Benchmarking and Adaptive Load Balancing of the Virtual Reactor Application on the Russian-Dutch Grid

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

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

Abstract. This paper addresses a problem of porting a distributed parallel application to the Grid. As a case study we use the Virtual Reactor application on the Russian-Dutch Grid testbed. We sketch the Grid testbed infrastructure and application modular architecture, and concentrate on performance issues of one of the core parallel solvers on the Grid. We compare the performance achieved on homogeneous resources with that observed on heterogeneous computing and networking infrastructure. To increase the parallel efficiency of the solver on heterogeneous resources we developed an adaptive load balancing algorithm. We demonstrate the speedup achieved with this technique and indicate the ways to further enhance the algorithm and develop an automated procedure for optimal utilization of Grid resources for parallel computing.

Keywords: Grid, benchmarking, adaptive load balancing, heterogeneous resources, parallel distributed application, Virtual Reactor, PECVD.

1 Introduction

The importance of fully integrated simulators is recognized by various research groups and scientific software companies [1]. Our Virtual Reactor application [2,3] was developed for simulation of plasma enhanced chemical vapour deposition (PECVD) reactors, multiphysics systems spanning a wide range of spatial and temporal scales. Simulation of three-dimensional flow with chemical reactions and plasma discharge in complex geometries is one of the most challenging and demanding problems in computational science and engineering, requiring both high-performance and high-throughput computing. This application serves as a test-case driving and validating the development of the Russian-Dutch computational Grid (RDG) for distributed high performance simulation [4]. The Virtual Reactor is particularly suitable for "gridification" since it can be decomposed into a number of functional components. Moreover, this application requires large parameter space exploration, which can be efficiently organized on the Grid. For that we rely upon the Nimrod-G parameter sweep middleware [5]. Our work on porting the Virtual Reactor to the Grid started within the CrossGrid EU project [6]. Some results of these efforts were reported in [2]. The RDG Grid is the successor of the CrossGrid as it is based on CrossGrid software and serves as a testbed for the Virtual Reactor.

In this paper we present the results of ongoing work on porting the Virtual Reactor application to the Russian-Dutch Grid. We demonstrate the results of benchmarking

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 530 – 538, 2006.

of one of the parallel solvers on the RDG, indicate the bottleneck of the parallel algorithm used on heterogeneous Grid resources, and propose a generic approach for adaptive workload balancing that takes into account the processors power and interprocessor communications. Further we show the results of implementation of the load balancing algorithm, and conclude the paper with discussion and future plans.

2 Russian-Dutch Grid Testbed Infrastructure

Generally a site within a Grid testbed can be of one of the four types depending on homo- or heterogeneity of underlying resources: homogeneous worker nodes on uniform (I) or non-uniform (II) links; and heterogeneous nodes on uniform (III) or non-uniform (IV) links. Currently the Russian-Dutch Grid testbed consists of five sites with different infrastructures: Amsterdam (3 nodes, 4 processors) and St. Petersburg (4 nodes, 6 processors) – type IV; Novosibirsk (3 processors) – type II; Moscow1 (12 nodes, 24 processors) and Moscow2 (14 processors) – type I.

The Russian-Dutch Grid testbed is built with the CrossGrid middleware [6] based on the LCG-2 distributions, and sustains the interoperability with the CrossGrid testbed. More information on the RDG testbed can be found in [4].

3 Porting of the Virtual Reactor to the Grid

The Virtual Reactor application includes the basic components for reactor geometry design; computational mesh generation; plasma, flow and chemistry simulation; editors of chemical processes and gas properties connected to the corresponding databases; pre- and postprocessors, visualization and archiving modules [2]. This is schematically shown in Fig. 1, where we emphasize the *simulation* components.

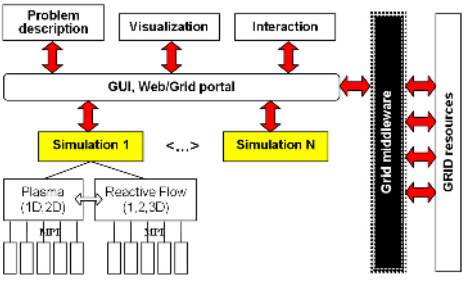

Fig. 1. Functional scheme of the Virtual Reactor application

The aim of our research is to virtualize separate components of the application to run them as services and combine on the Grid. The core components are modules simulating gas flow, chemical reactions and film deposition processes occurring in a PECVD reactor. The details on numerical methods and parallel algorithm employed in the solver are described in [7]. The most important features are the following: for

stability reasons, implicit schemes were applied, thus forcing us to use a sweep-type algorithm for solving equations in every "beam" of computational cells in each spatial direction of the Cartesian mesh. A special parallel algorithm was developed with beams distribution among the processors with communications exploiting a synchronous Master-Slave model [7]. The algorithm was implemented using the MPI message passing interface. In the testbed we use generic MPICH-P4 built binaries that can be executed on all the testbed machines using Globus job submission service.

In order to make a Grid application more efficient it is necessary to perform initial benchmarking of the application modules on Grid resources to reveal existing bottlenecks in the application architecture and possible mismatch with the Grid environment. The results of this benchmarking activity are described in the next section.

4 Benchmarking of the Virtual Reactor on the Grid

The tests performed on the CrossGrid testbed showed that most of the interactive components of the Virtual Reactor do not set restrictions on the environment and can be effectively run on distributed Grid resources. Here we concentrate on benchmarking of the *simulation* modules. Each simulation consists of two basic components: one for plasma simulation and another for reactive flow simulation (see Fig. 1). These two components exchange only a small amount of data every hundred or thousand time steps, therefore the network bandwidth is not critical for their communication. Next, we focus on benchmarking the individual parallel solvers, starting from a 2D reactive flow solver. To measure the dependency of solver performance upon the input data, multiparameter variation (of the computational mesh size, number of simulation time steps, and number of processors) has been applied. We started from a light-weighted problem not simulating the chemical and plasma processes, with a simplified reactor geometry consisting of a single block that allows easy tracking of parameter influence on the execution time. In these tests, a single-block topology was used. The block was subdivided into a (*ncell* x *ncell*) number of computational mesh cells, with *ncell* running from 40 to 100, thus forming 1600–10000 cells.

4.1 Benchmark Results for Homogeneous Sites

The measurements were carried out on the five Grid sites within the RDG testbed. Figures 2 and 3 demonstrate the total execution time and speedup of the parallel solver for different computational mesh sizes on the Moscow site of Type I (homogeneous cluster). The speedup decreases for larger problem size (with more computational mesh cells). This fact indicates that the ratio of the interprocess communications bandwidth to the processor performance is not high enough for the light-weighted problems with relatively small number of operations per computational cell. To get insight into the computation/communication relations within the solver we measured the communication time for different types of the problem and for varying mesh sizes. We observed that for light-weighted problems interprocess communication time grows super-linearly with increasing the mesh size, although the amount of data transferred is linearly proportional to the number of mesh cells. This behaviour shall be studied further by extensive benchmarking of the network links. Some

peculiarities in the communication time can be seen in Fig. 4: (1) The communication time grows non-monotonically with the number of processors, but drops down on every processor with an even number; and (2) The time of MPI Receive calls is an order of magnitude higher for the first few processors.

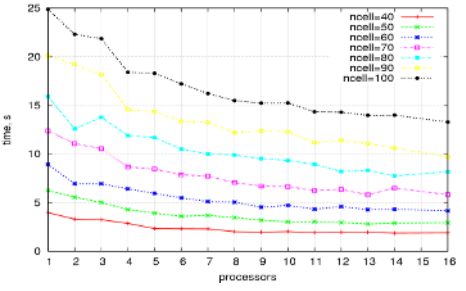

Fig. 2. Dependency of the total execution time on the number of processors

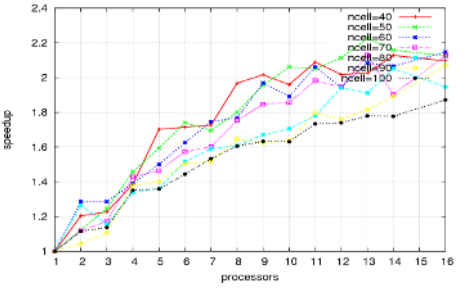

Fig. 3. Speedup achieved by the solver for different computational mesh sizes

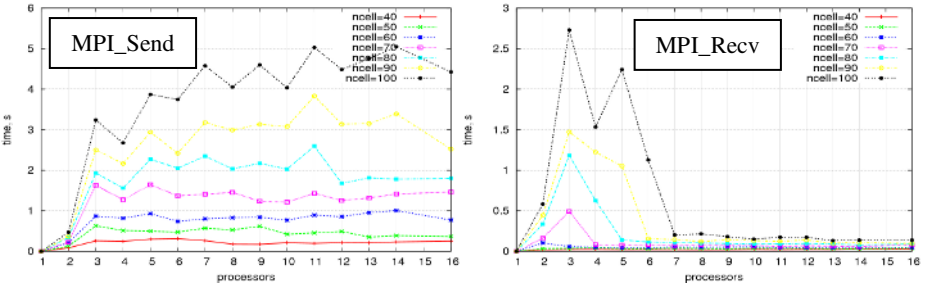

Fig. 4. Dependency of the communication time on the number of processors for different computational mesh sizes

These results reflect the topology, network and nodes features of the tested Grid site:

(1) Since the site consists of two-processor nodes, the network channels work more efficiently for data transfers between the Master and a Slave processor if a connection was already established with another Slave processor on the same node. This can be explained by implementation of the MPI library which saves network resources while opening and maintaining connections for concurrent processes on the same node.

(2) The "peaks" of the MPI Receive time for the first few processors (see Fig. 4 right) are caused by the constraints on the portions of data that could be accommodated at once. The constraining factors could be the network bandwidth distribution, the processor cache size, the memory available on the node, or a combination of these factors.

In Figure 5 the total execution time is presented along with the contributions of calculation and communication. For a smaller mesh (Fig. 5 left), the communication time makes a relatively small contribution into the total execution time even for a large number of processors involved. For a larger mesh (Fig. 5 right), communication makes up to 30% of the execution time. This result confirms that the network bandwidth is not sufficient for this type of problem (see also explanations to Fig. 3).

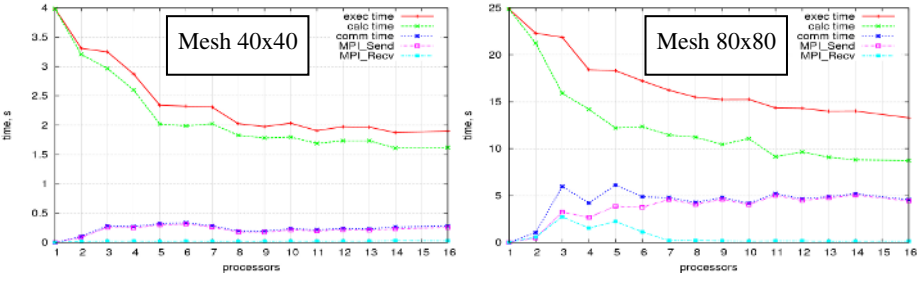

Fig. 5. Total execution time and contributions of the calculation and communication depending on the number of processors for different computational mesh sizes

All the previous results were shown for a light-weighted problem not simulating the chemical processes. Figure 6 demonstrates the influence of taking into account the chemistry on the solver performance. Here we plot the ratio of computation to communication time for different mesh sizes. The higher this ratio, the better the parallel efficiency is. One can see that for the chemistry-enabled simulations the ratio behaviour does not depend on the mesh size in the tested range of parameters, while this behaviour for the chemistry-disabled simulations significantly differs for small and large mesh sizes. For a small mesh size, the ratio stays decently high, and for 6 processors and more it reaches the level of the chemistry-enabled simulations. For a larger mesh, the computation/communication ratio for the no-chemistry simulations is very low, thus diminishing the overall parallel efficiency.

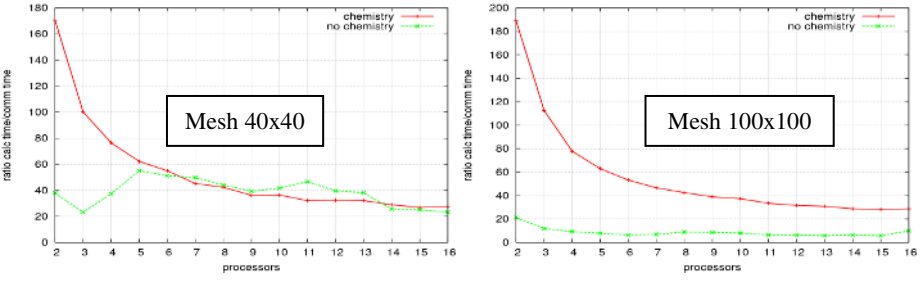

Fig. 6. The ratio of computation to communication time for chemistry-enabled and chemistry-disabled simulations

4.2 Heterogeneous Sites: Load Balancing and Benchmarking

The parallel algorithm was originally developed for homogeneous computer clusters with equal processor power, memory and interprocessor communication bandwidth. In case of submitting equal portions of a parallel job to the nodes with different performance, all the fast processors have to wait at the barrier synchronization point till the slowest ones get the job done. The same problem occurs if the network connection from the Master processor to some of the Slave processors is much slower than to the others. As we have shown in the previous section, for communication-bound simulations (chemistry-disabled simulation with large computational meshes), the communication time on low-bandwidth networks is of the order of calculation time, therefore the heterogeneity of the interprocessor communication links is a hindrance as considerable as the diversity of the processor power. One of the natural ways to adapt the solver to the heterogeneous Grid resources is to distribute the portions of job among the processors proportionally to the processor performance and network connections.

The issue of load balancing in Grid environment 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 [11]. Less often load balancing code is included into the application source code to improve performance in specific cases [12]. Some research projects concern load balancing techniques that use source code transformations to improve the execution of the application [10]. We employ the application-centric approach where the balancing decisions are taken by the application itself, however the algorithm and the code estimating available resources and suggesting the optimal load balancing of a parallel job is generic and can be employed in any parallel application to be executed on heterogeneous resources.

We developed a mechanism for estimating the "weight" of a processor according to its processing power and network connection to the Master processor. The values of the weights determine how much work will be executed by each processor. Similar approach was used in [9] for heterogeneous computer clusters, however the same tools can not be used in Grid environments, where the weights shall be calculated every time the solver is started on a new set of dynamically assigned processors.

The link bandwidth between the Master and Slave processors is estimated using MPI_Send transfers of a predefined data block (MPI buffer size is 10^6 of MPI_DOUBLEs) in the beginning of the solver execution, after the resources were allocated. The CPU power was obtained by a function from the *perfsuite* library [8]. The node weights were calculated as follows:

$$weight_i = c\_CPU \cdot CPUweight_i + c\_NET \cdot NETweight_i;$$

$$CPUweight_i = CPU_i \Big/ \sum_j CPU_j;\ NETweight_i = MPI\_SendTime_i^{-1} \Big/ \sum_j MPI\_SendTime_j^{-1}$$

The main factor in distributing the load was the processor power (c_CPU=1.0), and to take into account the influence of the network connections we introduced the *cn* parameter (cn=c_NET=[0.0 … 2.0]). The value of $cn=0$ means that the diversity of communication links is not taken into account, and $cn=1$ means that influence of the links bandwidth is considered as important as the processor power.

To illustrate the approach described above, we present the results obtained for a light-weighted problem of chemistry-disabled simulation of a real reactor geometry with 10678 cells on St. Petersburg Grid site, which is heterogeneous in both CPU power and network connections of the nodes (Type IV). There are two 3 GHz nodes and two dual 450 MHz nodes. One of the dual nodes is placed in a separate network segment with 10 times lower bandwidth (10 Mbit/s against 100 Mbit/s in the main segment). Figures 7 and 8 illustrate the speedup achieved by applying the workload balancing technique with different values of the network influence parameter *cn*. The speedup was calculated as the ratio of the execution time without load balancing to that applying the balancing algorithm. The most noticeable speedup is observed for 3 and 4 processors in the considered resource configuration. This is explained by the fact that in these cases the solver was run on equal-performance processors connected with the network links of different bandwidth (Type II infrastructure). Figure 8 shows that in this case the speedup grows linearly with the increase of the network influence coefficient *cn*. The slowdown observed on 2 processors with our balancing algorithm is discussed in the next section (item 4).

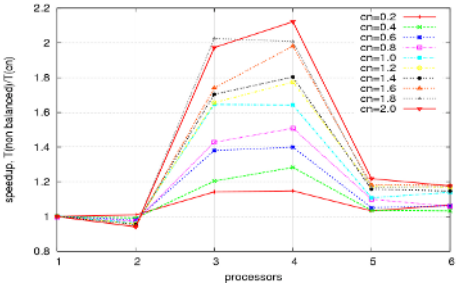

Fig. 7. Speedup of the load-balanced version compared to the non-balanced solver

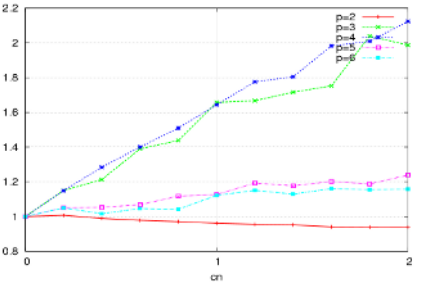

Fig. 8. Dependency of speedup on the network influence parameter *cn*

5 Discussion

Analysis of the results achieved with the workload balancing algorithm suggests that the following issues shall be addressed in order to optimize the balancing technique:

1. The type of resources assigned to the parallel solver shall play a role in choosing the *c_CPU* and *cn* coefficients: for the Type II resources (homogeneous worker nodes with heterogeneous interconnections) *c_CPU* shall be set to 0.0, as only the network heterogeneity shall be compensated by load balancing. For the Type III (heterogeneous worker nodes with uniform interconnections) *cn* shall be 0.0; and only for the Type IV (heterogeneous nodes with heterogeneous interconnections) both *c_CPU* and *cn* parameters shall be adjusted optimally. This can be done automatically by enriching the weighting algorithm with a function analyzing the CPU and network responses of the nodes participating in the simulation.
2. To choose optimal values of the network weighting coefficient *cn* for the Type IV resources, for each particular problem to be simulated we shall analyze the ratio of

the computation to communication time. This can be also theoretically estimated as a function of the CPU power to the network bandwidth ratio.

3. To measure the interprocess communication rate, we sent a fixed amount of data from the Master to each Slave processor. However the response of the communication channels to increasing amount of data is not scaled linearly. For the slower networks this tendency is even more pronounced. This brings us to a conclusion that the amount of data sent to measure the links performance shall be close to the amount really transferred within the solver for every particular problem, mesh size, geometry and number of processors in a parallel job.
4. To calculate the weight of the Master processor, we used a fixed artificial value of the *MPI_SendTime* for this processor. Often it was much lower than the values of measured connections to the Slaves. It caused assigning excessive load for the Master processor, which slowed down the simulation because the Master shall perform co-ordination and execute some additional functions. A simple solution would be to dynamically set this parameter to the value of a Slave processor with the fastest link to the Master.

7 Conclusions and Future Work

In this paper we addressed the issue of porting a cluster-based problem solving environment to the Grid using as a test case a distributed parallel Virtual Reactor on the Russian-Dutch Grid testbed. We illustrated the performance issues that occur while porting computational components from homogeneous cluster environment to the Grid. To adapt the parallel programs to the heterogeneous Grid resources, we developed a generic workload balancing technique that takes into account specific parameters of the Grid resources dynamically assigned to a parallel job. We plan to enhance the algorithm and create a library for automatic load balancing on the Grid..

Benchmarking the components of a distributed application allowed us to evaluate their performance dependencies. Applying these results to improve Grid resource management for the Virtual Reactor is another direction of our future work.

Acknowledgments. The authors would like to thank Irina Shoshmina, Breanndán Ó Nualláin and the RDG Grid deployment team for their assistance. The research was conducted with financial support from the NWO/RFBR projects 047.016.007 and 047.016.018, and from the Virtual Laboratory for e-Science project (www.vl-e.nl).

References

1. www.cfdrc.com, www.fluent.com, www.semitech.us, www.softimpact.ru
2. V.V. Krzhizhanovskaya et al. *Grid -based Simulation of Industrial Thin-Film Production*. Simulation: Transactions of the Society for Modeling and Simulation International, V. 81, No. 1, pp. 77-85 (2005)
3. V.V. Krzhizhanovskaya et al. *A 3D Virtual Reactor for Simulation of Silicon-Based Film Production.* Proceedings of the ASME/JSME PVP Conference. ASME PVP-Vol. 491-2, pp. 59-68, PVP2004-3120 (2004)
4. Project "High performance simulation on the Grid" http://grid.csa.ru/
5. Nimrod-G: http://www.csse.monash.edu.au/~davida/nimrod/

6. CrossGrid EU Science project: http://www.eu-CrossGrid.org
7. V.V. Krzhizhanovskaya et al. *Distributed Simulation of Silicon-Based Film Growth.* Proceedings of the 4th PPAM conference, LNCS, V. 2328, pp. 879-888. Springer-Verlag 2002
8. R. Kufrin. *PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux.* 6th International Conference on Linux Clusters. Chapel Hill, NC. (2005)
9. J.D. Teresco et al. *Resource-Aware Scientific Computation on a Heterogeneous Cluster*. Computing in Science & Engineering, V. 7, N 2, pp. 40-50, 2005
10. R. David et al. *Source Code Transformations Strategies to Load-Balance Grid Applications*. LNCS vol. 2536, pp. 82-87, Springer-Verlag, 2002
11. A. Barak et al. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, LNCS, vol. 672, Springer-Verlag, 1993
12. G. Shao et al. *Master/Slave Computing on the Grid.* Proceedings of Heterogeneous Computing Workshop, pp 3-16, IEEE Computer Society (2000)