Hierarchical resource management in grid computing

Korkhov, V.V.

Publication date
2009
Document Version
Final published version

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.

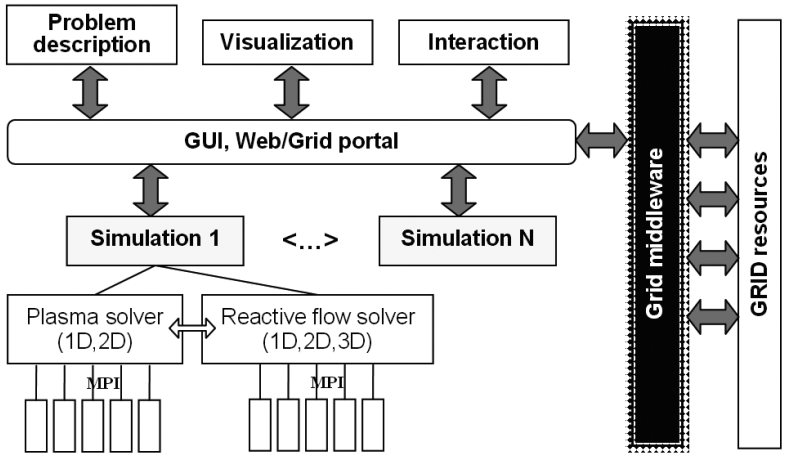

Figure 3.1: Functional scheme of the Virtual Reactor application. Interface components (problem description, visualization, interaction) are mediated by the Grid portal layer that provides access to the resources and controls the simulation execution.

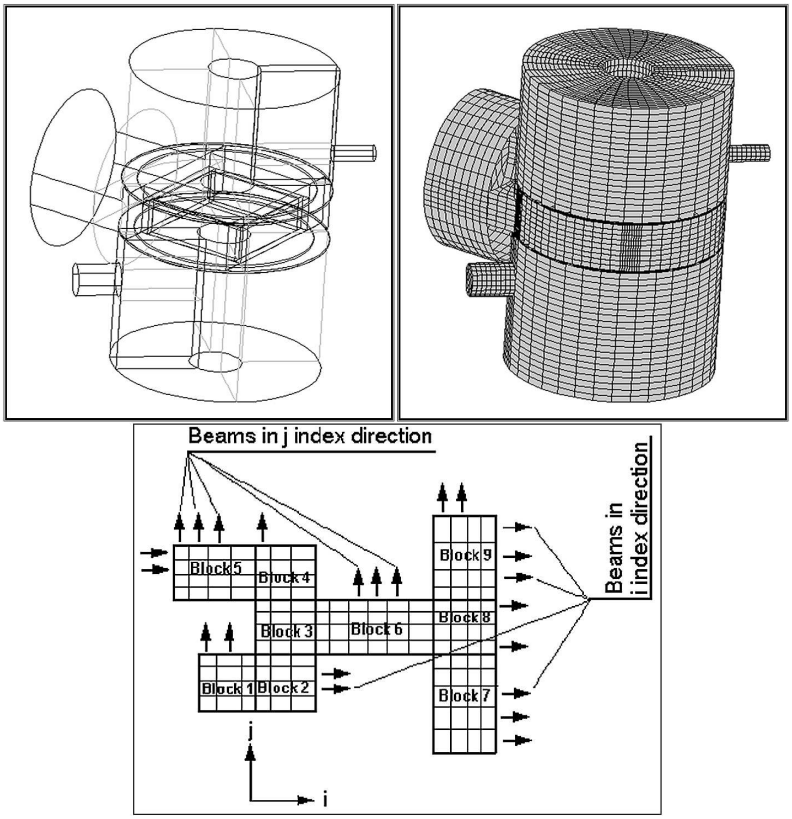

Figure 3.2: Virtual reactor geometry (meshed) and sample beams of computational cells

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.

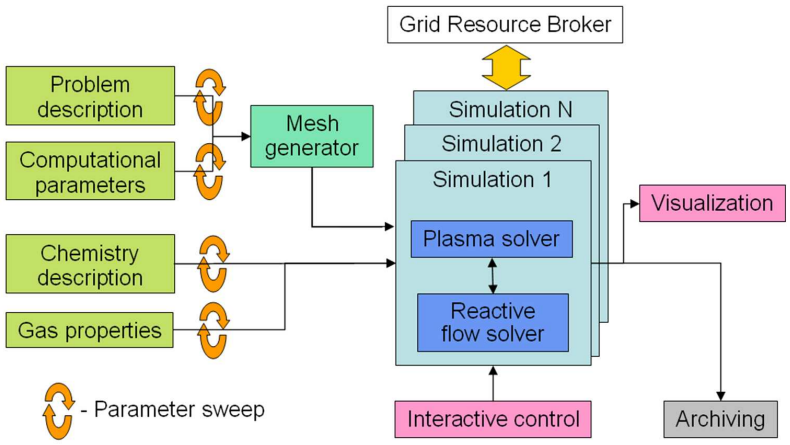

Figure 3.3: Scheme of the Virtual Reactor workflow: the arrows indicate some of the interdependencies of the components e.g, plasma and reactive flow solvers as input data use: description of the physical and chemical properties created by the editor components; parameters of the plasma chemical deposition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

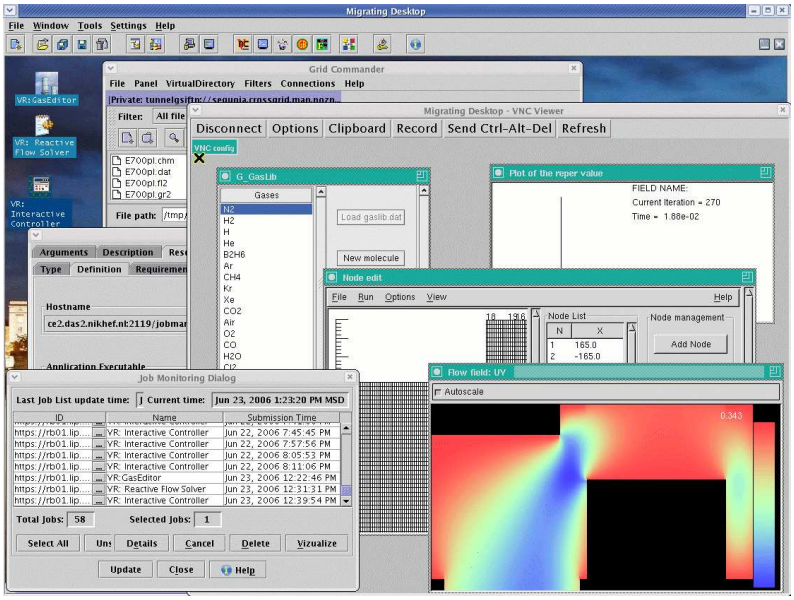

Figure 3.4: A running experiment of the Virtual Reactor on the Grid. The distributed components of the application, including the interactive editors and visualizers, are accessible via the virtual folder of the Migrating Desktop portal. Separate PSE components and simulations are run on distributed 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).

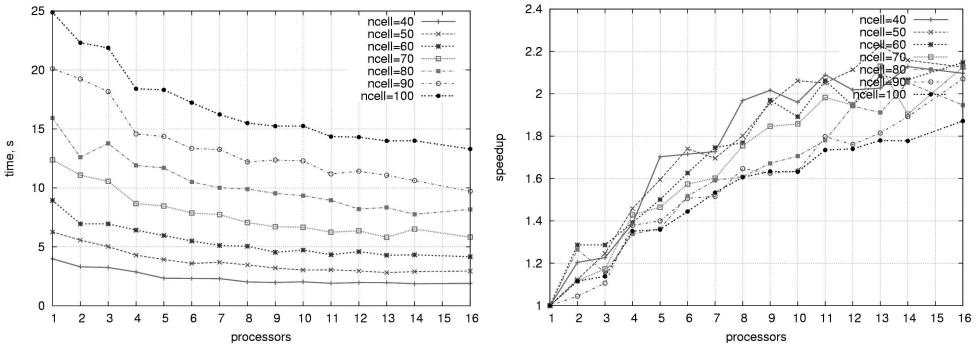

Figure 3.5: Light-weight (no chemistry) simulation: total execution time and speedup for different computational mesh sizes

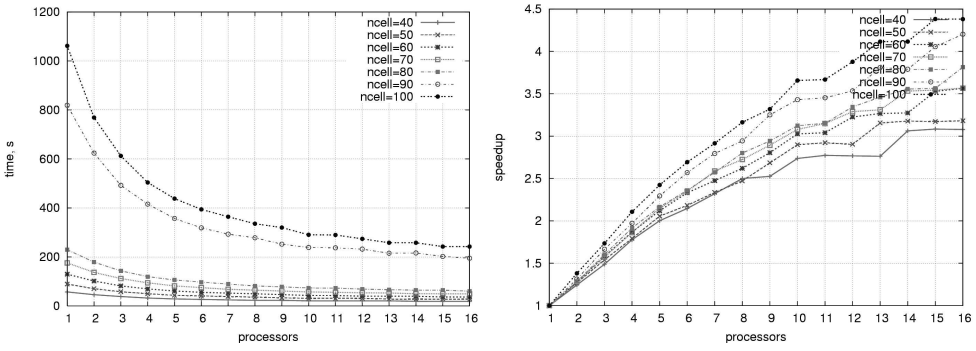

Figure 3.6: Chemistry-enabled simulation: total execution time and speedup for different computational mesh sizes

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).

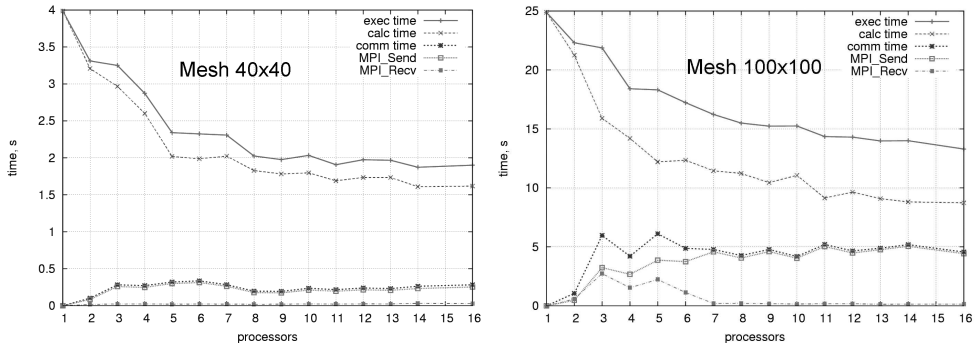

Figure 3.7: Total execution time and contributions of the calculation and communication depending on the number of processors for different computational mesh sizes (light-weighted simulation)

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.

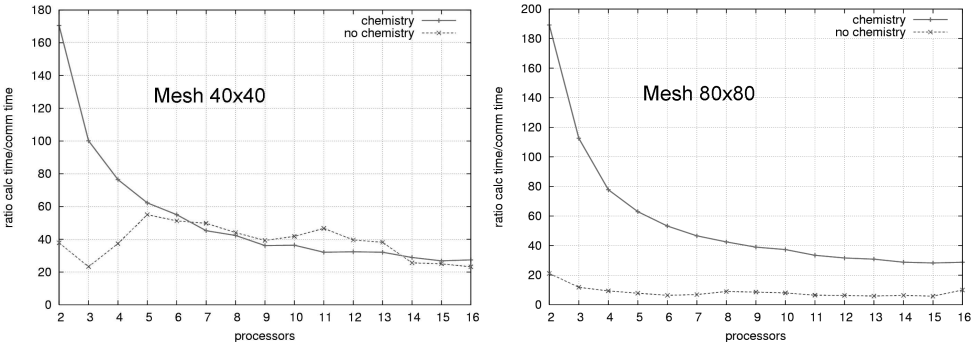

Figure 3.8: The ratio of the computation to communication time for chemistry-enabled and light-weighted simulations

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.

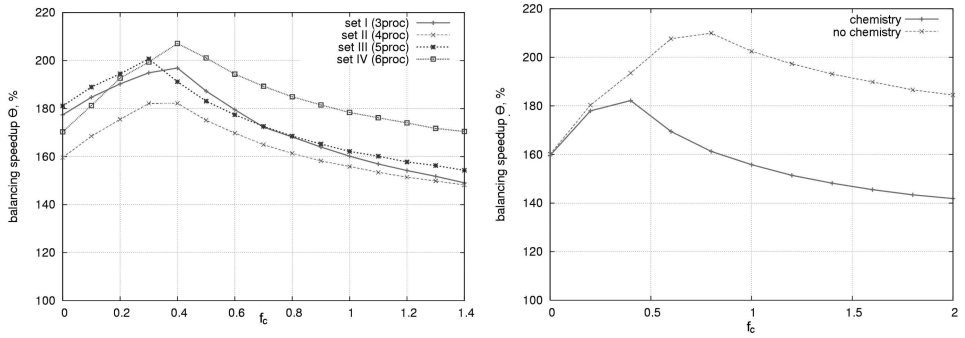

Figure 3.9: Dependency of the balancing speedup Θ on the parameter f_c. Left: single simulation on different sets of resources. Right: different types of simulation on the same set of resources.

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

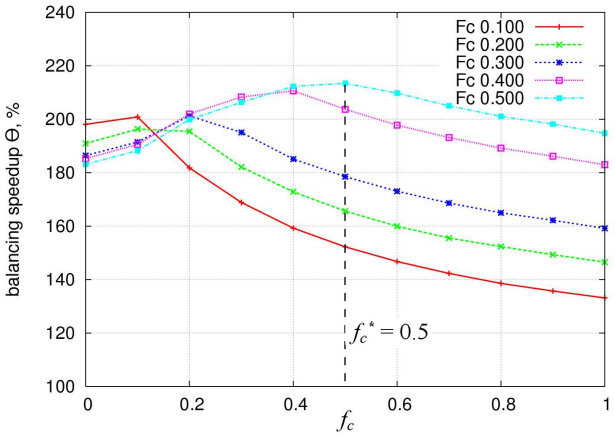

Figure 3.10: Dependency of the load balancing speedup Θ on the "guessed" application parameter f_c for 5 synthetic applications with different values of F_c.

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.

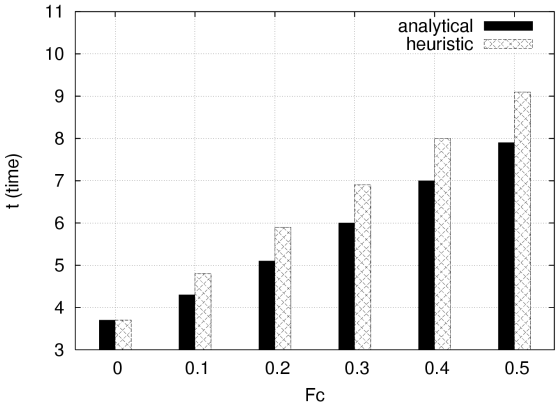

Figure 3.11: Comparison of the execution times for different weighting: the best theoretical distribution versus the generic heuristic load balancing.

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.