Dynamic Load Balancing of *Black-Box* Applications with a Resource Selection Mechanism on Heterogeneous Resources of the Grid

Valeria V. Krzhizhanovskaya[1,2] and Vladimir V. Korkhov[1,2]

[1] University of Amsterdam, Faculty of Science, Section Computational Science
[2] St. Petersburg State Polytechnic University, Russia
{valeria,vkorkhov}@science.uva.nl

Abstract. In this paper we address the critical issues of efficient resource management and high-performance parallel distributed computing on the Grid by introducing a new hierarchical approach that combines a user-level job scheduling with a dynamic load balancing technique that automatically adapts a *black-box* distributed or parallel application to the heterogeneous resources. The algorithm developed dynamically selects the resources best suited for a particular task or parallel process of the executed application, and optimizes the load balance based on the dynamically measured resource parameters and estimated requirements of the application. We describe the proposed algorithm for automated load balancing, paying attention to the influence of resource heterogeneity metrics, demonstrate the speedup achieved with this technique for different types of applications and resources, and propose a way to extend the approach to a wider class of applications.

Keywords: dynamic load balancing, resource management, high-performance computing, Grid, heterogeneous resources, parallel distributed application.

1 Introduction and Motivation

Grid-based problem-solving environments (PSEs) play an increasingly more important role in a broad range of applications stemming from fundamental and applied sciences, engineering, industry, medicine and economy. In [1,2] we provide an extensive overview of the Grid-aware problem-solving environments and virtual laboratories for complex applications. A great number of noticeable advances were achieved as a result of joint efforts of the multidisciplinary research society, such as the development of widely acknowledged standards in methodologies, formats and protocols used within the environments [11]. Another manifesting development concerns the move from specific one-application PSEs to the high-level generic environments that provide services, tools and resources to formulate and solve a problem using standardized methods, modules, workflows and resource managers [1]. Our research in this field has started from porting a Virtual Reactor problem-solving environment to the Grid [1-5], pioneering the move of fully integrated simulators from a single PC via computer clusters with a remote user interface [5] to fully distributed heterogeneous

Grid systems [2,3]. A detailed description of the Virtual Reactor application and our "gridification" activities can be found in [1-5].

We have implemented and tested several approaches, and adapted an existing interactive distributed application to the peculiarities of the Grid, thanks to the complementary projects developing Grid middleware, tools and portals [3,9,10]. However a few things shadow the overall optimistic picture of the major advances in Grid usability as observed in our extensive experiments with different Grid implementations. Among the most prominent and as yet unsolved problems we experienced are efficient resource management at the application and system levels, and optimization of the workload allocation for parallel and distributed applications on highly diverse and dynamically changing Grid resources. These two intertwined fundamental issues hindering the progress of Grid computing have pulled the forces of a vast computer society that strive to extrapolate an efficient high-performance computing on the Grid from a single demo test-case to a ubiquitous reality. A huge number of algorithms, approaches and tools have been developed to bring Grid resource management and job scheduling issues to a more advanced level of efficiency and, even more importantly, usability (see for instance [12-21]). In addition to that, an excessive number of load balancing techniques have been implemented and tested since the times when heterogeneous cluster computing emerged. We could not find a recent book providing a good overview of the state-of-the-art in load balancing, and a list of relevant papers would take at least several pages, so we will give references only to those intimately related to the technique we propose hereunder.

In a seemingly successful research field teeming with various solutions at hand, when things came to practice it turned out to be impossible to find a tool/library for automatic load balancing of a parallel distributed application on heterogeneous resources of the Grid. The first-priority consideration we had in mind was instrumenting our Virtual Reactor application with a library that would require minimal intrusion into the code and that would adapt the parallel codes previously developed for homogeneous computer clusters to the heterogeneous and dynamically changing Grid resources. Another goal was finding the means to enable "smart" resource selection and efficient utilization for the whole problem-solving environment, i.e. distributing the PSE disparate modules wisely, according to their individual requirements. The stumbling-block is that these *application requirements are not known beforehand* in most real-life complex applications, where only the key developers can embrace the complexity and dependencies of the PSE components. And even the code designers aware of the numerical methods' particularities can not predict the exact application requirements, which differ in each new computational experiment, depending on initial conditions, combination of real-life processes to be simulated, numerical schemes chosen, computational parameters, etc. This uncertainty prompted us to use the term *black-box* applications in the title of this article; we certainly do not mean that the user does not know what application he is running and of what avail. Our extensive benchmarking and performance assessment of the Virtual Reactor application clearly showed 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 [6-8].

A countless number of parallel and distributed applications have been developed for traditional (i.e. static homogeneous) parallel computers or cluster systems. Porting such applications from homogeneous computing environments to dynamic heterogeneous computing and networking resources poses a challenge to keep up a high level of application efficiency. To assure efficient utilization of Grid resources, special methods for workload distribution control should be applied. *An adequate workload optimization method should take into account* two aspects:

– (1) The **application characteristics**, such as the amount of data transferred between the processes, logical network topology, amount of floating point operations, memory requirements, hard disk or other I/O activity, etc.
– (2) The **resource characteristics**, like computational power and memory of the worker nodes, network links bandwidth, disk I/O speed, and the level of heterogeneity of the resources randomly assigned to the application by the Grid resource broker.

The method should be (a) self-adapting and flexible with respect to the type of application, (b) computationally inexpensive not to induce a large overhead on the application performance, and (c) should not require significant modifications in the code. On top of that, the load balancing shall be (d) dynamic and fully automated since we want to hide the "ugly" features of the Grid from innocent users.

2 Background: Automated Load Balancing on the Grid

The issue of load balancing in Grid environments is addressed by a number of research groups. Generally studies on load balancing consider distribution of processes to computational resources on the system/library level with no modifications in the application code [22,23]. Less often, load balancing code is included into the application source-code to improve performance in specific cases [24,25]. Some research projects concern load balancing techniques that use source code transformations to speedup the execution of the application [26]. We employ an application-centric approach where the balancing decisions are taken by the application itself. This is dictated by two arguments: first, the immaturity (or the lack of "intelligence") of the middleware or system-level resource managers; and second, the complexity of the problem-solving environments such as our Virtual Reactor, which has a number of communicating modules, some of which are parallel programs. An important feature of our approach is that although it is application-centric, the algorithm that estimates available resources and suggests the optimal load balancing of a parallel job is generic and can be employed in any parallel application to be executed on heterogeneous resources by instrumenting it with the load-balancing library.

A detailed description of global load optimization approaches for heterogeneous resources and adaptive mesh refinement applications can be found for instance in [29,30,31]. We shall note however, that in [29] and [31] no network links heterogeneity was considered and only static resource estimation (initialization) was performed in [29] and [30]. These two issues are the major challenges of Grid high-performance computing: 1) the heterogeneity of the network links can be two orders of magnitude higher that that of the processing power; and 2) Grid resources are inherently

dynamic. Developing our algorithm, we tried to address specifically these two issues. The approaches discussed in [29] and [31] are only valid for batch sequential applications (specifically for the queuing systems and computer cluster schedulers), whereas our effort is directed towards *parallel* programs utilizing heterogeneous resources.
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, multiparameter optimization, numerous heuristics, etc.), but all 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. In our view, this is an essential step to be made in order to make Grid computing efficient and user-friendly. Although some tools are already available for "smart" system-level process-resource matching and job scheduling on the Grid, none of them is automatic yet, and none is coupled with a mechanism evaluating the application requirements. We aim to bridge this gap by building a hierarchical approach that combines a user-level job scheduling [32,33] with a dynamic load balancing technique that automatically adapts a *black-box* distributed or parallel application to the heterogeneous Grid resources.

To summarize, the existing algorithms and tools provide only a partial solution. Our target is to combine the best achievements and to design a flexible tool for automated load balancing on the Grid. In this paper we present the results of the ongoing work in this direction. In Section 3 we introduce the basic ideas and steps of a generalized automated load balancing technique for a *black-box* application on the Grid. Section 4 presents the results of implementation of the load balancing algorithm, describes a synthetic test application developed for experiments, and shows the trends of the load balancing speedup and the influence of the resource heterogeneity level. Section 5 concludes the paper with discussion and future plans.

3 Generalized Automated Load Balancing with Resource Selection

Based on our previous experience [6-8], we developed a load balancing technique that takes into account the heterogeneity and the dynamics of the Grid resources, estimates the initially unknown application requirements, and provides the resource selection and most optimal mapping of the parallel processes to the available resources. In the most general case we consider that the resources have been randomly assigned to the application by a Grid resource broker via the User-Level Scheduler [32,33], or that the application can request the desirable resources with a set of parameters. An important feature of the proposed mechanism is that all the functionality described below is implemented as an external library, and the application is instrumented by linking to this library. As we mentioned in the introduction, this is a work in progress: The technique described below has not been fully implemented yet. A part of coupling the parallel load balancer with the user-level job scheduler is under development now. It will be published with additional details after deployment and testing.

3.1 The Basic Algorithm of the Automated Load Balancing

The load balancing meta-algorithm includes 8 basic Steps. Below we provide a descriptive explanation of each Step, mentioning special cases to be considered at each stage. We shall note that this is a conceptual description, rather than a mathematically strict algorithm. An exact formulation of the core load balancing heuristic is provided in the next subsection.

Step 1. Benchmarking resources: Measuring the computational power and memory available on the worker nodes, 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. These can be various parameters of databases, storages, sensors, scanners, and other attached devices.

Step 2. Ranking resources: The priority of ranking parameters shall be dependent on the type of application. For traditional parallel computing solvers, which we consider as test-case applications in present work, the first ranking parameter shall be computational power (CPU) of the processor, the second parameter being the network bandwidth to this processor. For memory-critical applications, memory shall be 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.

Step 3. Checking the level of heterogeneity: This parameter is often not considered in the load balancing heuristics; however it plays a crucial role in the choice of load balancing approach to be taken. The first and most obvious argument is that if the resources happen to be almost homogeneous, for traditional parallel applications no additional load rebalancing is required (and parallel tasks are distributed in equal chunks). In subsection 3.2, we discuss how the levels of heterogeneity affect the weighting factors used for calculating the workload per processor. We introduce the heterogeneity metrics and pay special attention to the way it influences the load balancing performance for our parallel computing test-case application.

Step 4. Testing application components and their interconnections: For that, run a small subset of the routine operations on the resources given. For a majority of traditional computational applications, the best is to perform one or a few time steps, iterations or events (depending on the type of simulation) in order to ensure that no time is wasted just for the testing, and the simulation is already running, though not in the most optimal way yet. This Step will measure the application performance on a given set of resources and collect the data needed to calculate the application requirements.

Step 5. Estimating the application requirements: The idea is to *quantitatively* estimate the requirements of the application based on the results of resource benchmarking (Step 1) and measurements of the application response (Step 4). For our parallel computing test-case application, the requirements to be calculated are the communication to computation ratio and the minimally required memory per node. An extensive description of the theoretical background and details of the corresponding heuristic can be found in [6-8]. In the next subsection we give an excerpt completing this meta-algorithm.

Step 6. 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 2 and 5. In our computational application example, memory can be the constraining factor: In case of sufficient memory on allocated processors, the load balancing can be performed further, taking into account all the other factors. In the unfavourable case of insufficient memory on some of the processors, they must be disregarded from the parallel computation or replaced by other, better suited processors. This shall be done on the level of job scheduling and resource allocation, within the framework of a combined approach coupling the application-centered load balancing with a system-level resource management. For this, we consider the User-Level Scheduler [32,33] as a feasible application-level intermediate resource managing approach.

Step 7. Matching resources II. Selecting resources: This is the second stage requiring a hierarchical approach we are developing. It provides the means to select the best-suited resources for each of the PSE components. This Step consists of 3 basic functionalities: finding an optimal number of processors for each application component, the actual resource matching, and rejecting some of the resources and requesting some others -depending on the approach taken and resource availability. The resource matching procedure (to be distinguished from process *mapping*) shall take into account the application requirements derived in Step 5 and can be implemented using some standard multi-parameter optimization method. In our parallel computing test-case, selecting resources might look fairly simple: we always want the fastest processors with the fastest links between them. But with the severe heterogeneity of Grid resources, this is not so trivial anymore. What is better, fast worker nodes connected by the slow links or slower processors with the fast links? The answer 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 application will not care about the network bandwidth. Another open question to be answered is how many worker nodes shall be assigned to a parallel solver. Again, the answer will be different depending on the solver characteristics: For a majority of "pleasingly" parallel applications (employing the resource farming concept), the more processors the better, so the actual number of processors to be allocated is an issue of availability and competition with the other PSE components. On the other hand, for a wide class of "normal" 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.

Step 8. Load balancing: After selecting the best suited set of resources, we need to perform the actual optimization of the workload distribution within the parallel modules, in order words *mapping* the processes onto the allocated resources. This Step is based on the heuristic developed earlier [6-8], which includes a technique to calculate the weighting factors for each processor depending on the resource characteristics and application requirements established in Step 5. In Section 3.2 we summarize the methodology, introduce some corrections in the theoretical formulation and discuss the role of the heterogeneity function.

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 and load re-distribution shall be performed. This leads to repeating all the meta-algorithm Steps except of Step 4 and Step 5. In most cases this can be done by running the application with a few consecutive time steps or iterations (see comments to Step 4). NB: if the selected resources did not change much, Steps 6 and 7 can be omitted not to incur unnecessary overhead.

If the application is dynamically changing (for instance due to adaptive meshes, moving interfaces or different combinations of physical processes modeled at different simulation stages) then the application requirements must be periodically re-estimated even on a static set of resources. In this case, the periodic re-estimation loop stars from Step 4, with the same remark on skipping Steps 6 and 7 if the application change is not dramatic.

Periodic re-estimations shall be performed frequently during the runtime of the application to correct the load imbalance with a reasonably short delay. The minimally required frequency of rebalancing can be estimated and dynamically tuned by calculating the relative imbalance introduced during the controlled period of time.

In the next subsection we provide a strict formulation of the most important aspects essential for understanding the experimental results shown in Section 4. A scrupulous mathematical description of all the conditions, metrics and algorithms in a complete meta-algorithm we save for another paper.

3.2 Adaptive Load Balancing on Heterogeneous Resources: Theoretical Approach

In [6,7] we proposed a methodology for adaptive load balancing of parallel applications on heterogeneous resources, extending it to *dynamic* load balancing and introducing the heterogeneity metrics in [8]. In this section we give a theoretical description of the basic concepts and parameters mentioned in the meta-algorithm, and concentrate on the two most important issues: (1) estimating the application requirements (Step 4 and Step 5) and (2) the actual load balancing of parallel or distributed *black-box* applications on heterogeneous Grid resources (Step 8). The load balancing Step aims at optimizing the load distribution among the resources already selected in previous Steps (after performing the check against the restricting factors such as the memory deficiency). Therefore the theory is given under the assumption that the resources are "fixed" for a single load-balancing loop, and that using *all* these resources provides a reasonably good performance result (e.g. parallel speedup for traditional parallel computing applications). Another prerequisite is that the application is already implemented as a parallel (or distributed) program, and is able to distribute the workload by chunks of controllable size. Saying this we kept in mind the Master-Worker model, but the technique is applicable to other communication logical topologies, given that the measurements are carried out along the links used within the application. The load balancing procedure we describe is implemented as an external library, and after linking with the application provides a recommendation on how much work shall be done by each of the assigned processors to ensure the fastest possible execution time –taking into account the specific parameters of the resources and the estimated application requirements [6-8]. We designed the algorithm in such a way that the knowledge of these resource and application characteristics would give an instant solution to the workload distribution, thus making the procedure very lightweight and suitable for dynamic load balancing at runtime.

The main generic parameters that define a parallel application performance are:

- An application parameter $f_c = N_{comm}/N_{calc}$, where N_{comm} is the total amount of application communications, i.e. data to be exchanged (measured in bit) and N_{calc} is the total amount of computations to be performed (measured in Flop);
- The resource parameters $\mu_i = p_i/n_i$, where p_i is the available performance of the i^{th} processor (measured in Flop/s) and n_i is the network bandwidth to this node (measured in bit/s).

The resource characteristics p_i and n_i we obtain in Step 1 after benchmarking the resources, but the application parameters N_{comm} and N_{calc} are not known beforehand in real-life applications. The target is to experimentally determine the value of the application parameter f_c that provides the best workload distribution, i.e. minimal runtime of the application mapped to the resources characterized by a parameter set $\mu = \{\mu_i\}$.

A natural way to do that is to run through the range of possible values of f_c with a discrete step, calculating a corresponding load distribution and performing one time step/iteration with a new load distribution. Measuring the execution time of this iteration and comparing it for different values of f_c, we find an optimal value f_c^*, which provides the minimal execution time. This idea is implemented in Step 5 and will be illustrated in the Results section (4.2). A detailed algorithm is described in [8]. There we suggested estimating the range of possible values of the application parameter f_c as following: 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 possible value was calculated as $f_c^{\max} = \max(n_i)/\min(p_i)$. Experimenting with this rough upper bound evaluation, we found that in many cases it gives a too high value of f_c^{max} , unnecessarily extending the search range and thus reducing the efficiency of the load balancing procedure. Another approach to search for the optimal value f_c^* can be borrowed from the optimization theory, for instance using an adaptive 1-dimensional non-linear constrained optimization method with a correction for small stochastic perturbations in resource performance [34]. This approach can reduce the number of the load balancing loops needed to find the best load distribution.

To calculate the amount of the work per processor in the load balancing Step 8, we assign a weight-factor to each processor according to its processing power and network connection. A similar approach was applied in [25] and in [27] for heterogeneous computer clusters, but the mechanism for adaptive calculation of the weights -taking into account the 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.

The weighting factor w_i determines the final workload to be assigned to each of N processors: $W_i = w_i W$, where W is the total workload. The weighting factor w_i shall reflect both the capacity of resources according to the estimated infrastructure parameters μ_i and the application parameter f_c. In [8] we derived an expression for processor weights analogous to that used by other authors [25,27]. Extensive experimentation and analysis of this expression revealed that the optimal balance for computation-intensive applications running on fast network links is not computed correctly. To correct this, we modified the equation for weights calculation, deriving it from the first principles of equalizing the time spent by each processor. In the simplified model of communication that can suite as the first approximation of real communication topologies, the weights can be calculated as follows:

$$w_i = q_i \Big/ \sum_{i=1}^{N} q_i; \quad q_i = p_i \big/ (1 + \varphi f_c \mu_i) \tag{1}$$

Here q_i is the dimensional weight calculated from the resource parameters p_i and μ_i, and from the guessed application parameter f_c. φ is the heterogeneity metrics of the network links that can be expressed as a standard deviation of the set of normalized dimensionless resource parameters:

$$\varphi = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(1 - n_i / n_{avg})^2}, \quad n_{avg} = \frac{1}{N}\sum_{i=1}^{N} n_i \tag{2}$$

The purpose of this heterogeneity metrics is to ensure that if the network links are homogeneous, i.e. $n_i = n_{avg}$, then the weighting is done only according to the processors capacity. In this case $\varphi = 0$, and the last term in the denominator of Eq.(1) is nullified, thus providing that the weights w_i are linearly proportional to the processing power p_i. Then we can see that in the infrastructure of heterogeneous processors connected by homogeneous network links the value of application parameter f_c does not affect the load distribution, which is exactly the case in the Master-Worker lock-step synchronous communication model. Generally speaking, in other communication models this can be different, so a bit more sophistication is needed in order to design a generic algorithm that would suit well the majority of logical topology models.

To evaluate the efficiency of the workload distribution we introduce the load balancing speedup $\Theta = T_{non-balanced} / T_{balanced} \cdot 100\%$, where $T_{non\text{-}balanced}$ is the execution time of the parallel application without the load balancing (even distribution of the prosesses), and $T_{balanced}$ is the execution time after load balancing on the same set of resources. This metric is used to estimate the application parameter f_c^* that provides the best performance on given resources, that is the largest value of speedup 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, which means that the application requirements fit best the resources in this particular workload distribution. The case of $f_c^* = 0$ while $\varphi \neq 0$ 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.

While deriving Eq. (1), 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 memory can be one of the determining factors of the application performance and play a role in the load balancing process. 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, but the idea and the load balancing mechanism remain the same. Similar considerations shall be applied for the other types of applications. For instance, in a widely used class of applications performing sequential computing with hard disk intensive operations, the network link bandwidth parameter n_i shall be replaced with the disk I/O speed for finding an optimal load distribution in "farming" computations on the Grid.

4 Performance Results

In this section we provide some details on implementing the load balancing algorithm and show the results illustrating the load balancing technique for our computational application case-study and demonstrating the speedup achieved with this technique for different types of applications and resources. The adaptive load balancing technique we propose was first applied while deploying the Virtual Reactor parallel components on heterogeneous Grid resources [3]. Several simulation types have been extensively tested on various sets of resources, demonstrating how the algorithm works. However one application can obviously provide only a limited freedom for experiments. To be able to examine the behavior of an *arbitrary* parallel application (characterized by various values of the application parameter f_c and various interprocess communication topologies) on *arbitrary* sets of heterogeneous resources, we developed a synthetic parallel application that allowed us to model different combinations and to compare the best theoretically achievable performance results with those given by our workload-balancing approach.

4.1 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 f_c 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 their 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 as any other application would be. We use this experimental setup to examine how a specific parallel application defined by a combination of communication/computation ratio f_c 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 [7,8]. The experiments were carried out on the DAS-2 computer cluster [35], using MPICH-P4 implementation of MPI.

4.2 Load Balancing Speedup for Different Applications

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 3.2). Figure 1 presents the results of load balancing of our synthetic application with the Master-Worker non-lockstep asynchronous communication logical topology (when 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

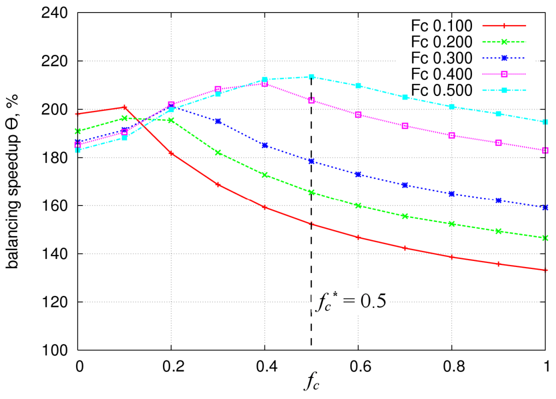

Fig. 1. Dependency of the load balancing speedup Θ on the "guessed" application parameter f_c for 5 synthetic applications with different values of F_c

dependency on application and resource parameters adds another 35 % percent 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 %.

4.3 Load Balancing for Master-Worker Model: Heuristic Versus Analytically Derived 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. (1). Omitting the mathematical details, we present the final recurrence relation for calculating the weights:

$$q_N = \left(1 + \sum_{i=2}^{N} \prod_{k=i}^{N} \frac{\tau_k + T_k}{T_{k-1}}\right)^{-1}, \quad q_{i-1} = q_i \frac{\tau_i + T_i}{T_{i-1}} \text{ for } i = N...2; \quad w_i = q_i \Big/ \sum_{i=1}^{N} q_i \tag{3}$$

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 Fig. 2 we present an example of comparison of the execution times achieved with these load balancing strategies on a set of highly heterogeneous

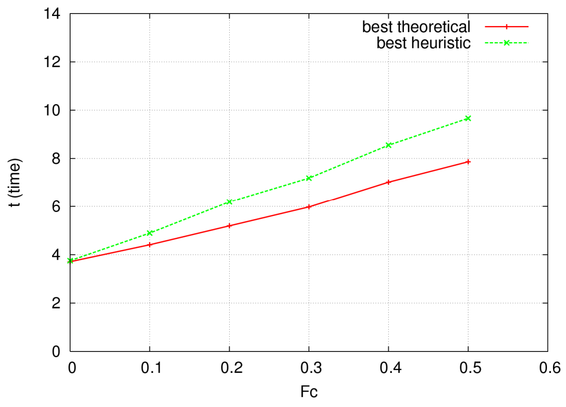

Fig. 2. Comparison of the execution times for different weighting: the best theoretical distribution versus the generic heuristic load balancing

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.

4.4 Influence of the Resource Heterogeneity on the Load Balancing Efficiency

Thorough testing of the different applications on different sets of resources showed a strong influence of the level of resource heterogeneity on the results achieved. We performed a series of targeted experiments varying the resource heterogeneity both in the processor power and the network links bandwidth. As a sample of these tests, in Fig. 3 we show the dependency of the load balancing speedup on the processing power heterogeneity metrics, analogous to that of the networks links heterogeneity introduced by Eq. (2). As we see, the speedup grows superlinearly with the heterogeneity level, thus indicating that our approach is especially beneficial on strongly heterogeneous resources, such as the Grid resources.

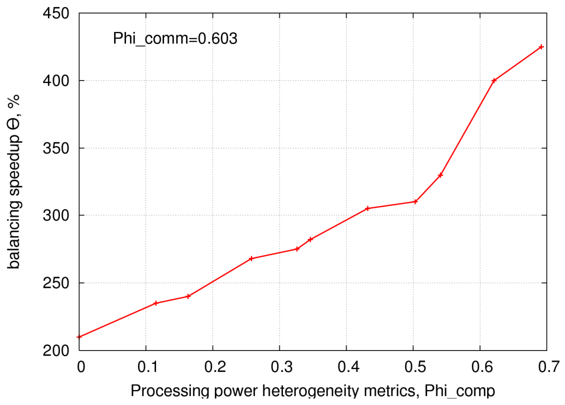

Fig. 3. Dependency of the load balancing speedup Θ on the resource heterogeneity metrics φ

5 Conclusions and Future Work

We introduced a new hierarchical approach that combines user-level job scheduling with dynamic load balancing technique that automatically adapts a *black-box* distributed or parallel application to the heterogeneous resources. The proposed algorithm dynamically selects the resources best suited for a particular task or parallel process of the application, and optimizes the load balance based on the dynamically measured resource parameters and estimated requirements of the application. We studied the performance of this load balancing approach by developing a synthetic application with flexible user-defined application parameters and logical network topologies, on artificially designed heterogeneous resources with a controlled level of heterogeneity. Some of the conclusions from our methodological experiments are as follows:

- The proposed algorithm adequately finds the application requirements;
- Based on that, our approach adapts the application to the set of heterogeneous resources with a very high load balancing speedup (up to 450 %);
- The novelty of our load balancing approach –dependency of the load distribution on the application and resource parameters– adds up to 150 % to the balancing speedup compared to the balancing that takes into account only the processors' performance;
- Analysis of the speedup achieved for different types of applications and resources indicates that the communication-intensive applications benefit most from the proposed load balancing technique.
- The speedup from applying our approach grows superlinearly with the increase of the resources' heterogeneity level, thus showing that it is especially useful for the severely heterogeneous Grid resources.
- Comparison of the performance of our heuristic load balancing with the performance achieved with the analytically derived weights, showed a relatively small discrepancy of 5-15 %, with a larger difference attributed to the very communication-intensive applications. This overhead is a relatively small impediment, considering that our approach is generic and suits any type of communication topology.

The results presented here were obtained for traditional parallel computing applications with the most widespread communication model: a Master-Worker scheme in a non-lockstep asynchronous mode. At present, we test other connectivity schemes, such as the different Master-Worker modes, as well as Mesh, Ring and Hypercube topologies. Another direction of our work is implementation and testing of hierarchical coupling of user-level job scheduling with the load balancing algorithm presented. The User-Level Scheduler [32,33] will provide a combined resource management strategy connecting the application-level resource selection mechanism to the system-level job management. In addition to that, it can support resource usage optimization and fault tolerance [23], as a desirable functionality increasing the usability of the Grid. We also plan to extend our approach to a wider class of applications, including memory-critical applications, multimedia streaming applications, and a widely used class of applications performing sequential computing with hard disk intensive operations.

Acknowledgments. The authors would like to thank Peter Sloot for fruitful discussions of this paper. The research was conducted with financial support from the Dutch National Science Foundation NWO and the Russian Foundation for Basic Research under projects # 047.016.007 and 047.016.018, and with partial support from the Virtual Laboratory for e-Science Bsik project.

References

1. Krzhizhanovskaya, V.V., Korkhov, V.V.: 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, pp. 917–932. Trapeznikov Institute of Control Sciences RAS, Moscow (2006)

2. Krzhizhanovskaya, V.V., Sloot, P.M.A., Gorbachev, Y.E.: Grid-based Simulation of Industrial Thin-Film Production. Simulation: Transactions of the Society for Modeling and Simulation International 81(1), 77–85 (2005)
3. Krzhizhanovskaya, V.V., Korkhov, V.V., Tirado-Ramos, A., Groen, D.J., Shoshmina, I.V., Valuev, I.A., Morozov, I.V., Malyshkin, N.V., Gorbachev, Y.E., Sloot, P.M.A.: Computational Engineering on the Grid: Crafting a Distributed Virtual Reactor. In: Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), p. 101 (2006)
4. Krzhizhanovskaya, V.V., et al.: 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)
5. Krzhizhanovskaya, V.V., Zatevakhin, M.A., Ignatiev, A.A., Gorbachev, Y.E., Sloot, P.M.A.: Distributed Simulation of Silicon-Based Film Growth. In: Wyrzykowski, R., Dongarra, J.J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 879–888. Springer, Heidelberg (2002)
6. Korkhov, V.V., Krzhizhanovskaya, V.V.: Workload Balancing in Heterogeneous Grid Environment: A Virtual Reactor Case Study. In: Proceedings of the Second International Conference Distributed Computing and Grid Technologies in Science and Education, pp. 103–113. Publ: JINR, Dubna, D11-2006-167 (2006)
7. Korkhov, V.V., Krzhizhanovskaya, V.V.: Benchmarking and Adaptive Load Balancing of the Virtual Reactor Application on the Russian-Dutch Grid. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J.J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 530–538. Springer, Heidelberg (2006)
8. Korkhov, V.V., Krzhizhanovskaya, V.V., Sloot, P.M.A.: A Grid Based Virtual Reactor: Parallel performance and adaptive load balancing. Revised version submitted to the Journal of Parallel and Distributed Computing (2007)
9. CrossGrid EU Science project: http://www.eu-CrossGrid.org
10. Nimrod-G: http://www.csse.monash.edu.au/~davida/nimrod/
11. Fox, G.: Grid Computing environments. IEEE Computers in Science and Engineering 10, 68–72 (2003)
12. Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.): Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers, Boston (2004)
13. Foster, I., Kesselman, C. (eds.): The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, Seattle (2003)
14. Buyya, R., Cortes, T., Jin, H.: Single System Image. The International Journal of High Performance Computing Applications 15(2), 124–135 (2001)
15. Maghraoui, K.E., Desell, T.J., Szymanski, B.K., Varela, C.A.: The Internet Operating System: Middleware for Adaptive Distributed Computting. The International Journal of High Performance Computing Applications 20(4), 467–480 (2006)
16. Sonmez, O.O., Gursoy, A.: A Novel Economic-Based Scheduling Heuristic for Computational Grids. The International Journal of High Performance Computing Applications 21(1), 21–29 (2007)
17. Boyera, W.F., Hura, G.S.: Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. J. Parallel Distrib. Comput. 65, 1035–1046 (2005)
18. Collins, D.E., George, A.D.: Parallel and Sequential Job Scheduling in Heterogeneous Clusters: A Simulation Study Using Software in the Loop. SIMULATION 77, 169–184 (2001)
19. Schoneveld, A., de Ronde, J.F., Sloot, P.M.A.: On the Complexity of Task Allocation. Complexity 3, 52–60 (1997)

20. de Ronde, J.F., Schoneveld, A., Sloot, P.M.A.: Load Balancing by Redundant Decomposition and Mapping. Future Generation Computer Systems 12(5), 391–407 (1997)
21. Karatza, H.D., Hilzer, R.C.: Parallel Job Scheduling in Homogeneous Distributed Systems. SIMULATION 79(5-6), 287–298 (2003)
22. Barak, A., Wheeler, R.G., Guday, S.: The MOSIX Distributed Operating System. LNCS, vol. 672. Springer, Heidelberg (1993)
23. Overeinder, B.J., Sloot, P.M.A., Heederik, R.N., Hertzberger, L.O.: A Dynamic Load Balancing System for Parallel Cluster Computing. Future Generation Computer Systems 12(1), 101–115 (1996)
24. Shao, G., et al.: Master/Slave Computing on the Grid. In: Proceedings of Heterogeneous Computing Workshop, pp. 3–16. IEEE Computer Society Press, Los Alamitos (2000)
25. Sinha, S., Parashar, M.: Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. In: Proceedings of 3rd IEEE Intl. Conference on Cluster Computing, pp. 435–442 (2001)
26. David, R., et al.: Source Code Transformations Strategies to Load-Balance Grid Applications. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 82–87. Springer, Heidelberg (2002)
27. Teresco, J.D., et al.: Resource-Aware Scientific Computation on a Heterogeneous Cluster. Computing in Science & Engineering 7(2), 40–50 (2005)
28. Kufrin, R.: PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. In: 6th International Conference on Linux Clusters, Chapel Hill, NC (2005)
29. Lu, C., Lau, S.-M.: An Adaptive Load Balancing Algorithm forHeterogeneous Distributed Systems with Multiple Task Classes. In: International Conference on Distributed Computing Systems (1996)
30. Lan, Z., Taylor, V.E., Bryan, G.: Dynamic Load Balancing of SAMR Applications on Distributed Systems. In: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (2001)
31. Zhang, Y., Hakozaki, K., Kameda, H., Shimizu, K.: A performance comparison of adaptive and static load balancing in heterogeneous distributed systems. In: The 28th Annual Simulation Symposium, p. 332 (1995)
32. Germain-Renaud, C., Loomis, C., Moscicki, J.T., Texier, R.: Scheduling for Responsive Grids. Grid Computing Journal (Special Issue on EGEE User Forum) (2006)
33. Moscicki, J.T., Bubak, M., Lee, H.-C., Muraru, A., Sloot, P.: Quality of Service on the Grid with User Level Scheduling. In: Cracow Grid Workshop Proceedings (2006)
34. Calvin, J.M.: A One-Dimensional Optimization Algorithm and Its Convergence Rate under the Wiener Measure. Journal of Complexity N 17, 306–344 (2001)
35. http://www.cs.vu.nl/das2/