spectrum of approaches, which differ in terms of who supplies the performance model: the system, the programmer or some combination of both. The performance model is commonly parameterized by both static and dynamic information. The scheduling policy ensures the achievement of the application performance goal. A usual goal is to minimize the execution time although different parameters can be optimized (e.g. execution cost). In many current efforts the scheduling policy is to choose according to a performance model the best set of resources among the candidate resource choices.

Consider a simple model of a distributed workflow (based on the meta-application model described in [115]). A workflow consists of a set of components that communicate with each other during the application run-time. The components are separate tasks that can be computational or data-processing units, interfaces to specific devices, data storage interfaces etc. Some of the components may be schedulable (like computation or data processing) and some are fixed (as data storage or remote device). In general, fixed components may affect the placement of the other components (i.e. a data processing component requires high speed link with a fixed database component).

Workflows can be represented as a data-flow graph. In the simple approach they can be divided in two categories: concurrent and pipeline. Concurrent application consists of a set of components running on different sites concurrently and exchanging data at the same time (Fig. 6.1-a). Pipeline application components start processing one after another and not concurrently, in a chain-like fashion (Fig. 6.1-b). A mixture of these types may also exist.

The scheduling of a workflow requires a cost model to evaluate the efficiency of a candidate schedule. The cost functions depend on the information about the application.

Consider application component $C_i, (i \leq N)$ where N is the number of components

a) concurrent b) pipeline

Figure 6.1: Basic types of data-driven workflows

(for simplicity here we consider all components as schedulable). For each component we define:

- $comp(C_i)$ - the computation load of the component C_i, may be counted in the number of instructions to be executed;
- $comm(C_i, C_j)$ - the communication load between C_i and C_j, may be counted in the number of bytes transferred between the components;

Let us now consider $S_k, (k \leq M)$, M - number of sites. Thus we can define:

- $compT(C_i, S_k)$ - the computation time for the component C_i running on the site S_k when the site provides all its resources to the application. In case other components are scheduled in S_k then the function may be degraded.
- $commT(C_i, C_j, S_k, S_l)$ - the communication time taken for data transfers between C_i scheduled on S_k and C_j scheduled on S_l . The function may depend on the number of components sharing the link.
- $commTT(C_i, S_k)$ - the total communication cost for component C_i placed on site S_k , is a function of $commT$ functions for the component's links. In the simplest case is the sum of those costs.

For a candidate schedule we get a set of pairs $(i, k) \in P_q$, that assigns components C_i to resources S_k , where P_q describes a particular candidate schedule; $q \in Q$, where Q is a set of all estimated schedules.

Thus we can define cost functions for a workflow as a whole in terms of the component cost functions. For concurrent applications we get:

$$T_c = \underset{(i,k)\in P_q}{max} [compT(C_i, S_k) + commTT(C_i, S_k)]$$

For pipeline workflows we get:

$$T_p = \underset{(i,k)\in P_q}{sum} [compT(C_i, S_k) + commTT(C_i, S_k)]$$

The scheduling problem is to determine such an assignment of components to resources that minimizes the cost function T for the workflow.

6.3.2 Heuristic algorithms for workflow scheduling

To create an optimal schedule of a workflow means to assign all the tasks composing the application to appropriate resources to minimize the cost of execution, usually counted as overall execution time. The model for estimation of the cost (performance metric) was presented in Section 6.3.1. Here we address the algorithms used to achieve sub-optimal schedules based on the abovementioned performance metric.

Each component in the modeled workflow is provided with the execution requirements for processing: computational load (CPU cycles needed), communication load (data transfers size), memory and storage requirements. The information on the characteristics of the available resources (CPU speed, available memory and storage, network links speed) can be retrieved from the Grid information services, thus the target is to perform the matchmaking of the workflow components and resources to ensure the optimal execution. We propose and evaluate the following algorithms to create the schedule for the workflow execution: two types of basic greedy algorithm, computation-network prioritized algorithm, and simulated annealing.

Basic greedy algorithm (BG):

This is the straightforward version of the greedy heuristic we propose and use as the starting point of the experiments. We assume that a resource used to execute a workflow component can be shared between several components or even utilized by other background jobs, thus the component is not in the full possession of computing resources and only a share of computing power is available. This heuristic consists of the following steps:

```
Reqs = get_requirements(WFcomps) // Find out the minimal requirements
               // for the workflow components (set of minimal memory
               // and storage requirements)
Res = discover_resources(Reqs, GIS) // From the Grid indexing service
               // discover a set of available resources satisfying the
               // minimal requirements
sort_descend(Res, CPU) // Sort the created resource pool according to
               // available CPU power
sort_descend(WFcomps, Reqs.CPU) // Sort all the workflow components
               // according to CPU requirement
compsPerRes = 1 // Maximal number of components allowed to be mapped
                // on the same resource
for comp in WFcomps do // Staring from the component with highest CPU
                       // requirement iterate all the components
  comp.mapped = false
  while(!comp.mapped) do
    for res in Res do // Iterate all the resources available in the
                      // pool beginning from the most powerful one
      if(res.fits(comp.reqs) && (res.alreadyMapped < compsPerRes))
      then // resource fits (memory/storage requirements satisfied
           // the number of already mapped components does not
```

```
        res.map(comp)                          // exceed the limit)
        res.alreadyMapped++       // map this component
        comp.mapped = true        // to this resource and
        break                     // stop iterations
      end if
    end for
    if(!comp.mapped) then
      compPerRes++       // If all the resources have been
                  // iterated and none could fit then increase
                  // permissible number of components per resource
      if(compPerRes > WFcomps.size()) exit(NO_MAPPING_FOUND)
    end if
  end while
end for
```

Modified basic greedy algorithm (BGM):

The basic greedy algorithm is modified so that the same resource could be used for executing several components simultaneously unless it decreases the overall performance of the workflow compared to the case when all the components are placed to separate nodes. Matchmaking for each component is changed in the following way:

```
for comp in WFcomps do
  for res in Res do                  // Perform the estimation of the
    if(res.fits(comp.reqs)) then     // run-time (Section 6.3.1) for all
      res.map(comp)                  // the available resources, select
      T=estimate_runtime()           // the optimal one (the one with
      res.unmap(comp)                // minimal run-time) and map the
      if(T < bestfitT) then          // component to this resource
        bestfitT = T; bestfitRes = res
      end if
    end if
  end for
  bestfitRes.map(comp)
end for
```

The main difference with basic greedy algorithm is that here the components can be scheduled to the same resource before all the resources are occupied with at least one component. This can be efficient in the case of a single powerful resource among a set of much slower resources.

Computation-network prioritized algorithm (CNP):

This algorithm introduces a way to describe the level of a relative workflow intensity of data transfers and computations, which is a reflection of the Advanced Workload Balancing (AWLB) algorithm introduced in Chapter 3 and was used for resource ranking

in User-Level Scheduling environment (Section 5.2.4). Again, we use the parameter f_c which defines priority either of networking communications or computational operations for the experiment.

The f_c parameter affects the calculation of the resource rank which is used in resource sorting and selection procedures. We define 'the base resource' as the resource for the component with the highest requirements - the most powerful of the available resources. The rank of the resources is calculated using the bandwidth to the base resource and available computing power of the resource. The heuristic formula for resource rank is similar to the one used in Section 5.2.4:

$$r_i \sim p_i(1 + f_c/\mu_i) \tag{6.1}$$

where r_i is the rank of resource i, p_i - processing power of resource i, and μ_i is the resource capacity parameter. Here $\mu_i = p_i/n_i$, where n_i is the bandwidth between the base resource and the resource i.

The f_c coefficient here reflects an aggregate average communications/computations characteristic of the whole workflow. By varying the value of f_c different resource priorities can be achieved and different resulting schedules can be obtained. Compared to the adaptive workload balancing algorithm presented in Chapter 3, the optimal value of f_c can be obtained using a similar procedure as described in Section 3.3.

Simulated annealing algorithm (SA):

Simulated annealing is a technique, which was developed to help solve large combinatorial optimization problems [1]. It is based on probabilistic methods that avoid being stuck at local (non-global) minima. It has proven to be a simple but powerful method for large-scale combinatorial optimization.

In the simulated annealing algorithm an objective function to be minimized is defined. Here it is the overall execution time of an experiment. The execution time of each workflow component is equivalent to the "energy" of the component. Then, "temperature" is the average of these times. Starting from some initial schedule and initial temperature, the algorithm randomly selects a component to be re-mapped, randomly selects a suitable resource and re-maps the component. The total "energy" (execution time) of the experiment is estimated. Any downhill step is accepted and the process repeats. An uphill step may be accepted. Thus, the algorithm can escape from local minima. This uphill decision is made by the Metropolis criterion. The Metropolis criterion attempts to permit small uphill moves while rejecting large uphill moves. To allow this a uniformly distributed random number p in the range [0...1] is generated and compared with the Metropolis number:

$$m = \frac{e^{delta}}{temperature}$$

where 'delta' is the difference in energy between current schedule and candidate schedule. Since delta/temperature is negative (candidate schedule energy is larger then

current one), e is raised to a negative power so that m will also be in the range [0...1]. The decision to accept an uphill move is made if randomly generated number p is less than Metropolis number m. The process of finding an optimum is divided to a number of series of iterations (during each iteration series a number of re-mapping steps are performed) with decreasing temperature on each step. Decreasing values for temperature as well as large negative values of delta make acceptance of an uphill move less likely. As the optimization process proceeds, the algorithm closes in on the global minimum. Since the algorithm makes very few assumptions regarding the objective function, it is quite robust. The degree of robustness can be tuned by the user by adjusting the following parameters:

- factor - annealing temperature reduction factor
- ntemps - number of temperature steps to try
- nlimit - number of trials at each temperature
- glimit - number of successful trials (or swaps)

Pseudo code of a basic process:

```
initialize temperature

for i := 1...ntemps do
  temperature := factor * temperature
  for j := 1...nlimit do
    calculate current_cost
    trial_cost := randomResourceSwap()
      // swap resources randomly
    delta := current_cost − trial_cost
      // cost value change

    if delta > 0 then
      make the swap permanent
      increment good_swaps
    else
      p := random number in range [0...1]
      m := exp( delta / temperature )
      if p < m then
        // Metropolis criterion
        make the swap permanent
        increment good_swaps
      end if
    end if
    exit when good_swaps > glimit
  end for
end for
```

6.3.3 Simulation results and discussion

For the experiments we allocated a resource pool of 20 available machines of various computing power, memory and storage capabilities. The machines were distributed across 5 domains with different bandwidths within the domains and between the domains. We combined the domains with powerful computational resources linked by a low bandwith links with faster connected, but slower in performance resource domains. All the mentioned algorithms were tested in this environment on several experiment topologies consisting of various number of components: from 3 to 10. Figure 6.2 presents the results for a set of experiment topologies.

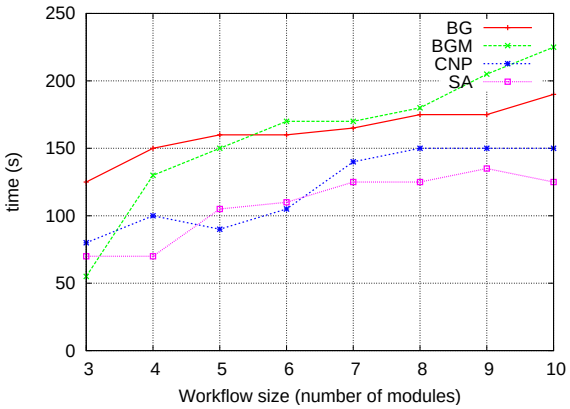


Figure 6.2: Simulation results of scheduling performance for different scheduling algorithms. The dependency of the workflow runtime on the amount of components executed.

We can see that the basic greedy algorithm usually gives one of the two worst results. The modified basic greedy algorithm has different behaviors: in topologies with small number of components it is very efficient and gives good results, though the more the size of experiment grows the less efficient it becomes. For the topologies with more than 6 components BGM gives the worst results among all the algorithms. Computation-network prioritized algorithm is examined with the value of f_c coefficient equal to 0.3, which proved to be the best for the test topologies during a separate experiment. In all cases it gives better results than basic greedy algorithms (both standard and modified) only except the case of 3 components when BGM is the most effective. The best results (except 3 components case) have been shown by simulated annealing algorithm, though it was the most time consuming one.

The simulation results have shown that heuristic mapping algorithms might be efficient in some system configurations, especially in small homogeneous environment, but for generic case and complex system topologies the algorithms of random search like simulated annealing are more promising. The drawback of such algorithms is the requirement for increased processing time caused by much more complex and

extensive computations taken. Typically, the time consumed by SA for scheduling the experimental topologies was around an order of magnitude higher that the time required by the straightforward heuristic algorithms, though the absolute value can still be neglected in the scale of workflow execution time: typical time for heuristic algorithms was limited by 1 second while SA took at most 10 seconds during the experiments.

6.4 VLAM-G: interactive data driven workflow management system for the Grid

6.4.1 The vision

The aim of the VLAM-G system is to provide and support coordinated execution of distributed Grid-enabled components combined in a workflow. This system combines the ability to take advantage of the underlying Grid infrastructure and a flexible high level rapid prototyping environment. On the high level, a distributed application is composed as a data driven workflow where each component represents a process or service on the Grid. Processes are activated only when the data is available on their input ports. The significant difference from other similar systems is the support for simultaneous execution of co-allocated processes on the Grid which enables direct data streaming between the distributed components: traditional batch processing of grid jobs and workflow execution based on input/output files exchange between the components is not suitable for many use case scenarios. This feature is highly required for semi-realtime distributed applications e.g. in the bio-medical domain or in online video processing and analysis [101].

Grid technology is maturing very quickly, the new paradigm based on SOA architecture is replacing the original resource oriented approach. However, the transition to the new paradigm will not take place overnight; moreover, the old approaches remain more efficient in a number of cases. One of the targets of VLAM-G is, thus, to enable support for co-existence of different types of grid execution models within a single workflow. This goal is achieved by abstracting a particular Grid execution model to an intermediate common representation. In VLAM-G a workflow is not composed of particular specific Grid jobs or services but of components with a special interface. These components are called modules, and they are the core entities of the VLAM-G data-driven workflow. Thus a module can represent a specially developed application which uses the VLAM-G native module API (`VLport`), a web service, or a legacy application.

The runtime control of the execution of a distributed workflow provides the ability to monitor the execution and influences the behavior of workflow components. VLAM-G supports several ways of runtime control: direct interaction with the user interface of a module (remote X GUI access) and module parameter control (reading flags and values set by a module and updating these values from outside the module). Monitoring delivers all the log data from remote modules to the VLAM-G user interface thus all the issues in module execution can be tracked centrally.

Intensive distributed data processing might take a long time. To facilitate the handling of the executing workflow, the system is capable of closing the user interface, detaching from the workflow engine and re-attaching later on during runtime.

The core features of the system we present are:

(1) Dataflow (and not control flow) is used as a driving force;

(2) Workflow components (VLAM-G modules) are versatile: a module may be either a specially developed software component (written in C++/Java/Python using the VLAM-G API), or an interface to legacy applications or web-services;

(3) Distributed execution: support for Grid job submissions together with web services and local tasks within a single workflow;

(4) Support for legacy applications wrapped as modules (flexible XML configuration);

(5) Support for remote graphical output: remote X display for Grid jobs is provided;

(6) Interactivity support: online control via parameters, and via remote graphical output;

(7) Decentralized handling of intermediate data;

(8) Decoupling of GUI and engine;

6.4.2 The architecture

In this section we present the architecture of the VLAM-G workflow engine which enables the execution and the runtime control of distributed data-driven workflows on the Grid. A workflow is composed of a set of components called modules which represent an executing entity (remote application or a web service). As computing resources VLAM-G can use: (1) grid enabled local or remote sites (so far VLAM-G works only with the Globus middleware) that enable inbound and outbound communications; this allows data streams to be established between workflow components located on remote grid resources; (2) web and grid services. Below we will concentrate on the architecture of the workflow engine starting from modules as workflow components and finishing with the description of the components the engine is built of.

The VLAM-G environment is constructed from two core components: the graphical user interface (VL-GUI) and the Run-Time System (RTS) – the engine which prepares and executes the workflow, handles the intermediate data and allows the monitoring and the online control. The VL-GUI is used to create a complex scientific experiment interactively by composing a workflow, setting the parameters for each workflow component before and at runtime, and transferring the workflow description to the engine using standard SOAP protocol.

In the following sub-sections, we describe the architecture of the RTS which consists of the RTS Manager (RTSM), the RTSM factory, and the Resource Manager.

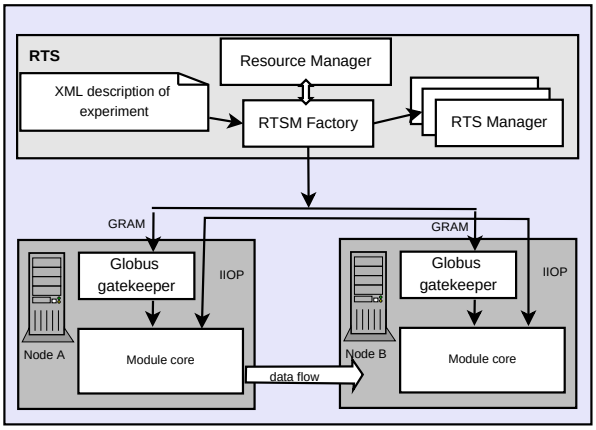


Figure 6.3: Run-Time System Architecture.

The workflow modules

Modules are the core composition elements of the VLAM-G framework. A module is an entity which represents a task to be executed on the Grid. It can be a specially developed application, a web service, or a legacy application. A workflow is composed of a set of modules. The modules have input and output ports which are used to compose the workflow by connecting the output of a module to the input of other modules.

VLAM-G workflows often target coordinated and simultaneous distributed processing of streaming data. Assuming this, a model of a module can be represented as the sequential retrieval of data from input ports, data processing and distribution of the result data to output ports. The execution is performed in a continuous fashion, so that these steps are repeated until interrupted or until the incoming data stream is closed.

A module represents a task being executed on the Grid and from the design point of view it consists of a processing part and a service part. The processing part represents the application itself: the code developed using the VLAM-G module API, a call of a web/grid service, or a launch of a legacy application. The service part provides the interface and the basic facilities for runtime control and the management of data transfers between modules. All available modules are stored in a repository, the creation and deployment of a new module is simple and straightforward and will be explained in the course of the chapter.

The Run-Time System architecture

Figure 6.3 shows a high level overview of the VLAM-G workflow engine: the RTS. The main components of the system are the RTSM Factory, the RTSM and the Resource Manager (RM). The scenario of a workflow enactment using the RTS is the following:

- The RTSM Factory creates an instance of the RTSM, the VLAM-G workflow engine. The RTSM instance is responsible for the given workflow execution. Multiple simultaneous executing workflows are supported. The engine takes as input a directed acyclic graph representing the dataflow of the workflow where the nodes correspond to the modules (workflow components) and the edges reflect the data streams.

- The computing resources where the modules are scheduled can be specified explicitly or retrieved from the RM. If the execution host of a module is not specified then the RM searches for optimal resources for executing the workflow by utilizing the information on module resource requirements. The RM handles the discovery, location, and selection of the necessary resources according to the VLAM-G module requirements. It maps the application tasks to the appropriate resources to optimize the workflow performance, utilizing a number of algorithms and scheduling techniques [65].

- After the resources have been selected, the workflow becomes fully concrete, and the RTSM schedules the workflow components using the Globus job submission mechanism (with non-web service modules), connects the module ports according to the workflow description and sets the module parameters to their default values.

- The RTSM starts the workflow and monitors the execution of each of the VLAM-G modules in this workflow.

VLAM-G resource management

To create an optimal schedule for a data-driven workflow we need to assign all the components composing the workflow to appropriate resources with the goal of minimizing the cost of the workflow execution. For this, each module in the VLAM-G framework is provided with a module description file that contains information about module resource requirements, including a default execution location which may be left blank.

Before creating an RTSM instance, the RTSM Factory contacts the RM. In turn, the RM processes the description of the workflow, with execution location missing for some modules, together with the requirements of the modules. The RM performs scheduling decisions based on the application information, the available resources information, and cost and application models (Figure 6.4). The application information includes the requirements that define the quality of the service requested by the modules. These requirements include the amount of memory needed, the approximate number of processing cycles (i.e. processor load), the storage and the communication load between modules. VLAM-G uses a RSL-like language to specify these requirements (RSL is a resource specification language used in the Globus toolkit to specify the job to be submitted to the Grid Resource Allocation Manager, [21]). The resource information is obtained from the Grid Monitoring and Discovery Service (MDS) [21], which also provides forecasts of the resource state from the Network Weather Service

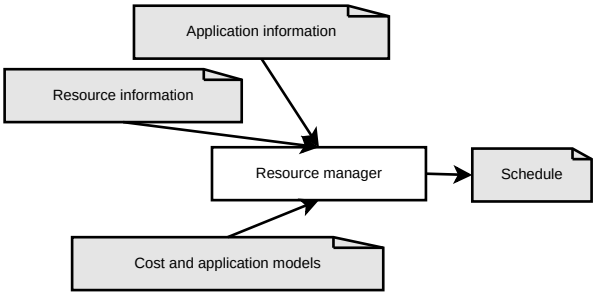


Figure 6.4: Resource Management in VLAM-G: Application and resource information along with cost and application models are processed by the resource manager to produce an execution schedule.

(NWS) [117]. This helps to estimate the resource load in the specified time frame in the future and to model the application performance. The cost and application models are used by the resource manager to evaluate a set of candidate schedules for the application. The cost model refers to the execution time of the workflow as the performance metric, and the application model specifies how the execution of the workflow is simulated. The application model is usually represented as a DAG which can be executed in a pipeline or concurrently, and the execution performance depends on the application and on the resource information.

The RM uses several types of heuristics and simulated annealing to achieve sub-optimal schedules based on a performance metric (generally the overall execution time is evaluated). For simulated annealing the RM tries a number of candidate schedules and simulates the execution of the workflow with given requirements on given resources. The results are estimated using a cost model, resource state information, and predictions. The algorithm converges to a sub-optimal schedule which is transferred to the RTSM Factory. Different types of meta-scheduling algorithms (heuristic algorithms and simulated annealing), the evaluation results, and analysis are discussed in detail in [65].

VLAM-G interactive module control

Each VLAM-G module may have parameters that can be monitored and changed during runtime by the module itself or by the user. A parameter in the VLAM-G context is a named entity with a value that can be changed either from within the module code or from outside, i.e. from the VL-GUI. A VLAM-G parameter can be compared to an environment variable of an operating system. This feature allows the users to interact directly with every module composing the application workflow, thus the execution of modules can be controlled on the fly without the need to stop and restart the whole workflow. A parameter is usually associated with a metric that needs to be controlled or is used to monitor the internal state of a module during execution.

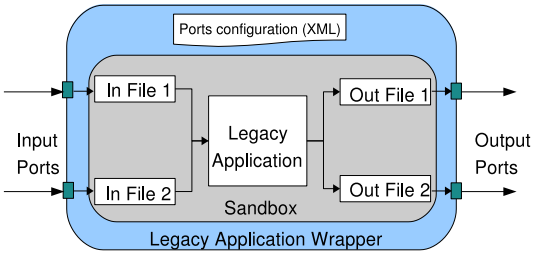


Figure 6.5: Legacy application wrapper: legacy applications operate input and output files that are mapped to VLAM-G ports by the wrapper. The wrapper is generic and can be configured using an XML configuration file.

Legacy application support

We define a legacy application (LA) as follows: LA is an application that has no source code available and cannot be modified. LAs are modeled as black box applications exchanging data files; both the input and output data needed for the execution of a LA is assumed to be a set of files. This model can be mapped directly to the VLAM-G module concept because input and output ports can be bound to the exchanged files. For the legacy applications, a generic wrapper has been developed (Figure 6.5).

Each LA has a configuration file where all input and output data files are described and bound to the input/output ports. The LA wrapper allows the RTSM to instantiate the LA in the same way as it instantiates a native VLAM-G module. The LA wrapper reads the data from input ports and stores them in the sandbox as files, then executes the LA code. The LA produces a set of output files stored again in the sandbox after processing the data. After the LA finishes the execution, the output files are sent to the associated output ports. This process is repeated until the stream of input files is over. Thus the processing is performed in a loop fashion: continuously repeating the retrieval of input files, the processing by the LA and the transfer of the output files, thus emulating the streams of input and output data.

The same technique is also applied to RPC style web services. Web services expose their operations in WSDL description. Each operation defined in this WSDL can be imported as a VLAM-G module by applying a similar approach as the one used by an LA. Thus a web service with several operations can be imported as a collection of VLAM-G modules. In turn, each of the parameters of every operation described in the web service WSDL is imported as an input port of the corresponding module, and its return values are described as output ports. VLAM-G provides a generic wrapper for this kind of web service operations The web service wrapper collects data from input ports and stores it in input files. A generic client for web services performs the web service calls, setting up operation parameters based on input files, and stores the result in an output file. Finally this output file is sent to the next module via the output port.

Other projects also address the problem of using legacy code in Grid environment. GEMLCA [31] enables the deployment of legacy code applications as Grid services

without the need of code re-engineering. Similarly to VLAM-G, the legacy code here is also provided as a black box with specified input and output parameters and environment requirements. The difference is that VLAM-G wraps an LA not as a web-service but as an entity supporting implicit data streaming.

6.4.3 VLport library: design and implementation

To provide implicit data exchange between the modules (remember that modules are not aware about existence of each other, they only read and write data through ports) a special connectivity layer was required. This layer maps abstract ports of modules to standard networking concepts (as sockets) and ensures that the modules are connected according to a workflow description and can communicate. Thus the `VLport` library was developed. The `VLport` library is responsible for maintaining the stream of data from an output port of a module to an input port of another module running on a different computing node. The library is a key component in moving intermediate data produced at a given step of the application workflow to the next step, regardless of the physical location of two processes.

The `VLport` provides the runtime environment for any VLAM-G module. It offers a basic API for the creation of I/O ports, changing the parameters of the VLAM-G modules, and other utility functions. From the implementation point of view the library provides a class, which must be used as base for developing any VLAM-G module. A module developer implements the abstract method `vlmain` (Figure 6.6 and listing 6.1), which contains all computational logic of the module. The RTSM instantiates, connects, and executes all modules composing the workflow. The input and output ports have a simple interface compatible with standard C++ streams. Legacy applications and web services are wrapped and represented to the workflow system as VLAM-G modules as well. Interfaces to Java and Python languages are provided so that native VLAM-G modules can be developed using these languages as well.

The library provides a number of utility functions that gives VLAM-G module developers easy access to all GASS data sources using GridFTP, FTP, HTTP and HTTPS protocol. The data streamed to/from ports can be serialized to the eXternal Data Representation (XDR). This makes it possible to exploit heterogeneous computational resources. However, module developers can also work with a raw stream, which is similar to the network sockets. The RTSM establishes all connections, a developer works with opened streams like any C++-compatible streams (i.e. iostream).

The RTS performs internal module control by using CORBA technology. CORBA has been selected as a stable and mature technology, not as resource-demanding as web services, to enable the management of centrally controlled distributed objects. For the RTS, each module is represented as a CORBA object [137]. Each instance of a module has IIOP CORBA reference and can be accessed remotely. In order to avoid firewall problems, the library chooses TCP ports from a range specified in GLOBUS_TCP_RANGE variable. Thus, a module can be instantiated even behind a firewall.

Listing 6.1: Example of a native VLAM-G module

```
#include <vlapp.h>
#include <fstream>

class MyVLApplication : public VL::VLApplication
{
public:
    MyVLApplication(int argc , char **argv)
        : VL::VLApplication(argc, argv), log("test_vlapp.log")
    {
        myOstream = createDefaultOPort("port_out");
    };
    virtual ~MyVLApplication()
    {
        delete myOstream;
    };
    int vlmain(int argc , char **argv)
    {
        std::ifstream fstr("File.orig.dat");
        time_t t1;
        time(&t1);
        *myOstream << fstr.rdbuf();
        std::cout  << "Time:" << time(NULL)-t1 << std::flush;
        return 0;
    };
private:
    VL::vostream  *myOstream;
    std::ofstream log;
};

int main(int argc , char **argv)
{
    globus_module_activate(GLOBUS_COMMON_MODULE);
    globus_module_activate(GLOBUS_IO_MODULE);
    try
    {
        MyVLApplication app(argc, argv);
        app.run();
    }
    catch(VL::Exception *e)
    {
        std::cerr << e->what();
        delete e;
    }
    globus_module_deactivate(GLOBUS_IO_MODULE);
    globus_module_deactivate(GLOBUS_COMMON_MODULE);
    return 0;
}
```

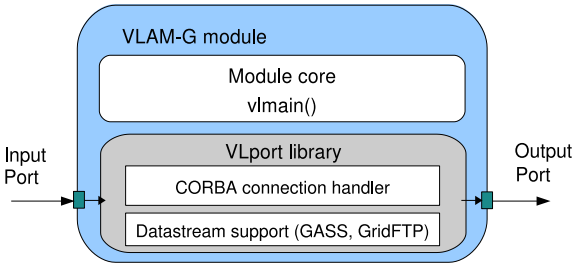


Figure 6.6: VLAM-G module structure: method vlmain() implemented by a module developer and supported by VLport library for transparent data transfers to other modules within a workflow.

The life cycle of a module can be described as follows:

1. *Initializing a module:* The RTSM instantiates a module on a grid-enabled (using the GRAM protocol) or local node. All necessary environment variables are set for proper module initialization.

2. *Creating input/output ports:* During the initialization stage the input and output ports are created. The number of ports is predefined for a module and can not be changed during the runtime.

3. *Registering a module:* The module contacts the RTSM and registers itself. The RTSM keeps sending keep-alive messages to the module during the runtime. If acknowledgment is not received during a predefined timeout the module is considered crashed.

4. *Connecting modules:* The RTSM connects modules with each other according to the dataflow; an output port can be connected with many input ports (one-to-many relationship);

5. *Scheduling a module:* The RTSM schedules the modules for execution (the method `vlmain` is executed);

6. *Exchanging data with other modules:* The module reads the data from input ports, processes it and writes results to output ports: loop fashion execution;

7. *Module termination:* When a module exits (i.e. input port has been closed by the previous module) it returns from `vlmain` method. All pending buffers are flushed, and ports are closed. The module unregisters itself from RTSM registry and exits.

The control of the parameters of a VLAM-G module is implemented as CORBA remote procedure calls. All the modules in the worklfow are controlled by the RTSM using IIOP protocol.

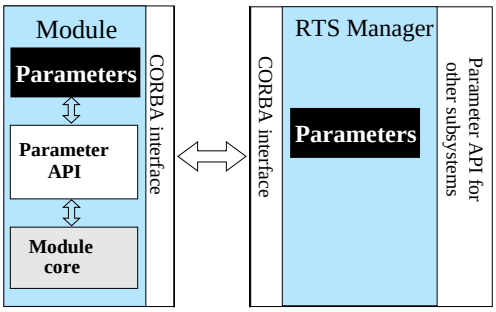


Figure 6.7: Module parameter interface: RTSM controls (sets and gets) parameters of a module via CORBA interface; module core controls parameters via specified API from VLport library.

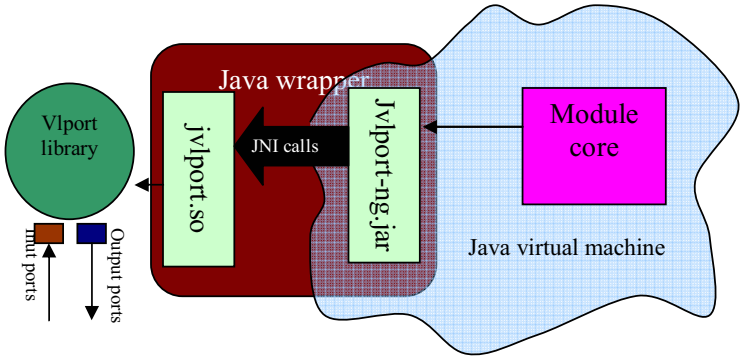


Figure 6.8: Java wrapper: module developers can implement modules in Java; Java calls are translated to C++ `VLport` library by the wrapper using Java Native Interface.

To utilize various programming languages such as Java and Python a set of wrappers has been developed. The wrappers translate the calls from the target language to the C++ `VLport` library. Figure 6.8 shows the Java wrapper library based on Java Native Interface (JNI). The VLAM-G module core and the wrapper are executed in the same Java virtual machine, the wrapper functions translate the calls to the `VLport` library. Thus a Java module has the same functionality and life-cycle as a native module. The Python wrapper employs similar techniques using the SWIG (simplified wrapper and interface generator) to generate a wrapper around the `VLport` library. Similar solutions for code wrapping have been employed in other projects, for example Jopera [94] provides a wrapper for Java snippets, small blocks of Java code usually needed to preform small computation. It also provides a wrapper for Unix applications (legacy applications) using the shell command line and the pipe-based inter-processes communication.

The VLAM-G port library is developed in C++ and is implemented as a dynamic library for UNIX systems. It uses the threaded versions of Globus libraries and

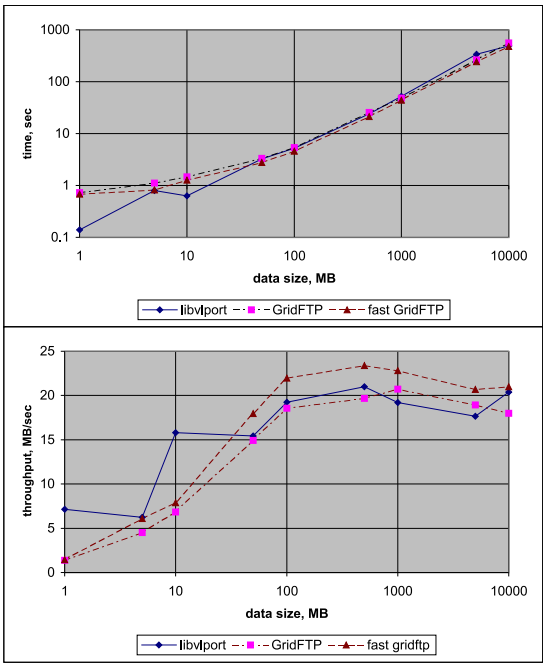


Figure 6.9: Average performance of the RTS library on WAN compared to standard Globus data transfer tool.

OmniORB - a free high performance ORB for C++ [137]. The RTSM is a part of the VLAM-G front-end and is developed in Java. It uses Commodity Grid Kit for Java [125] and Community OpenORB Project [126].

6.4.4 Performance evaluation

The RTS library is designed to provide maximal performance for distributed applications while hiding all low level details from the application developers. It must thus provide a throughput comparable with the standard protocols. In this section we compare the performance of the library with the performance of standard data transfer tools included to the Globus toolkit, and measure the introduced overhead.

We evaluated the dependency of the data transfer performance on the data block size using both standard Globus utilities and `VLport` library. The data transfers took place between the nodes of clusters connected through a high-speed WAN (wide area network). The clusters are part of the ASCI Supercomputer 2 (DAS 2), which is a cluster system distributed over different universities in the Netherlands [130]. To measure the overhead introduced by the VLAM-G library, the average throughput of the link was evaluated with the help of standard Globus tool 'globus-url-copy' using GridFTP protocol. Figure 6.9 shows the data transfer rate as a function of the data block size. Compared to the throughput obtained with the VLAM-G RTS library,

the globus-url-copy shows slightly better performance for the test with parallel stripes (fast GridFTP) and almost the same outcome in the case of a standard GridFTP protocol. The maximum data throughput for large data blocks is about 20 MB/sec for the library. Since the RTS Library is designed to support data streams, the performed tests show encouraging results as the speed of the data transfer achieved using the library is comparable to the one achieved using standard Globus tools.

6.5 Multi-layered application as a workflow

Chapter 3 outlined the challenges posed by the Grid to the multi-layered complex applications. We illustrated the generic method to deploy such kind of multi-component applications on a case study of the Virtual Reactor application. The proposed workflow for the Virtual Reactor experiment is shown in Figure 3.3.

Several approaches to the execution of the Virtual Reactor in high-performance computing environments have been examined. The initial version has been developed to be executed in a parallel computing environment via a Web portal interface [70]. Later, the Migrating Desktop was used to run the VR components on the Grid resources (section 3.2.2). But no solution was proposed to seamlessly integrate all the components into a single sequence of tasks that would not require the manual intervention and control on each stage of the VR workflow. Such a solution can be provided by a workflow management system as it is specifically designed for automatic sequencing of multiple tasks.

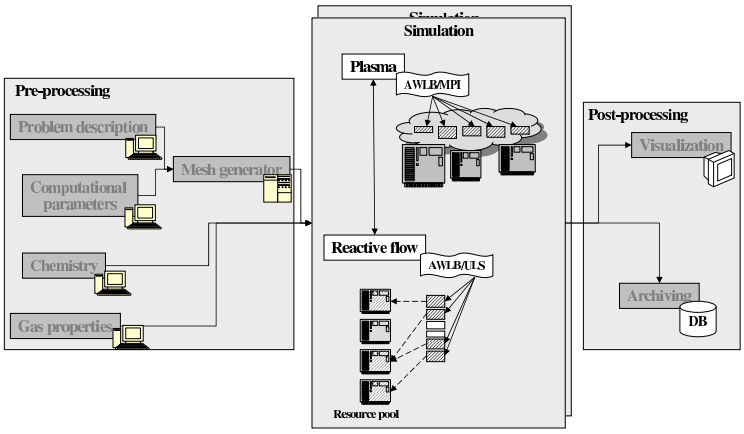


Figure 6.10: Multiple layers of processing within the Virtual Reactor workflow.

In the previous chapters we described the resource management and workload balancing for the components from different layers of a complex distributed application: starting from the task-level parallelism on heterogeneous systems in Chapter 3, its development on multi-cluster environment in Chapter 4 to job-level parallelism in Chapter 5. Each chapter covers a specific type of application execution in a dis-

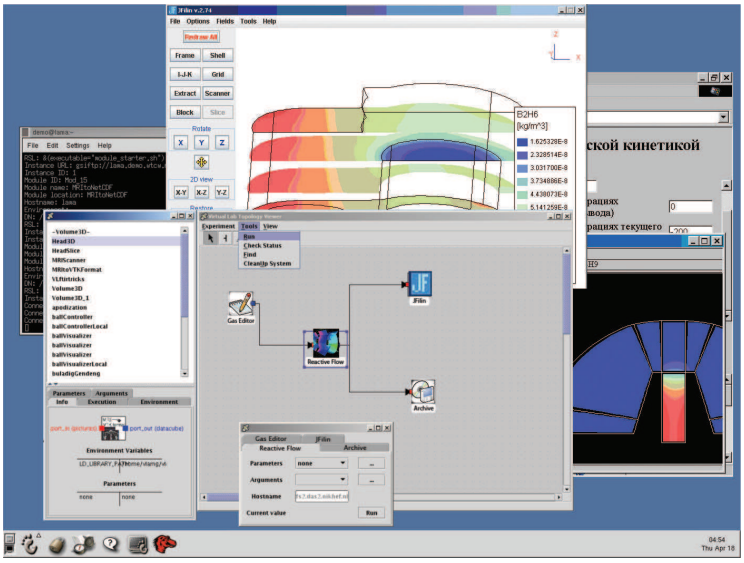


Figure 6.11: Virtual Reactor workflow in VLAM-G.

tributed environment, and typically each of these types can be used by a component of the complex distributed application, in particular the Virtual Reactor. A single workflow can include all these types of processing as its components. The workflow management system performs high-level resource management, leaving the particular management decisions for the components itself, thus the hierarchy is built: WMS assigns the components to computing systems, in particular homo- and heterogeneous clusters, and task management systems (like User-Level Scheduling). Figure 6.10 shows the Virtual Reactor workflow with the types of processing assigned to its components.

The prototype multilayer Virtual Reactor workflow has been implemented in the VLAM-G environment. The components are wrapped as VLAM-G modules. The workflow management system controls the data exchange between the modules (the standard way provided by VLAM-G is used instead of file transfers) and the sequence of modules execution. The results of processing can be archived or visualized in a virtual desktop provided by VLAM-G. The basic VR workflow is shown in Figure 6.11.

6.6 Conclusions

In this chapter we propose and compare several algorithms based on ad-hoc greedy model and on the random search technique that are adapted for data-driven workflow scheduling in the Grid environment. We study the behavior of these algorithms using simulation model of a Grid workflow. The results show that the heuristic greedy algorithms may be effective for mapping in specific network configurations especially

in small and homogeneous environment but for effective scheduling in heterogeneous environment more complex algorithms are needed. For the latter we propose using random search algorithms e.g. simulated annealing algorithm. These algorithms allow to avoid local minima and to get result closer to global optimum, but their drawback is the significant time needed to find sub-optimal schedule. Our experiments show that the random search approach is promising for scheduling in Grid environments.

As the implementation of a workflow management system we present the VLAM-G – a data-driven WMS, and the Run-Time System, its engine. This engine allows the execution of data-driven workflows in a transparent way on the available Grid resources. A VLAM-G workflow is composed of entities called modules that interface native VLAM-G applications, web services and legacy applications even if the source code is not available for modification. Interactive control of the execution is provided: the end-users can interact with any workflow component at runtime via a simple parameter interface or by accessing a virtual display with remote graphical output. The API for native VLAM-G modules is provided by `VLport` runtime libraries which have been designed to support different languages: C / C++, Java and Python. A generic wrapper provides the means to port an existing application or a web service as a standard VLAM-G workflow component.

The VLAM-G workflow management system can be used for distributed applications requiring direct data streaming between the remote components, e.g. semi-realtime applications, remote devices access and control etc. The performance evaluation showed that the overhead brought by intermediate `VLport` library is negligible.

As the top layer of the complex application hierarchy, a workflow embraces the components representing the application of lower layers of the hierarchy, introduced in the previous chapters. A single framework performs the management of these different layers, and the execution of each component is coordinated with the execution of the other components. The organization of such a composite workflow is illustrated with a sample workflow developed for the driving application - the Virtual Reactor.

Chapter 7. Summary and conclusions

The work presented in this thesis addresses the problems of resource management in distributed computing from the perspective of multi-layer complex distributed applications. The challenge is to study all the different layers of application parallelism bound into a single hierarchy and propose methods of workload balancing and resource management that are generic enough not to depend on a particular technology of distributed computing and can be reflected and used on the different layers of the application hierarchy.

While talking about technology invariant approaches to efficient resource management for complex parallel applications in distributed heterogeneous environment, we propose and study the adaptive workload balancing (AWLB) approach that can be used in different environments and with different distributed computing technologies. This method of adaptive workload balancing depends on the dynamic characteristics of both application and resources. Traditionally there are two approaches to workload balancing of parallel applications: (1) carefully calculate the distribution of the workload, taking into account all the properties of the environment and application – a time and resource consuming task requiring expert knowledge of the application structure and algorithms used; and (2) distribute the workload in a straightforward way, at best considering only the processing power of the worker nodes – a fast but not very efficient way in terms of parallel performance. We propose an intermediate approach that combines the fast calculation of initial approximation of the workload distribution and its further iterative refinement converging close to the optimum.

Discussing the complex application hierarchy and the ways to decompose complex applications to different layers and manage these layers, we consider a three-layer structure of distribution and parallelism: task (e.g. single parallel application), job (e.g. parameter sweep and task farming), and workflow (functional decomposition of the application). Each layer of this structure has its own requirements to maximize the performance and enable good scalability. Thus appropriate resource management and workload balancing has to be addressed while building the distributed multi-layer and multi-component application as a whole.

Chapter 3 addresses the lowest layer in the complex application hierarchy - a single parallel application running on heterogeneous resources. One of the most challenging problems in porting parallel distributed applications from homogeneous cluster environments to heterogeneous resources is to keep up a high level of parallel efficiency of the computational components. To tackle this problem, we developed a theoretical approach and a generic workload balancing technique that takes into account specific parameters of the resources dynamically assigned to a parallel job, as well as the

application requirements. Here we introduce the AWLB methodology and apply it to the parallel solvers of the Virtual Reactor - one of the driving application of this research.

The case of a single parallel application spanning over a set of parallel computers is described in Chapter 4. The possibility to use a large amount of resources raises the question if and when several parallel computers will indeed bring any performance benefit when used together for a single parallel application. The theoretical approach to estimate the possible speedup of a parallel application in a homogeneous computational Grid is introduced by Hoekstra and Sloot in [49]; here we examine the proposed approach using a real application as a test-case: a Lattice Boltzmann Equation (LBM) solver; A simple model of this application is used for prediction of possible performance gain from the multi-cluster distribution. Compared to the solution for parallel applications spanning over a set of heterogeneous resources proposed in Chapter 3, the approach presented in this chapter is valid for homogeneous Grids, but it gives rather high accuracy in predicting the application speedup using a simple application model and a set of basic environment characteristics.

Another application hierarchy layer that is called a multi-job application is presented in Chapter 5. In this type of the distributed application data processing is organized in a set of separate components that perform information exchange only at the start and finish of the execution. In this chapter the AWLB method developed in Chapter 3 is applied to multi-job applications with divisible workload corroborating the technology independence of this method. We present a hybrid resource management environment, operating on both application and system levels, developed for minimizing the execution time of parallel applications with divisible workload on heterogeneous Grid resources. This integrated environment consists of application-level AWLB applied to User-Level Scheduling (ULS) environment. The latter fills the gap between the application and Grid resource managers: this user-level middleware is a customizable, application-centric scheduler and application hosting environment. Dynamic benchmarking of resources and estimation of the application characteristics is used to optimize the usage of a dynamic user-level pool of Grid resources maintained by the ULS. To prove the concept we perform the experiments with a synthetic application with configurable requirements using EGEE Grid resources and the AWLB algorithm incorporated into the DIANE user-level scheduler. We present the experimental results and discuss the ways to manage the workload of divisible load parallel applications on the Grid, compare different workload distribution methods, illustrate the usage of dynamic resource pool and application performance dependencies with adaptive resource selection.

Chapter 6 discusses the highest layer of the complex distributed application hierarchy: the Grid workflow. The workflows controlled by a dataflow are in the focus of this study: a model of a data-driven workflow is discussed and different strategies of resource management for this type of workflows are evaluated. To be executed on the Grid a distributed workflow needs a framework that enables the workflow enactment and execution control: a workflow management system (WMS). This chapter presents the VLAM-G and its core component, the Run-Time System (RTS), as an implementation of a data-driven WMS. The RTS is a dataflow driven workflow engine

which utilizes Grid resources, hiding the complexity of the Grid from users. Special attention is paid to the concept of dataflow and direct data streaming between distributed workflow components. The architecture, components of the RTS, and the features of VLAM-G workflow execution are presented. As the top layer of the complex application hierarchy, a workflow can perform the management of components representing different lower layers, and the execution of each component is coordinated with the others. This capability of workflows and in particular the VLAM-G WMS is illustrated with a workflow developed for the Virtual Reactor application, which embraces the components of different layers.

The presented work covers the area of execution of multi-component and multi-layer applications in the distributed environment. From the bottom to the top layers of the application hierarchy we describe the methods of appropriate resource management, present implementation and experimental validation of the proposed approaches.

Now it's time to refer back and reconsider the questions posed in Chapter 1. The adaptive workload balancing algorithm was developed to be technology independent and to contain as little application specific properties as possible. On the other hand, it proved its efficiency being used in different environments and employed by different technologies: Message Passing Interface in Chapter 3 and User-Level Scheduling in Chapter 5. Moreover, we were able to use it on different layers of the application hierarchy. Mapping of the application layers and their management was studied throughout Chapters 3, 5, and 6 starting from parallel applications on heterogeneous resources and finishing with distributed workflows managed by a workflow management system. The question about the benefits of a distribution of a single parallel application across a set of Grid resources was addressed in Chapter 4 where the Grid speedup and efficiency metrics were introduced and used to evaluate possible benefits.

In the future research we plan to address the issues of resource and service management within Grids built with Service Oriented Architecture. This modern paradigm seems to require new approaches at a first glance, and the challenge is to apply the ideas and methods presented in this thesis to this type of computing environment. Efficient management and orchestration of Grid services is an essential part of modern workflow management systems, and the plans are to enhance the newest generation of VLAM-G WMS with the capabilities of advanced resource management for service-oriented workflows.

Publications

Journal publications

1. V.V. Korkhov, J.T. Moscicki, V.V. Krzhizhanovskaya. User-Level Scheduling of Divisible Load Parallel Applications with Resource Selection and Adaptive Workload Balancing on the Grid. IEEE Systems Journal - Special Issue on Grid Resource Management, Vol 3(1), 2009, pp. 121-130.

2. V.V. Korkhov, J.T. Moscicki, V.V. Krzhizhanovskaya. Dynamic Workload Balancing of Parallel Applications with User-Level Scheduling on the Grid. Future Generation Computer Systems, Vol 25(1) 2009, pp. 28-34.

3. V.V. Korkhov, V.V. Krzhizhanovskaya and P.M.A. Sloot. A Grid Based Virtual Reactor: Parallel Performance and Adaptive Load Balancing. Journal of Parallel and Distributed Computing, Vol 68(5), 2008, pp 596-608, Elsevier

4. V. Korkhov, D. Vasunin, A. Wibisono, A. Belloum, M. Inda, M. Roos, T. Breit, L.O. Hertzberger. VLAM-G: Interactive Dataflow Driven Engine for Grid-enabled Resources. Journal of Scientific Programming Vol 15(3), 2007, pp. 173-188

5. Z.W. Hendrikse, A. Belloum, P.M.R. Jonkergouw, G.B. Eijkel, R.M.A. Heeren, L.O. Hertzberger, V.Korkhov, C. de Laat, D. Vasunin. Evaluating the VLAM-G toolkit on the DAS-2. Future Generation Computer Systems Vol 19(6), 2003, pp. 815-824

6. A.S.Z Belloum, D.Groep, L.O. Hertzberger, V. Korkhov, C. de Laat, D. Vasunin. VLAM-G: a Grid-based Virtual Laboratory. Future Generation Computer Systems, Vol 19(2), 2003, pp. 209-217.

7. H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.J. van den Brand, G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D. Vasunin, A. Visser and H.H. Yakali. VLAM-G: A Grid-based Virtual Laboratory. Scientific Programming, (Special issue on Grid Computing) Vol 10(2), 2002, pp. 173-181. (R.H. Perrott and B.K. Szymanski, editors), IOS Press

8. A. Chevel, V. Korkhov. Experiments with Grid network fragments. Open Systems, No 05-06, 2001, Open Systems Publishing, Russia

Books

1. A. Bogdanov, V. Korkhov, V. Mareev, E. Stankova. Architectures and Topologies of Multiprocessor Computing Systems (book of lecture notes). Published by Internet University of Information Technologies (INTUIT), 2004, ISBN 5-9556-0018-3, 176 pages

Conference proceedings

IEEE, ACM and LNCS publications

1. V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, A.Belloum, C. de Laat, P. Adriaans, L.O. Hertzberger. WS-VLAM: Towards a Scalable Workflow System on the Grid. 16th IEEE International Symposium on High Performance Distributed Computing, Proceedings of the 2nd workshop on Workflows in support of large-scale science (WORKS'07), pp 63-68, ISBN 978-1-59593-715-5, Monterey Bay, California, USA, June 25-29, 2007 Publisher: ACM New York, NY, USA

2. V.V. Krzhizhanovskaya and V.V. Korkhov. Dynamic Load Balancing of Black-Box Applications with a Resource Selection Mechanism on Heterogeneous Resources of the Grid. Proceedings of Conference on Parallel Computing Technologies (PaCT'07), Lecture Notes in Computer Science, Vol 4671, pp. 245-260 (2007), September 3-7, 2007, Pereslavl-Zalessky, Russia

3. A. Wibisono, D. Vasyunin, V. Korkhov, Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, B. Hertzberger. WS-VLAM: a GT4 based workflow management system. Proceedings of International Conference on Computational Science (ICCS'07), Lecture Notes in Computer Science, Vol 4489, pp. 191-198 (2007), ISBN 978-3-540-72587-9

4. V.V. Krzhizhanovskaya, V.V. Korkhov, A. Tirado-Ramos, D.J. Groen, I.V. Shoshmina, I.A. Valuev, I.V. Morozov, N.V. Malyshkin, Y.E. Gorbachev, P.M.A. Sloot. Computational Engineering on the Grid: Crafting a Distributed Virtual Reactor. Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), Amsterdam, the Netherlands, December 4-6, 2006, pp.101. IEEE CS Press.

5. V. Korkhov, V. Krzhizhanovskaya. Benchmarking and Adaptive Load Balancing of the Virtual Reactor Application on the Russian-Dutch Grid, Proceedings of the 6th International Conference on Computational Science (ICCS'06), Reading, UK, May 28-31, 2006, Part I, Lecture Notes in Computer Science, Vol 3991, pp. 530-538. Springer Berlin / Heidelberg 2006. ISBN: 3-540-34379-2. DOI: 10.1007/11758501

Other publications

1. V.V. Krzhizhanovskaya, V.V. Korkhov, M.A. Zatevakhin, Yu.E. Gorbachev Parallel Distributed Computing in Modeling of the Nanomaterials Production Technologies. Proceedings of Parallel Computing Technologies (PAVT'2008) international scientific conference (St.Petersburg, 28 Jan - 1 Feb 2008). ISBN 978-5-696-03720-2, pp. 585-590, YUSU, Chelyabinsk, Russia, 2008.

2. V.V. Krzhizhanovskaya and V.V. Korkhov. Problem-Solving Environments for Simulation and Optimization on Heterogeneous Distributed Computational Resources of the Grid. Proceedings of the Third International Conference on Parallel Computations and Control Problems (PACO 2006), Moscow, Russia, October 2-4, 2006. Publ: V.A. Trapeznikov Institute of Control Sciences RAS, ISBN 5-201-14990-1, pp. 917-932, Moscow, Russia, 2006.

3. V.V. Krzhizhanovskaya, V.V. Korkhov, P.M.A. Sloot. Virtual Reactor: a distributed computing environment for simulation of plasma chemical processes on heterogeneous resources of the Grid. All-Russian conference "Scientific services on the Internet: Parallel Programming Technologies". Novorossiysk, Russia, 18-23 September 2006.

4. V.V. Korkhov, V.V. Krzhizhanovskaya. Workload Balancing in Heterogeneous Grid Environment: A Virtual Reactor Case Study. Proceedings of the Second International Conference on Distributed Computing and Grid Technologies in Science and Education. JINR, D11-2006-167, ISBN 5-9530-0138-X, pp. 103-113, Dubna, Russia, 2006.

5. V.V. Korkhov, V.V. Krzhizhanovskaya. Workload Balancing in Heterogeneous Grid Environment: A Virtual Reactor Case Study. Book of abstracts of the Second International Conference on Distributed Computing and Grid Technologies in Science and Education. JINR, p. 93. ISBN 5-9530-0117-7. Dubna, Russia, 2006

6. I. Morozov, I. Shoshmina, A. Evlampiev, E.Stankova, A. Bogdanov, A. Luzan, D. Malashonok, I. Valuev, V. Korkhov. Experience in setting up an experimental Grid testbed for heavy scientific applications, All-Russian Scientific Conference on Scientific Service in Internet: distributed computing technologies, Abrau-Durso, Russia, September 19-24, 2005.

7. I. Shoshmina, V. Korkhov, D. Malashonok, A. Bogdanov, Evaluation of a Grid testbed between IHPC&IS and SPbSU PTC, Proceedings of XI All-Russian Guidance Conference "Telematics'2004", St.Petersburg, Russia, 2004

8. V. Korkhov, A. Bogdanov, L.O. Hertzberger. On Issues of Resource Management in Grid Environment, Proceedings of Intl Conference on Distributed Computing and Grid Technologies in science and education, Dubna, Russia, 2004

9. V. Korkhov, A.S.Z Belloum, and L.O. Hertzberger, VL-e: Approach to design a Grid-based Virtual Laboratory, Proceedings of 5th Workshop on Distributed and Parallel Systems, pp 21-28, ISBN 0-387-23094-7, Budapest, Hungary, September 19-22, 2004.

10. V. Korkhov, A.S.Z Belloum, and L.O. Hertzberger. Evaluating Meta-scheduling Algorithms in VLAM-G environment, Proceedings of ASCI'04 conference, Zeewolde, The Netherlands, pp. 87-94, 2004.

11. V. Korkhov, A. Bogdanov. Application of Grid technologies and Resource Management Issues, Proceedings of X All-Russian Guidance Conference "Telematics'2003", St.Petersburg, Russia, 2003.

12. A. Chevel, V. Korkhov. Deployment of Globus Tools at St.Petersburg (Russia), International Conference on Computing in High Energy and Nuclear Physics (CHEP'01), Beijing, China, September 3-7, 2001.

Bibliography

[1] A. Abraham, R. Buyya, and B. Nath. Nature's Heuristics for Scheduling Jobs on Computational Grids. In *Proceedings of ADCOM 2000, Cochin, India*, pages 45–52, December 14-16 2000.

[2] H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.J. van den Brand, G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D. Vasunin, A. Visser, and H.H. Yakali. VLAM-G: A Grid-Based Virtual Laboratory. *Scientific Programming Vol 10, No 2, pp. 173-181*, 2002.

[3] I. Ahmad and A. Ghafoor. Semi-distributed load balancing for massively parallel multicomputer systems, *IEEE Transactions on Software Engineering, Vol. 17(10) pp. 987-1004*, 1991.

[4] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a Research Framework for High-performance Grid Programming Environments. In *Jose C. Cunha and Omer F. Rana, editors, Grid Computing: Software environments and Tools, chapter 10, pp. 230-256*, Springer, Jan. 2006.

[5] G. Allen, K. Davis, K. Dolkas, N. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling Applications on the Grid: A GridLab Overview. *International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications*, August 2003.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM*, pages 423–424, 2004.

[7] K. Antonis, J. Garofalakis, I. Mourtos, and P. Spirakis. A hierarchical adaptive distributive algorithm for load balancing. *Journal of Parallel and Distributed Computing Vol.64(1) pp 151-162*, 2004.

[8] L. Axner. *High Performance Computational Hemodynamics with the Lattics Boltzmann Method*. PhD thesis, Universiteit van Amsterdam, (Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Dr. A.G. Hoekstra), December 2007. ISBN: 978-90-5776-170-6.

[9] A. Barak and O. La'adan. The MOSIX Multicomputer Operating System for High Perfomance Cluster Computing. *Future Generation Computer Systems (13) vol.4-5, pp. 361-372*, 1998.

[10] A. Barak and A. Shiloh. A distributed load balancing policy multicomputing, *Software Practice and Experience, Vol. 15(9) pp. 901-913*, 1985.

[11] K. Benedyczak, A. Nowinski, K. Nowinski, and P. Bala. UniGrids Streaming Framework. Enabling streaming for the new generation grid. In *ICM, PARA 2006 Umea Sweden*, 2006.

[12] F. Berman. High-Performance Schedulers, In *I. Foster, C. Kesselman (Eds), The Grid: Blueprint for a New Computing Infrastructure, 1st edition, Morgan Kaufmann*, 1999.

[13] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS, *IEEE Transactions on Parallel and Distributed Systems, Vol. 14(4) pp. 369-382*, 2003.

[14] J. Blower, K. Haines, and E. Llewellin. Data streaming, workflow and firewall-friendly Grid Services with Styx. In *Proceedings of the UK e-Science All Hands Meeting, 19-22 September*, 2005.

[15] S.H. Bokhari. On the mapping problem. *IEEE Transactions on Computers, C-30 (3) pp. 207-214*, 1981.

[16] T.N. Bui and C. Jones. Parallel algorithms for partitioning simple classes of graphs. In *Proceedings of the 19th International Conference on Parallel Processing, Urbana-Champaign, IL. Pennsylvania State University Press: University Park, PA, pp. 150-153*, 1990.

[17] G. Carrera, E. de Andres, J.T. Moscicki, A. Muraru, S.H.W. Scheres, and J.M. Carazo. Heavy computational tasks on the EGEE Grid: 2D/3D maximum-likelihood refinement. *Network of Excellence 3DEM Annual Meeting, Palma*, Jan. 2007.

[18] T.L. Casavant and J.G. Kuhl. Analysis of three dynamic load balancing strategies with varying global information requirements. In *Proceedings of the 7th International Conference on Distributed Computing Systems, pp. 185-192*, 1987.

[19] S. M. Charters, N. S. Holliman, and M. Munro. Visualisation on the Grid: A Web Service Approach. *UK e-Science All Hands Meeting*, September 2004.

[20] G. Chin, L.R. Leung, K. Schuchardt, and D. Gracio. New paradigms in problem solving environments for scientific computing. In *Proceedings of the international conference of Intelligent User Interface*, San Francisco, 2002.

[21] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. *The Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press*, August 2001.

[22] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 2002.

[23] K. Czajkowski, I. Foster, and C. Kesselman. Resource and Service Management, In *I. Foster, C. Kesselman (Eds), The Grid: Blueprint for a New Computing Infrastructure, 2nd edition, Morgan Kaufmann*, 2004.

[24] S.P. Dandamudi. Sensitivity evaluation of dynamic load sharing in distributed systems. *IEEE Concurrency, Vol. 6(3) pp 62-72*, 1998.

[25] F. Darema. SPMD model: past, present and future. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001. Lecture Notes in Computer Science 2131, p. 1*, 2001.

[26] A. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *4th International Conference on Linux Clusters: The HPC Revolution 2003 in conjunction with ClusterWorld Conference and Expo*, 2003.

[27] R. David, S. Genaud, A. Giersch, B. Schwarz, and E. Violard. Source code transformations strategies to load-balance grid applications. In *In Grid Computing GRID 2002: Third International Workshop, volume 2536 of Lecture Notes in Computer Science*, pages 82–87. Springer-Verlag, 2002.

[28] J. de Ronde. *Mapping in High performance Computing. A case study on Finite Element Simulation.* PhD thesis, University of Amsterdam (Promotor: Prof. Dr. P.M.A. Sloot, co-promotor: Prof. Dr. L.O. Hertzberger), 1998.

[29] J. de Ronde, A. Schoneveld, and P.M.A. Sloot. Load Balancing by Redundant Decomposition and Mapping, *Future Generation Computer Systems, V. 12, N 5 pp. 391-407*, April 1997.

[30] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems, Vol. 25(5), pp. 528-540*, 2009.

[31] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter, and P. Kacsuk. GEMLCA: Running Legacy Code Applications as Grid Services. *Journal of Grid Computing Vol. 3. No. 1-2. pp 75-90*, June 2005.

[32] T. Fahringer, A Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr, and H. Truong. Askalon: a tool set for cluster and Grid computing. *Concurrency and Computation: Practice and Experience, Vol. 17 pp. 143-169*, 2005.

[33] I. Foster and C. Kesselman (Eds). *The Grid: Blueprint for a New Computing Infrastructure. Second Edition.* Morgan Kaufmann, 2004.

[34] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications, Vol. 15(3), pp. 200-222*, 2001.

[35] G. Fox. Grid Computing environments. *IEEE Computers in Science and Engineering., V. 10, pp. 68-72*, 2003.

[36] G. Fox, G. Aydin, H. Bulut, H. Gadgil, S. Pallickara, M. Pierce, and W. Wu. Management of Real-Time Streaming Data Grid Services. *Concurrency and Computation: Practice and Experience, Special Issue from Grid and Cooperative Computing 4th International Conference November 30 to December 3 2005 Beijing China*, 2006.

[37] G. Fox and D. Gannon. Workflow in Grid Systems. *Concurrency and Computation: Practice and Experience, Vol. 18(10), pp. 1009-1019*, 2006.

[38] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D.Walker. Solving Problems on Concurrent Processors, volume 1, Prentice-Hall, 1988.

[39] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Cluster Computing Journal, V. 5, N 3, pp. 237-246*, 2002.

[40] F. Gagliardi, B. Jones, F. Grey, M.E. Begin, and M. Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project, *Philosophical Transactions of the Royal Society A. V. 363, Issue 1833, pp. 1729 - 1742*, 2005.

[41] E. Gallopoulos, E. Houstis, and J.R. Rice. Computer as thinker doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 2:13–23, 1994.

[42] D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy, and L. Fang. Building Applications from a Web Service based Component Architecture. *Proc. of the Workshop on Component Models and Systems for Grid Applications, June 26, 2004, Saint Malo, France. Springer*, 2005.

[43] C. Germain-Renaud, C. Loomis, J.T. Moscicki, and R. Texier. Scheduling for Responsive Grids. *Grid Computing Journal, Special Issue on EGEE User Forum*, 2006.

[44] C. Germain-Renaud, C. Loomis, J.T. Moscicki, and R. Texier. Scheduling for Responsive Grids. *Grid Computing Journal, Special Issue on EGEE User Forum*, 2006.

[45] C. Germain-Renaud, R. Texier, and A. Osorio. Interactive Reconstruction and Measurement on the Grid. *Methods of Information in Medicine, Vol. 44(2) pp. 227-232*, 2005.

[46] A. Hac and T.J. Johnson. Sensitivity study of load balancing algorithm in a distributed system, *Journal of Parallel Distributed Computing, 10 (1) pp 85-89*, 1990.

[47] B. Hendrickson and R.W. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of Supercomputing, San Diego, CA, 1995. IEEE Computer Society Press: Los Alamitos, CA, 1995*, 1995.

[48] T. Hey and A. Trefethen. The UK e-science core programme and the grid. *Future Generation Computer System*, 18(8):1017–1031, 2002.

[49] A.G. Hoekstra and P.M.A. Sloot. Introducing Grid Speedup Γ: A Scalability Metric for Parallel Applications on the Grid. In *in P.M.A. Sloot; A.G. Hoekstra; T. Priol; A. Reinefeld and M.T. Bubak, editors, Advances in Grid Computing - EGC 2005, in Lecture Notes in Computer Science, vol. 3470, pp. 245-254*, 2005.

[50] E.N. Houstis and J.R. Rice. Future problem solving environments for computational science. *Mathematics and Computers in Simulation, Vol. 54, pp. 243-257*, 2000.

[51] K. Hwang. Advanced Computer Architecture, Parallelism, Scalability, Programmability (McGraw-Hill, New York), chapter 3, 1993.

[52] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the Ptolemy Project. Technical report, University of California at Berkeley, July 2003.

[53] Internet2. Virtual Laboratory: An Application Environment for computational Science and Engineering. http://www.internet2.edu/html/virtual_laboratory.html.

[54] S. Iqbal. Load balancing strategies for parallel architectures, *Ph.D. Thesis, Univeristy of Texas at Austin*, May 2003.

[55] K. A. Iskra, F. van der Linden, Z. W. Hendrikse, B. J. Overeinder, G. D. van Albada, and P. M. A. Sloot. The implementation of dynamite: an environment for migrating PVM tasks. *SIGOPS Oper. Syst. Rev.*, 34(3):40–55, 2000.

[56] K.A. Iskra. *Time Warp - from Cluster to Grid.* PhD thesis, University of Amsterdam (Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Dr. G.D. van Albada), June 2005.

[57] P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombas. P-GRADE: A Grid Programming Environment. *Journal of Grid Computing, Volume 1(2), pp. 171-197*, 2003.

[58] M. Kafeel and I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems, *IEEE Concurrency, Vol. 6(3) pp. 42-51*, 1998.

[59] L.V. Kale, S. Kumar, J. DeSouza, M. Potnuru, and S. Bandhakavi. Faucets: Efficient Resource Allocation in the Computational Grid. In *Proceedings of the 2004 International Conference on Parallel Processing*, 2004.

[60] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing, Vol. 63(5), pp. 551-563*, 2003.

[61] R. Keller, E. Gabriel, B. Krammer, M. Mueller, and M. Resch. Towards Efficient Execution of MPI Applications on the Grid: Porting and Optimization Issues, *Journal of Grid Computing vol. 1, no. 2, pp. 133-149*, 2003.

[62] A. Kertsz and P. Kacsuk. A taxonomy of grid resource brokers, In *Proc. 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems, pp. 201-210*, 2007.

[63] G. Kickinger, J. Hofer, A. M. Tjoa, and P. Brezany. Workflow Management in GridMiner. In *Proceedings of the 3rd Cracow Grid Workshop, Cracow, Poland*, October 2003.

[64] J. Kommineni, D. Abramson, and J. Tan. Communication over a Secured Heterogeneous Grid with the GriddLeS runtime environment. In *2nd IEEE International Conference on e-Science and Grid Computing. Amsterdam, Netherlands*, 2006.

[65] V. Korkhov, A. Belloum, and L.O. Hertzberger. Evaluating Meta-scheduling Algorithms in VLAM-G Environment. *Tenth Annual Conference of the Advanced School for Computing and Imaging (ASCI)*, June 2004.

[66] V.V. Korkhov and V.V. Krzhizhanovskaya. Benchmarking and Adaptive Load Balancing of the Virtual Reactor Application on the Russian-Dutch Grid. In *Lecture Notes in Computer Science, V. 3991, pp. 530-538*, 2006.

[67] V.V. Korkhov and V.V. Krzhizhanovskaya. Workload Balancing in Heterogeneous Grid Environment: A Virtual Reactor Case Study. In *Proc. of the Second International Conference on Distributed Computing and Grid Technologies in Science and Education. Publ: JINR, Dubna, ISBN 5-9530-0138-X, pp. 103-113*, 2006.

[68] V.V. Korkhov, V.V. Krzhizhanovskaya, and P.M.A. Sloot. A Grid Based Virtual Reactor: Parallel performance and adaptive load balancing. *Journal of Parallel and Distributed Computing, Vol 68/5, pp 596-608, DOI: 10.1016/j.jpdc.2007.08.010, Elsevier*, 2008.

[69] S. Krishnan, K. K. Baldridge, J. P. Greenberg, B. Stearn, and K. Bhatia. An End-to-end Web Services-based Infrastructure for Biomedical Application. *Proceedings of Grid 2005, 6th IEEE/ACM International Workshop on Grid Computing*, Nov 2005.

[70] V. Krzhizhanovskaya. *A virtual reactor for simulation of Plasma Enhanched Chemical Vapor Deposition.* PhD thesis, Universiteit van Amsterdam, Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Prof. Dr. Yu. E. Gorbachev, June 2008. ISBN: 978-90-9023166-2.

[71] V. V. Krzhizhanovskaya, M. A. Zatevakhin, A. A. Ignatiev, Yuri E. Gorbachev, and Peter M. A. Sloot. Distributed Simulation of Silicon-Based Film Growth. In *PPAM '01: Proceedings of the th International Conference on Parallel Processing and Applied Mathematics-Revised Papers*, pages 879–887, London, UK, 2002. Springer-Verlag.

[72] V.V. Krzhizhanovskaya and V.V. Korkhov. Problem-Solving Environments for Simulation and Optimization on Heterogeneous Distributed Computational Resources of the Grid. In *Proceedings of the Third International Conference on Parallel Computations and Control Problems PACO'2006, Moscow, Russia. Publ: Moscow, V.A. Trapeznikov Institute of Control Sciences RAS, pp. 917-932*, 2006.

[73] V.V. Krzhizhanovskaya and V.V. Korkhov. Dynamic Load Balancing of Black-Box Applications with a Resource Selection Mechanism on Heterogeneous Resources of the Grid. In *Proceedings of International Conference on Parallel Computing Technologies (PaCT-2007), in LNCS, V. 4671, pp. 245-260, Springer Berlin / Heidelberg*, 2007.

[74] V.V. Krzhizhanovskaya, V.V. Korkhov, A. Tirado-Ramos, D.J. Groen, I.V. Shoshmina, I.A. Valuev, I.V. Morozov, N.V. Malyshkin, Y.E. Gorbachev, and P.M.A. Sloot. Computational Engineering on the Grid: Crafting a Distributed Virtual Reactor. In *Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), Amsterdam, the Netherlands, December 4-6 2006, pp.101. IEEE CS Press.*, 2006.

[75] V.V. Krzhizhanovskaya, P.M.A. Sloot, and Yu. E. Gorbachev. Grid-based Simulation of Industrial Thin-Film Production. *Simulation: Transactions of the Society for Modeling and Simulation International, V. 81, No. 1, pp. 77-85*, 2005.

[76] V.V. Krzhizhanovskaya, M.A. Zatevakhin, A.A. Ignatiev, Y.E. Gorbachev, W.J. Goedheer, and P.M.A. Sloot. A 3D Virtual Reactor for Simulation of Silicon-Based Film Production. In *Proceedings of the ASME/JSME PVP Conference. ASME PVP-Vol. 491-2, pp. 59-68, PVP2004-3120*, 2004.

[77] R. Kufrin. PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. In *6th International Conference on Linux Clusters. Chapel Hill, NC.*, 2005.

[78] V. Kumar and A. Gupta. Analyzing Scalability of Parallel Algoritms and Architectures, *Journal of Parallel and Distributed Computing, Vol. 22, pp. 379-391*, 1994.

[79] Z. Lan, V.E. Taylor, and G. Bryan. A novel dynamic load balancing scheme for parallel systems, *Journal of Parallel Distribiuted Computing. 62 (12) pp.1763-1781*, 2002.

[80] A. Laszloffy, J. Long, and A.K. Patra. Simple data management and scheduling and solution strategies for managing the irregularities in parallel adaptive hp finite element simulations, *Parallel Computing, 26 (13-14), pp. 1765-1788*, 2000.

[81] C. Lee and D. Talia. Grid Programming Models: Current Tools, Issues and Directions, In *Berman, F., Fox, G.C. and Hey, A.J.G. (Eds.), Grid Computing, Making the Global Infrastructure a Reality, (Wiley), chapter 21, specifically section 21.2.3*, 2003.

[82] H.C. Lee, J. Salzemann, N. Jacq, H.Y. Chen, L.Y. Ho, I. Merelli, L. Milanesi, V. Breton, S.C. Lin, and Y.T. Wu. Grid-enabled High-throughput in silico Screening against influenza A Neuraminidase, *IEEE Transaction on Nanobioscience, V. 5, no.4 pp. 288-295*, 2006.

[83] M. Litzkow, M. Livny, and M.W. Mutka. Condor: a hunter of idle workstations. In *8th IEEE conference on distributed computing systems, IEEE, New York, 1998, pp. 104111.*, 1998.

[84] A. Manara. Integration of new communities in the Grid for mission critical applications: distributed radio-frequency compatibility analysis for the ITU RRC06 conference. *EGEE'06 Conference, 25-29 September 2006, Geneva, Switzerland.*

[85] R. McClatchey and G. Vossen. Workshop on workflow management in scientific and engineering applications report. *SIGMOD Rec.*, 26(4):49–53, 1997.

[86] J.T. Moscicki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research A. V. 502 pp. 426-429*, 2003.

[87] J.T. Moscicki. Ganga - a computational task management tool for easy access to Grid. *To appear in Computer Physics Communications*, 2009.

[88] J.T. Moscicki, M. Bubak, H.-C. Lee, A. Muraru, and P. Sloot. Quality of Service on the Grid with User Level Scheduling. *Cracow Grid Workshop Proceedings*, 2006.

[89] J. Nabrzyski, J.M. Schopf, and J. Weglarz (Eds). *Grid Resource Management: State of the Art and Future Trends.* Kluwer Academic Publishers, 2004.

[90] T. M. Nguyen, A. M. Tjoa, G. Kickinger, and P. Brezany. Towards Service Collaboration Model in Grid-based Zero Latency Data Stream Warehouse (GZLDSWH). *Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 04).*

[91] M.G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers, *ACM Computing Surveys Vol. 25(3) pp. 263-302*, 1993.

[92] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[93] S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003. Lecture Notes in Computer Science, Vol. 2672, pp 41-61*, 2003.

[94] C. Pautasso and G. Alonso. JOpera: a Toolkit for Efficient Visual Composition of Web Services. *International Journal of Electronic Commerce (IJEC), Vol. 9, No. 2*, Winter 2004/2005.

[95] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC), Edinburgh, Scotland*, 2002.

[96] P. Saiz, L. Aphecetche, P. Buncic, R. Piskac, J.E. Revsbech, and V. Sego. AliEn - ALICE environment on the GRID. In *Nuclear Instruments and Methods in Physics Research Section A: Vol. 502(2-3), pp. 437-440*, 2003.

[97] L. Sanglu and X. Li. A scalable load balancing system for NOWs. *ACM SIGOPS Operating Systems Review, 32 (3) pp. 55-63*, 1998.

[98] K. Schloegel, G. Karypis, and V. Kumar. A new algorithm for multi-objective graph partitioning. In *European Conference on Parallel Processing (EuroPar), Toulouse, France, Lecture Notes in Computer Science, vol. 1685, pp. 322-331*, 1999.

[99] J. Schopf. Ten Actions When Grid Scheduling, In *J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds), Grid Resource Management: State of the Art and Future Trends, Kluwer*, 2004.

[100] J. Schopf and L. Yang. Using Predicted Variance for Conservative Scheduling on Shared Resources, In *J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds), Grid Resource Management: State of the Art and Future Trends, Kluwer*, 2004.

[101] F.J. Seinstra and J.M. Geusebroek. Color-Based Object Recognition on a Grid. *Proceedings of the 9th European Conference on Computer Vision (ECCV 2006) Workshop on Computation Intensive Methods for Computer Vision (CIMCV 2006), Graz, Austria, May 7-13*, 2006.

[102] G. Shao, F. Berman, and R. Wolski. Master/Slave Computing on the Grid. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 3, Washington, DC, USA, 2000. IEEE Computer Society.

[103] I. Shoshmina, A. Evlampiev, D. Malashonok, and A. Bogdanov. Experience of Exploiting the RiDGrid Segment. In *Proc. of the Second International Conference on Distributed Computing and Grid Technologies in Science and Education. Publ: JINR, Dubna, D11-2006-167, ISBN 5-9530-0138-X*, 2006.

[104] S. Sinha and M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. *Proc. 3rd IEEE Intl. Conference on Cluster Computing, pp.435-442*, 2001.

[105] S. Succi. The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. New York: Oxford, 2001.

[106] X.-H. Sun and M. Wu. Grid Harvest Service A System for Long-Term, Application-Level Task Scheduling. In *Proc. of IEEE International Parallel and Distributed Processing Symposium*, 2003.

[107] X.N. Sun and L.M. Ni. Scalable Problems and Memory-Bounded Speedup, *Journal of Parallel and Distributed Computing, Vol. 19, pp. 27-37*, 1993.

[108] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed computing with Triana on the Grid. *Concurrency and Computation: Practice and Experience, vol. 17(9), pp. 1197-1214*, 2005.

[109] J. Teresco, J. Faik, and J.E. Flaherty. Resource-Aware Scientific Computation on a Heterogeneous Cluster. *Computing in Science and Engg.*, 7(2):40–50, 2005.

[110] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees. DIRAC: A Scalable lightweight Architecture for High Throughput Computing. In *In Procs 5th IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*, 2004.

[111] B. Veeravalli, D. Ghose, and T.G. Robertazzi. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems, *Cluster Computing, Volume 6, Issue 1, pp.7-17*, 2003.

[112] D.W. Walker, M. Li, O. Rana, M.S. Shields, and Y. Huang. The software architecture of a distributed problem-solving environment. *Concurrency - Practice and Experience, Vol. 12(15), pp. 1455-1480*, 2000.

[113] J. Watts and S. Taylor. A Practical Approach to Dynamic Load Balancing. *IEEE Transactions on Parallel and Distributed Systems, v.9(3), pp. 235-248*, 1998.

[114] D. Weissenbach and E. Clevede. Faster earthquake source mechanism determination with EGEE. In *1st EGEE Conference, Geneva*, 2006.

[115] J. Weissman. Metascheduling: A Scheduling Model for Metacomputing Systems. In *Proceedings of HPDC 1998*, pages 348–349, 1998.

[116] B. Wilson, B. Tang, G. Manipon, D. Mazzoni, E. Fetzer, A. Eldering, A. Braverman, E. R. Dobinson, and T. Yunck. GENESIS SciFlo: scientific knowledge creation on the grid using a semantically-enabled dataflow execution environment. In *Proceedings of the 17th international conference on Scientific and statistical database management, 2005, Santa Barbara, CA, pp.83-86*, 2005.

[117] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768*, October 1999.

[118] J. Yu and R. Buyya. A taxonomy of Scientific Workflow Systems for Grid Computing, *SIGMOD Record, vol 34, no 3*, 2005.

[119] M. J. Zaki, Wei Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. In *HPDC '96: Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, page 282, Washington, DC, USA, 1996. IEEE Computer Society.

[120] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems, *Software Practice and Experience, v.23(12), pp.1305-1336*, 1993.

[121] Atlas Computing - Technical Design Report CERN-LHCC-2005-022.

[122] Biomedical Informatics Research Network. http://www.nbirn.net/.

[123] CERN Openlab project, http://proj-openlab-datagrid-public.web.cern.ch.

[124] Collaboratory for Multi-scale Chemical Science. http://cmcs.ca.sandia.gov/.

[125] Commodity Grid Kits, http://www.globus.org/cog/.

[126] Community OpenORB Project, http://openorb.sourceforge.net/.

[127] CrossGrid EU Science project: http://www.eu-CrossGrid.org/.

[128] DAGMan - Directed Acyclic Graph Manager, http://www.cs.wisc.edu/condor/dagman/.

[129] DIstributed ANalysis Environment, http://cern.ch/diane.

[130] Distributed ASCI Supercomputer 2 (DAS-2), http://www.cs.vu.nl/das2/.

[131] Distributed.Net, http://distributed.net/.

[132] FusionGrid Collaboratory Project. http://www.fusiongrid.org/.

[133] Geant4 VO. http://lcg-voms.cern.ch:8443/vo/geant4.

[134] Globus Alliance: e-Science and e-Business projects. http://www.globus.org/.

[135] Interactive European Grid Project http://www.interactive-grid.eu/.

[136] Nimrod-G. http://www.csse.monash.edu.au/ davida/nimrod.

[137] omniORB, Free High Performance ORB. http://omniorb.sourceforge.net.

[138] SETI@Home, http://setiathome.berkeley.edu/.

[139] Virtual Observatory project. http://www.us-vo.org/.

[140] Virtual Laboratory for e-Science, http://www.vl-e.nl/.

[141] Wikipedia, http://en.wikipedia.org/wiki/Load_balancing_(computing).

Samenvatting

Het in dit proefschrift beschreven onderzoek behandelt het probleem van resource management in complexe en meervoudig gelaagde gedistribueerde applicaties. Het voornaamste doel is de bestudering van parallellisme op alle niveaus in dergelijke applicaties, en het definiëren van nieuwe methoden voor resource management en load balancing (het evenredig verdelen van de werklast) die niet afhankelijk zijn van specifieke gedistribueerde rekentechnologieën, en die toegepast kunnen worden op alle niveaus in de applicatiehiërarchie.

Na een in inleidende verhandeling over technologie-invariante methoden voor resource management in complexe parallelle applicaties die draaien in heterogene gedistribueerde systemen, introduceren wij onze methode van adaptieve workload balancing (AWLB), die toegepast kan worden in een grote verscheidenheid aan computeromgevingen. De AWLB methode is afhankelijk van de dynamische karakteristieken van zowel de applicatie zelf als van de omgeving waarin deze applicatie draait. Traditioneel bestaan er twee verschillende load balancing strategieën voor parallelle applicaties: (1) bepaal vooraf grondig de verdeling van de werklast, rekening houdend met de specifieke aspecten van zowel de applicatie als het onderliggende systeem, en (2) verspreid de werklast op een simpele, voor de hand liggende manier, hooguit rekening houdend met de relatieve rekenkracht van elke deelnemende computer. De eerstgenoemde methode is tijdrovend, en vereist expertkennis van zowel de complete applicatiestructuur, alsook van de specifieke details van de toegepaste algoritmiek. De tweede methode is veel sneller, maar niet efficiënt voor wat betreft de behaalde rekensnelheid. Wij stellen daarom een middenweg voor die is gebaseerd op een snelle eerste benadering, gevolgd door een iteratieve verfijning in de richting van het optimum.

Gegeven de complexiteit van applicaties, en de vele manieren om complexe applicaties op te delen in een gelaagde hiërarchie, beschouwen we een structuur voor parallellisme en distributie die bestaat uit drie niveaus: (1) het taakniveau, d.w.z. een enkele parallelle applicatie, (2) het jobniveau, d.w.z. een verzameling taken, bijvoorbeeld een parameter sweep, en (3) het workflowniveau, d.w.z. een functionele decompositie van de applicatie. Op elk van deze niveaus moet aan specifieke eisen worden voldaan om de rekensnelheid te maximaliseren, en om schaalbaarheid te garanderen. Bijgevolg moet het probleem van load balancing en resource management in beschouwing worden genomen tijdens het creëren van een applicatie als geheel.

Hoofdstuk 3 behandelt de onderste laag in deze complexe applicatiehiërarchie: een enkele parallelle applicatie draaiend op een heterogeen systeem. Eén van de grootste uitdagingen in het overbrengen van een parallelle applicatie van een homogene clusteromgeving naar een heterogene verzameling van computers is het verkrijgen van een vergelijkbaar niveau van parallelle efficiëntie. Voor de oplossing van dit probleem hebben we een theoretisch raamwerk ontwikkeld, tezamen met een generieke workload balancing techniek die rekening houdt met de specifieke parameters van zowel de applicatie als het onderliggende systeem. We introduceren hier de AWLB methode, en passen het toe op één van de sturende applicaties van dit onderzoek: een parallelle implementatie van de Virtuele Reactor.

Het geval van een enkele parallelle applicatie die wordt geëxecuteerd op een verzameling van parallelle computers wordt beschreven in Hoofdstuk 4. De beschikbaarheid van een groot aantal rekensystemen roept de vraag op of, en wanneer, het gebruik van meerdere parallelle computers voor een enkele parallelle applicatie een versnelling kan betekenen. Een theoretische methode tot het schatten van de mogelijke versnelling van een parallelle applicatie in een homogene Grid omgeving is gegeven door Hoekstra en Sloot in [49]. Hier valideren we

onze aanpak op basis van een bestaande applicatie: een Lattice Boltzmann Equation (LBM) solver. We gebruiken een eenvoudig model van deze applicatie voor de schatting van de snelheidswinst die mogelijk te behalen is op een multicluster systeem. In vergelijking met onze oplossing voor parallelle applicaties draaiend op een heterogeen systeem (in Hoofdstuk 3) is de aanpak die in dit hoofdstuk wordt gepresenteerd toepasbaar op homogene Grids. De aanpak is vrij accuraat in het voorspellen van de snelheidswinst van applicaties, op basis van een simpel applicatiemodel en een eenvoudige verzameling van systeemkarakteristieken.

Het volgende niveau in de applicatiehiërarchie wordt behandeld in Hoofdstuk 5. In applicaties die uit een aantal gedistribueerde taken bestaan (ofwel: multi-job applicaties) is het rekenwerk georganiseerd als een verzameling aparte componenten die alleen informatie uitwisselen aan het begin en aan het eind van de executie. In dit hoofdstuk wordt de AWLB methode toegepast op multi-job applicaties waarvan de werklast verder op te splitsen is. We presenteren een hybride resource management omgeving, werkend zowel op applicatieniveau als op systeemniveau, die de executietijd van dergelijke applicaties in heterogene Grid systemen minimaliseert. Deze omgeving bestaat uit op applicatieniveau toegepaste AWLB die wordt geïntegreerd met zogenaamde User-Level Scheduling (ULS). Deze laatste component dicht het gat tussen de applicatie zelf en de Grid resource managers: het is een door de gebruiker aan te passen middleware systeem voor het schedulen van applicaties. Het onderliggende systeem wordt dynamisch gemeten en applicatiekarakteristieken worden bij voortduring geschat ter optimalisatie van het gebruik van de door ULS gecontroleerde dynamische verzameling van Grid computersystemen. Ter validatie hebben we experimenten uitgevoerd met een synthetische en configureerbare applicatie, gebruik makend van het EGEE Grid systeem, de AWLB methode, en de DIANE user-level scheduler. Op basis van onze resultaten bespreken we verschillende manieren voor het managen van de werklast van splitsbare multi-job applicaties op het Grid. Bovendien vergelijken we verschillende methoden voor werklastdistributie, en illustreren we het gebruik van een dynamische set van computers onder dynamische variatie in het applicatiegedrag door middel van adaptieve selectie van computerresources.

Hoofdstuk 6 behandelt de bovenste laag van onze applicatiehiërarchie: de Grid workflow. Onze voornaamste aandacht gaat uit naar workflows die worden gestuurd door een dataflow, en we evalueren verschillende resource management strategieën voor dit type dataflow. Voor het executeren van een gedistribueerde workflow op het Grid is een workflow management systeem (WMS) benodigd dat de controle heeft over de executie. In dit hoofdstuk presenteren we VLAM-G en het belangrijkste kerncomponent, het Run-Time Systeem (RTS), als een implementatie van een data-gestuurd WMS. Het RTS maakt gebruik van Grid resources, terwijl het de inherente complexiteit van het Grid voor gebruikers verbergt. We behandelen de architectuur, de verschillende onderdelen van het RTS, en de speciale mogelijkheden van VLAM-G. Speciale aandacht wordt besteed aan het concept van dataflow, en het direct laten vloeien van data tussen verschillende gedistribueerde workflow componenten. Als bovenste niveau in de applicatiehiërarchie kan een workflow de management van de verschillende onderliggende niveaus verzorgen, alsook de coördinatie van de executie van elk component in samenhang met alle andere. Deze mogelijkheden van het VLAM-G WMS wordt geïllustreerd aan de hand van een workflow voor de Virtuele Reactor applicatie, die relevante onderdelen bevat op alle niveaus van onze applicatiehiërarchie.

Het in dit proefschrift gepresenteerde werk behandelt de gedistribueerde executie van meervoudig gelaagde applicaties die bestaan uit meerdere complexe componenten. Voor alle niveaus in de applicatiehiërarchie, van laag tot hoog, beschrijven we de meest geschikte methoden voor resource management. We presenteren onze implementatie alsook een experimentele evaluatie van de voorgestelde methodieken.

Acknowledgements

I would like to express my gratitude to my promotor Peter Sloot who could always find some time in his tight schedule to share his thoughts on my research and transfuse his enormous energy that inspired my work. I am endlessly thankful to my co-promotor, Adam Belloum, who was nearby me all these years as a supervisor and just a friend with advice and help on any subject starting from distributed computing and up to any kind of problem I could be stuck upon during my living in Amsterdam. I am grateful to Bob Hertzberger whose influence made it possible for to me to come to Amsterdam, join his research group many years ago and since then feel his implicit protection. I am deeply thankful to Alexander Bogdanov who initiated the collaboration between St.Petersburg Institute for High Performance Computing and Data Bases and University of Amsterdam that was the first step on the way to this thesis.

I appreciate very much the involvement of the Promotion Committee: Pieter Adriaans, Alexander Bogdanov, Marian Bubak, Bob Hertzberger, and Alfons Hoekstra. Special thanks to Alfons Hoekstra for fruitful discussions we had about my work on Grid speedup and for his thoroughness in finding inaccuracies in my manuscript.

I thank Valeria Krzhizhanovskaya very much for all the joined work we have accomplished together. It was a great pleasure to work together with Valeria who was always full with ideas, maybe even crazy sometimes, which made them even more attractive. Together we worked on the adaptive workload balancing algorithm for parallel applications in heterogeneous environment (Chapter 3 in this thesis). Later this approach was transformed to be used for applications run in a user-level scheduling environment (Chapter 5). Here Kuba Moscicki, the author of DIANE user-level scheduler, deserves many thanks, as a lot of help was required from him on this stage. The collaboration we could establish still amazes me as I have never yet met Kuba in person, but our joined work resulted in two journal publications.

I want to especially mention and thank Dmitry Vasyunin, my colleague, flat-mate and a good friend. He made the greatest contribution to the work on VLAM-G workflow management system (Chapter 6) and his expertise in lots of programming languages, computer technologies, software and hardware was always at hand when I needed it. Apart from working hours my sweetest memories come back to the apartment on Bestevaerstraat that we used to share for a long time, and lasting discussions we used to have in the soviet-style kitchen there.

Of course, I would like to thank all the people who were around me in the University for their friendliness, helpfulness and just a nice time we spent together. First of all I want to mention my group and room-mates Adianto Wibisono, Victor Guevara, and Zhiming Zhao. Victor's spontaneity and funny jokes always brought joy to our office life. Zhiming's stories about Chinese traditions often amazed me, especially I remember the traditional Chinese recipe to cook tender duck paws. Wibi's persistence in learning Russian words and phrases rejoiced my heart, and to Wibi I wish to cheer up and accomplish his thesis finally. I thank all the other members of the group and VL-e team even though some of them have already left the University: Hakan Yakali for his concern, invariable good mood and KGB jokes, Ersin Kaletas and Jamel Abtroun for database support and a nice time at football games, Zeger Hendrikse and Philip Jonkergouw for their participation in the early stages of VLAM-G, Piter de Boer for his great Vbrowser and for interesting tales about Dutch and especially Frisian culture, Silvia Olabarriaga for her collaboration and energy in arranging social events, Marcia de Inda for her active usage and feedback on VLAM-G environment, Scott Marshall for being always helpful in solving complicated issues of English, Marco Roos

for his contribution in bio-informatics workflows, Frank Terpstra for controlling the discipline of keeping VLAM-G technical meetings, and Cees de Laat for showing interest in my research and support.

I am happy to be a part of CSP lab and I thank all the CSP people who influenced my work and sometimes leisure in Amsterdam: Alfredo Tirado-Ramos, Breanndan O Nuallain, Rob Belleman, Kamil Iskra, Derek Groen, Jaap Kaandorp, Syed Murtaza, Michael Scarpa and many more people.

Special thanks to the Russian community that was also formed on the grounds of the University and nearby territories: multi-scientist/painter/photographer Ilya Korjoukov, visualization expert Denis Shamonin, rigorous Roman Shulakov, Max Yurkin with his incredible mixture of seriousness and ingenuity in having fun, Victoria Yanulevskaya with her bursting emotions especially at the famous 'Argentina-Jamaica' game, the best cook in town Yevgen Bilevych, always charming Elena Zudilova-Seinstra, delicate Ekaterina Ermilova (good luck with your thesis!), serious and lovely Lilit Axner, focused and persevering Lev Naumov, Vlad Sergeev and Nelly Hoffman with their two cats Tesla and Tor and a cosy apartment I was lucky to stay at, and math and climbing maniac Jevgeny Ivanov.

My biggest thanks go to Jacqueline van der Velde and Erik Hitipeuw who handled all the administration and paper work. It would be much more difficult without your friendliness and help. I also thank Peter Boven, Amanda Collins and Thes Smeets for their attention and assistance in solving personnel related issues.

A large part of my work has been accomplished with the Center of Supercomputing Applications at St.Petersburg Institute for High Performance Computing and Databases, and I would like to thank the people I worked with: Dmitry Malashonok, Irina Shoshmina, Sergey Romanov, Grigory Davydov, Elena Stankova, Alexander Degtyarev. I thank my recent colleagues from St.Petersburg State University for their interest in my research, in particular Sergey Andrianov.

I am very grateful to Frank Seinstra who is always ready to help in any circumstances. Frank always was my only hope when I was stuck upon anything related to advanced knowledge of Dutch.

I thank Dutch families who let me in to their homes and showed real Dutch life for their warmth and hospitality: Niek van der Kade from Oudenbosch and Ralph Kämena from Den Haag.

My infinite gratitude is to my mother who always gave me her moral support in my studies and has been patiently waiting for this thesis to be accomplished.