Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 143–150. http://www.scpe.org
ISSN 1895-1767

PERFORMANCE MEASUREMENTS OF COMPUTING NETWORKS

IGOR ROZMAN*, MARJAN ŠTERK*, JAKA MOČNIK†, BORUT ROBIČ‡, AND ROMAN TROBEC§

Abstract. This paper presents measured communication and computational performance results on an ad-hoc computing network composed of several geographically distributed nodes. Communication speed was measured for three basic communication patterns commonly used in parallel and distributed applications: point-to-point, bidirectional ring, and all-to-all communication. For that purpose simple test programs were made, based on the MPI library. The parallel execution time of two real-life computationally intensive parallel applications is also presented. Results show that only the applications with low communication requirements could be beneficially executed on computing networks.

Key words: computing network, ad-hoc networks, grid, Internet, communication speed, MPI, performance evaluation

1. Introduction. Scientific and industrial applications require ever more computing capacity, and for the majority of institutions and enterprises it is hard to follow this trend by buying more and more powerful computers [1]. On the other hand, there is an increasing number of networked computers, which are only partially utilized. Grid technology has been developed and proposed in recent years [2], which provides the infrastructure required to harness the combined computing power of these machines for computationally intensive tasks at a considerably lower price than special-purpose supercomputers. Grid technology is a standardized solution for distributed computing on a larger number of geographically separated computers.

A Grid could represents a group of computers, which are originally installed for some other general tasks, like for office workstations, but with unexploited resources made available for executing Grid computing tasks. These computers are generally not located in the same place and can have very heterogeneous resources in terms of processor type, storage capacity, available network bandwidth, operating system type, etc. Also, their availability can change with time, depending on their primary use. A Grid differs from a computer cluster in the way of resource management. While centralized in a cluster, it is distributed in a Grid, and far more complicated with every node having its own resource manager. The user should not care about details of resource allocation. This should be done by the Grid middleware transparently to the user. Resources must first be discovered and their access and use granted, which includes user and computer authentication, according to assigned rights and priorities. Computational nodes must be assigned according to their processing and communication capacities. Communication between nodes is usually performed over public networks and must be made secure. Broad public usage of computational and storage resources would also require charging according to use in the way electric power, phone service, Internet access or other public services are charged nowadays [3, 4].

Users of the Grid computing power should not bother about where the computing is executed. Instead, a suitable Grid computing tool should provide for fast, secure and repeatable execution of computing tasks, enabling the use of computer resources available in the network. Grids, which are specialized for the execution of computationally intensive tasks, are called computational Grids. The most important resources here are processors, and such a network would consist of a large number of computers with large computational power or even of supercomputers and computer clusters. Tasks can run on a single computer, on a number of them, or they can even move between networked computers during their execution. However, it was experienced that the efficiency of computational Grids is not always satisfactory, particularly if the communication requirements are high.

It is known that only scalable algorithms can be implemented efficiently on massively parallel machines [5]. Typically, such algorithms need only a small amount of communication between processors [6]. By increasing the number of processors, the computational load on each processor decreases. There is a limit beyond which further increase in the number of processors is not profitable, because the computational load becomes too small when compared to the communication load. The latter also depends on the characteristic communication patterns of the algorithm, in particular, the amount of data to be transferred and the frequency of communication needed during calculation.

*Jozef Stefan Institute, Jamova 39, Ljubljana, Slovenia.
†XLAB d.o.o., Teslova 30, Ljubljana, Slovenia.
‡Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, Ljubljana, Slovenia.
§Jozef Stefan Institute, Jamova 39, Ljubljana, Slovenia (roman.trobec@ijs.si).

The goal of our work was to build an ad-hoc computing network, inspired by Grid technologies, and to investigate the impact of communication performance on computation performance. Nodes of our computing network are workstations and home computers at three different locations: Ljubljana, Salzburg and Zagreb. The communication speeds in different communication patterns were measured with simple test programs based on the MPI library [7, 8] in order to prevent the overhead of a Grid middleware [9]. Additionally, the computational performance is checked by running two real-life computation-intensive parallel applications.

The rest of the paper is organized as follows. In the next section, the testing methods and the tested ad-hoc computing network are described. In Section 3, the measured communication results for asynchronous point-to-point communication, bidirectional communication in a ring topology, and all-to-all communication are presented. Section 4 is devoted to the measured execution times of two real-life applications. The paper concludes with the summary of obtained results and with some directions for further work.

2. Testing environment.

2.1. Terminology. The *execution time* of a parallel application is the sum of *calculation time*, *idle time* and *communication time*. The calculation time is spent for data transfer between internal memory and CPU and their manipulation in the processing unit. The idle time is a period during which processing unit is available, but is not in use, for example due to the lack of data from other resources. The communication time consists of the time needed to transfer the required application messages, and is for each message the sum of the *set-up time* and the *transfer time*. The set-up time dominates for short messages, while the transfer time is more important by longer messages. The *bandwidth* is the message size divided by its communication time.

The following standard abbreviations are used: bit (b), byte (B), second (s), minutes (min), giga (G), mega (M) and kilo (k).

2.2. Measuring Methodology. Communication bandwidth was measured as *(message length/round-trip time)* for the following lengths of messages: 80 B, 320 B, 640 B, 16 kB and 64 kB. Each measurement was performed eight times, every 30 minutes, in a period of 24 hours. Two measurements with the lowest communication bandwidth were eliminated in order to exclude various system activities, which could have a significant impact on measured results. Therefore the shown communication bandwidth is given as an average of six measurements.

2.3. Computing Network. We set up a heterogeneous computing network composed of distributed nodes on the following distant locations:

- computing node at Jozef Stefan Institute (PC_IJS), located in Ljubljana,
- workstation at Jozef Stefan Institute (Zelda), located in Ljubljana,
- home computer (PC_Lj), located in Ljubljana,
- computing node at University of Salzburg (PC_Slz), located in Salzburg,
- workstation at Rudjer Boskovic Institute (PC_Zg), located in Zagreb.

The computer PC_IJS is locally connected to Zelda, via Fast Ethernet (LAN) and further to the wide area network (WAN). All remote nodes PC_Lj, PC_Slz and PC_Zg are also connected to the WAN (see Figure 2.1). Some technical details of our network nodes are given in Table 2.1.

The Grid infrastructural services (authentication, authorization, data transfer, job management etc.) on our network are provided by the Globus Toolkit [10], a widely used open source middleware for building Grid systems. However, all these services contribute to a substantial run-time overhead, therefore we chose to run our parallel test applications that use solely the MPI communication library MPICH v1.2.7 [11] with the ch_p4 device, which supports communication between workstations in heterogeneous TCP/IP networks. It enables practical, portable, efficient and simple message communication among processes. MPI is usually applied in parallel computers and clusters, if efficient resource management is important, while the safety issues are not imperative.

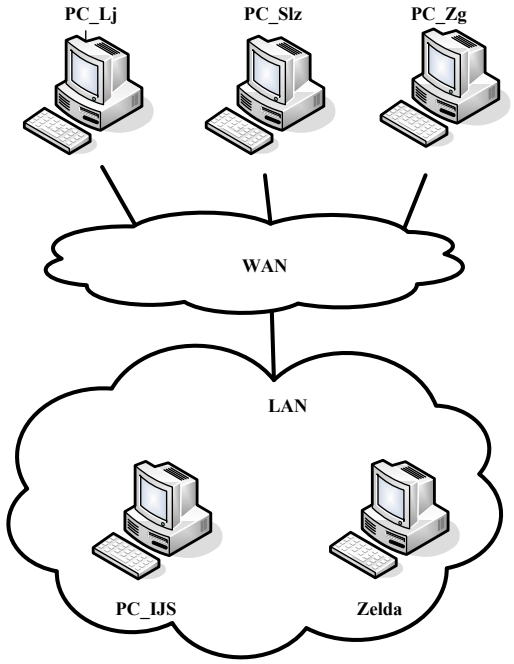


FIG. 2.1. *The topology of the tested ad-hoc computing network.*

TABLE 2.1
Hardware configuration of tested computing network nodes.

Computer	OS	CPU	RAM
PC_IJS	Fedora 6-2.6.19	Athlon XP 2000+	512 MB
PC_Lj	Fedora 6-2.6.19	Celeron 1700	512 MB
PC_Slz	Debian 3.3.5-2.6.9	Pentium 4 3.0	512 MB
Zelda	Fedora 2-2.6.10	Opteron 1.8	2 GB
PC_Zg	Debian 3.3.5-2.6.8	Xeon 2.8	2 GB

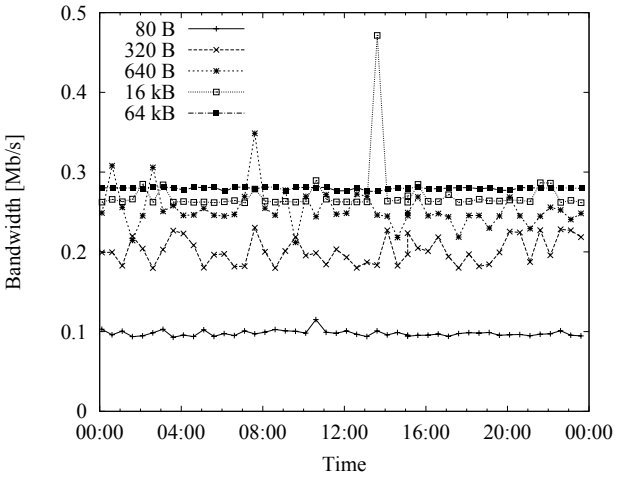
3. Results of Communication Performance.

3.1. Point-to-point Communication. The simplest communication action between a pair of processes, one side sending and the other receiving, is called point-to-point communication. Its bandwidth was measured with a test program, which uses asynchronous MPI functions `MPI_Isend` and `MPI_Irecv`. The program implements "ping-pong" by sending/receiving messages of various lengths between two nodes and measures the round-trip time.

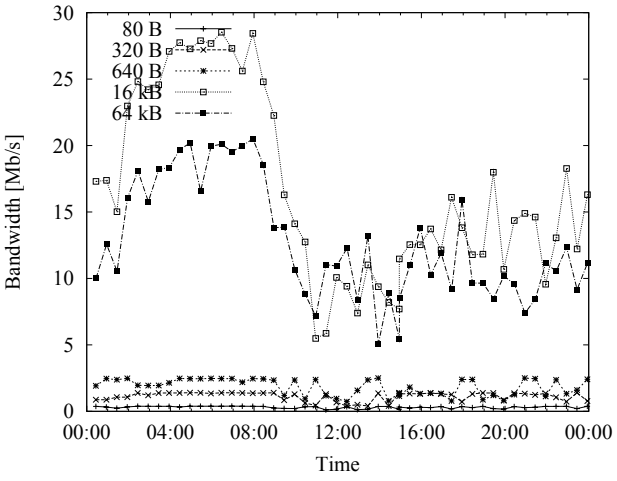
Measurements of asynchronous communication bandwidth between nodes of our test network were conducted in the period from March 13 to 21, 2007, using the methodology described in Subsection 2.2. The results are shown in Figure 3.1. Note that different scales are used on y-axes. On all graphs the bandwidth is low for short messages, which is due to the set-up time, and rises with the message length.

The slowest connection is between PC_IJS and PC_Lj (Figure 3.1(a)), because the node PC_Lj has limited upload speed of 0.25 Mb/s by Internet service provider. In average the highest bandwidth of 0.29 Mb/s is reached at 64 kB messages. In contrast, the highest bandwidth on most of other connections is reached at 16 kB messages. According to our previous experiments [12] this could be explained with the limited sizes of system and network buffers. Longer messages are fragmented into more packets, further, some of them can be lost and therefore resubmit, which could result in significant lowering of the communication bandwidth.

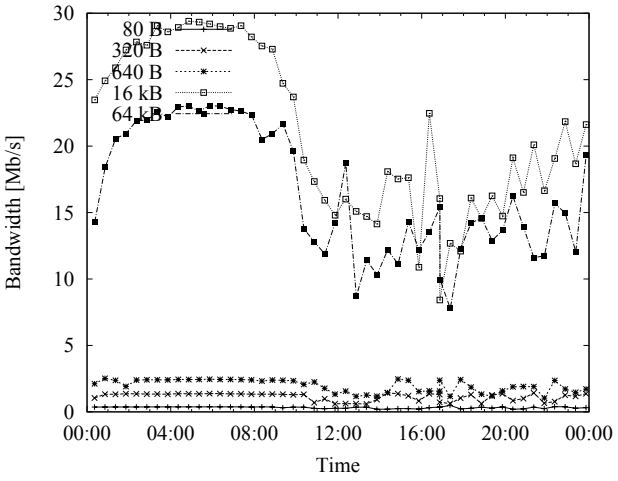
The communication bandwidth of the connection between Ljubljana and Zagreb (Figures 3.1(b), 3.1(c)) is the fastest among all Internet connections, however, it is very variable. It reaches the minimum at 5 Mb/s and the maximum at 28 Mb/s for 16 kB messages. Again, longest messages reached lower bandwidth as expected. Two measurements conducted in two different days show the same pattern of the bandwidth, which starts to



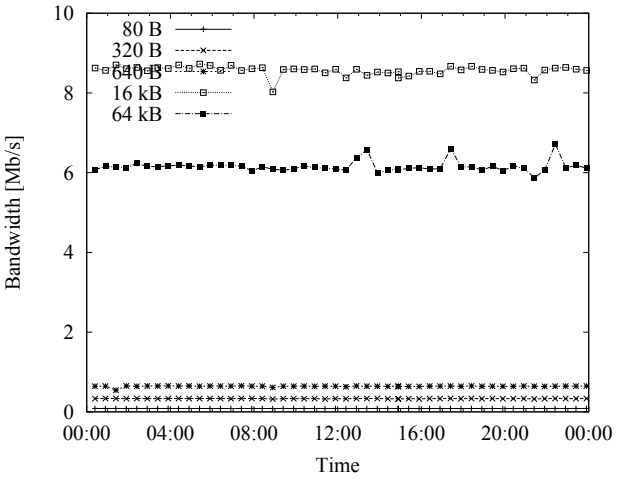
(a) PC_IJS and PC_Lj on March 13, 2007.



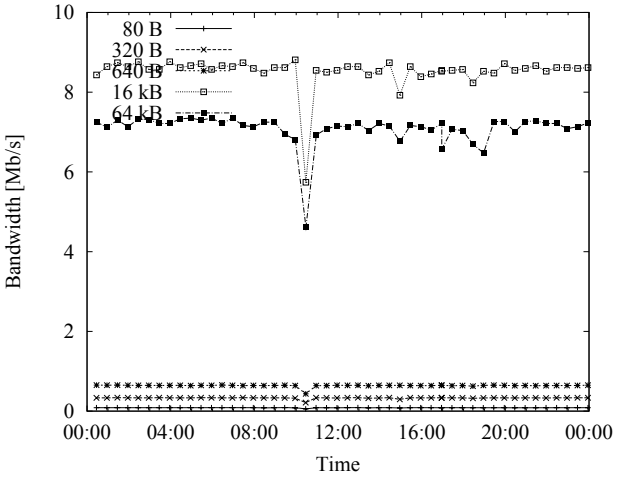
(b) PC_IJS and PC_Zg on March 13, 2007.



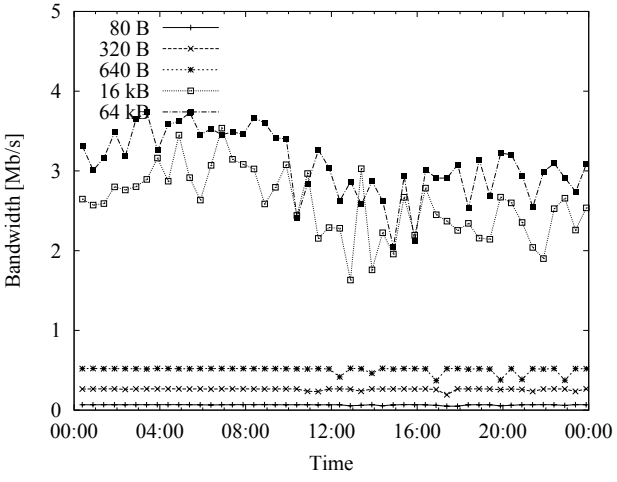
(c) Zelda and PC_Zg on March 15, 2007.



(d) PC_IJS and PC_Slz on March 13, 2007.



(e) Zelda and PC_Slz on March 15, 2007.



(f) PC_Zg and PC_Slz on March 21, 2007.

FIG. 3.1. *The bandwidth of asynchronous point-to-point communication between nodes of test network.*

increase after 1. a.m. and decrease after 8 a.m. The reason is probably the daily usage of Internet which also increases the load on the connection between Ljubljana and Zagreb.

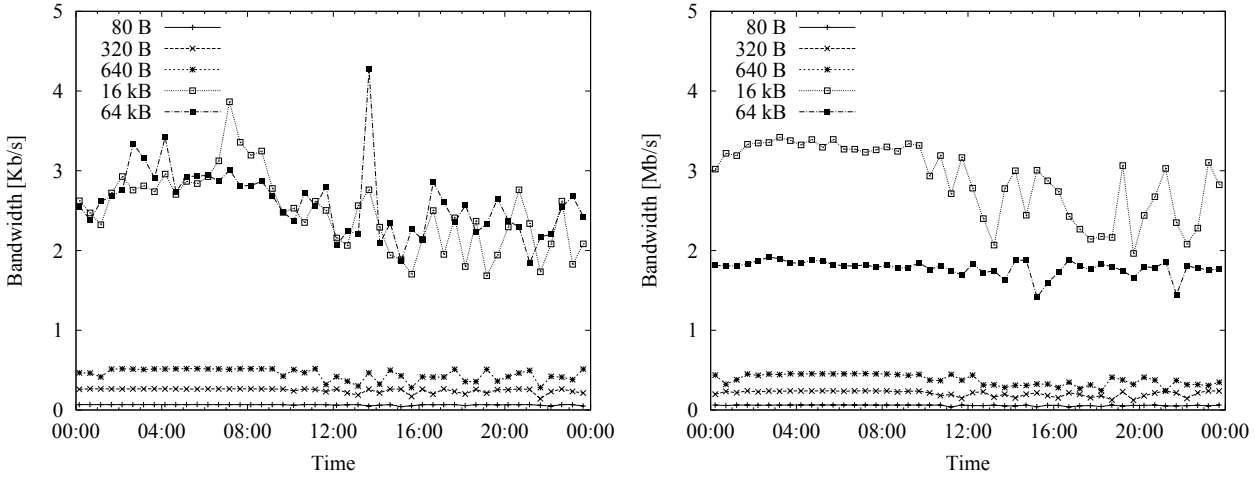
The connection between Ljubljana and PC_Slz has quite a constant bandwidth of 9 Mb/s in case of 16 kB messages (Figures 3.1(d), 3.1(e)) with no significant difference during the day and night period. The bandwidth of the connection between PC_Slz and PC_Zg (Figure 3.1(f)) is not as constant, with highest values in the interval between 2 Mb/s and 3.5 MB/s for 64 kB messages. Also, the bandwidth is a little higher between 1 a.m. to 8 a.m., similar to the previously described connections from Figures 3.1(b), 3.1(c).

We may conclude that there is no uniform pattern for communication bandwidth because it is limited by physical connection, daily usage, or by limitations posed from connection providers.

3.2. Bidirectional Communication in Ring. In parallel numerical simulations, the problem domain is often decomposed in subdomains, e.g., slices in 1-D decomposition, which are distributed among processors [13]. In every simulation step, an exchange of boundary values between neighboring processors is needed. In the case of a linear array or ring topology, each node thus sends the same amount of data to its left and right neighbor, and also receives the same amount from both, all at the same time. Ring topologies are vulnerable to node and link failures. A single crashed process or communication channel makes it impossible for some other processes to communicate data further along the ring. Thus, the ring topology and algorithms based on it are not suitable for the Internet environment.

The test program shifts data with messages of the same length applying non-blocking MPI functions `MPI_Isend` and `MPI_Irecv`. The bandwidth is now the length of a single message divided by the time needed for message exchange with left and right neighbors, i. e. two sent and two received messages in each node of the ring.

The performance of bidirectional communication among PC_IJS, PC_Slz, PC_Zg and Zelda in the ring topology is shown in Figure 3.2(a). From the point-to-point measurements we know that the connection between PC_Lj and WAN is more than ten times slower than other connections, therefore we eliminated it from the further measurements of bidirectional and all-to-all communications. The communication bandwidth remains limited, again with by the slowest link, which is in this case the link between Zagreb and Salzburg.



(a) Bidirectional communication in ring.

(b) All-to-all communication.

FIG. 3.2. *Collective communication among PC_IJS, PC_Slz, PC_Zg and Zelda on March 21, 2007.*

3.3. All-to-all Communication. In all-to-all communication, each node sends a distinct message of size m to every other node. Many parallel applications, for example, molecular dynamics [14] are based on such a collective communication. All-to-all communication test was implemented by the MPI function `MPI_Allgather`, which, on all processes, gathers data from all other processes and distributes local data to all other processes. The bandwidth represents the length of a single message divided by the time needed for execution of the function `MPI_Allgather`.

The performance of all-to-all communication for PC_IJS, PC_Slz, PC_Zg and Zelda is shown in Figure 3.2(b). The communication is limited by the slowest link (Zagreb-Salzburg). Because of relatively low bandwidth we did not detect any problems that could be attributed to switch or router congestion, which would slow down the collective communication speed in clusters.

4. Results of Computational Performance. The execution times of two applications were measured to evaluate computational performance: simulation of a human knee cooled after surgery by gel packs [15], and searching for the optimal solution to the Rubik cube [16].

4.1. Description of Applications. In the *knee cooling simulation*, heat transfer is modeled by a linear diffusion equation and solved by the finite difference method and explicit time stepping. This implies a relatively small amount of calculation and an exchange of boundary values for each time-step. The problem domain was 3-D and composed of several million points with substantial amount of communication.

The parallel version of the simulation was implemented using 1-D domain decomposition—domain slicing [17]. Load balancing is easily achieved in a cluster architecture with equal nodes and communication speed. The calculation time is inversely proportional to the number of processors, while the communication time remains constant. It can be expected that the speedup will behave as a monotonously increasing function of number of processors, soon approaching some constant value limited by the communication time. Performance results in a cluster environment confirmed these expectations and were already presented in [12]. However, the parallel execution time on a computing network will be saturated much earlier because of slow communication links.

The *Rubik cube solver* attempts to solve (order) one initial setup of the Rubik cube in an optimal (minimal) number of moves. To this end, it uses a highly-optimized IDA* (Iterative Deepening A*) depth-first search using a memory-based heuristics based on partial solutions of the Rubik cube [18, 16]. The algorithm searches the problem space which is represented by a search tree with the initial cube setup at the root. Each parent-child link in the tree represents one (valid) move during solving the Rubik cube. Each move is a rotation of one layer of the cube. At the first level (link stemming from the root), all 18 different moves are possible, giving the root a branching factor of 18. At the lower levels, some links are pruned from the tree *a priori* as they represent redundant moves that do not lead to an optimal solution: as an example, consider the move that reverses the previous one, which is clearly redundant. Therefore, an average number of children (branching factor) for nodes at all the lower levels is 13.3 [16].

Note that simple depth-first search does not find the optimal solution, while full A* search is not appropriate because of its exponential memory requirements. Iterative deepening, on the other hand, first executes a depth-first search with maximum depth limited to 1, followed by 2 etc, ensuring the optimal solution is found. A typical Rubik cube can be solved, with the described algorithm, in 16-20 moves and thus finding the optimal solution requires a search over about 10^9–10^{14} nodes. In our test we use a single initial setup that is known to have an optimal solution consisting of 16 moves and the times to discover that it can not be solved in 15 moves is measured.

The parallel implementation of the solver is a simple master-slave type of computation. The master node first searches the top two levels of the search tree and then distributes the next 243 (as the average branching factor on the second level is 13.5, see above) subtrees of the problem space to the slave machines. Each slave executes the job, i. e. searches the subtree to the current maximum depth, and either returns the solution or replies that no solution can be found up to the current maximum depth. The communication in each IDA* iteration consists solely of distributing the jobs to the slaves and collecting the results.

4.2. Measured Execution Time. Both applications were run first on each single node. Except on the node PC_Slz, there was enough internal memory to prevent swapping with the hard disk. Then the applications were run in parallel on various sets of computing nodes. Difficulties appeared when we tried to use the node Zelda with other nodes. We found out that the application programs do not strictly use complex MPI data types which results in incompatibilities between nodes. Zelda runs on 64-bit operating system while all other nodes run on 32-bit operating systems. Consequently, the node Zelda was not a part of measurements in our computing network. The execution time of both test applications is presented in Table 4.1. Speed-up is not calculated because of heterogeneity in computation and communication performances.

From the separate runs of the *simulation of knee cooling* on each node we see that the execution time was quite similar, with the shortest on the node Zelda and the longest on the node PC_Zg. The node PC_Slz reported "Out of memory".

TABLE 4.1
Execution time of the knee cooling simulation for 1 *s of the real time, and of the Rubik cube solution.*

Computing nodes	Execution time Knee cooling	Execution time Rubik cube
Zelda	11.6 s	4.63 min
PC_IJS	12.2 s	9.08 min
PC_Zg	13.8 s	6.55 min
PC_Slz	Out of memory	4.48 min
PC_Lj	17.0 s	13.5 min
Zelda (use of both CPUs)	6.4 s	2.55 min
PC_IJS, PC_Zg	11.0 s	3.92 min
PC_IJS, PC_Zg and PC_Slz	22.9 s	2.27 min
PC_IJS, PC_Zg, PC_Slz, PC_Lj	223.4 s	1.97 min

Next the simulation was run on Zelda with the use of both CPUs. The execution time was almost halved compared to a single CPU, as expected. This is a consequence of efficient usage of two processors concurrently, and fast communication speed of internal bus between them, which did not slow down the execution time. Then we ran the simulation on nodes PC_Zg and PC_IJS, because they are connected with the fastest connection as we know from our previous measurements of communication performance. The execution time was about 20% shorter compared to the average execution time of both nodes, if they run as single nodes. Further, the simulation was run on the nodes: PC_IJS, PC_Zg and PC_Slz. The execution time was longer than on the slowest single node. This happened because the communication bandwidth between PC_Slz and other nodes is too low too support the required data transfer. The same phenomena is visible even more if the node PCLj is joined to the computing pool. The execution time becomes excessively long as a consequence of increased communication time of PC_Lj.

Data decomposition was always even for all nodes, which works well if the nodes have similar computation and communication ability as, e.g. on computing clusters. This was evidently not the case in our last experiment, therefore some method for load balancing has to be implemented to achieve more efficient execution on computing networks with substantial heterogeneity in node performance [19, 20, 21].

The results show that the *Rubik cube solver* scales well with the number of nodes, regardless of either their communication bandwidth or their computational capabilities for two reasons.

The first reason is that the execution time to communication time ratio is very high. This is first due to the fact that both input and output of each job are in the order of magnitude of 100 B with the tested implementation. Then, due to the master-slave paradigm and a small number of jobs, only 243 master to slave round-trips are required. This behavior results in only minimal impact of the communication bandwidth.

The second reason for good scaling is an adaptive load distribution strategy that first distributes one job to each slave node, and assigns a new job to any slave only after receiving the results of its previous job. In this manner, the faster slaves process more jobs, and the impact of heterogeneous nodes with different computational capabilities is much lesser when compared to the knee cooling simulation where a fixed load distribution proves inappropriate. Once again, due to very small size of the input, the fact that we are not prestaging input data to slaves does not lessen the performance.

We can conclude that such applications, characterized with high execution to communication ratio and with the capability of adaptive load distribution, prove more suited to network computing.

5. Conclusions. The obtained results of communication bandwidth in computing network show that it is impossible to predict in advance, and for a longer period, expected communication bandwidth of a particular connection. Some slow connections can appear, which represent communication bottlenecks. Therefore, for efficient distributed computationally intensive applications some additional mechanism is needed for on-line monitoring of the available communication resources.

Grids and other computing networks are generally composed of heterogeneous computers not tailored for intensive communication and prone to frequent faults. Any communication or computer failure can deteriorate the whole application if implemented without fault-tolerance measures. Computing networks are limited to simple topologies because each node has typically just a single network interface.

In parallel applications, which require many short messages, very distant nodes can represent a serious drawback, because of extra time needed for messages to reach these distant nodes, even if travelling with the speed of light. Travelling time can be much larger than the node set-up time, however, it must be added to the node set-up time.

The current results obtained and the mentioned drawbacks of computing networks indicate that there will always be a need to distinguish between distributed and parallel systems. For most scientific applications, except those with almost no communication, parallel systems will be preferred as they have balanced computation and communication speed. Computational networks or Grids need on-line monitoring of computation and communication resources and some mechanism for load balancing in order to be useful for parallel applications. In further work, we will focus on required approaches needed to increase the competitiveness of computing networks.

Acknowledgments. The authors acknowledge financial support from the state budget by the Slovenian Research Agency under grant P2-0095, and would also like to thank University of Salzburg and Rudjer Boskovic Institute for their support in setting-up our testing environment.

REFERENCES

[1] World Community Grid on WEB, `http://www.worldcommunitygrid.org/index.jsp`
[2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, 1999.
[3] C. S. Yeo, M. D. de Assuncao, J. Yu, A. Sulistio, S. Venugopal, M. Placek and R. Buyya, *Utility Computing and Global Grids*, Technical Report, University of Melbourne, GRIDS-TR-2006-7, 2006.
[4] M. A. Rappa, *The utility business model and the future of computing services,* IBM Systems Journal 43 (2004), pp. 32–42.
[5] G. Golub and J. M. Ortega, *Scientific Computing, an Introduction with Parallel Computing*, Academic Press Inc., San Diego, 1993.
[6] V. Avbelj, M. Šterk and R. Trobec, *Lead Selection in Body Surface Electrocardiography by Exhaustive Search Optimisation*, in *Parallel Numerics '02 theory and applications*, Jozef Stefan Institute and Salzburg University, Ljubljana, 2002, pp. 201–207.
[7] W. Gropp, E. Lusk and A. Skjellum *Using MPI: Portable Parallel Programming with the Message-passing Interface*, MITPress, 1999.
[8] MPI on WEB, `http://www-unix.mcs.anl.gov/mpi/`
[9] R. Prodan and T. Fahringer, *Overhead analysis of scientific workflows in grid environments*, IEEE Transactions on Parallel and Distributed Systems 19 (2008), pp. 378-393.
[10] Globus Alliance on WEB: `http://www.globus.org/`
[11] MPICH on WEB, `http://www-unix.mcs.anl.gov/mpi/mpich/`
[12] I. Rozman, M. Šterk, R. Trobec, *Communication performance of LAM/MPI and MPICH on a Linux cluster*, Parallel Process. Lett., 16 (2006), pp. 323–334.
[13] M. Šterk, R. Trobec, *Biomedical simulation of heat transfer in a human heart*, J. chem. inf. mod., 45 (2005), pp. 1558–1563.
[14] U. Borštnik, M. Hodošček, D. Janežič, *Improving the performance of molecular dynamics simulations on parallel clusters*, J. chem. inf. comput. sci., 44 (2004), pp. 359–364.
[15] R. Trobec, M. Šterk, S. AlMawed, M. Veselko, *Computer Simulation Of Topical Knee Cooling*, Computes in Biology and Medicine, accepted and under revision.
[16] R. Korf, *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases* in *Proceedings of the Workshop on Computer Games (W31), IJCAI'97*, Nagoya Japan, 1997, pp. 21–26.
[17] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1995.
[18] R. Korf, *Artificial Intelligence Search Algorithms* in *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.
[19] C. H. Hsu, T. L. Chen, K. C. Li, *Performance effective pre-scheduling strategy for heterogeneous grid systems in the master slave paradigm*, Future Generation Computer Systems, 23 (2007), pp. 569–579.
[20] J. Močnik, R. Trobec, B. Robič, *Integration of load balancing in a CORBA environment*, Parallel Algorithms and Applications, 18 (2003), pp. 99–105.
[21] K. Power and J. P. Morrison, *Ad Hoc Metacomputing with Compeer*, Scalable Computing: Practice and Experience 6, no. 1, (2005), pp. 17–32.

Edited by: Dana Petcu, Marcin Paprzycki
Received: April 6, 2008
Accepted: May 20, 2008