The user-level scheduling of divisible load parallel applications with resource selection and adaptive workload balancing on the Grid

Korkhov, V.V.; Moscicki, J.T.; Krzhizhanovskaya, V.V.

DOI
10.1109/JSYST.2008.2011301

Publication date
2009

Document Version
Final published version

Published in
IEEE Systems Journal

The User-Level Scheduling of Divisible Load Parallel Applications With Resource Selection and Adaptive Workload Balancing on the Grid

Vladimir V. Korkhov, Jakub T. Moscicki, and Valeria V. Krzhizhanovskaya

Abstract—**This paper presents a hybrid resource management environment, operating on both application and system levels developed for minimizing the execution time of parallel applications with divisible workload on heterogeneous Grid resources. The system is based on the adaptive workload balancing algorithm (AWLB) incorporated into the distributed analysis environment (DIANE) User-Level Scheduling (ULS) environment. The AWLB ensures optimal workload distribution based on the discovered application requirements and measured resource parameters. The ULS maintains the user-level resource pool, enables resource selection and controls the execution. We present the results of performance comparison of default self-scheduling used in DIANE with AWLB-based scheduling, evaluate dynamic resource pool and resource selection mechanisms, and examine dependencies of application performance on aggregate characteristics of selected resources and application profile.**

Index Terms—**Adaptive workload balancing, grid, resource selection, user-level scheduling.**

I. Introduction

LARGE Grid infrastructures, such as enabling grids for E-science (EGEE) Grid [1], provide access to the computing resources at unprecedented scale. Designed for high-throughput applications, the Grid middleware and infrastructure comes with little support for the high-performance use-cases, especially the ones using a set of heterogeneous resources for a single application. Submitting, scheduling and mapping jobs on the Grid can take up to few orders of magnitude more time than the execution [2]. This is especially true for low-latency and short-deadline scenarios which are pervasive in a number of application domains from medical applications to physics data analysis. User-level scheduling (ULS) is one of the most promising ways to eliminate the difference in scale between short execution times and the large grid middleware latencies. The ULS system contains application-specific knowledge therefore it may provide customized resource selection and control mechanisms—a feature indispensable for enabling high performance applications on the Grid.

Efficient execution of parallel applications on heterogeneous and dynamic Grid resources is a challenging problem that requires the development of adaptive workload balancing algorithms that would take into account the application requirements and the resource characteristics. Generally studies on load balancing consider distribution of processes to computational resources on the system/library level with no modifications in the application code [16], [17]. Less often, load balancing code is included into the application source-code to improve performance in specific cases [18], [19]. Some research projects concern load balancing techniques that use source code transformations to speedup the execution [20]. In the proposed integrated system, a hybrid approach is employed, where the balancing decision is taken in interaction of the application with the execution environment.

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

In this paper we suggest an approach based on the integration of the distributed analysis environment (DIANE) ULS environment [2], [3] with the adaptive workload balancing algorithm (AWLB) [4]–[6]. The benefits of the integration are twofold: the AWLB is enriched with the capability to select resources most suitable for the application, and the ULS environment is equipped with an advanced strategy to optimize resource usage. Optimization of the workload performed by the AWLB is adaptable to the resource characteristics (CPU power, memory, network bandwidth, input/output (I/O) speed, etc.) and to the corresponding application requirements. The ULS environment acquires the most appropriate resources to the user-level resource pool, and the AWLB controls the workload distribution. We perform the studies using a model application with tunable characteristics (communication to computation ratio, memory usage,

Manuscript received September 09, 2007; revised June 30, 2008. First published February 06, 2009; current version published February 25, 2009. This work was supported in part by the Dutch National Science Foundation NWO, by the Russian Foundation for Basic Research under Projects #047.016.007 and 047.016.018, and by the Virtual Laboratory for e-Science Bsik Project.

V. V. Korkhov is with the Section Computational Science, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands (e-mail: v.korkhov@uva.nl).

J. T. Moscicki is with CERN, 1211, Geneva 23, Switzerland (e-mail: jakub.moscicki@cern.ch).

V. V. Krzhizhanovskaya is with the Section Computational Science, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands (e-mail: v.krzhizhanovskaya@uva.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2008.2011301

communication topology, etc.). In this paper, we show some results for the master-worker communication model, one of the most popular and widely used in scientific applications.

This paper is organized as follows. Section II presents the overview of related work on application-level scheduling; Section III briefly summarizes the AWLB algorithm, introduces the resource pooling and adaptive resource selection, and describes the strategy for integration of the adaptive workload balancing algorithm with the user-level scheduling environment; Section IV gives an overview of the DIANE user-level scheduling environment used for implementation and experiments, describes mapping of the real-life applications onto the synthetic model application used for experiments, and shows experimental results comparing the AWLB with the standard self-scheduling algorithm. Additionally, the results of adaptive resource selection for applications of different types are discussed. Section V concludes this paper.

II. Related Work

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

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

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

Numerous software packages have been developed as hard-wired solutions to specific applications. gPTM3D [12] is a Grid-enabled implementation of medical image reconstruction program in a master/worker model with self-scheduling. MPI-BLAST [15] is a parallel implementation of the genomic alignment search tool and may be run in-conjunction with Grid-enabled MPI implementations such as MPICH-G2 [29]. Client-server architecture has been exploited as *ad-hoc* solution for parallel earthquake source determination in EGEE Grid [13]. While being useful for their respective applications such implementations may not be easily reused in other contexts and outside of their application domains.

Some large virtual organizations (VOs), such as the ones in high energy physics (HEP), require central coordination of the Grid activities for data production, simulation, and analysis. Alien [10] and DIRAC [11] are the systems that have been implemented as end-to-end solutions deployed on the Grid at the level of HEP VOs. Job scheduling is based on pilot agents executing on the ordinary Grid resources and pulling tasks from central VO task queue. Resource negotiation sometimes involves additional services deployed at directly at the Grid sites. Alien and DIRAC systems support thousands of concurrent jobs from hundreds of users with higher efficiency than the generic Grid middleware. However, central and site services require systematic deployment and maintenance which can only be afforded by very large collaborations. Additionally, such systems also tend to be application-specific and difficult to reuse.

Finally, a number of reusable application-level scheduling systems have been developed in recent years. Nimrod [9] is a top-down solution for parameter-sweep applications which is able to negotiate resources using standard Globus protocols. Nimrod handles file transfers and automatic partitioning of the computation based on the user-supplied declarations in a special-purpose language. Based on the declarations the job wrapper scripts are automatically created for black-box application executables. Nimrod then controls the execution of jobs and resubmits the failed ones if necessary. The resource performance monitoring uses the Network Weather Service.

AppLES [8] provide generic templates for creating application-level schedulers and AppLES Parameter Sweep Template (APST) is an AppLES-based execution environment for parameter-sweep applications. XML-based specification is used for the description of the execution workflows and customization of task scheduling priorities. In this respect the APST supports more flexible application execution models than Nimrod. Both approaches aim at the "farming" applications, such as parameter sweeps, and lack sufficient support for resource-adaptive scheduling of parallel applications.

III. Integrated AWLB + ULS Environment

In [4] and [5], we developed an application-level AWLB for parallel applications on heterogeneous resources and validated its MPI implementation on the Grid. Intensive experimentation showed the necessity of a combined resource management on system and application levels. In [6], we introduced such an approach, where the environment selects and acquires resources according to the application requirements, while the application controls workload distribution. In this section we further develop this approach.

The application consists of a set of parallel tasks that process the workload scheduled by the Master task bound to the ULS environment. The ULS collects the information about the available resources and monitors the application responses. As the application may comprise heterogeneous tasks executed at different times with different performance characteristics the total application performance requirements may vary at runtime. Similarly, the capacity of Grid resources may vary with time due to inherent Grid dynamics. The ULS may respond to changing application or resource conditions and more suitable resource set may be selected to execute the application at a given time. This may happen at the individual task execution boundary or at other natural boundaries specific to the application model (for example at each iteration for the iterative simulations). This is a distinctive feature, in contrast to the traditional parallel programs where resources are allocated once and fixed during the execution (unless special migration libraries are used like Dynamite [17]). To support the replacement of resources during runtime, the concept of user-level resource pool is employed. This resource pool is maintained and supplied by the ULS environment which dynamically selects the resources most suitable for the application. The suitability of resources is determined by the application requirements; for traditional parallel computing applications considered here as a test case, it depends on the processing power and network connectivity correlated with the application communication to computation ratio.

After resources have been selected and assigned to the tasks, the workload balancing is performed. The AWLB algorithm is summarized in Section III-A. The computation is performed as an iterative process; at each iteration the distribution of the workload is reevaluated on a new set of resources, and the AWLB parameters are reestimated.

A. AWLB Algorithm

In this section, we briefly summarize the approach for AWLB of parallel applications on heterogeneous resources, introduced in [4] and [5]. The AWLB provides an optimal distribution of the divisible workload between participating processors according to the computing environment characteristics and the application requirements.

The main generic parameters that define a parallel application performance are as follows.

- The application parameter $f_c = N_{\mathrm{comm}}/N_{\mathrm{calc}}$, 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 ith processor (measured in Flop per second) and n_i is the network bandwidth to this node (measured in bits per second).

The AWLB algorithm is based on the benchmarking of the available resources capacity, defined as a set of individual resource parameters $\boldsymbol{\mu} = \{\mu_i\}$, and experimental estimation of the application parameter f_c. The value of the application parameter f_c is determined by running through the space of possible f_c and finding the value f_c^* which provides minimal runtime of the application on this set of resources.

The suitability of resources is determined by the application requirements; for traditional parallel computing applications considered here as a test case, it depends on the processing power and network connectivity correlated with the application communication to computation ratio. Thus the combination of $\boldsymbol{\mu}$ and f_c^* determines the distribution of the workload between the processors. To calculate the amount of workload per processor, we assign a weight-factor w_i to each processor according to its processing power and network connection. In [5], we derived an expression for the weighting factors w_i

$$w_i = p_i(1 + f_c\varphi/\mu_i)$$

where φ is the resource heterogeneity metric also introduced in [5]. The workload for a processor is given by $W_i = w_iW$, where W is the total application workload. To evaluate the efficiency of the workload distribution we introduce the load balancing speedup Θ as

$$\Theta = \frac{T_{\mathrm{non-balanced}}}{T_{\mathrm{balanced}}}$$

where $T_{\mathrm{non-balanced}}$ is the execution time without load balancing, and T_{balanced} is the execution time using load balancing on the same resource set (the time taken to execute the algorithm itself is included). The algorithm is flexible, lightweight and suitable for different types of applications, e.g., I/O intensive ones [6].

B. Resource Pooling and Selection

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

The refinement of resource selection in the pool is based on the basic optimality principle in divisible load scheduling problems. The optimality principle states that to obtain the optimal processing time, all the participating processors must stop computing at the same instant in time [30]. Although the optimality principle remains valid for arbitrary network topologies, the optimal time performance depends crucially on the selection of a proper subset of the available processors: elimination of slow processor-link pairs may lead to better performance if it improves the load distribution. Thus, using a larger set of nodes may yield an inferior performance compared to an optimal subset of nodes among which the load is distributed according to the optimality principle.

Division of resources into fast and slow may be done with a help of a ranking algorithm. Evidently, the meaning of "fast" and "slow" for a resource depends on the type of application it is supposed to run. Thus, we have to introduce a metric for appropriate resource ranking dependent on f_c of a particular application.

To rank the resources, the metric similar to the one used for processor weighting in AWLB

$$r_i \sim p_i(1 + f_c/\mu_i)$$

where r_i is the rank of processor i.

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

$$\begin{aligned}&M = 1\\&\text{while}\,(t(M+1)/t(M) < 1 - \xi)\\&M = M + 1\end{aligned} \tag{1}$$

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

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

C. *Structural Scheme of the* AWLB + ULS *Approach*

The outline of adaptive workload balancing in ULS environment is presented in Fig. 1 as a meta-algorithm based on the concepts introduced in Sections III-A, III-B, and in [6]. The application execution control is performed within the framework of the ULS environment, and two levels are distinguished: the resource pool level and the application level. The resource pool is managed by the ULS environment that receives the application feedback on the changing performance requirements. In turn, the application level retrieves the information on available pooled resources and controls the workload distribution on these resources using AWLB. The dynamic application requirements determined and updated by AWLB are forwarded to the ULS environment to be used in resource selection and pooling.

The interaction between the resource pool and application levels is performed via the Master task (in the current version of the environment we consider the Master/Worker model) that can access the resource information retrieved by ULS. This is the part of the application that has to be modified in case of porting existing codes to the integrated AWLB+ULS environment. The master task retrieves the resource pool information from the ULS, controls the workload distribution using AWLB, updates application requirements and forwards them to ULS to be used for resource selection.

Explanation of the steps in Fig. 1 is provided as follows.

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

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

Step R2. Benchmark resources: Measure the worker node parameters such as the computational power, memory, bandwidth of the network links, hard disk capacity and I/O speed. In a more generic sense of resources, some other metrics may be added characterizing the equipment and tools associated with a particular Grid node.

Step R3. Rank resources: Update and reorder the list of pooled resources (used by the resource selection procedure in Steps A2 and A3).

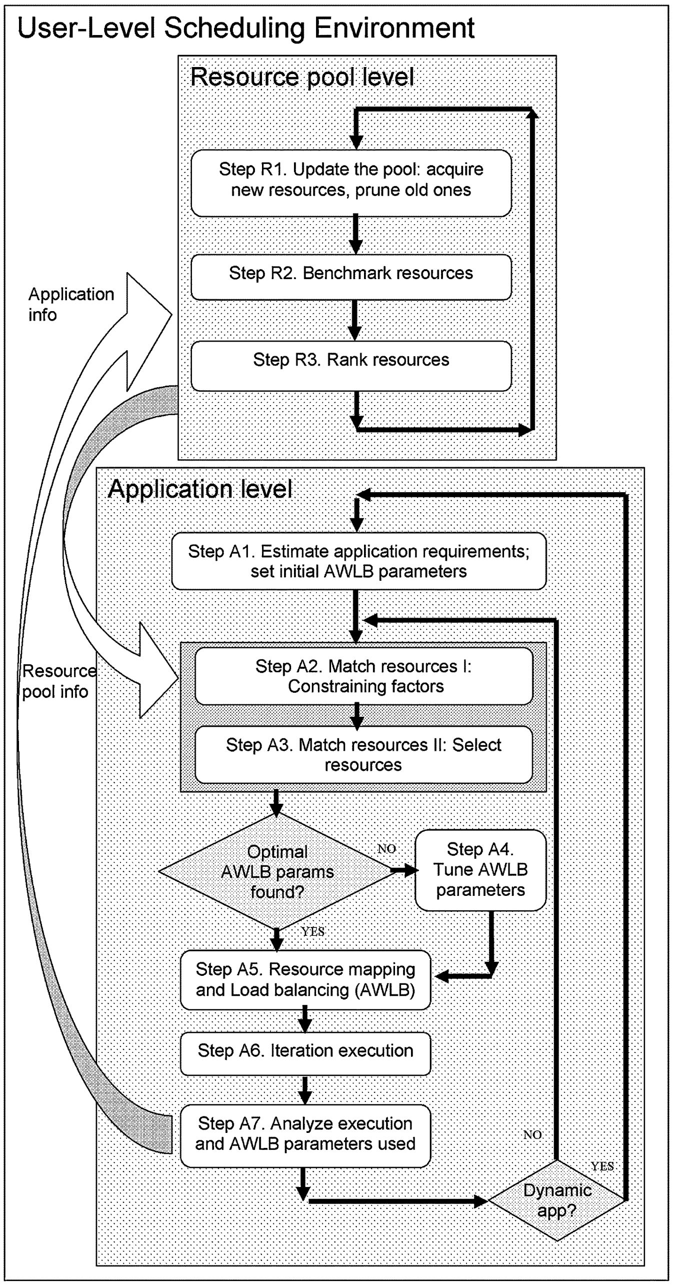
User-Level Scheduling Environment
Resource pool level
Step R1. Update the pool: acquire new resources, prune old ones
Step R2. Benchmark resources
Step R3. Rank resources
Application info
Resource pool info
Application level
Step A1. Estimate application requirements; set initial AWLB parameters
Step A2. Match resources I: Constraining factors
Step A3. Match resources II: Select resources
Optimal AWLB params found?
NO
YES
Step A4. Tune AWLB parameters
Step A5. Resource mapping and Load balancing (AWLB)
Step A6. Iteration execution
Step A7. Analyze execution and AWLB parameters used
NO
YES
Dynamic app?

Fig. 1. Iterative execution of a parallel application in the integrated AWLS + ULS environment with a dynamic resource pool.

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

Step A1. Estimate application requirements; set initial AWLB parameters: Application requirements are used to set initial values of the AWLB parameters (see Section III-A). These parameters are automatically tuned during runtime (Step A4).

Step A2. Matching resources I. Constraining factors: Use the ranked resource list (Step R3) to filter out the resources which do not meet the minimal application requirements determined in Steps A1 and A7. In our computational application example, memory is the constraining factor: in case of insufficient memory, the processor is disregarded from the computation and removed from the resource pool. For a large emerging class of multimedia streaming applications, free disk space may be a constraining factor.

Step A3. Matching resources II. Selecting resources: 1) Find the optimal number of processors. It is either defined by the user or provided by the AWLB mechanism, depending on the performance data of the processors. 2) Select best suited resources. The suitability is determined by the application characteristics: for the communication-bound applications, network links bandwidth is the top priority ranking parameter, for the computation-intensive applications the processing power is, and for the intermediate cases the resources are chosen to best fit the application communication to computation ratio, which is discovered by the AWLB algorithm. Analogously, resource selection for the memory-critical applications is based on the memory metric, and for streaming applications network bandwidth and disk I/O speed are the key parameters.

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

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

Step A5. Resource mapping and load balancing: Actual optimization of the workload distribution within the parallel tasks is performed, i.e., mapping the processes and workload onto the allocated resources. This step is based on the AWLB algorithm described in Section III-A. It includes a method to calculate the weighting factors for each processor depending on the resource characteristics measured in Steps R3–R5 and application requirements estimated in Steps A1 and A7.

Step A6. Iteration execution: Perform an iteration (including calculations and communications) with the workload distribution defined in Step A4.

Step A7. Analyze execution and AWLB parameters: Measure the execution time of a single iteration with the current AWLB parameters and quantitatively estimate the requirements of the application based on the results of resource benchmarking (Step R2) and measurements of the application response.

For dynamic resources, when performance is influenced by other factors, a periodic re-estimation of resource parameters and load redistribution is performed. If the application is dynamically changing (for instance due to adaptive meshes or different combinations of physical processes modeled at different simulation stages), then the application requirements must be periodically re-estimated even on the same set of resources.

IV. Implementation and Experimental Results

To implement and validate the hybrid $\mathrm{AWLB}+\mathrm{ULS}$ approach described in Section III-C, we have chosen the DIANE [21], [22], which is a realization of user-level scheduling environment developed at CERN. The framework provides an execution environment for parametric parallel applications, i.e., the applications that are not communication-bound and the communications occur in regular patterns. This covers a broad class of applications including parameter sweep, data-analysis if data locality is assumed, Monte Carlo simulations, etc. The communication backbone of the environment is based on the Master/Worker model, however customization allows to achieve more complex task synchronization patterns. DIANE allows to plug-in user-defined scheduling algorithms and failure-recovery strategies. DIANE layer runs as a set of regular user jobs, and therefore it operates entirely inside the user space. User-level scheduling does not require any modification to the Grid middleware and infrastructure, nor the deployment of special services in the Grid sites, thus it provides immediate exploitation of Grid resources available to the user. Lightweight, transient services, such as the Directory Service, are run locally on user-controlled computer and may be enabled or disabled at any time. The Directory Service controls the resource pool.

DIANE has been used with a number of applications, from black-box executables (e.g., image processing [24], telecommunications [25], physics simulation, regression testing, data analysis [26]) to interfacing of the applications at the source-code level (e.g., medical physics and bio-informatics [14]). DIANE allows to increase the application performance, minimize the feedback latency [2], and improve certain quality of service (QoS) parameters of the Grid, such as reliability and predictability.

DIANE provides a software plug-in framework and uses Ganga [23] as a job abstraction and management layer. DIANE user-level scheduling exploits the concept of late-binding also known as placeholders or pilot agents similarly to Condor-G glide-ins [7]. The Grid jobs run generic agents which get the workload from a scheduler—a Run Master service. Deferring the mapping of workload until the runtime helps to short-circuit the overhead of hierarchical scheduling and to accurately react to the dynamically changing characteristics of the resources or of the application. The execution of tasks is controlled by the Run Master service running on local user computer.

Typically the Grid worker nodes have the constraints of outbound-only direct connectivity with the outside networks. Certain subsets of Grid sites may provide inbound connectivity on certain ports in order to support the cross-cluster MPI applications [31]. However, such support is not ubiquitous and may not be relied upon for an average Grid user in an average VO. The worker node crosstalk is still possible in DIANE, however, the communication is routed via the Run Master service. In DIANE model, the output of one task may trigger execution of another task if decided by the Run Master. This is the only allowed way of inter-worker communication. This model is more efficient for applications with low f_c parameter because the communication even between neighboring workers is routed via a distant point in the wide area network. On the other hand the AWLB methodology may be applied directly because the resource parameter $\boldsymbol{\mu}$ is evaluated always against the same worker endpoint (Run Master). Thus, the $\boldsymbol{\mu}$ is invariant of any communicating worker node pair.

The experiments were carried out on the EGEE Grid production infrastructure [1], in the Geant4 VO [27]. The model application is implemented as a python-based plug-in for DIANE environment. The master-worker model of execution is employed, where the master selects the amount of workload to be processed by a worker, sends the data to workers and receives the results.

We compare performance results achieved using the AWLB algorithm and dynamic resource pool with the results shown by the standard DIANE task dispatching technique—self-scheduling [also called a first-input–first-output (FIFO) scheduling algorithm]. In self-scheduling all the workload is divided into tasks of equal size; typically the number of tasks largely exceeds the number of available worker nodes. As soon as a worker becomes available, the next task from the list is assigned to it. In AWLB all the workload is divided into the number of available workers, and the size of the workload assigned to each task is calculated by the heuristic algorithm, using the resource and application characteristics.

A. AWLB Parameters in Real Applications

In this section, we examine the Feynsect application [28] which computes the generic Feynman diagrams with one and two loops using one of the numerical algorithms available and the Monte Carlo integration technique. The 5-D parameter space of elementary particles in the minimal super-symmetric standard model (MSSM) consists of around 1000 independently computable points, so the problem is trivial to parallelize. The computation of each MSSM point consists of execution of a set of independent algorithms which correspond to different Feynman diagrams and Monte Carlo integration sectors. The number of algorithms (around 1000) is the same for all points. Task parameters consist of the MSSM point (on average 500 bytes text file) and the algorithm source code (on average 50 kB gzipped Fortran code). The algorithm is sent to the worker node and compiled on the fly, unless it is already in the local worker cache. Additionally, common Feynsect C++ libraries (1 MB) are downloaded and compiled once for each worker node. The algorithm execution time varies between 100 and 1000s. The output of the algorithm execution is a text file (3–5 kB).

The Feynsect application is the simplest case of the application with divisible workload, as all the points and algorithms are independent from one another, thus the unit of granularity is a single point with a single algorithm to execute. The more powerful the worker node is, the more points and algorithms it can process during a single workload balancing iteration.

The calibration runs showed that the value of f_c for this application lies in the range of [0.001, 0.5] b/Flop, but has only a few discrete points corresponding to the different algorithms used. The exact values of f_c are automatically found by the AWLB algorithm during several initial iterations. A detailed description of this procedure is given in [5]. In short, to discover the

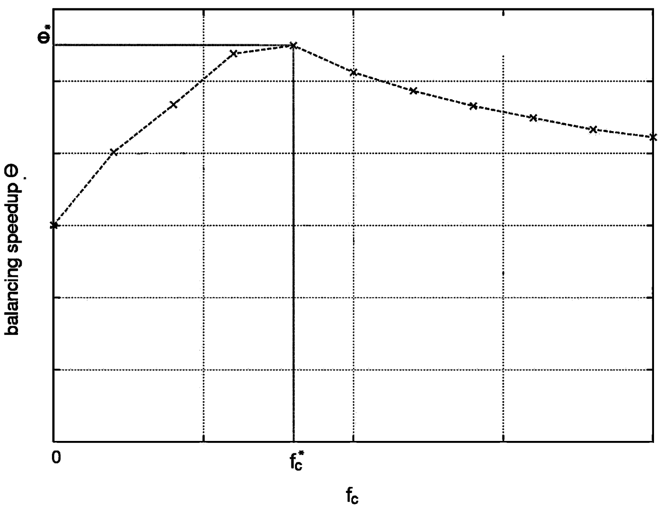

Fig. 2. Search through the range of possible application parameters f_c. Automatically discovered $f_c = f_c^*$ provides the best load-balancing speedup.

application parameter f_c, the search through the range of possible values of f_c is performed. For each value of f_c the corresponding weighting of resources and the load distribution is calculated based on the resource parameters $\boldsymbol{\mu}$ determined separately (see Section III-A). Then one time step (iteration) is performed, and the execution time and balancing speedup Θ are measured. Selection of the next value of f_c may be done by any optimization method for unimodal smooth functions, such as a simple line-search method. Fig. 2 shows a sample search through the range of possible values of f_c to find the f_c^* that provides the best balancing speedup Θ. This value f_c^* is the application parameter we were looking for; it is later used for calculating the load distribution and selecting the best fitting resources from the pool.

To validate the methodology of the integrated $\mathrm{AWLB} + \mathrm{ULS}$ approach in a broader range of application parameters, we experimented with a model application with a synthesized workload and tunable ratio f_c. This model application may represent the whole range of possible use cases of Feynsect algorithm, and in addition simulate a broader range of application parameters f_c.

B. AWLB and Self-Scheduling Performance

Fig. 3 presents the comparison of execution times of a single iteration with the AWLB and self-scheduling (FIFO) algorithms. In this figure COMP is the total amount of computational operations (in Flops) and COMM is the total amount of communications for each simulation (in bytes transferred). In this experiment the number of processors used is increased by one processor for each iteration. In all cases, the AWLB significantly outperforms the self-scheduling almost twice. In some cases, the gain can be up to several times (data not shown).

Fig. 4 shows how the application communication to computation ratio f_c influences the execution time. During this experiment, the computational load (COMP) was kept constant on a fixed set of 16 processors, while the amount of data transferred

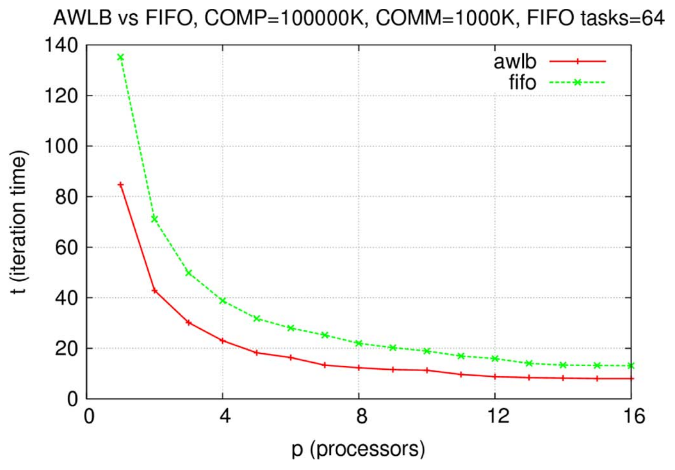

Fig. 3. Comparison of AWLB and self-scheduling algorithms: Runtime dependency on the number of processors acquired. $f_c = 0.01$.

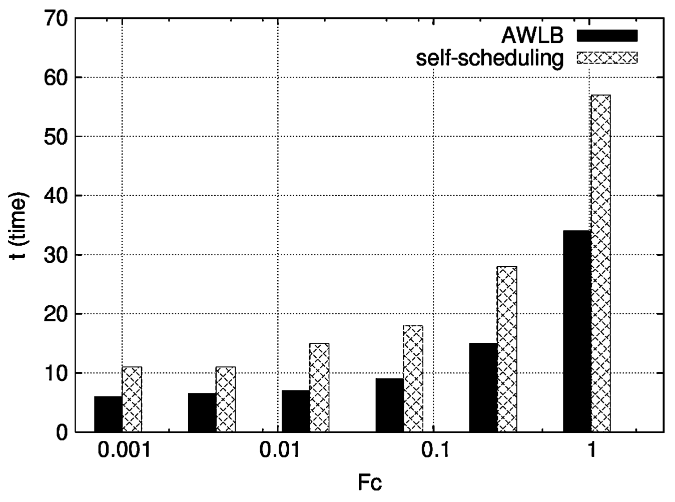

Fig. 4. Comparison of AWLB and self-scheduling algorithms for different values of application communication/computation ratio f_c with 16 workers and 64 self-scheduling tasks.

between the master and the workers (COMM) is varied. For all types of simulations, the AWLB algorithm is significantly faster than the self-scheduling.

Table I illustrates a typical example of AWLB parameters on a set of heterogeneous resources. Each worker is weighted according to the resource parameters and application characteristics (see Section III-A). In the table, PROC and NET are the relative processor and network capacity of a worker (results of benchmarking). The application values COMM and COMP are the same as in Fig. 3.

Fig. 5 presents the statistics of an actual run using a dynamically populated resource pool. The core feature of the ULS environment is the ability to change resources during the execution runtime, such that every iteration can run on a different set of resources. Fig. 5(a) shows how the resources are gradually added to the resource pool, depending on their availability. We can also notice that some workers are removed from the pool: the number of workers is not growing steadily, but experiences dips every few iterations.

TABLE I
SAMPLE DISTRIBUTION OF PROCESSOR WEIGHTS (W) BY AWLB ON A SET OF HETEROGENEOUS WORKERS

Worker ID	PROC	NET	Weight
1	57.9	4.3	0.011
2	42.4	13.1	0.019
3	52.3	28.8	0.037
4	54.8	28.9	0.038
5	55.5	28.7	0.037
6	96.0	16.4	0.028
7	42.4	28.9	0.036
8	42.4	28.5	0.036
9	41.2	28.4	0.035
10	53.0	8.9	0.015
11	53.2	28.3	0.037
12	33.6	3.2	0.07
13	41.8	28.9	0.036
14	42.1	28.1	0.035
15	49.9	42.6	0.052
16	47.0	27.8	0.035
17	53.8	27.9	0.036
18	22.2	7.4	0.010
19	55.2	28.4	0.037
20	57.8	4.3	0.011
21	39.5	28.0	0.035
22	64.9	11.9	0.020
23	29.9	9.8	0.014
24	41.5	28.9	0.036
25	41.1	28.8	0.036
26	41.1	28.6	0.036
27	41.0	28.5	0.036
28	41.1	19.0	0.025
29	45.9	42.0	0.051
30	41.7	28.6	0.036
31	40.9	29.1	0.036
32	41.4	28.7	0.036

In Fig. 5(b), the basic resource matching and selection mechanism is demonstrated (Step A3). All the workers acquired at each iteration [shown in Fig. 5(a)] are used for the execution, and the execution time depends on the number of available workers at the moment. Notice that at iterations number 9,13,16 when some resources left the pool, the execution time increases.

C. *Adaptive Resource Selection*

To illustrate the adaptive resource selection algorithm presented in Section III-B, we analyze the execution of the model application on the dynamically acquired Grid resources. The distribution of resources obtained for the experiments is presented in Fig. 6(a). Each point in the plot reflects a single available worker with the corresponding processor performance and network connectivity to the master.

To check the behavior of different types of applications we modeled different amount of computations and communications (i.e., different f_c). The resource ranking used for resource selection (see Section III-B) is based on application properties, thus the rank of a single resource depends on the application it is used to execute. Fig. 6(b) illustrates resource ranking for different application types (i.e., different values of f_c) for the same processor/network parameter distribution shown in Fig. 6(a).

The general idea of resource selection is to provide the best resource subset from the set of available resources to ensure the fastest possible execution of the application. To estimate the performance gain from the resource selection procedure, we

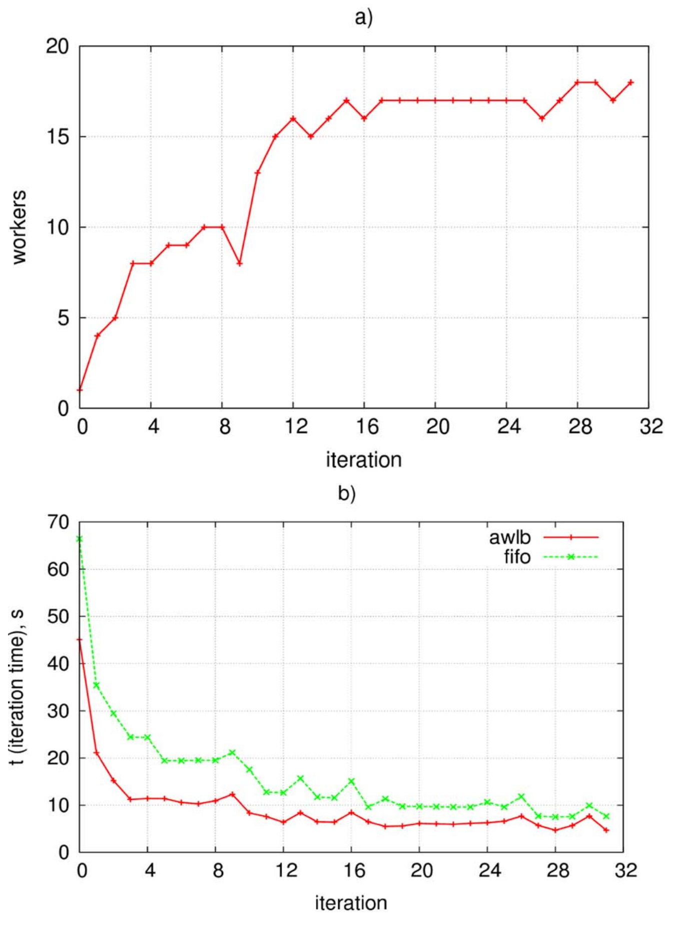

Fig. 5. (a) Dynamic resource pool population. (b) Sample execution times using dynamic pool.

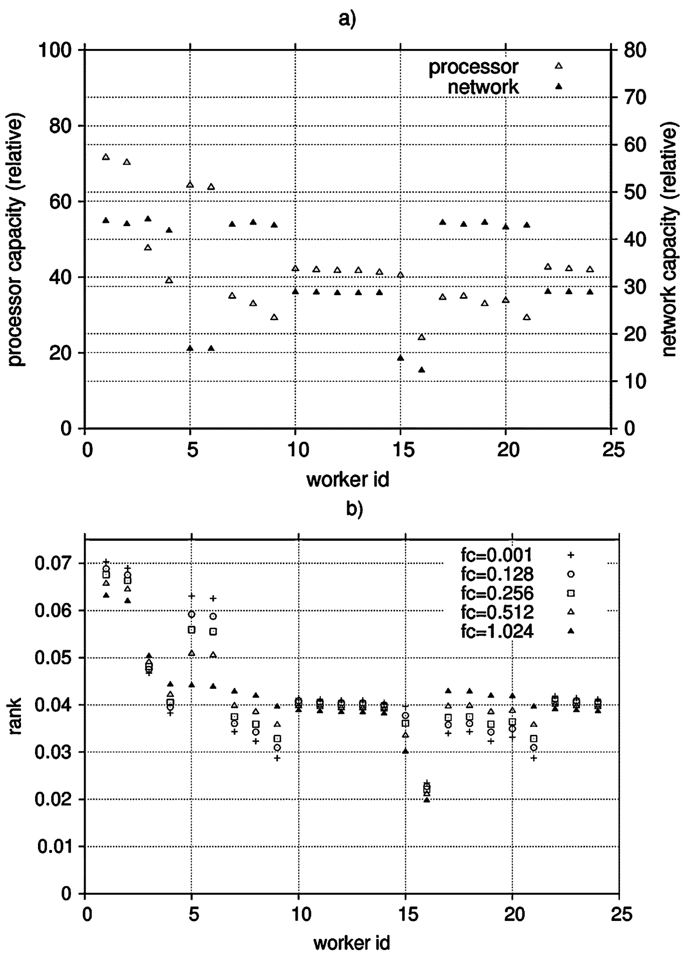

Fig. 6. (a) Sample resource distribution: processor and network capacity for the workers in the resource pool. (b) Resource ranks for the workers (dependency on f_c).

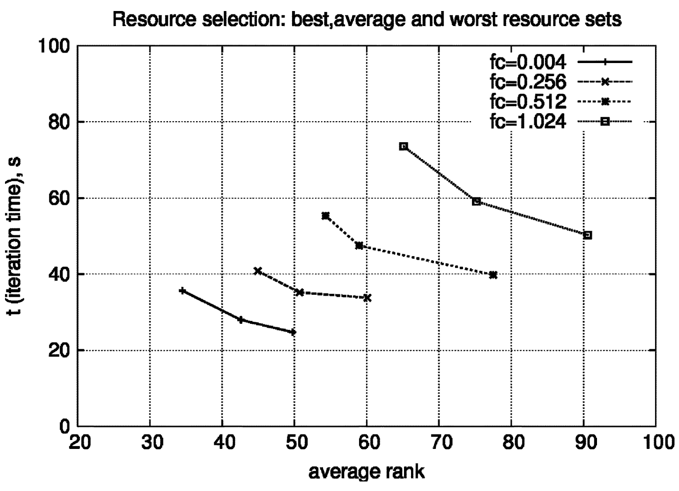

Fig. 7. Resource selection performance: comparison of the best, average, and worst ranked resources. On X-axis the average rank of each resource set is shown.

analyzed execution times on the best, average, and worst sets of available resources. The number of workers was fixed and the resources were selected from the top, middle and bottom of the ranked resource list. Thus, the performance was estimated for the same number of workers with the highest, average and lowest ranks. Fig. 7 illustrates the efficiency of the resource selection algorithm by presenting the resulted performance difference.

The number of processors to be used for efficient execution of a parallel application depends on both the application and resource characteristics; therefore, it is not possible to predict the speedup saturation point (the number of workers to be used). The adaptive resource selection mechanism (see Section III-B) analyses the speedup growth with addition of every new worker to the set of workers already used. The threshold ξ of the minimal acceptable speedup growth is defined by the user.

Fig. 8 presents sample execution of applications with different f_c on the same resource pool and $\xi = 0.1$. For each f_c two curves are shown: the execution time (using AWLB algorithm) and the number of workers used for each iteration. The number of workers is determined automatically using resource selection algorithm and performance data from the previous iterations. As expected, the speedup of the application with higher communication demand saturates on a smaller number of workers, which is tracked by the resource selection algorithm. The resources providing only marginal speedup growth are not acquired from

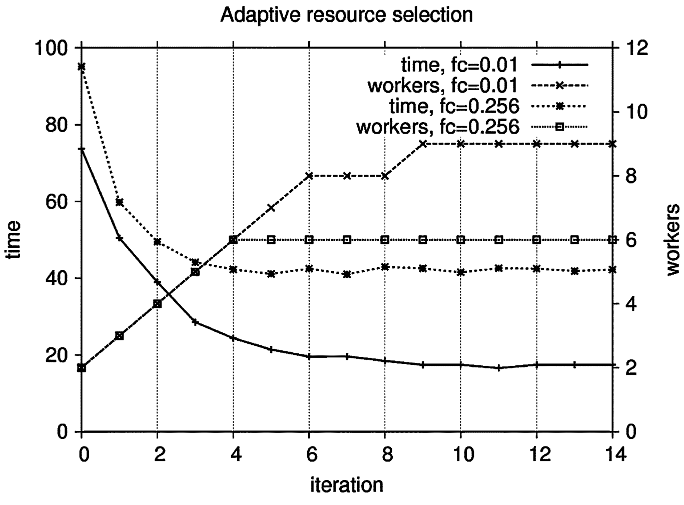

Fig. 8. Adaptive selection of the number of workers, $f_c = \{0.01, 0.256\}$.

the resource pool, and can be more efficiently used for another application accessing the same resource pool.

V. Conclusion

Evolution of the Grid paradigm to support High Performance Computing is indispensable for a large number of applications which require on-demand access to the Grid resources. In this paper, we proposed an approach to enhance the quality of handling multi-task jobs in Grid environment by integrating the AWLB developed for parallel applications on heterogeneous resources and ULS environment. The latter provides the missing link between the application and Grid resource managers: this user-level middleware is a customizable, application-centric scheduler and application hosting environment. Dynamic benchmarking of resources and estimation of the application characteristics is used to optimize the usage of a dynamic user-level pool of Grid resources maintained by the ULS. We devise a generic recipe on how to solve the workload balancing problem on the Grid for arbitrary applications. To prove the concept we performed a series of tests using a model application with configurable requirements. The EGEE Grid resources and the DIANE user-level scheduler with incorporated AWLB algorithm formed the infrastructure for the experiments. We study the range of parameters which correspond to an existing Feynsect application from theoretical physics and present experimental results and discussion on the way to manage the workload of divisible load parallel applications on the Grid. We compare different workload distribution methods and illustrate the usage of dynamic resource pool and application performance dependencies with adaptive resource selection. We plan to enhance the resource selection and match-making mechanisms by further development of the automated application performance analysis, improvement of handling the dynamic resource pool, and validation of the approach by implementing a real-life distributed computing application in the AWLB+ULS environment.

References

[1] F. Gagliardi, B. Jones, F. Grey, M.-E. Begin, and M. Heikkurinen, "Building an infrastructure for scientific grid computing: Status and goals of the EGEE project," *Philosophical Trans. Royal Soc. A.*, vol. 363, no. 1833, pp. 1729–1742, 2005.

[2] C. Germain-Renaud, C. Loomis, J. Moscicki, and R. Texier, "Scheduling for responsive grids," *Grid Comput. J.*, vol. 6, no. 1, pp. 15–27, 2006.

[3] J. T. Moscicki, M. Bubak, H.-C. Lee, A. Muraru, and P. Sloot, "Quality of service on the grid with user level scheduling," in *Cracow Grid Workshop Proc.*, 2006, pp. 119–129.

[4] V. V. Korkhov and V. V. Krzhizhanovskaya, "Benchmarking and adaptive load balancing of the virtual reactor application on the Russian-Dutch grid," *LNCS*, vol. 3991, pp. 530–538, 2006.

[5] V. V. Korkhov, V. V. Krzhizhanovskaya, and P. M. A. Sloot, "A grid based virtual reactor: Parallel performance and adaptive load balancing," *J. Parallel Distrib. Comput.*, vol. 68, no. 5, pp. 596–608, 2008.

[6] V. V. Krzhizhanovskaya and V. V. Korkhov, "Dynamic load balancing of black-box applications with a resource selection mechanism on heterogeneous resources of the grid," in *Conf. Parallel Comput. Technol. (PaCT), LNCS*, Berlin/Heidelberg, 2007, vol. 4671, pp. 245–260.

[7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "A computation management agent for multi-institutional grids," *Cluster Comput. J.*, vol. 5, no. 3, pp. 237–246, 2002.

[8] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using AppLeS," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 4, pp. 369–382, Apr. 2003.

[9] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS)*, Cancun, Mexico, May 2000, pp. 520–528.

[10] P. Saiz, L. Aphecetche, P. Buncic, R. Piskac, J.E. Revsbech, and V. Sego, "AliEn—ALICE environment on the GRID," *Nucl. Instruments Methods Phys. Res. Sect. A*, vol. 502, no. 2–3, pp. 437–440, 2003.

[11] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: A scalable lightweight architecture for high throughput computing," in *Proc. 5th IEEE/ACM Int. Workshop Grid Comput. (GRID)*, 2004, pp. 19–25.

[12] C. Germain-Renaud, R. Texier, and A. Osorio, "Interactive reconstruction and measurement on the grid," *Methods Inf. Med.*, vol. 44, no. 2, pp. 227–232, 2005.

[13] D. Weissenbach and E. Clevede, "Faster earthquake source mechanism determination with EGEE," presented at the 1st EGEE Conf., Geneva, Switzerland, 2006.

[14] H. C. Lee, J. Salzemann, N. Jacq, H.Y. Chen, L.Y. Ho, I. Merelli, L. Milanesi, V. Breton, S.C. Lin, and Y.T. Wu, "Grid-enabled high-throughput in silico screening against influenza a neuraminidase," *IEEE Trans. Nanobiosci.*, vol. 5, no. 4, pp. 288–295, Apr. 2006.

[15] A. Darling, L. Carey, and W. Feng, "The design, implementation, and evaluation of mpiBLAST," presented at the 4th Int. Conf. Linux Clusters: HPC Revolution Conjunction With ClusterWorld Conf. Expo, San Jose, CA, Jun. 2003.

[16] A. Barak and O. La'adan, "The MOSIX multicomputer operating system for high perfomance cluster computing," *FGCS*, vol. 13, no. 4–5, pp. 361–372, 1998.

[17] K. A. Iskra, F. van der Linden, Z.W. Hendrikse, B.J. Overeinder, G.D. van Albada, and P.M.A. Sloot, "The implementation of dynamite—An environment for migrating PVM tasks," *Operat. Syst. Rev.*, vol. 34, no. 3, pp. 40–55, 2000.

[18] G. Shao, F. Berman, and R. Wolski, "Master/Slave computing on the grid," in *Proc. Heterogeneous Comput. Workshop*, 2000, pp. 3–16.

[19] S. Sinha and M. Parashar, "Adaptive runtime partitioning of AMR applications on heterogeneous clusters," in *Proc. 3rd IEEE Int. Conf. Cluster Comput.*, 2001, pp. 435–442.

[20] R. David, S. Genaud, A. Giersch, B. Schwarz, and E. Violard, "Source code transformations strategies to load-balance grid applications," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 2002, vol. 2536, pp. 82–87.

[21] J. T. Moscicki, "Distributed analysis environment for HEP and interdisciplinary applications," *Nucl. Instruments Methods Phys. Res. A*, vol. 502, pp. 426–429, 2003.

[22] CERN, Geneva, Switzerland, "DIstributed ANalysis Environment," [Online]. Available: http://cern.ch/diane

[23] J. T. Moscicki, "Ganga—A computational task management tool for easy access to Grid," *Comput. Phys. Commun.*, accepted for publication.

[24] G. Carrera, E. de Andres, J.T. Moscicki, A. Muraru, S.H.W. Scheres, and J.M. Carazo, "Heavy computational tasks on the EGEE Grid: 2D/3D maximum-likelihood refinement," presented at the Netw. Excellence 3DEM Ann. Meet., Palma, Jan. 2007.

[25] A. Manara, "Integration of new communities in the Grid for mission critical applications: Distributed radio-frequency compatibility analysis for the ITU RRC06 conference," in *Proc. EGEE Conf.*, Geneva, Switzerland, Sep. 2006.

[26] CERN, Geneva, Switzerland, "Atlas Computing—Technical design report," CERN-LHCC-2005-022, 2005.

[27] CERN, Geneva, Switzerland, "Geant4 VO," 2005. [Online]. Available: http://lcg-voms.cern.ch/vo/geant4

[28] C. Anastasiou, S. Beerli, and A. Daleo, Evaluating multi-loop feynman diagrams with infrared and threshold singularities numerically CERN, Geneva, Switzerland, CERN-PH-TH-2007-058, Mar. 2007.

[29] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A grid-enabled implementation of the message passing interface," *J. Parallel Distrib. Comput. (JPDC)*, vol. 63, no. 5, pp. 551–563, May 2003.

[30] B. Veeravalli, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Comput.*, vol. 6, no. 1, pp. 7–17, 2003.

[31] G. Borges, "Interactive European Grid Project," INGRID, Lacco Ameno, Ischia, Italy, 2008 [Online]. Available: http://www.interactive-grid.eu/

Vladimir V. Korkhov received the Master's degree in mathematics and computer science from St. Petersburg Institute of Fine Mechanics and Optics, St. Petersburg, Russia, in 2001. He is currently finalizing the Ph.D. thesis at University of Amsterdam (UvA), Amsterdam, The Netherlands.

He is a Researcher with the Faculty of Science, UvA. His research interests include grid computing, distributed software systems, resource management and workload balancing in heterogeneous environment, and workflows on the grid. He is the author of more than 20 conference and journal papers.

Jakub T. Moscicki received the M.Sc. degree in computing from the AGH University of Science and Technology, Krakow, Poland.

He is a Software Engineer and Researcher with CERN, Geneva, Switzerland. His research interests include the distributed and parallel applications deployed on large-scale computing infrastructures such as the Grids. The multidisciplinary application portfolio includes high-energy physics, theoretical physics, medical and radiation studies, bio-informatics and drug design, telecommunications and simulation.

Valeria V. Krzhizhanovskaya received the M.Sc. degree in applied mathematics and physics from St. Petersburg State Polytechnic University (StPSPU), St. Petersburg, Russia, and the Ph.D. degree in computational science from University of Amsterdam (UvA), Amsterdam, The Netherlands.

She is a Researcher with UvA, and a Senior Lecturer with StPSPU. She has published over 40 papers. Her research interests include parallel distributed computing in heterogeneous systems, Grid computing, problem solving environments; modeling, simulation and numerical methods in physics.

Dr. Krzhizhanovskaya worked as a guest editor of 4 special issues of the *International Journal of Multiscale Computational Engineering*, organized 5 *International Symposia on Simulation of Multiphysics Multiscale Systems*, served as a program committee member and a reviewer in over 20 conferences and 6 international journals, participated in more than 40 conferences, and worked in about 20 international projects.