Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 453–457
ISBN 978-83-60810-14-9
ISSN 1896-7094

Unicore 6 as a Platform for Desktop Grid

Jakub Jurkiewicz*‡, Krzysztof Nowiński*, Piotr Bała* †
* Interdisciplinary Center for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5a, 02-106 Warsaw, Poland
†Faculty of Mathematics and Computer Science, Nicolaus Copernicus University
Chopina 12/18, 87-100 Toruń, Poland
‡Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
Banacha 2, 02-907 Warsaw, Poland

***Abstract*—The paper shows a possibility of arranging a desktop grid based on the UNICORE 6 which is a well established grid middleware. The grid consists of a number of PC computers which can communicate with the server in a secure way and perform scheduled computational tasks. The main advantage of this system is ease of deployment, flexibility and ease of integration with the large scale grid. We present here results of a simple performance test as well.**

I. Introduction

Nowadays the community grid computing becomes more and more popular with a number of architectures available. Boinc [2] package, Condor [3] are good examples here. At the same time we observe rapid development of full featured grid middlewares such as Unicore and Globus Toolkit. Intense works are carried out which aim at connecting desktop grids and full size grid infrastructures. One of the simplest propositions is to create an interface that would allow to run Globus or Unicore jobs on the desktop grid. Condor/G is a good example here. Currently, interfaces that would allow using full size grid nodes for desktop grid are available. Both solutions have one great disadvantage—they make a connection between two different systems and middlewares which causes technical problems. Of course there are some current works which uses this solutions, however they usually mean creating some kind of bridge between middlewares[4][5][6].

In our work we present a new solution: Unicore 6 middleware is used to create a desktop grid. This solution makes the connection of systems really easy, simplifies all problems related with authorisation and authentication and minimises cost of middleware. This, only a very beginning stage of creating a desktop grid middleware proves that Unicore suits well as a middleware for creation of a desktop grid.

II. Unicore

Unicore is a Java based grid middleware. Early versions of the system (up to Unicore 5) had communication based on the exchange of serialised Java objects. This solution was very fast and easy to implement, but it worked only if both sides of a given communication used the same version of Java. Unicore 5 has been still in use, and one of its main advantages is good separation of the user from the computing system executing his job. This allows connecting computing nodes with completely different architectures. For a certain time now a completely new version of Unicore has been available with the communication based on web-services. It combines simplicity of previous version with portability and independence of Java version from web services. In this paper we refer to Unicore 6 [1].

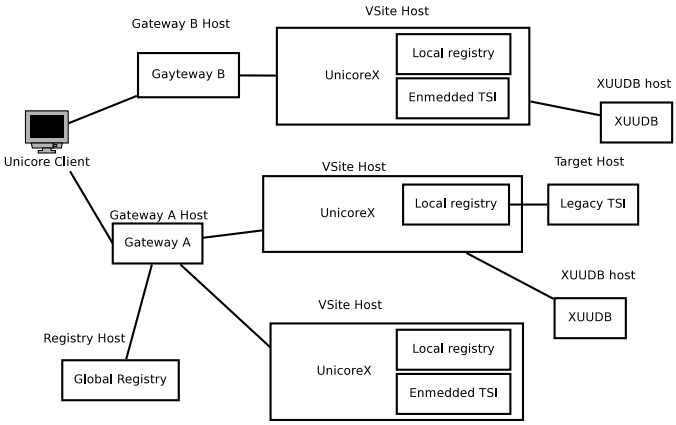

Fig. 1. Example Unicore installation

The Unicore system consists of the following parts:

- Gateway - a module that allows other modules to connect to the grid. It ensures that no unauthenticated user has any access to the protected part of a grid.
- Virtual site (VSite) and a target system (i.e. UnicoreX) - the part of Unicore that is responsible for execution of applications.
- Registry - a holder of information on services used by clients.
- Storage - used for data storage on the grid.
- XUUDB - user database for authentication of users.
- Client - part of Unicore which runs on client's side.

An example Unicore 6 installation is presented on Fig. 1.

A virtual site splits into UnicoreX and Legacy TSI which provides for connecting to target systems built using Unicore 5.

Additionally Unicore has been extended by UVOS system for Virtual Organisations [7]. This extension will allow integrating created desktop grid with the computational grid, as one of possible sites.

1) Why Unicore?: Unicore and Globus Toolkit are two most popular grid middlewares. They both, in their new versions are based on web services. Unicore has two great advantages—it is easy to be configured and run, and it has a simple but very extensible and powerful authentication and authorisation infrastructure which have been based on industry standards such as the X.509 PKI.

III. Desktop Grid Architecture

The goal of this work is to build, using Unicore modules, an architecture model consisting of:

- computing element (desktop node, potentially unreliable)
- manager node

Current architecture of desktop grid is presented in the Fig. 2.

A. Computing Element—Node

Computing element which does the most of computations has been built based on gateway and target system—UnicoreX. It uses XUUDB that works as manager for authorisation.

The problem was to minimise the application so that to make it running on the computing element under the target system. An ideal application consists of code that could be downloaded with a job to be executed. Unfortunately, this leads to serious security problems. It is possible only if the application run inside the target system, has the same security as unsecured java applet. This means that the application can not:

- run or read from disk,
- connect to host other then a Desktop Grid Manager,
- use only a secure classloader.

At the early stage of project we still use a disk for keeping logs and keystores, but in the future, after all optimisations we plan to totally disallow the application to use disk for execution. Data for computation is kept on the manager computer and downloaded straight to the memory space of application.

1) Application Runner: All jobs submitted to computing element could be divided into three parts:

1) obtaining data from grid storage using the key provided in the job description,
2) doing actual computations using a module described in the job description (Application),
3) sending back the data to the grid storage.

All data is kept in the memory, thus involving no need to use a disk. The private key used for data transferring is encoded into the ASCII string and it could be given to the application as a normal run time argument.

B. Manager Node

Desktop Grid Manager node software consists of the following elements:

- gateway—which allows accessing to the registry and storage,
- registry,
- grid storage accessible via RBYTEIO,
- XUUDB which is accessible on its own port,
- Desktop Grid Manager.

Additionally, on the manager node we separate the storage space for finished tasks results. It would increase the security, because Desktop Grid Manager is responsible for moving the results there and, after such moving, is the only one who has access to the data. More detailed description of Desktop Grid Manager is presented in III-C.

It is also possible to split Desktop Grid Manager among registry, storage, XUUDB and computations manager. This would allow running jobs from a computer that has no external IP address and/or open ports. Of course, another solution is to create Desktop Grid Manager that works as a service under Unicore.

The architecture we have chosen is very good at this stage of development of desktop grid. It allows us to easily experiment and change parameters of tested architecture.

C. Desktop Grid Manager

Desktop Grid Manager is a multithreaded application. It uses threads for controlling different aspects of desktop grid work. It allows running a job on a desktop grid and getting results, and is responsible for:

1) checking available nodes in registry,
2) dividing job into sub-tasks and merging results,
3) submitting tasks to a computing element,
4) fetching sub-tasks results,
5) monitoring state of nodes.

Desktop Manager uses separate threads for:

- checking registry—this thread is used for checking if any new computing element has registered, and if there is a new node, thread tries to do the rescheduling.
- checking node—checks if node is still alive, and what is state of computations. If node is down, or if it has finished computations, the thread tries to do rescheduling.
- running manager tasks—some tasks have to be done on the manager server, i.e. dividing job into sub-tasks and merging the results. There is a specially designated thread with its own queue for doing such job. When it finishes a task it tries to do the rescheduling.

1) Checking Available Nodes in the Registry: When the owner of a private computer turns on desktop grid infrastructure on his computer, the UnicoreX registers in the Registry located at the Desktop Grid Manager. A thread that checks the registry finds out whenever a new node becomes available and, if so it runs a new thread for checking the node.

2) Dividing Job Into Subtask and Merging the Results: When the desktop grid receives a job to do, it divides it into sub-tasks. It is performed by a thread used for running manager tasks. This thread also runs a part that merges results after they are fetched.

3) Submission of Task to a Computing Element: If any of the following events happen in the system, such as:

- node is up or node is down,
- new job has been submitted,

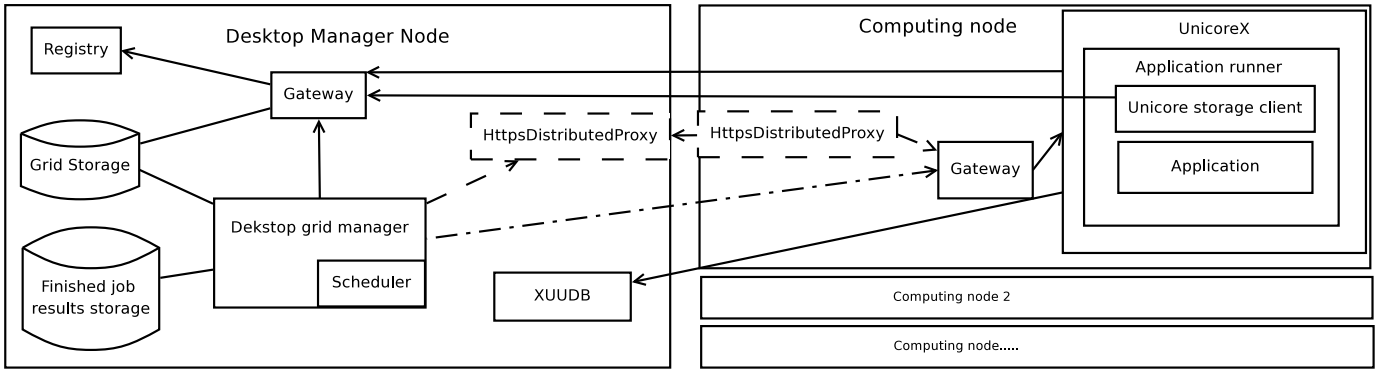

Fig. 2. Architecture of desktop grid

- node has finished computations,
- manager task has finished,

the Desktop Manager tries to run a new task on free computing elements. It runs a scheduler module which is responsible for matching tasks with the computing elements. The actual running and fetching of a task to a private computer is done by Node Manager which is part of Desktop Manager.

4) Fetching Sub-Tasks Results: When a job finishes computations, which means that all data is sent to a grid storage (see III-A1),the node monitoring thread receives information that the job has finished (by executing Unicore check job state call). Then, it copies all data from desktop grid storage to the finished job storage, and tries to perform the rescheduling. Or, possibly, if all tasks for the job have finished, it tries to queue a new manager thread for merging the results.

5) Monitoring State of Nodes: The system monitors two indicators for node activity:

- node down or up,
- state of job on the node

Because the registry can have out-of-date information about nodes, we have to monitor its status by our own. For this purpose, we call the Unicore check node time call, which is a common method of testing if the whole node Unicore infrastructure is running.

6) Use of Desktop Grid Manager: Desktop Grid Manager is built of two parts—core manger described above and the User interface which can be easily modified and adopted to the user's needs. Different possible settings of usage of the desktop grid are presented in Fig. 3, Fig. 4, Fig. 5 and Fig. 6.

In Fig. 3 Desktop Grid Manager plays a role of target system. Additionally, because the desktop grid uses the same middleware as Unicore grid, we can utilise free time of Unicore grid computing nodes.

In Fig. 4 there is presented a desktop grid as a standalone system. This would be achieved by changing authorisation database for Desktop Grid Manager from Unicore grid authorisation database to local one.

Standalone settings could be slightly modified by allowing clients to be computing units at the same time. Such settings are presented in Fig. 5.

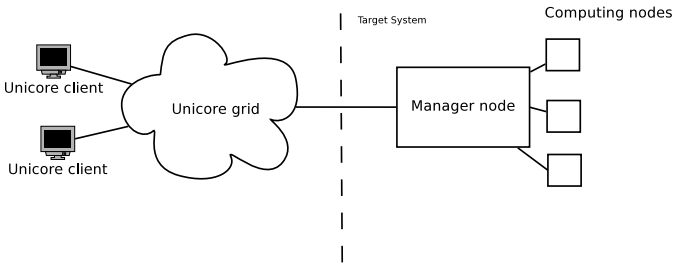

Fig. 3. Desktop grid manager server as target system for Unicore

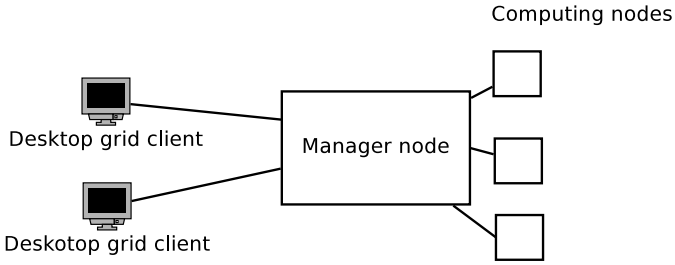

Fig. 4. Desktop grid manager as standalone server

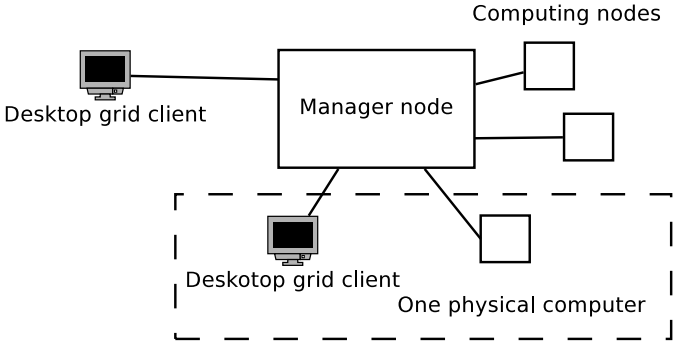

Fig. 5. Desktop grid manager as standalone cooperative system

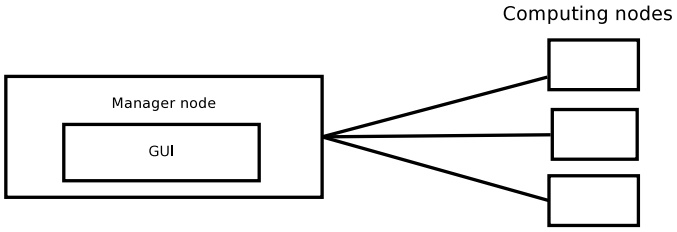

Fig. 6. Desktop grid system—development setting

Finally, Fig. 6 presents a currently implemented development setting.

With respect to this setting we created a simple GUI for Desktop Grid Manager. This solution makes monitoring of jobs execution really easy and, moreover, it simplifies debugging.

7) Scheduler: Scheduler used in our Desktop Manager is a random scheduler with added backup policy. It tries to assign tasks with lower number of running instances first. If tasks have the same number of running instances, then the one belonging to job that was assigned first is run.

D. HttpsDistributedProxy

As it is presented in Fig. 2, the system could contain distributed https proxy. This part has already been implemented but it hasn't been incorporated into the desktop grid yet. Because the computing elements could be put behind firewall or NAT, we introduce https distributed proxy. This proxy is built of two parts:

- server part—where https client connects and asks for page
- computing node part—the part that opens a connection to a server part and waits for data coming from server, that should be tunnelled to a site.

Because https protocol does not allow looking into packets by the proxy, the tunnelling is the only available option. Because the whole communication is started by a computing node part, the computing node may be located behind NAT gateway and be not visible from Internet.

E. Security

The security in the Desktop Grid is based on X.509 certificates. Desktop Server has one certificate, and every computing elements group should have their own one, too. Because certificates should be generated when the owner of computers gets a software package, and we cannot guarantee that few instances of one package will not work at the same time, we introduce a computing elements group—i.e. different computers working with the same certificate set.

Additionally, every sub-task is given its own certificate that will allow getting and putting on the desktop grid storage only such files which belong to it.

Currently, our desktop grid works with one set of certificates—all parts use the same certificate.

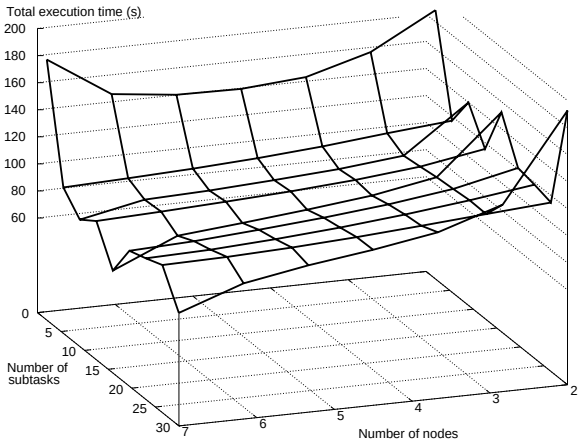

Fig. 7. Efficiency of presented system

IV. Efficiency Tests

Desktop grid was tested on naive implementation of concurrent Mandelbrot set computing algorithm. For purposes of tests we used jobs that took 236 seconds on a single computer (run locally).

In Fig. 7 there are presented results of efficiency tests for the created system. On one axis there is presented the number of working nodes (manager node is not counted), on another one the number of sub-tasks which the job was divided to. What seems unclear here is a fact that the minimum time of execution has been achieved for a number of tasks larger than the number of nodes(not slightly larger but double or triple). This is caused by pure balancing of tasks in naive concurrent version of algorithm.

In Fig. 8 there is presented the speedup of computations as a function of number of nodes. Speedup is a time taken by computations on one machine, divided by minimum total execution time for specified number of nodes. Below 5 nodes this function is of linear nature, however worse than optimum line $y = x$. Above 5 nodes, the constant cost gains in importance, thus making the difference in speedup between 6 and 7 nodes much smaller than that between 3 and 4 nodes. These results show that our system is quite effective, although it should be optimised.

V. Conclusions

Our work shows that Unicore 6 could be used as a basic middleware for desktop grid. It is easy to be configured and it needs only small efforts to develop Desktop Grid Manager.

VI. Future Work

In the future we plan to:

- incorporate https distributed proxy to desktop grid,
- incorporate architecture for managing certificates from project Chemomentum [8], and add Uvos in a further step,
- complete detaching the application run by nodes from disks—by using Java policy,

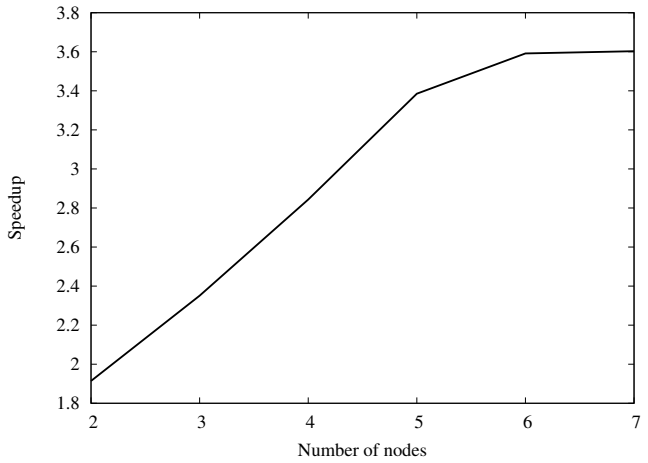

Fig. 8. Scalability of system—speed up

- attaching Java code to be executed, as an part of a task, to the task submission.

ACKNOWLEDGMENT

Work supported by the joint project of ICM UW and Telekomunikacja Polska S.A. 22/06/6727/K/2006/YCZ268 Grid System Monitoring and Control Tools under grant FSO 023/2006.

REFERENCES

[1] *Installation and Configuration of UNICORE 6* http://www.unicore.eu/documentation/manuals/unicore6/files/Installation_UNICORE6.pdf

[2] David P. Anderson, *BOINC: A System for Public-Resource Computing and Storage*, 5th IEEE/ACM International Workshop on Grid Computing. November 8, 2004, Pittsburgh, USA.

[3] Douglas Thain, Todd Tannenbaum, and Miron Livny, *Distributed Computing in Practice: The Condor Experience*, Concurrency and Computation: Practice and Experience, Vol. 17, No. 2–4, pages 323–356, February–April, 2005.

[4] Konstantinos Georgakopoulos, Konstantinos Margaritis, *Integrating Condor Desktop Clusters with Grid*, Distributed and Parallel Systems In focus: Desktop Grid Computing, September 2008.

[5] Zoltán Farkas, Péter Kacsuk, Manuel Rubio,*Utilizing EGEE for Desktop Grids*, Distributed and Parallel Systems In focus: Desktop Grid Computing, September 2008.

[6] Ian Kelley, Ian Taylor, *Bridging the Data Management Gap Between Service and Desktop Grids*, Distributed and Parallel Systems In focus: Desktop Grid Computing, September 2008.

[7] A. Faroughi, R. Faroughi, P. Wieder, W. Ziegler, *Attributes and VOs: Extending the UNICORE authorisation capabilities*, Proceedings of 3rd UNICORE Summit 2007 in conjunction with EuroPar 2007, Rennes, France, LNCS 4854, pages 121–130.

[8] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bala, E. del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt, K. Baldrige, *Chemomentum—UNICORE 6 based infrastructure for complex applications in science and technology*, Proceedings of 3rd UNICORE Summit 2007 in conjunction with EuroPar 2007, Rennes, France, LNCS 4854, pages 82–93.