HORIA-NICOLAI TEODORESCU
MITICĂ CRAUS
(Editors)

Scientific and Educational Grid Applications

Editura Politehnium

Iaşi 2008

CONTENTS

FORWARD

The present volume is the outcome of the collaboration between computer scientists, computer engineers and communication engineers, directed toward the establishment of an academic GRID infrastructure and of a set of applications for scientific and educational use.

The target readers are GRID application developers and graduate students in the fields of computer science and engineering.

The Editors are very much grateful to Professor Cornelius Croitoru for his numerous and extremely useful comments on a preliminary version of this volume, to Professor Mihail Voicu, correspondent member of the Romanian Academy, for his review, and to all the contributors for their patience in writing several versions of the papers. The Editors express their thanks to all referees.

The Editors acknowledge the support of the CEEX research grant *GRAI* (code 74/2006) in the preparation and publication of this volume.

The Editors

PRELIMINARIES

Horia-Nicolai Teodorescu

Preliminary issues

GRID computing appeared as a distinct concept only in 1998. However, cluster computing is arguably as old as the early 1970s, or even earlier according to some authors. After 1990, distributed and particularly GRID technology has established themselves as a competitor for massively parallel computing and for standard clusters. Making use of distributed computation and storage (memory) power located at tens of thousands of locations – virtually at a regional or global level – GRID computing may achieve performances as high as an average super-computer, with the costs shared by a large number of users, with the virtual advantage of accessibility for all the users. Importantly, GRID systems are based on commodity clusters that are not well determined in their parameters (number and types of machines, minimal computation power in nodes, reliability, risk levels) and are evolving in a manner unpredictable to the users, as any user can add and upgrade the installed computation power, as well as tools at any moment. Basically, GRID represents parallel computing on heterogeneous networks.

The evolution of GRID computing is not linear – fast progresses in microprocessors power has made distributed computing less fruitful when the parallel (distributed) tasks are too small. On the other hand, steadily increasing network speeds tend to decrease the optimal amount of computation performed by nodes. The tradeoff between these tendencies constantly changes the optimal task partitioning for distributed computing systems, including GRID. Moreover, limited data transfer speeds determine a threshold below which serial computing is faster than GRID distributed computation. In fact, cluster and GRID distributed computing is heavily relying on the communication means – the network component of the distributed computing is as important as the computational component. On the other hand, the heterogeneity of the networks and commodity hardware make the GRID computing behavior much less predictable than standard (homogeneous) cluster computing. This drawback is reflected in some of the articles in this volume. In *Migrating an Expert System towards Service Oriented Architecture and Multicore Systems*, Dana Petcu deals, among others, with the need of change of the task partition because of the evolution of the computing power of the serial computers. The articles *Sequential and Distributed 3D Terrain Model Generation. Performance Analysis* and *GRID Modeling*

Results for Large Economic Systems carefully analyzes the performances of implementation under serial and locally parallel implementations, moreover the influence of remote distribution (GRID-type) of the computation tasks. The same topic is discussed also in the article *On the Partitioning of Tasks for Parallelization of Fuzzy Coupled Map Models*. Other articles in this volume also tangentially treat the problem.

The applications presented in the articles in this volume are, almost all, typical for GRID implementations. Somewhat atypical are the articles on *Linguistic Processing Architecture in a GRID Environment* and the last article, on the modeling based on fuzzy coupled map lattices.

The future of GRID systems is difficult to predict, because of the vague definition of the GRID technology and because of the high adaptability of the GRID to new computer systems and networks. Based on the evolution of supercomputers, for which "generally, it will take six to eight years for any system to move from position one to 500 and eight to ten years to move from position 500 to notebook level" [Hans Werner Meuer, The TOP500 Project: Looking Back over 15 Years of Supercomputing Experience, Informatik-Spektrum, Volume 31, Number 3, June 2008, pp. 203-222 (**20**)], GRID application will need to constantly adopt to the increased power of the GRID nodes.

We may expect to see in the near future the advent of GRID applications that self-adapt to the change in computation power; else, every few years, GRID applications may need re-writing. On the other hand, the intrinsic adaptability power of the GRID technology may prove decisive in its long-term survival.

Volume structure and characteristics

The volume is divided in three parts, titled *Introduction*, *GRID architecture and services*, and *Applications*. In the first part, comprising two articles, a discussion of a GRID infrastructure and the design of components to integrate MPI applications are presented. The four articles in the second part of the volume deal with topics at the border of applications and GRID services. Two articles discuss issue related to linguistic services, while the other two articles in this section deal with issues related to expert and decision support systems related services. Readers should not expect a detailed analysis of the services, but rather an introductory one.

By far the most consistent part in the volume is the third one, where several articles on applications are gathered. Modeling of terrain for geographic applications, neuro-fuzzy models of economic processes, epidemic models, coupled-map fuzzy models, and visualization issues are presented at the application level in varying degrees of detail and with somewhat heterogeneous degrees of readability for the readers in the addressed fields.

This volume represents the outcome of a symposium primarily fuelled by a research grant aimed at establishing of an academic GRID for educational and research activities. Collaborations with colleagues from Timişoara (Romania) and Cluj-Napoca (Romania) added two articles to this volume.

The *GRAI* project – as the acronym in Romanian runs, standing for **GR**ID **A**cademic (from) **I**aşi – has contributed to build a small GRID sub-network and the required expertise in five universities and research institutes from Iaşi. The support of the research grant CEEX-74-2006 is acknowledged.

Audience

The volume does not offer a comprehensive view of the field; in fact, it has not been intended to. The volume reflects, mainly, the views and the teaching and research interests of a particular academic group having only academic expertise in the field. Therefore, the reader should have reasonable expectations regarding the content and the scientific level of this book.

The main audience of the volume is represented by master and Ph.D. degree students that look for current applications or wish to document their theses on the state of the art in GRID developments in Romania.

Originality

The Editors and the referees have made all reasonable efforts to check for the full originality of the chapters. However, neither the Editors nor the anonymous referees can take responsibility for this matter. The whole responsibility remains with the authors of the respective articles.

PART I INTRODUCTION

On the Development of a GRID Infrastructure

Cristian-Mihai Amarandei, Andrei Rusan,

Alexandru Archip, Simona Aruştei

Department of Computer Science and Engineering
Technical University of Iasi
{camarand,alexandru.archip,sarustei}@cs.tuiasi.ro,
char@tuiasi.ro

Abstract. *Building a Grid infrastructure from scratch requires experience with various operating systems, middleware applications and large infrastructure deployment tools. A Grid infrastructure developed from scratch implies testing various tools and applications in order to reduce the time for further development of the project and infrastructure maintenance. Computational clusters have become the dominant platform for a wide range of scientific disciplines and are the base of any Grid systems. This paper presents the development of a Grid project, starting with choosing of the right hardware, network infrastructure and cluster design and installation of the operating system and management tools across clusters, security technologies.*

Keywords: *Grid, Grid infrastructure, Grid security, Cluster design, Cluster management*

1. Introduction

A Grid is a system that coordinates resources that are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. Another definition presents a Grid as flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources - what we refer to as virtual organizations [1].

The easiest use of grid computing is to run an existing application on a different machine. The machine which the application usually runs on might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid. In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5% of the time. In some organizations, even the server machines can often be relatively idle [2]. If all computers available in organizations (PC's, servers and clusters) are used to provide extra computing power, sometimes it is not enough and they have to either upgrade their computers or get computing power from somewhere else and run the applications remotely. If the internal infrastructure is not always used, there will be computing power available and the organizations connected in a Grid system can share it. Under these circumstances, computing power is one of the most attractive features of a Grid. All Grid projects have clusters that are shared in the virtual organization.

The administrator should understand the organization's requirements for the Grid in order to better choose the Grid technologies that satisfy those requirements [2]. In order to achieve this purpose, it is necessary to find ways to implement a hardware and software base infrastructure that allows administrators to quickly install, update and deploy new nodes in the Grid.

Understanding the fail-over scenarios for the given Grid system is a crucial step, as any grid system must continue operating even if any of the management machines fails in some way. Machines should be configured and connected to facilitate recovery scenarios. Any critical databases or other essential data for keeping track of the jobs in the grid, members of the grid and machines on the grid should have suitable backups [2].

2. Cluster Architecture and Grid Systems

The administrator should design the local clusters using high speed interconnection network only for internal cluster communications to achieve a distinct collision domain (Figure 1). All other computers that will be used as CPU power in the grid can be connected via common network infrastructure that is already in place.

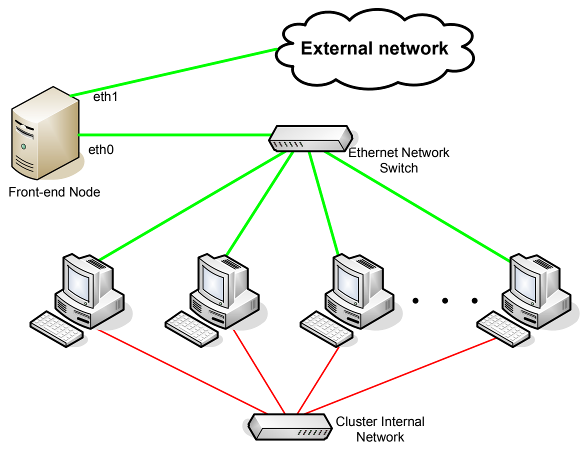


Fig. 1. A sample of a cluster architecture that will be a part of a Grid system

A High-Performance Computing cluster typically has a large number of computers (often called nodes) and most of these nodes would be configured identically. The idea is that the individual tasks that make up a parallel application should run equally well on whatever node they are dispatched on. However, some nodes in a cluster often have some physical and logical differences [10]: multiple logical functions may reside on the same physical node, and in other cases, a logical function may be spread across multiple physical nodes (Figure 2).

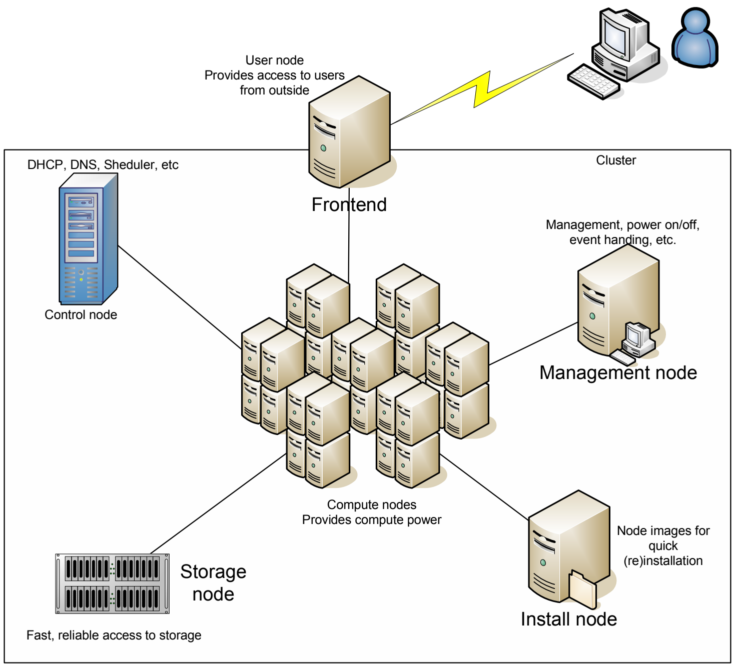

User node
Provides access to users from outside
DHCP, DNS, Sheduler, etc
Cluster
Frontend
Management, power on/off, event handing, etc.
Control node
Management node
Compute nodes
Provides compute power
Node images for quick (re)installation
Storage node
Install node
Fast, reliable access to storage

Fig. 2. The logical structure of a cluster

From the administrator's point of view, with this kind of structure, many problems can be avoided:

- same security settings and means to manage it
- a way to install/reinstall nodes through an install node that provides installation images for the whole cluster.
- the same software installed on all machines – avoid version conflict problem

Remotely located clusters must be connected and secured using frontends. Also, setting up a single, central access point for this kind of infrastructure is required both for the administration of the cluster (same security settings, same software version), as well as for the simple user that runs a job (to avoid problems with access rights, library versions etc.).

The logical structure of the clusters depicted in figure 2 may be used in Grid systems: Control nodes, Storage nodes, Management nodes, Install nodes and user interface nodes may be located on distinct sites within the grid system.

Services provided by organizations that are members in the Grid systems may be different, but the software that provides control and management is the same on all platforms, or at least there are applications that provide some kind of interconnection. For example, the processing node that provides processing power can run Linux or Windows as the base operating system, and the application provided to users runs on MacOS.

All Grid systems are affected by the availability of the services provided by Grid nodes and these are affected by the availability of computers and the time needed to reinstall them and get them back on line. Thus, providing a no-single-point-of-failure should be one of the goals of cluster design and a hardware configuration that can provide it is shown in the Figure 3.

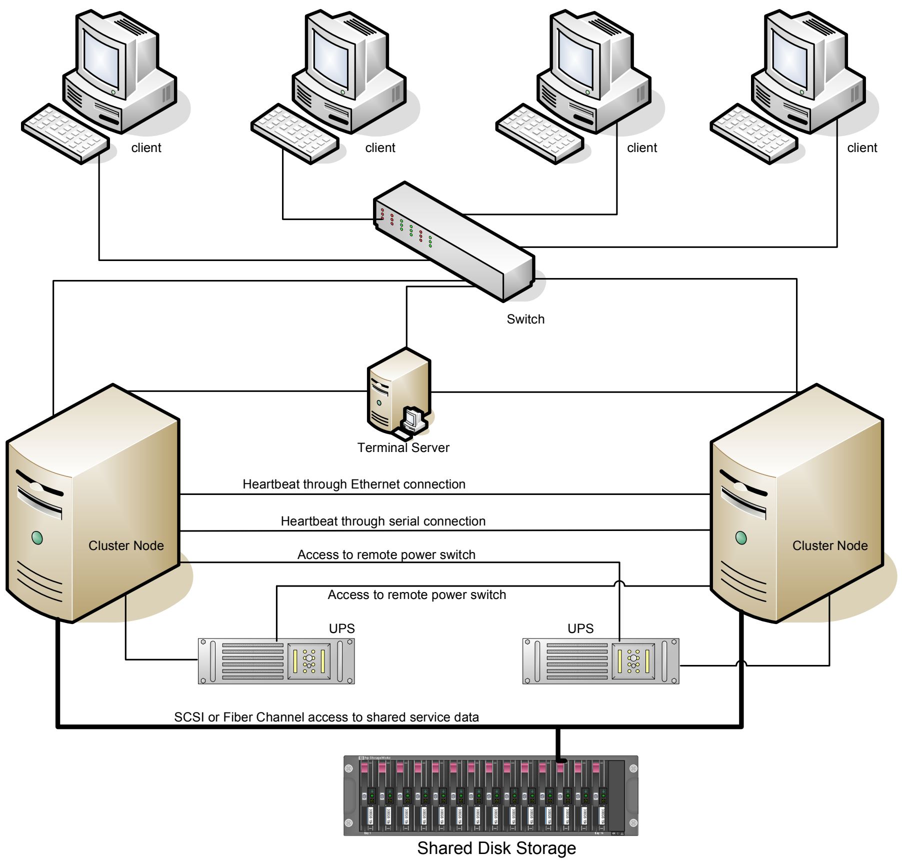

client
client
client
client
Switch
Terminal Server
Heartbeat through Ethernet connection
Heartbeat through serial connection
Access to remote power switch
Access to remote power switch
Cluster Node
Cluster Node
UPS
UPS
SCSI or Fiber Channel access to shared service data
Shared Disk Storage

Fig. 3. No-Single-Point-Of-Failure configuration example. Terminal server is not a required component.

If database servers are required for Grid systems, these servers must provide access and reasonable response time for clients. For those database servers we can choose Oracle, DB2, Sybase, Informix or any commercial product, or we can choose open source solutions with proven high performance: MySQL and PostgreSQL. MySQL cluster may prove to be very useful in Grid infrastructure development, if we consider that the cluster hardware solution is to have a tight coupled system for the storage nodes, like the one in Figure 1. Because one database cluster provides a single point of failure in the Grid infrastructure, replicas of this cluster in different Grid sites can be created.

The software for the cluster management is one of the most important aspect that the system administrator must take care of, and here are a plenty of choices: RocksClusters, Platform Open Cluster (free to use but with commercial support version of RocksClusters), RedHat Cluster Suite, Linux Cluster Manager, OSCAR (Open Source Cluster Application Resources), BOINC, ComputeMode, Clustermatic or Perceus/Warewulf Cluster (if diskless clients are used to build the cluster). In the following we describe the RocksClusters as a solution for cluster management.

3. RocksClusters – Cluster Deployment and Management Tool in Grid Systems

NPACI Rocks is a complete cluster-aware Linux distribution based upon Red Hat with additional packages and programmed configuration to automate the deployment of high-performance Linux clusters. The Red Hat distribution was chosen because of the two key mechanisms found within: software packaging tool (RPM), and script-driven software installation tool that describes a node’s software stack (kickstart) [5]. When scaling clusters, RocksClusters provides mechanisms that fully-automate node installation using RPM and kickstart file.

The software installation process contains two components: the software package installation and configuration of installed package. When configuring software packages, it is common to accept the defaults – the case of a desktop system, or to have different configuration that must be customized – the case of a network with various requirements. The common approach is to install software and, through a process of manual data entry, the software is configured. To extend this process to a cluster, we need to configure one node, create an image and replicate it on all nodes. The process looks simple, but when you have various software configurations and/or various hardware configurations, things becomes complicated.

RocksClusters simplify this process by treating software installation and software configuration as separate components. Software installation and configuration is done in the form of packages installs according to the functional role of a single cluster node and is referred as an appliance [5]. To help cluster architects to design new appliances, and re-use their system configuration, RocksCluster distribution provides a simple framework described with XML files. To integrate this software with RedHat based Linux distributions, a tool called *rocks-dist* is provided. This tool, gathers software components from the Linux base distribution, other software developed by the Rocks community and from third-party software. This software is used to create an up-to-date Linux distribution, fully compatible with RedHat Linux (Figure 4).

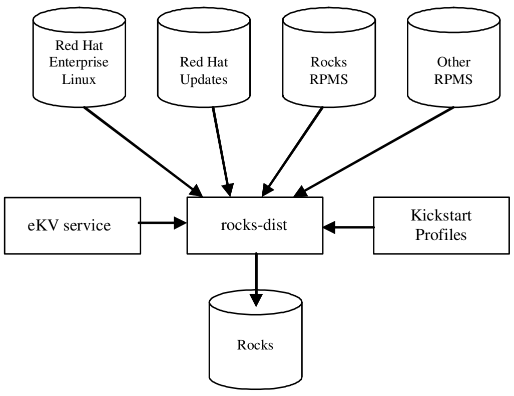


Fig. 4. Rocks distribution generation [8]

By managing the software packages independently from software configuration, the same configuration can be applied to different software distributions. Or, in the case of cluster testbeds, the same software distribution can be used with multiple software configurations [5].

Rocks cluster use the following components to generate the software distribution [7]:

- **Anaconda**: RedHat Linux installer is the tool that requests a kickstart file (either locally form the installation media or over the network), parses the keywords from the file and execute the appropriate commands to complete software installation. After reboot, the computer is ready to use.

- **CGI**: Rocks use CGI scripts to serve the appropriate kickstart files to Anaconda installer. On each node installation, kickstart files are requested via HTTP. The node constructs the URL by combining information from the DHCP response and node-specific information (e.g., hard disk name(s) and architecture type). An example URL for a x86-based node with two SCSI disks is:

http://frontend-0/install/kickstart. .cgi?devnames=sda,sdb&arch=i386

The kickstart file is generated by CGI scripts: extract node specific fields from the query component of the URL; query the SQL database and passing those values to KPP to generate XML file and to KGen, which transforms the XML file into a valid kickstart file and send it to the installing node.

- **SQL database**: The configuration database stores information about the cluster as a whole and information about specific machines and groups of machines.

- **KPP**: The Kickstart Pre-Processor traverses the configuration graph, requests machine state from the SQL configuration database and builds a single monolithic XML-based Kickstart file for a specific cluster node.

- **KGen**: The Kickstart Generator transforms an XML Kickstart file into RedHat Kickstart syntax. This additional step exists to allow future formats to be used.

The entire installation process using kickstart files is presented in Figure 5.

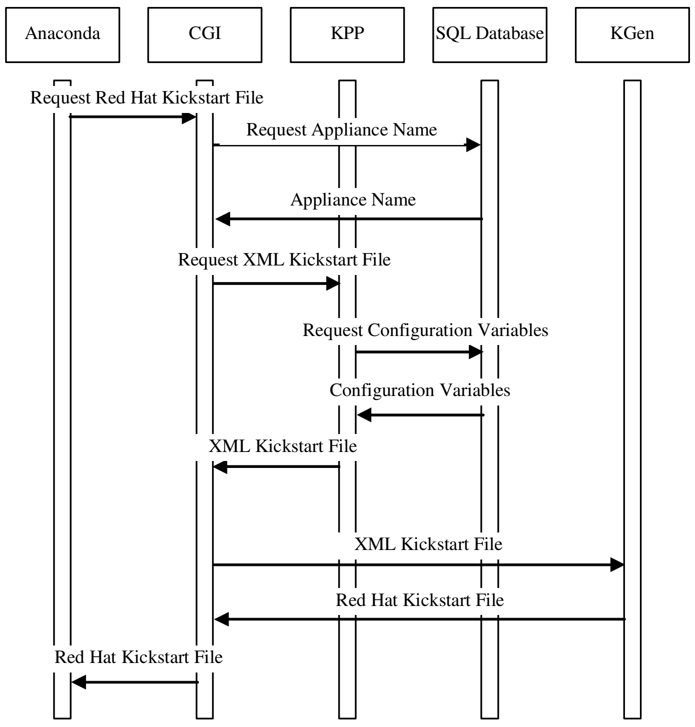


Fig. 5. The process of a machine requesting and receiving its Kickstart files [7].

4. Wide Area Deployment and Security in RocksClusters

If RocksClusters distribution looks like a good solution to manage a cluster, the next step in Grid development is to try to use it with remotely located clusters. Rocks distribution can be used to perform full cluster installations over wide area networks. This wide-area cluster integration involves a central server that holds the entire software stack and local frontends can obtain it from this server. Because this server will be a critical resource in the whole Grid system, a no-single-point-of-failure configuration as is shown in Figure 3 or replication is recommended. The wide-area rocks cluster architecture is shown in Figure 6.

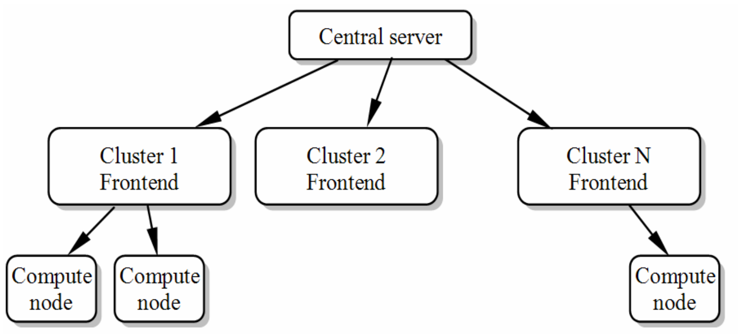


Fig. 6. Wide Area Cluster Integration [9]

The wide-area cluster integration facility enables a Grid infrastructure to push the entire software stack to nodes over the Internet with low administration overhead. If site specific additions and customizations of software components are required, the roll structure of RocksClusters is useful to system administrators – they only need to create and install a specific roll to the local frontend and roll that will be deployed by request on the cluster nodes.

Deploying software, configuration files and user credentials over the Internet can be a major security risk to the entire Grid system. To avoid this kind of problems, RocksClusters distribute password files, user and group configuration files and the like, uses 411 Secure Information Service. The 411 service provides a NIS-like service for Rocks and mimics the NIS interface for system administrators and uses Public Key Cryptography to protect files' contents. It operates on a file level, rather than the RPC-based per-line maps of NIS. The 411 service does not rely on RPC, and instead distributes the files themselves using HTTP (web service). Its central task is to securely maintain critical login/password files on the worker nodes of a cluster. It does this by implementing a file-based distributed database with weak consistency semantics. The design goals of 411 include scalability, security, low-latency when changes occur, and resilience to failures [8].

Frontends encrypt and serve 411 files (called 411 messages once they are encrypted) using their local Apache web server. Cluster nodes retrieve 411 messages using HTTP, decrypt them, and save the resultant file to their local. Cluster nodes can recognize multiple servers, and make some attempt to load balance their 411 message retrievals across the set of frontends servers to reduce strain on the cluster file system. In the case of wide-area cluster integration, the 411 service functionality is extended to frontends [8].

5. Case Study

The entire software stack provided by RocksClusters and other providers described above was studied and we started the implementation in the GRAI Grid project. In this project are involved five partners from "Gheorghe Asachi" Technical University of Iasi

(UTI) – project coordinator, “Al. I Cuza” University of Iaşi (UAIC), University of Agricultural Sciences and Veterinary Medicine Iasi (USAMV), “Gr.T. Popa” University of Medicine and Pharmacy Iasi (UMF) and from Romanian Academy-Institute for Computer Science Iasi (AR-IIT). Each partner will build its own cluster to be included in the project. The architecture that will be implemented for each partner cluster connected in the GRAI Grid as is shown in Figure 7.

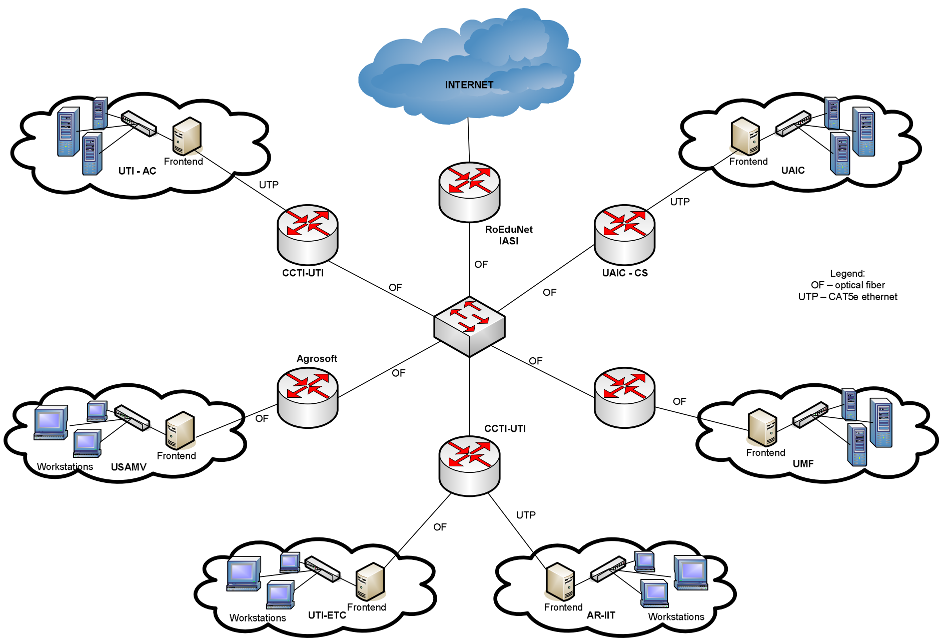


Fig. 7. GRAI Grid system architecture

A no-single-point-of-failure architecture will be implemented for the frontend located at UTI-AC GRAI site. Particularities for our frontend will include the following: two servers, heartbeat connection through crossover cable, two Gigabit Ethernet cables, two UPS devices with serial management cables like in Figure 3, and two switches for both external and local interfaces respectively.

As a wide area cluster integration solution (Figure 6) for the GRAI project, a central server has been established in UTI-AC location (Figure 7) to distribute the software to all Grid sites. Also, local frontends have been installed for each partner involved in the GRAI Grid project. Each of this frontends may be managed through the central site. The security mechanism provided by the 411 service of RocksClusters fits perfectly on the GRAI Grid project architecture as it allows for a reliable user and system security credentials management.

6. Conclusions

This paper discusses the development of a Grid project, starting with choosing of the right hardware, network infrastructure and cluster design and installation of the operating system and management tools across clusters and also security technologies. In Grid architectures the key problem is to manage an extremely heterogeneous hardware, computing, storage, and networking environment. The complexity of management of such Grid systems is challenging. The RocksCluster facilities establish a rational software distribution and management plan, enabling any Grid systems to easily expand by adding new computing hardware or any new software stack with minimal effort. The software stack required by GRAI project includes: middleware, workload management system, job scheduling and monitoring tools. RocksCluster distribution helps with automatic install of the:

- GlobusToolkit 4 – for middleware;
- Condor, SunGrid Engine or Torque – as resource managers;
- OpenPBS – for job cheduling;
- Ganglia, OpenSCE – as monitoring tools.

Many private companies, universities and research centers develop new software or integrate existing software in RocksClusters distribution. Using existing software tools (e.g., RedHat installer) and providing a way to add new software packages (Figure 5) as part of the distribution with the description files that are almost completely hardware independent or even build an entire customized distribution, RocksCluster is used to build the base infrastructure for the GRAI Grid project.

Acknowledgments. The Excellence Research Program, through grant 74 CEEX-II03 – "Academic Grid for Complex Applications", has supported the research for this paper.

References

[1]. Ian Foster, Carl Kesselman, and Steven Tuecke: ”The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, International Journal of Supercomputer Applications, (2001), 15(3): 200-222
[2]. Viktors Berstis, Fundamentals of Grid Computing, IBM Redbooks, REDP3613
[3]. MySQL Reference Manual
[4]. The Red Hat Cluster Manager Installation and Administration Guide, (2002)
[5]. P. M. Papadopoulos, C. A. Papadopoulos, M. J. Katz, W. J. Link, G. Bruno: “Configuring Large High-Performance Clusters at Lightspeed: A Case Study”, Clusters and Computational Grids for Scientific Computing, (2002)
[6]. P. M. Papadopoulos, M. J. Katz, G. Bruno, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters," cluster, 258, 3rd IEEE International Conference on Cluster Computing (CLUSTER'01), (2001)

[7]. M. J. Katz, P. M. Papadopoulos, G. Bruno, “Leveraging Standard Core Technologies to Programmatically Build Linux Cluster Appliances”, IEEE International Conference on Cluster Computing (CLUSTER'02), 47, (2002)
[8]. http://www.rocksclusters.org - Rocks Cluster Distribution manuals: Users Guide, Introduction to Clusters and Rocks Overview
[9]. F.D. Sacerdoti, S. Chandra, K. Bhatia, "Grid systems deployment & management using Rocks," cluster, 337-345, 2004 IEEE International Conference on Cluster Computing (CLUSTER'04), (2004)
[10]. E. Ford, B. Elkin, S. Denham, B. Khoo, M. Bohnsack, C. Turcksin, L. Ferreira, “Building a Linux HPC Cluster with xCAT”, IBM Redbooks, (2002)
[11]. Federico D Sacerdoti, Mason J. Katz, and Philip M. Papadopoulos, July 2005, IEEE High Performance Distributed Computing Conference, North Carolina

On the Design of Higher Order Components to integrate MPI Applications in Grid Services

Alexandru Archip, Simona Aruştei, Cristian-Mihai Amarandei and Andrei Rusan

"Gheorghe Asachi" Technical University of Iasi
Department of Computer Science and Engineering
{alexandru.archip,sarustei,camarand}@cs.tuiasi.ro, char@tuiasi.ro

Abstract. *Although Grid systems have greatly evolved during the past few years and now have support for parallel applications, this support is somehow limited. We present a new method of integrating MPI- based parallel applications as Grid Services. Our implementation aims to fully incorporate MPI applications within Java Grid services, through the use of Java COG Kits. Tests were conducted on the GRAI Grid, using Globus Toolkit 4 as the base middleware.*

Keywords: *MPI, Grid Computing, Grid Services, GT 4, Java COG*

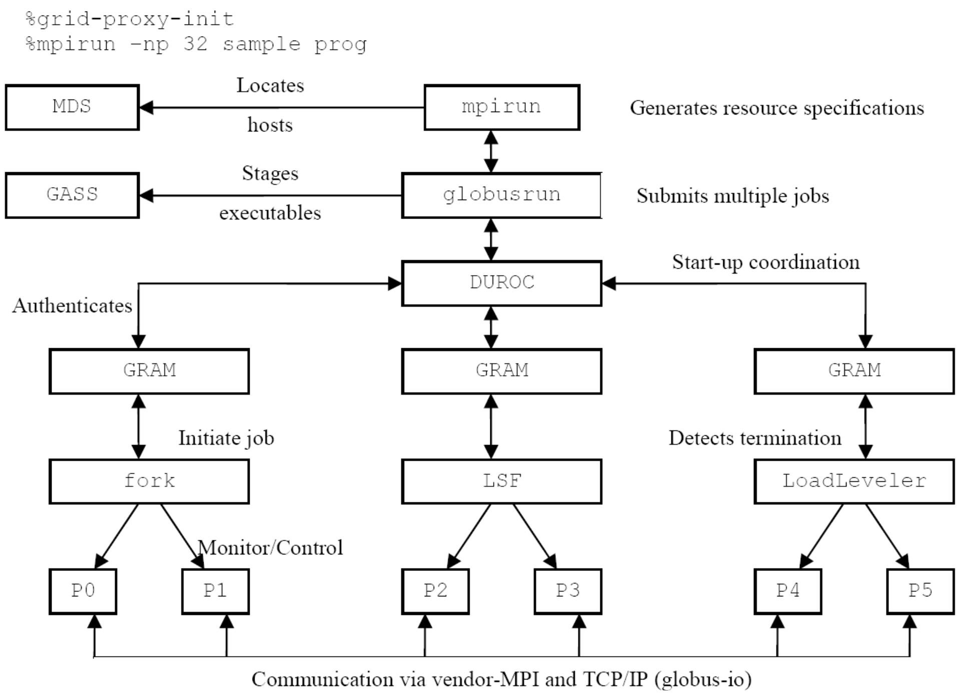
1. Introduction

Grid computing naturally incorporates distributed applications in order solve complex problems. When it comes to parallelism in Grid applications, specialists [1] agree that an useful standard is provided by the MPI (Message Passing Interface) libraries. This is due to the fact that this standardization relies on the most commonly used and best-understood parallel models. As presented in [2], Globus Toolkit – including version 4 of the middleware – offers support for parallel application design and implementation through the use of MPICH-G2 package. As depicted in [2] and [3], MPICH-G2 offers full support for MPI v1.1 standard and support for parallel file I/O. However, full support of MPI v2.0 standards has not been implemented. Another important aspect is that MPICH-G2 relies on GRAM for job submitting, as will be detailed in the following section. Tendencies in current implementations of Globus Toolkit 4 (GT4) indicate a migration of the grid services towards different approaches [1], [6].

In order to overcome these drawbacks, this paper presents a way of integrating other implementations of the MPI standards (such as LAM 7.1.2) with grid services. Tests for the current paper have been performed on an experimental installation of Globus Toolkit 4.0.5.

2. Drawbacks of MPICH-G2

MPICH-G2 relies on Globus Toolkit services to achieve transparent and efficient execution of parallel applications throughout heterogeneous Grid systems, while ensuring a good manageability for the parallel application. The general start-up of MPI applications through MPICH-G2 is given in Figure 1 [3].



```
%grid-proxy-init
%mpirun -np 32 sample prog
```

Fig. 1. General start-up diagram of MPI based applications [3]

As depicted in Figure 1, MPICH-G2 relies on DUROC and GRAM for forking the corresponding parallel processes. In order to run such an application, the user must submit the following RSL script:

```
+( &(resourceManagerContact="frontend.tuiasi.ro")
   (count=4)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
                (LD_LIBRARY_PATH /opt/globus/lib/))
   (arguments="arg 1" "arg 2" "arg 3")
   (directory="/export/home/alex/testing/mpich_test1")
   (executable="/export/home/alex/testing/mpich_test1/a.out")
   (stdout="/tmp/a.out.stdout")
   (stderr="/tmp/a.out.stderr")
)
```

We noticed during tests that, even though compiling and linking of the MPI code has been performed without errors, GRAM may fail in correctly initiating the required number of processes. These defective runs have been noticed especially when the underlying applications required the use of STL libraries.

Another important disadvantage is that MPICH-G2 is not fully compatible with MPI 2.0 standards. Therefore, message exchanges between communicators or dynamic management of processing nodes are not supported [3].

Also, [4] clearly states that MPICH-G2 implementations (as well as other MPI implementations) offer poor or no support for Grid Service interaction.

3. WS-GRAM Approach for MPI Jobs

The WS-GRAM component integrated in GT4 represents in fact a web service based approach to the already existent GRAM module. GT4 documentation [8] underlines the following benefits of this component:

- Jobs now have a submission ID, automatically generated by the globusrun-ws command line tool. This ID is used for better job management.
- Better support for multiple jobs.

In order to submit a MPI based application as a grid job using WS-GRAM, an authorized grid user must write a XML based RSL file. A simplistic example is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
    <executable>/home/alex/ws_gram/a.out</executable>
    <directory>/home/alex/ws_gram</directory>
    <argument>"arg 1"</argument>
    <argument>"arg 2"</argument>
    <argument>"arg 3"</argument>
    <stdout>/tmp/alex/a.out.stdout</stdout>
```

```
        <stderr>/tmp/alex/a.out.stderr</stderr>
        <count>8</count>
        <jobType>mpi</jobType>
    </job>
```

In the given example, *a.out* is a MPI-based C++ source code compiled with the LAM 7.1.2 tool suite. The authorized grid user must then perform the following steps (through the use of command line instructions):

1. Check if an appropriate proxy exists and initialize the proxy if needed.
2. Edit a text file with an appropriate job description (like the one listed above).
3. Check whether *lamd* is running and start if necessary.
4. Issue the following command: *globusrun-ws –submit –f job_description_file* (for further details please see [7] and [8]).

In case of a successful submission, the following are displayed:

```
Submitting job...Done.
Job ID: uuid:6b64e3d2-0afe-11dd-9d6d-001bfcd8bca3
Termination time: 04/16/2008 15:12 GMT
Current job state: Active
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
```

The listing above clearly depicts one of the main advantages of WS-GRAM over GRAM. In line two of the listing the reader may observe the submitting ID for the corresponding job. Using specific methods, an authorized Grid user may submit a time consuming job and may then monitor its execution at a later time.

WS-GRAM is made available as of version 4 of the Globus Toolkit. Although GRAM functionally is still maintained due to backward compatibility issues, Grid users and developers are strongly advised not to use this approach for further development. The complete migrating guide between GRAM and WS-GRAM [9] has been made available at the beginning of this year.

The next two sections describe a method used to integrate MPI based applications in Java Grid Services. Also, section 5 of the current paper presents a set of advantages of the presented model.

4. Integration of MPI Jobs with Java Grid Applications

A major drawback of MPI based implementations is that the preferred languages for programming are machine dependent languages [4]. Almost all parallel applications developed using various implementations of the MPI specifications are written mainly in

C/C++. Although this supports the speed of the application, it narrows down the possible execution platforms.

A first attempt to integrate MPI based jobs with Grid applications/services is to translate the C/C++ MPI code in mpiJava [11]. This solution, while being platform independent and ensuring a good interaction with other Grid services (such data services), has a major drawback when it comes to the speed of the application. Also, mpiJava relies on MPICH-G2 [11], which fully supports only MPI v1.1 standards. Therefore, this first method does not completely solve the problem of integrating MPI Jobs with Grid applications/services.

A second method to integrate native MPI implementations within Grid applications/services is suggested in [4] by Dunnwebera et al (see Figure 2).

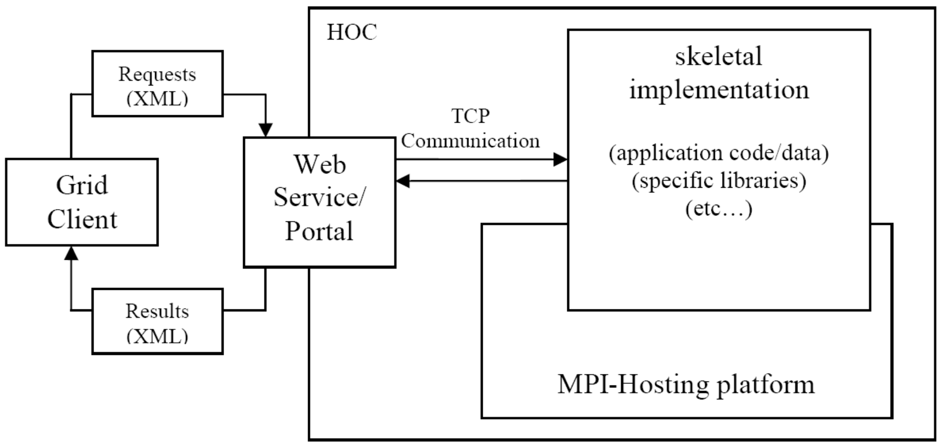


Fig. 2. General Grid integration of MPI C/C++ based applications [4]

The idea presented by Dunnwebera in [4] is focused on using some higher level programming language (the preferred programming language being Java) to design and implement a Web Service that would serve as a wrapper for the MPI C/C++ code. Such a component is called Higher Order Component (HOC). The communication between the MPI C/C++ application and HOC can be achieved through pipe-lines (however, this is also operating system dependent) or through the use of CORBA designed modules.

Based on [4], we have implemented two Java classes in order to support MPI job submission from Java applications. These classes have been developed using Java COG API (such as the one presented in [7]) supported by Globus Toolkit 4.0.3.

The *MyGridProxy* Class

The first step of every WS-GRAM job submission is checking for valid user credentials. In Grid environments these credentials are represented by user proxy files [10]. A succinct code listing is given below:

```
public class MyGridProxy {
   private X509Certificate certificate;
   private PrivateKey userKey = null;
   private GlobusCredential proxy = null;
   private ProxyCertInfo proxyCertInfo = null;
   private int bits = 512; //strength
   //12 hours life time for new proxy
   private int lifetime = 3600 * 12;
   private int proxyType;
   private GlobusGSSCredentialImpl credimpl;

   private String proxyFile;
   private String keyFile;
   private String certFile;
   private String issuer;

   private String user, password;
   public MyGridProxy() {}
   public void environmentSetup() {}
   public void createProxy() throws Exception {}
   public boolean checkProxy() {}
   public boolean destroyProxy() {}
   public GSSCredential buildProxy() {}
}
```

This class is responsible for user proxy management. Any instance of the above listed class should first check for a correct environment setup. This implies that user certificate and user key files are under their default location (.globus directory in user's home). After locating the corresponding files, the application must check whether or not a valid proxy exists and whether proxy's lifetime is sufficient for the target given job.

The *WSJobWrapper* Class

For job submission we have implemented the following Java class.

```
public class WSJobWrapper implements GramJobListener {
   private String rslFileName;
   private GramJob crtJob;
   //private String proxyPath;
   //predefined time interval
   //to wait for job notification
   private static final long STATE_CHANGE_BASE_TIMEOUT_MILLIS =
                     10000;
```

```
    private boolean jobCompleted;
    private GSSCredential proxy;
    private int exitCode;

    public void setRSL(String rslFileName) {}
    public String getRSL() {}
    public void setProxy(GSSCredential proxy) {}
    public GSSCredential getProxyPath() {}
    public boolean jobDone() {}
    public WSJobWrapper() {}
    public WSJobWrapper(String rslFileName,GSSCredential proxy){}
    public void stateChanged(GramJob job) {}
    public void submitRslFile() {}
    public int processCrtJob(
                 GSSCredential proxy,
                 EndpointReferenceType factoryEPR){}
    private synchronized void waitJobCompletion() {}
}
```

Any instance of the above listed class should first check whether the *globus container* is started or not. The main method of the class is *processCrtJob*. This method must receive a valid *EndpointReferenceType* indicating a valid *ManagedJobFactoryService* grid service. A notable difference from other implementations is that this class will always destroy the job after a given time interval. If no job state change notifications are received for a predetermined period of time (indicated by the corresponding *STATE_CHANGE_BASE_TIMEOUT_MILLIS*), the instance will assume job is completed. While this is not the best job management, it ensures that given jobs will not occupy their resources indefinitely. This consists as a great advantage as it guarantees that faulty code will not cause resource lockdowns.

The first step a Grid application using the current HOC is to check whether or not the authorized Grid user has an active valid proxy. In our case this is done by calling the public *checkProxy* method of the *MyGridProxy* class. If no valid user proxy is found or if proxy expired, the method will return *false*. In this case, the test application we have used will attempt to initialize the user proxy. The output of a successful run of the current model is given below:

```
01:$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION package1/MainHello
02:Proxy file (/tmp/x509up_u500) not found.
03:creating user proxy ... [BEGIN]
04:        loading user certificate ... [BEGIN]
05:                User Identity:\
   O=Grid,OU=GlobusTest,OU=simpleCA-ldap-c14,CN=Alexandru ARCHIP
06:        loading user certificate ... [DONE]
07:        loading user key ... [BEGIN]
```

```
08:                user key is encrypted
09:        loading user key ... [DONE]
10:        creating the proxy file ... [BEGIN]
11:                proxy created\
12:              valid until: Thu Apr 10 21:14:22 EEST 2008
13:                writing proxy file ...
14:               setting appropriate proxy file permissions ...
15:        creating the proxy file ... [DONE]
16:creating user proxy ... [DONE]
```

Initialization of user proxy is done performing the following steps:

- Load user certificate file (lines 04 – 05)
- Load user key file using a user supplied password (lines 07 – 09)
- Write the proxy file in its default location (directory */tmp* on Linux based systems, lines 10 – 15).

For job submission steps, the test application loads the appropriate, user specified, RSL job description and attempts connection with *ManagedJobFactoryService*. If no errors occur, job is submitted and the client application waits for notifications and for job completion.

The client side output is presented below:

```
01:$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION package1/MainHello
02:[MyGridProxy.buildProxy] Credential username:\
 /O=Grid/OU=GlobusTest/OU=simpleCA-ldap-c14/CN=Alexandru ARCHIP
03:GSSCredential proxy init ok ...
04:org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2934e4fb
05:Proxy lifetime: 43030
06:Job init OK
07:Submitting job with ID:\
 uuid:eb11ec40-0c45-11dd-9158-851d4e391aa2
08:Try number: 1
09:waiting for job to finish....
10:Job state changed: Active
11:Job state changed: CleanUp
12:Job state changed: Done
13:Job done [exit code: 0]
14:destroying job....
15:Job exited with code: 0
```

The Globus container also displays the following output:

```
01:2008-04-10 09:17:14,982 INFO exec.StateMachine\
   [RunQueueThread_0,logJobAccepted:3513]\
   Job ebfc9a60-0c45-11dd-935c-8a8cfba4e261 accepted\
   for local user 'alex'
02:2008-04-10 09:17:16,125 INFO exec.StateMachine\
   [RunQueueThread_0,logJobSubmitted:3525]\
   Job ebfc9a60-0c45-11dd-935c-8a8cfba4e261 submitted\
   with local job ID 'ed12ab60-0c45-11dd-8c66-0019b96d80b1:4109'
03:2008-04-10 09:17:18,893 INFO exec.StateMachine\
   [RunQueueThread_6,logJobSucceeded:3535]\
   Job ebfc9a60-0c45-11dd-935c-8a8cfba4e261 finished successfull
```

The MPI application we have used has been developed using LAM 7.1.2. A *lamboot* command has been issued prior to running the test application. As stated above, the idea for these code wrappers is given by Dunnwebera in [4]. The presented classes have the advantage of wrapping around any MPI application, provided that the target cluster is adequately configured.

Also, the brief code listing presented for our classes shows that no restrictions are made for the code of the embedded application. An immediate advantage is that these classes may be used to wrap any legacy code that may be executed on a target Grid platform. This may be achieved through a correct submission of an appropriate job description file.

5. Design of Wrapper Grid Services for MPI Applications

Using [4] as a base start, we suggest a new way of integrating MPI C/C++ applications in Grid Services. Our first assumption is that, according to [5], binary execution files – such as executable files – may be considered Resources for Grid Services. As a direct result, every MPI C/C++ code compiled with MPI related tools is a potential resource for our Grid Services. The Grid Service that would use such a Resource should comply with factory pattern design depicted in [5] – see Figure 3.

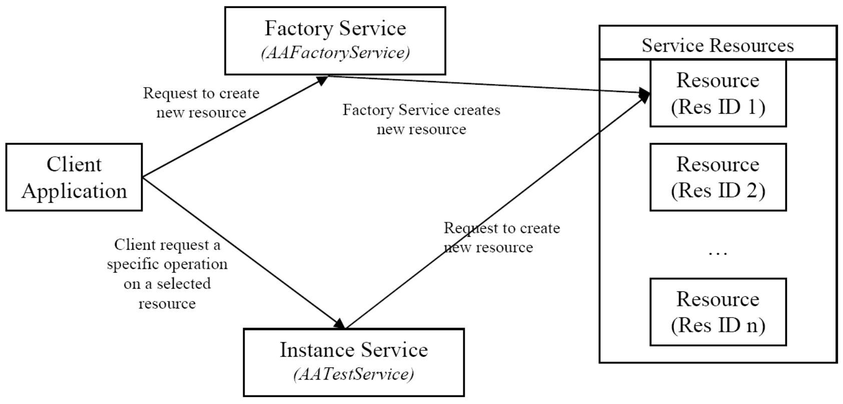


Fig. 3. The WS-Resource factory pattern[6]

In order for our service to run MPI applications as grid jobs through WS-GRAM, the Resources of the service makes use of Java classes presented in section 4. Essentially, the Resources will become a grid client for a corresponding *ManagedJobFactoryService*. Also, corresponding methods in the Instance Service must run with user credentials in order to properly monitor the job for status change notifications.

An immediate advantage of this method is that it resolves the issues concerning interaction with other required Grid services. As stated in section 2 of the present paper, according to [4], extremely poor or no support is provided for interconnection between MPI based applications and Grid Services. The Instance Service or the Resource itself may connect to various other needed services before submission of the job. One such example is given by input/output files that may be required by the MPI module. In such a case, prior to job submitting the Instance Service connects to the *ReliableFileTransfer* service (RFT for short) an uploads any needed input files. After the current job completed successfully, the Instance Service will be able to transfer result/output files back to its client in a similar manner.

A second advantage of this solution is related to security. The Resource must statically specify the MPI module it embeds and therefore the client cannot submit a foreign application code.

6. Conclusion and Further Research

Java COG Kits provide powerful tools for job submission and monitoring for Grid environments. Taking this into consideration, we implemented the above mentioned HOC as an Instance Service monitoring a given WS-Resource. By employing the Java COG API we also allow potential Grid clients to monitor the parallel application a particular Grid Service governs.

Aside from strict application monitoring, our method also eliminates the use of third party web service containers that would interact with the Grid middleware. A potential client for the Grid Service may be a standalone client directly accessing the Service, instead of a Grid Portal relying on third party service containers. Further research will involve service interaction with potential log files for each MPI application in order to further increase the level of job monitoring under Grid environments.

Acknowledgments. The Excellence Research Program, through grant 74 CEEX-II03 – "Academic Grid for Complex Applications", has supported the research for this paper.

References

[1]. http://www.gridtoday.com/grid/1735208.html
[2]. http://www.globus.org/grid_software/computation/mpich-g2.php
[3]. Nicholas T. Karonis, Brian Toonen, Ian Foster, MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, November (2002)

[4]. Jan Dunnwebera, Anne Benoitb, Murray Coleb, Sergei Gorlatcha, Integrating MPI-Skeletons with Web Services for Grid Programming, September (2005)

[5]. http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s03.html

[6]. http://gdp.globus.org/gt4-tutorial/multiplehtml/ch05s01.html

[7]. Gregor von Laszewski, Beulah Alunkal, Kaizar Amin, Jarek Gawor, Mihael Hategan, Sandeep Nijsure, The Java CoG Kit User Manual, Draft Version 1.1a, March 14, (2003)

[8]. http://www.globus.org/toolkit/docs/4.0/execution/wsgram/user-index.html

[9]. http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Migrating_Guide.html

[10]. http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Java_Scenarios.html#s-wsgram-developer-scenarios-java-creatingjob

[11]. http://www.hpjava.org/mpiJava.html

PART II GRID SERVICES

Migrating an Expert System towards Service Oriented Architecture and Multicore Systems

Dana Petcu

Western University of Timişoara
petcu@info.uvt.ro

Abstract. *The efficient use of existing software for scientific computing in the context of the new software and hardware technologies, like Web services and multicore systems, imposes a redesign of their software architecture. We describe the path that was followed in the case of an expert system for ordinary differential equations including facilities for parallel computing.*

1. Introduction

Software systems modernization using software-as-a-service (SasS) concepts represents a valuable option for extending the lifetime of legacy systems and reducing the costs of software maintenance by using part of software components running in big data centers. Unfortunately, the migration of a legacy system towards a service-oriented architecture is not a straightforward task. The first problem is that of establishing which part of the legacy system can be exposed as service. The second problem is that of establishing how the transformation will be done technically.

The most appropriate legacy systems for the migration towards Web services are those which are conceived as black-boxes that are callable through a command line and having a fixed-format input and output. Recently we have analyzed several cases that conforms to these characteristics – details are given in [10]. Unfortunately, the first problem mentioned above is not easy to be handled in the opposite case, that of migrating a legacy system with a rich user interface. We discuss in this paper such a case.

The second problem can be approached via several techniques. We review here them very shortly. A detailed analysis is presented in [5]. A first class of techniques comprises the black-box reengineering techniques which integrate systems via adaptors that wrap legacy code as service. A second class comprises white-box methods which require code analysis and modification to obtain the code components of the system to be presented as services.

The first class is mainly applied in the case when the code is not available. A recent paper on this subject is [1]. A solution for the particular case of interactive legacy systems is described in [4]. We have also proposed recently some technical solutions for the migration of the well-known interactive software tools used in the particular field of symbolic computations [6]. The second class mentioned above is based on invasive procedures on the legacy codes that usually improve the efficiency of legacy code. In this paper we make use of the third possible class, mentioned in [5], of the grey-box techniques, that combine wrapping and white-box approaches for integrating those parts of the system that are more valuable.

We present a case study on an interactive legacy system that provides numerical solutions for systems ordinary differential equations and incorporates an expert system, and which was designed ten years ago. The part of the legacy system that is the most computational intensive is migrated as Web service, while the user interface and the expert part are recoded in Java for portability reasons. Following this approach, the computational service can be accessed by any client that sends a message in a specific format containing the problem description and the method to be applied. Furthermore, the module that implements the parallel numerical methods, as one important component of the part wrapped as Web service, was extended to allow the efficient use of the multicore architectures. Taking into consideration the current trends to increase the number of processors on a chip, the extent to which software can be multithreaded to take advantage of the multicore chips is likely to be the main constraint on software performance in the future. Numerical computations requiring both CPU power and large memory are well-suited candidates for deriving advantages from multicore architectures. In this context, it is necessary to design and implement new libraries and tools for parallel numeric computations, especially for distributed-memory parallel computing environments using multicore processors.

One can notice that several parallel numeric computation packages were designed at the beginning of ’90s assuming a shared-memory parallel computing. The later evolution of the hardware towards distributed-memory parallel computers and clusters of workstations has lead to the impossibility to use the shared-memory parallel codes and to the need of designing and implement new versions that are suited for distributed memory. In particular, for the case of computing the numerical solutions of ordinary differential equations this architectural change had a tremendous effect – the class of parallel methods well suited for parallel implementation has been moved from that applying parallelism across method towards that applying parallelism across steps [3]. The question is if we can reconsider as efficient the parallelism across method by using the multicore architectures. We prove in this paper that there is a positive answer.

The paper is organized as follows. Section 2 describes shortly the system that is used as case study, while Section 3 presents the system’s computational component that is wrapped as Web service. The benefits of adding multithreaded functionality is discussed in Section 4. Finally, some conclusions are drawn in Section 5.

2. EpODE's Characteristics and Components

EpODE was designed as a tool for numerically solving large systems of ordinary differential equations (ODEs). It is also an expert system. EpODE provides not only an automatic identification of problem properties, but also of the properties of the solving method. Moreover, it can choose automatically the adequate method (including facilities for parallel computing in the case when the estimated time for solving the problem is too high). Furthermore, it can be also used as a tool for describing, analyzing and testing new types of iterative methods for ODEs, including those proposed for parallel or distributed implementation using real or simulated parallel machines. It is important to notice that EpODE is freely distributed with a rich database of test problems and of solving methods.

The main characteristics of EpODE which distinguish it from other ODE solving environments are the followings:

- friendly interpreter mode for describing problems and solving methods;
- the solvers are implemented in a uniform way: all solvers behave in a coherent way and have the same calling sequence and procedure;
- the tool is independent from other software packages with the exception of PVM used for parallel or distributed computations.

Details about EpODE design are given in [8]. Several experiments on parallel computers were reported in [9].

EpODE has five major components:

1. a user interface, the front end of which permits the description of an initial value problem for ODEs or an iterative method, the control of the solution computation process, and the interpretation of the results of the computation; help facilities are provided in order to assist the user in using the software;
2. a properties detection mechanism containing the procedures for establishing some properties of ODEs or those of an iterative method;
3. a mechanism for selecting the solving procedure, implementing a decision tree for the selection of the class of iterative methods according to the properties of the initial value problem for ODEs and for the selection of one method from this class according to the solution accuracy requirements and time restrictions;
4. a sequential computing procedure, a generic solving procedure whose parameters are selected according to the current problem and the serial method;
5. a parallel computing procedure, a generic solving procedure with message passing facilities for intercommunication of more than one computation process.

At the time of its design EpODE was the unique tool that allowed the above mentioned facilities. Only a recent developed tool reported in [2] has similar facilities (without the ones for parallelism). EpODE was written ten years ago in C++ and two graphical interfaces were provided, forWindows'95 and X Windows. One drawback is the fact that its interface is not portable. Moreover, the conclusions drawn relative to the efficiency of the parallel methods are no more valid due to the rapid development of the hardware. A rerun of the experiments reported in [9] revealed that the current hardware improvements leaded to a response time of the computational procedures hundreds time faster. In these conditions the problem dimension for which the parallelism across method is efficient (the computational time dominates the communication time) is increasing with at least ten times.

3. Wrapping the Computational Kernel as Web Service

The most intensive computational part of EpODE consists in the generic numerical solving procedure for sequential or parallel iterative methods applied to initial value problems for ordinary differential equations. The procedure is generic in the sense that it does not depend on the specific problem or the particular method – the concrete problem and methods are given as parameters. Since there is no need of user intervenience in the computational process, and simultaneously is a need for a fast response, this part of EpODE is well suited for transformation into a computational service lying on a remote high-performance server.

The component that implements in C++ and PVM the computational procedure is wrapped as a statefull Web service (WSRF implementation using Globus Toolkit 4 – see other wrapping examples in black-box style reported in [10]). The wrapper is written in Java and ensure the translation of the incoming client requests into the following actions:

1. write the problem and method descriptions provided as complex data structures, as well as supplementary information requested to proceed with the computation (e.g. the value of the method step, or the switch between sequential or parallel environment), into a file with a specific format;
2. transfer at the server site a given file with the problem, method, and computation parameters;
3. call the computational procedure through a line command that specifies the file;
4. retrieve an estimation of the computation time;
5. retrieve the status of the computation;
6. transfer the file with the computational results at the client side.

The container of the Web service is based on Tomcat technologies. Axis is used as implementation of the SOAP specification. The WSDL file of the service was generated with the Java2WSDL tool of Axis. The data structure describing the problem to be solved is present in the WSDL file:

```
<xsd:element name="Dim" type="xsd:int"/>
<xsd:element name="Vars" type="xsd1:ArrayStr"/>
<xsd:element name="Eqs" type="xsd1:ArrayStr"/>
<xsd:element name="BJacob" type="xsd1:ArrayBool"/>
<xsd:element name="Jacob" type="xsd1:ArrayStr"/>
<xsd:element name="T0" type="xsd:double"/>
<xsd:element name="InitV" type="xsd1:ArrayStr"/>
```

where ArrayStr for example is described in the <types> part of the WSDL:

```
<complexType name="ArrayStr">
      <complexContent>
            <restriction base="soapenc:Array">
                  <attribute ref="soapenc:arrayType"
                        wsdl:arrayType="string[]"/>
            </restriction>
      </complexContent>
</complexType>
```

Dim is the problem dimension, Vars is a vector with the problem variables, Eqs are the differential equations, BJacob is a Boolean matrix indicating the non-zero positions in the Jacobian matrix of the system, Jacob are the non-zero elements of the Jacobian matrix of the system, T0 is the initial value of the independent variable, and InitV is the vector of the initial values.

The data structure describing the method to be applied is more complex and cannot be described completely:

```
<xsd:element name="Implicit" type="xsd:Boolean"/>
<xsd:element name="MStep" type="xsd:Boolean"/>
<xsd:element name="MStage" type="xsd:Boolean"/>
<xsd:element name="MDeriv" type="xsd:Boolean"/>
<xsd:element name="Newton" type="xsd:Boolean"/>
<xsd:element name="Nsta" type="xsd:int"/>
<xsd:element name="Nfin" type="xsd:int"/>
<xsd:element name="Nplu" type="xsd:int"/>
<xsd:element name="Mpas" type="xsd:int"/>
...
<xsd:element name="VarMet" type="xsd1:ArrayStr"/>
...
<xsd:element name="FinEqs" type="xsd1:ArrayStr"/>
<xsd:element name="PluEqs" type="xsd1:ArrayStr"/>
...
```

The above data structure fields are referring some method properties: is implicit or explicit, is multistep or not, is multistage or note, is multiderivative or not, etc.

The data structure describing the options for the computations includes:

```
<xsd:element name="Step" type="xsd:double"/>
<xsd:element name="T1" type="xsd:double"/>
<xsd:element name="WhichV" type="xsd1:ArrayStr"/>
<xsd:element name="DisplaySteps" type="xsd:int"/>
<xsd:element name="PVM" type="xsd:Boolean"/>
...
```

In the case when PVM is set the computation will be done using the computational procedure for parallel methods that spawn the computational power between a small number of CPUs according to the degree of the method's parallelism that is automatically detected by the computational procedure.

Note that in the current implementation the equations should be provide in the Polish form (intermediate storage form in EpODE), but this inconvenient should be removed soon. Moreover, for the sake of the initial testing, only the option for parallelism across method is activated through the interface. Further developments will include into the computational kernel the EpODE's component that transforms any expression in its Polish form, the EpODE's facilities for parallelism across steps and parallelism across problem.

In order to ensure the portability, the EpODE's GUI and expert system are under a rebuilt process that uses Java technologies. At this moment the GUI has been almost entirely ported as Java frame and will be soon distributed with the new version of EpODE. The distributed computing facility that has been based on the usage of local networks of workstations through PVM, is now extended with a facility to send requests to the Web service in the following order. The EpODE's GUI user selects or describe the problem and optionally the method. Then he or she describes the computational parameters or approve the ones recommended by the expert system. In the case when the response time of the computational procedure is estimated to be at least of minutes order, the user can decide to call the Web service (press a button): a file with the problem, method and computation parameters will be generated and send to the Web service (the location of the container is know to the GUI).

As response, the new estimation of the computation time is received. The status of the computations can be found through a new request addressed to the Web service. Finally the numerical results can be retrieved in a file that is further displayed in the user interface as a list of values or can be interpreted by the visualization module. The client of the Web service can be also another tool that sends the input data in the requested format or generates an appropriate parameter file. We imagine the case when the Web service is called, for example, by another numerical software code that solves partial differential equations and during its solving procedure it transforms the problem into a large system of ordinary equations. Note that the largest ODE systems that are usually used in testing ODE software tools are provided by a such discretization process [3]. Moreover, the symbolic described Jacobian requested by EpODE computational procedures can be easily generated with a computer algebra system – for example an older software tool, ODEXPERT [7] uses Maple for this task.

We can further imagine a more complex scenario in which several Web services are composed: one that is generating the ODE system, another for computing the Jacobian, both wrapped as Web services, are sending the necessary information for the expert system (also aWeb service) that picks an appropriate method from a rich database (can be the EpODE component) and ask the above described Web service to perform the computation, and finally sends the numeric computation results to a visualization tool that is also wrapped as Web service. This scenario will be the basis of our future development.

4. Adding a Multi-threaded Facility for Multicore Architectures

EpODE was designed to allow the experimentation of parallel methods when solving initial value problems for ordinary differential equations. As mentioned above there are three classical approaches: parallelism across problem that depends on the degree on the sparsity of the system's Jacobian matrix, parallelism across method that depends on the number of the method variables that can be computed simultaneous, and parallelism across steps that allows a higher degree of parallelism with the drawback of heavy control of the convergency of the numerical solutions towards the exact one.

The parallelism across method was a viable solution ten years ago in the case of large systems. With the increase of the computational power faster than the communication speed, parallel computations based on parallelism across method are justified only in the case of systems with hundreds of equations. Indeed, we have re-run the experiments reported in [9] dealing with systems of almost one hundred equations on a new generation cluster (with 7 HP ProLiant DL-385 with 2 x CPU AMD Opteron 2.4 GHz, dual core, 1 MB L2 cache per core, 4 GB DDRAM, 2 network cards 1 Gb/s) and the results show that the parallel variant is no more efficient.

TABLE 1. Response times of the computational procedure with or without threads

Problem	Method		No.	Time (ms)	
	Acronym	Parallel. degree	steps	no threads	with threads
Plate81	DIRK4	2	5	2031	1519
ME140	FR2	2	50	3658	2068
	PC1	2	50	5165	3624
	PC6	2	50	10022	4992
	BL1	2	50	3428	2768
	BL2	3	10	3119	1431

The question is if we can still improve the efficiency of the computational procedures implementing parallelism across method by using multithreading when running on multicore architectures. To be able to answer to this question, we have rewrite some parts of the C++ code for the computational procedures of EpODE. The multithreading implementation is close to that based on the PVM library – instead PVM processes, threads are used, and instead message passing, threads are communicating through a common matrix.

The time response of the computational procedure is clearly improved using the multithreaded version. Table 1 shows the time results in the case of two classical problems of 81, respectively 140 equations solved by representative methods from different classes of parallel methods. DIRK4 is a 4-stage 4th order Diagonally Implicit Runge-Kutta method, PC1 is the predictor-corrector scheme based on the implicit trapezoidal rule, PC6 is another predictor-corrector scheme, while BL1 and BL2 are one-stage block methods, all of them available through the rich database of methods provided by EpODE. ME140 is a discretization of the Medical Akzo Nobel problem, using the method of lines. Plate81 is obtained suing the same procedure starting from a diffusion problem. Please refer to [3, 9] for the description of these problems and methods.

In order to incorporate the multithreading facility into the Web service described in the previous section, the following field should change its description:

```
<xsd:element name="PVM" type="xsd:int"/>
```

set on 1 when PVM is used (recommended for systems of hundred orders of equations), set on 2 when multithreading is used (recommended for systems of ten orders), and set on 0 when parallelism facilities are not used (recommended for small systems). The Web service is available in a container running on the interface node of the above mentioned multicore cluster, so each option can be properly exploited.

5. Conclusions

In order to prolong the lifetime of a legacy code we have used a grey-box technique for migrating it towards a service-oriented architecture. One of its unique components, the one that can profit from computational power of remote high-performance servers, was wrapped as Web service. This service can be accessed by the interface of the expert system redesigned due to portability issues or by a simple client that respects the format of the input data. The migration opens new possibilities to exploit the facilities provided by the legacy code by combining it with other services to offer more complex computational scientific services.

The transition towards the new version of the expert system is not completed. While the computational kernel was successfully adapted to make efficient use of multicore architectures, several other components are still remaining to be translated into the new user interface. The improvement and the integration of the expert system into the same Web service or another Web service is one of the next steps to be followed soon. Moreover, intensive tests should be completed before releasing the new free version. Complex usage scenarios, as the one described in Section 2, should be the context of these intensive tests.

Acknowledgments. The project no. 11064, PEGAF, of the Romanian PNII-Partnership Programme has supported part of the research for this paper.

References

[1]. B. Balis, M. Bubak, M.Wegiel, A Solution for Adapting Legacy Code as Web Services, Component Models and Systems for Grid Applications, V. Getov, T. Kiellmann (eds.), Springer (2005), 57-75.

[2]. B. Bunus, A Simulation and Decision Framework for Selection of Numerical Solvers in Scientific Computing, Procs. Annual Simulation Symposium vol. 39, IEEE Computer Press (2006), 178-187.

[3]. K. Burrage, Parallel and Sequential Methods for Ordinary Differential Equations, Numerical Mathematics and Scientific Computation, Oxford University Press, 1995.

[4]. G. Canfora, A.R. Fasolino, G. Frattolillo, P. Tramontana, Migrating Interactive Legacy System to Web Services, Procs. 10th European Conference on Software Maintenance and Reengineering, IEEE Computer Press (2006), 23-32.

[5]. G. Canfora, A.R. Fasolino, G. Frattolillo, P. Tramontana, A Wrapping Approach for Migrating Legacy System Interactive Functionalities to Service Oriented Architectures, J. Syst. Software, in print (2007).

[6]. A. Carstea, M. Frincu, G. Macariu, D. Petcu, K. Hammond, Generic Access to Web and Grid-based Symbolic Computing Services" Procs. ISPDC 2007, IEEE Computer Press (2007), 143-150.

[7]. M.S. Kamel, K.S. Ma, W.H. Enright, ODEXPERT - An Expert System to Select Numerical Solvers for Initial Value ODE Systems, ACM Transactions on Mathematical Software, vol. 19: 1 (1993), 44-62.

[8]. D. Petcu, M. Dragan, Designing an ODE Solving Environment, Lectures Notes in Computational Science and Engineering 10: H.P. Langtangen, A.M. Bruaset and E. Quak (eds.), Springer-Verlag, Berlin (2000), 319-338.

[9]. D. Petcu, Experiments with an ODE Solver on a Multiprocessor System, Computers & Mathematics with Appls. 42 (8-9), Pergamon-Elsevier Science (2001), 1189-1199.

[10]. D. Petcu, A. Eckstein and C. Giurgiu, Using Statefull Web Services to Expose the Functionality of Legacy Software Codes, Procs. SACCS 2007, Iasi (2007), 257–263.

Discovery Linguistic Services in a GRID Environment

Adrian Iftene

"Alexandru Ioan Cuza" University of Iasi, Romania
adiftene@info.uaic.ro

Abstract. *In the last years the computational Grids have become an important research area in large-scale scientific and engineering research. Our approach is based on Peer-to-peer (P2P) networks, which are recognized as one of the most used architectures in order to achieve scalability in key components of Grid systems. The main goal in using a computational Grid was to improve the computational speed of systems that solve complex problems from Natural Language processing field. One of the important components of our system will be the resource discovery component, whose duty is to provide system-wide up-to-date information about P2P peers, linguistic resources and tools, and Grid services.*

1. Introduction

Solving complex problems has become a usual task in the Natural Language Processing domain, where it is normal to use large information databases like lexicons, semantic relations, dictionaries. Our solution for solving some complex problems is related to using a Grid system with a high computational power, similar to Grids presented in [1, 2]. To achieve their envisioned global-scale deployment, Grid systems need to be scalable. Peer-to-peer (P2P) techniques are widely viewed as one of the prominent ways to reach the desired scalability.

Resource discovery is one of the most important functionalities of a Grid system and, at the same time, one of the most difficult to scale. Indeed, the duty of a resource discovery system (such as the Globus MDS [3]) is to provide system-wide up-to-date information, a task which has inherently limited scalability. To add to the challenge, Grid resource discovery systems need to manage not only static resources, but also resources whose characteristics change dynamically over time, making the design critical.

We will see how we can configure this peer-to-peer system in order to find the global solution for the RTE3 [4] competition task. The system architecture is based on the peer-to-peer model, using the Server Message Block (SMB) protocol [5] for file transfer.

After the peer-to-peer network is configured, one computer becomes the initiator and builds the list of available neighbors. Subsequently, the initiator has the following additional roles: split the initial problem into sub-problems, send the sub-problems to the list of neighbors for solving, receive the partial output files and build the final solution.

One on the most important remaining problem is related to system configuration aspects. For every computer from our network we must configure a local cache zone with information about the network, linguistic databases and tools, and Grid services. Until now, this configuration was done manually for every computer from our network. With configuration part we spend several minutes for a network with around 10 computers, while a single run took only few seconds. From this reason our current work is related to P2P discovery methods using PDP protocol from JXTA (short for Juxtapose) project [6].

2. Peer-to-Peer technologies

Peer-to-Peer (P2P) technologies enable any network-aware device to provide services to another network-aware device. A device in a P2P network can provide access to any type of resource that it has at its disposal: documents, storage capacity, computing power, or even its own human operator. Most Internet services are distributed using the traditional client/server architecture (see Fig. 1).

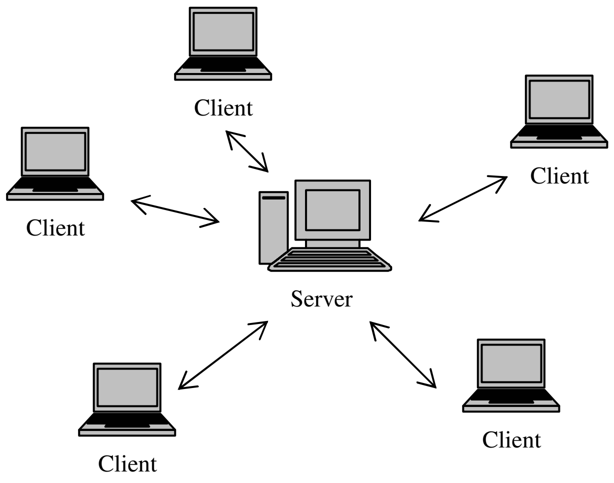


Fig. 1. Client/Server architecture

In this architecture, clients connect to a server using a specific communication protocol, such as the File Transfer Protocol (FTP), to obtain access to a specific resource.

Most of the processing involved in delivering a service usually occurs on the server, leaving the client relatively unburdened. Most popular Internet applications, including the World Wide Web, FTP, telnet, and email, use this service-delivery model.

Unfortunately, this architecture has a major drawback. As the number of clients increases, the load and bandwidth demands on the server also increase, eventually preventing the server from handling additional clients. The advantage of this architecture is that it requires less computational power on the client side.

The client in the client/server architecture acts in a passive role, capable of demanding services from servers but incapable of providing services to other clients. This model of service delivery was developed at a time when most machines on the Internet had a resolvable static IP address, meaning that all machines on the Internet could find each other easily using a simple name. If all machines on the network ran both a server and a client, they formed the foundation of a rudimentary P2P network (see Fig. 2).

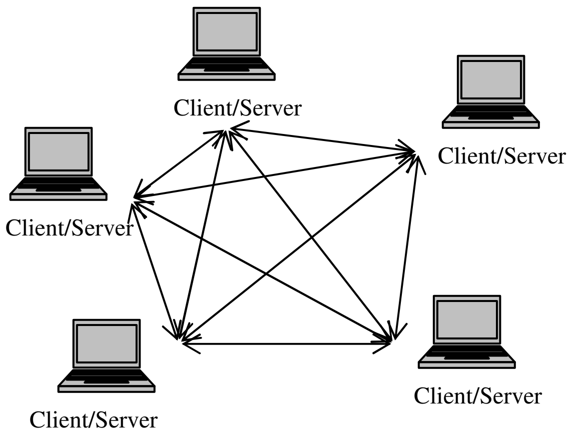


Fig. 2. Peer-to-peer architecture.

The main advantage of P2P networks is that they distribute the responsibility of providing services among all peers on the network; this eliminates service outages due to a single point of failure and provides a more scalable solution for offering services. In addition, P2P networks exploit available bandwidth across the entire network by using a variety of communication channels and by filling bandwidth to the “edge” of the Internet. Unlike traditional client/server communication, in which specific routes to popular destinations can become overtaxed, P2P enables communication via a variety of network routes, thereby reducing network congestion.

P2P has the capability of serving resources with high availability at a much lower cost while maximizing the use of resources from every peer connected to the P2P network. Whereas client/server solutions rely on the addition of costly bandwidth, equipment, and co-location facilities to maintain a robust solution, P2P can offer a similar level of robustness by spreading network and resource demands across the P2P network.

Distributed computing provides a way of solving difficult problems by splitting the problem into sub-problems that can be solved independently by a large number of computers. Our previous work was concerned with the parallel graph coloring approach in a P2P environment [7], and our recent efforts address the Natural Language Processing (NLP) area [8]. The main scope in both approaches was to use the computational power of the P2P network in order to increase the computational speed.

JXTA Project

In April 2001, Bill Joy placed Project JXTA in the hands of the P2P development community by adopting a license based on the Apache Software License Version 1.1. Currently, Project JXTA has a reference implementation available in Java, with implementations in C, Objective-C, Ruby, and Perl 5.0 under way.

The JXTA v1.0 Protocols Specification defines the basic building blocks and protocols of P2P networking:

- **Peer Discovery Protocol** – Enables peers to discover peer services on the network,
- **Peer Resolver Protocol** – Allows peers to send and process generic requests,
- **Rendezvous Protocol** – Handles the details of propagating messages between peers,
- **Peer Information Protocol** – Provides peers with a way to obtain status information from other peers on the network,
- **Pipe Binding Protocol** – Provides a mechanism to bind a virtual communication channel to a peer endpoint,
- **Endpoint Routing Protocol** – Provides a set of messages used to enable message routing from a source peer to a destination peer.

The JXTA protocols are language-independent, defining a set of XML messages to coordinate some aspect of P2P networking. Although some developers in the P2P community are reluctant in using of such a verbose language, the choice of XML allows implementers of the JXTA protocols to leverage existing toolsets for XML parsing and formatting.

3. Components of P2P Networks in JXTA

Peers

A *peer* is any node of a P2P network that forms the basic processing unit. Its definition from [9] is: "*Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly*."

The meaning of *useful work* depends on the type of peer. Three possible types of peers exist in any P2P network: **simple peers** (designed to serve a single end user), **rendezvous peers** (provide peers with a network location used for discover of other peers and peers resources) and **router peers** (provide mechanisms for peers to communicate with other peers separated from the network).

Peer Groups

A *peer group* is defined as follows [9]: "*A set of peers formed to serve a common interest or goal dictated by the peers involved. Peer groups can provide services to their member peers that aren't accessible by other peers in the P2P network.*"

Peer groups divide the P2P network into groups of peers with common goals based on the following:

- The **application** they want to collaborate on as a group,
- The **security** requirements of the peers involved,
- The need for **status information** on members of the group.

Peer group members can provide redundant access to a service, ensuring that a service is always available to a peer group as long as at least one member is providing the service.

Network Transport

To exchange data, peers must employ some type of mechanism to handle the transmission of data over the network. This layer, called the *network transport*, is responsible for all aspects of data transmission, including breaking the data into manageable packets, adding appropriate headers to a packet to control its destination, and in some cases, ensuring that a packet arrives at its destination.

The concept of a network transport in P2P can be broken into three constituent parts:

- **Endpoints** – The initial source or final destination of any piece of data being transmitted over the network.
- **Pipes** – Unidirectional, asynchronous, virtual communication channels connecting two or more endpoints.
- **Messages** – Containers for data being transmitted over a pipe from one endpoint to another.

To communicate using a pipe, a peer first needs to find the endpoints, one for the source of the message and one for each destination of the message, and connect them by binding a pipe to each of the endpoints.

Services

Services provide functionality that peers can engage to perform "useful work" on a remote peer. This work might include transferring a file, providing status information, performing a calculation, or basically doing anything that you might want a peer in a P2P network to be capable of doing. Services can be divided into two categories:

- **Peer services** – Functionality offered by a particular peer on the network to other peers.
- **Peer group services** – Functionality offered by a peer group to members of the peer group.

These *core services* provide the basic P2P foundation used to build other, more complex services.

Protocols

Every data exchange relies on a protocol to dictate what data gets sent and in what order it gets sent. A *protocol* is simply this [9]: "*A way of structuring the exchange of information between two or more parties using rules that have previously been agreed upon by all parties.*"

In P2P, protocols are needed to define every type of interaction that a peer can perform as part of the P2P network: *finding peers* on the network, *finding what services* a peer provides, *obtaining status information* from a peer, *invoking a service* on a peer, *creating, joining, and leaving peer groups*, *creating data connections* to peers, *routing messages* for other peers.

4. Discovery Advertisements in JXTA

A fundamental problem in P2P is "*How does a device find peers and services on a P2P network*?" The question is important because, without the knowledge of the existence of a peer or a service on the network, there's no possibility for a device to engage that service.

Finding Advertisements

Any of the basic building blocks discussed in the last section can be represented as an advertisement, and that characteristic considerably simplifies the problem of finding peers, peer groups, services, pipes, and endpoints. A peer can discover an advertisement in three ways:

- **No discovery** – Instead of actively searching for advertisements on the network, a peer can rely on a cache of previously discovered advertisements to provide information on peer resources.
- **Direct discovery** – Peers that exist on the same LAN might be capable of discovering each other directly without relying on an intermediate rendezvous peer to aid the discovery process. Direct discovery requires peers to use the broadcast or multicasting capabilities of their native network transport.
- **Indirect discovery** – Indirect discovery requires using a rendezvous peer to act as a source of known peers and advertisements, and to perform discovery on a peer's behalf.

Rendezvous peers provide peers with two possible ways of locating peers and other advertisements:

- **Propagation** – A rendezvous peer passes the discovery request to other peers on the network.
- **Cached advertisements** – A rendezvous can use cached advertisements to respond to a peer's discovery queries.

JXTA Peer Discovery Protocol

The Peer Discovery Protocol (PDP) defines a protocol for requesting advertisements from other peers and responding to other peers' requests for advertisements. The Peer Discovery Protocol consists of only two messages that define the following:

- A *request format* to use to discover advertisements,
- A *response format* for responding to a discovery request.

These two message formats, the *Discovery Query Message* and the *Discovery Response Message*, define all the elements required to perform a discovery transaction between two peers.

The Discovery Query Message

The Discovery Query Message is sent to other peers to find advertisements. It has a simple format, as shown in next table.

TABLE 1. The Discovery Query Message XML

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:DiscoveryQuery>
   <Type> . . . </Type>
   <Threshold> . . . </Threshold>
   <PeerAdv> . . .</PeerAdv>
   <Attr> . . . </Attr>
   <Value> . . .</Value>
</jxta:DiscoveryQuery>
```

The elements of the Discovery Query Message describe the discovery parameters for the query. Only advertisements that match all the requirements described by the query's discovery parameters are returned by a peer.

The Discovery Response Message

To reply to a Discovery Query Message, a peer creates a Discovery Response Message that contains advertisements that match the query's search criteria, such as the Attr/Value combination or Type of advertisement. The Discovery Response Message is formatted as shown in Table 2.

TABLE 2. The Discovery Response Message XML

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:DiscoveryResponse>
   <Type> . . . </Type>
   <Count> . . . </Count>
   <PeerAdv> . . . </PeerAdv>
   <Attr> . . . </Attr>
   <Value> . . . </Value>
   <Response Expiration="expiration time"> . . .  </Response>
</jxta:DiscoveryResponse>
```

The elements of the Discovery Response Message closely correspond to those of the Discovery Query Message.

5. The NLP System

The Recognition of Textual Entailment (RTE [10]) competition is organized by PASCAL (Pattern Analysis, Statistical Modelling and Computational Learning) [11] - the European Commission's IST-funded Network of Excellence for Multimodal Interfaces. This year the challenge arrived at its third edition. Competitors receive 800 pairs of text called *Text* and *Hypothesis*, and they must decide for every pair if the *Hypothesis* can be entailed from the *Text* (i.e. we have textual entailment (TE) for a specific pair).

Our system P2P architecture is based on CAN model [7]. The system presented below consists of more core modules (CMs), linguistic tools and databases of linguistic resources. In order to solve the task from RTE3 competition we must connect to a computer from this computational Grid in order to initiate the solving of the problem.

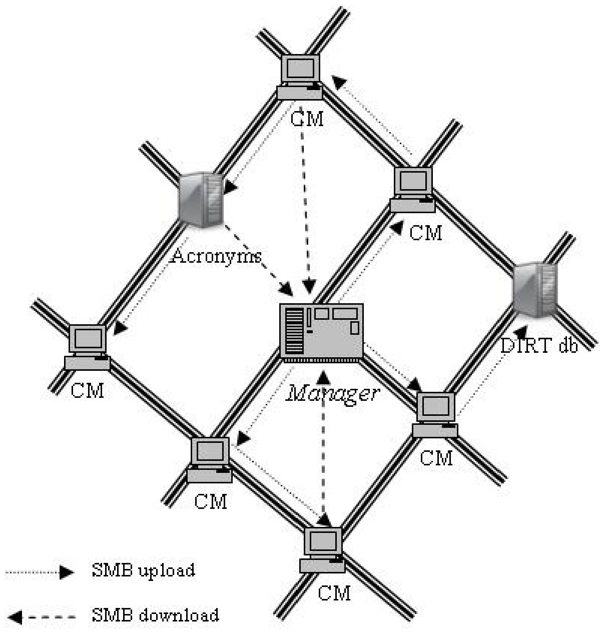


Fig. 3. P2P Network for RTE3 competition

The main **linguistic tools** used are: *LingPipe* [12] for name entities (NEs) identification and *Minipar* [13] for dependency tree building. The **linguistic resources** are *Extended WordNet*, *DIRT,* the *Acronyms Database* and the *Background Knowledge*.

- For non-verbs words from the hypothesis, if in the text we do not have words with the same lemma, we search for their synonyms in the *Extended WordNet* [14].
- If the word is a verb in the hypothesis, we use the *DIRT* resource [15] in order to transform the hypothesis into an equivalent one, with the same words except the verb. Our aim in performing this transformation is to find a new value for the verb which can be better mapped in the text.
- The *acronyms' database* [16] helps our program to find relations between the acronym and its meaning, for example "US - United States".
- *Background knowledge* was built semi-automatically, for the NEs and for numbers from the hypothesis without correspondence in the text. For these NEs, we used a module to extract from Wikipedia [17] snippets with information related to them. Subsequently, we use this file with snippets and some previously set patterns of relations between NEs, with the goal to identify a known relation between the NE for which we have a problem and another NE.

Between CMs, the upload and download operations are done using a special component based on the SMB protocol.

Any computer from this network can initiate the solving of the RTE task and it becomes the *Initiator*. First of all it checks its list with neighbors in order to know the number of computers which can be involved in the problem solving (the future CMs). After that it updates all these CMs with the last version of the TE module. In parallel, all pairs are sent to the LingPipe and Minipar modules, which send back the pairs on which the central module can run the TE module. After these steps, the initial problem (consisting of 800 pairs) was split in sub-problems (a range between the number of the first and last pair) according to the number of neighbors and using a dynamic quota. At first, this quota has an initial value, but in time it is decreased and eventually becomes the default minimal value. The main goal of using this quota is to send any neighbor a number of problems according to its computational power and to its load at runtime. In the end, the partial results from the other computers are downloaded to the initiator and are used in order to build the final solution.

An important step in building the system was the initial system configuration. At this step for every computer from this network we perform manually the following preparation steps:

- We add the "**neighbors**" in CAN terminology in the first zone of computer cache (IPs of computers from immediate proximity). The computer node will do a static discovery of the network through this cache, but in all next requests this computer will access the entire network only via these computers (indirect discovery).
- We add the addresses of "**NLP tools and resources**" in the second zone of computer cache (IPs of computers from entire P2P network). These tools and resources will be used by current CM in order to solve sub-problems assigned by the *Initiator*.
- We add the addresses of "**GRID services**" in the third zone of computer cache (computers IP from P2P network or from Internet). These Grid services are used for solving of sub-problems or for sending and receiving of sub-problems [18].

The big problems of this step come from the time consumption during the configuration of all CMs and from problems that appear in failure cases. If one computer from this network is temporary unavailable and another computer wants to access its tools, resources or services, after a few tries without success, the address of the computer with problems will be removed from computer cache. This situation conducts to the impossibility of using this computer in the future, without a reconfiguration of all neighbors caching.

In order to solve the configuration-reconfiguration inconvenient, a *special node* from our P2P network communicates with GRID server and it is informed by available GRID services. After that, this special node sends advertisements into the P2P network with following information:

<*service name, service address, service input parameters, service output values*>

Of course, it is possible like the same *service name* to be available on different computers with different addresses.

What we want to do in the next future? To every CM from the P2P network we will add a special component responsible with service discovery. This component will communicate with *special node* from P2P network. For that it will send *Discovery Query Messages* accordingly with node necessities into the P2P network, and will be able to process *Discovery Response Messages*.

6. Conclusions

The configuration of the network seems to be the most consuming task from our system. We worked in a network with different system configurations of computers (2 dual cores - running two processes, 3 normal - running one process and 1 Grid server). For every computer from P2P architecture we must specify three layers of cache: for neighbors, NLP tools and resources and for Grid services.

Also, in the current solution any failure problem of the network or any new adding of a computer in the current P2P architecture requires at least a local reconfiguration, or even a full configuration of all network.

PDP protocol from JXTA protocols is able to request advertisements from other peers and to respond to other peers' requests for advertisements. This work is done through two message formats, the *Discovery Query Message* and the *Discovery Response Message*. These messages define all the elements required to perform a discovery transaction between two peers.

In current work we use the JXTA Shell [19] and want to use through this shell the PDP protocol for cache building. The JXTA Shell is a demo application built on top of the JXTA platform that allows users to experiment with the functionality made available through the Java reference implementation of the JXTA protocols. The JXTA Shell provides a UNIX-like command-line interface that allows the user to perform P2P operations by manipulating peers, peer groups, and pipes using simple commands.

In the future, we want to have possibility to discover automatically all available GRID services from a P2P network. Also, the peers will have possibility to discover and to select between available GRID services, and in the case of failure of a node from the network, its will can change automatically the service provider.

Acknowledgment: The author thanks the members of the NLP group in Iasi for their help and support at different stages of the system development. The work on this project is partially financed by the CEEX GRAI (Grid Computing and Artificial Intelligence) project number 74 and by Siemens VDO Iaşi.

References

[1]. Geoffrey C. Fox, Denis Gannon, Computational Grids, Computing in Science and Engineering., Vol 3, No. 4, (2001), 74–77
[2]. Dennis Gannon and Andrew Grimshaw, Object-Based Approaches, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., (1999), 205–236
[3]. Globus MSD: http://www.globus.org/toolkit/mds
[4]. RTE3 Competition: http://www.pascal-network.org/Challenges/RTE3/
[5]. SMB Protocol: http://en.wikipedia.org/wiki/Server_Message_Block
[6]. JXTA Project: https://jxta.dev.java.net/
[7]. Adrian Iftene and Cornelius Croitoru, Graph Coloring using Peer-to-Peer Networks, In Proceedings, of 5th International Conference RoEduNet IEEE, Sibiu, Romania, (2006), 181–185
[8]. Adrian Iftene, Alexandra Balahur-Dobrescu, and Daniel Matei, A Distributed Architecture System for Recognizing Textual Entailment, In proceedings of 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE Computer Society , Timisoara, Romania, (2007), 219–226
[9]. Brendon J. Wilson, JXTA, Pearson Education (2002)
[10]. RTE site: http://www.pascal-network.org/Challenges/RTE/
[11]. Pascal: http://www.pascal-network.org/
[12]. LingPipe: http://www.alias-i.com/lingpipe/
[13]. Dekang Lin, Dependency-based Evaluation of MINIPAR, In Workshop on the Evaluation of Parsing Systems, Granada, Spain, (1998), 234–241
[14]. eXtended WordNet: http://xwn.hlt.utdallas.edu/downloads.html
[15]. Dekang Lin, and Patrick Pantel, DIRT - Discovery of Inference Rules from Text, In Proceedings of ACM Conference on Knowledge Discovery and Data Mining (KDD-01), San Francisco, CA. (2001), 323–328
[16]. Acronym Database: http://www.acronym-guide.com
[17]. English Wikipedia: http://en.wikipedia.org/wiki/Main_Page

[18]. Ionuț Cristian Pistol, Adrian Iftene, Linguistic Processing Architecture in a GRID Environment, In Proceedings of Workshop on Distributed and Parallel Computing and Systems, ECIT2008, Iasi, Romania (2008)
[19]. JXTA Shell: http://download.java.net/jxta/jxta-jxse/2.5/jnlp/shell.jnlp

Linguistic Processing Architecture in a Grid Environment

Ionuț Cristian Pistol, Adrian Iftene

"Alexandru Ioan Cuza" University of Iaşi, Romania
{ipistol, adiftene}@info.uaic.ro

Abstract. *This paper describes the planned integration of ALPE (Automated Linguistic Processing Environment) - a system designed to facilitate the management and usage of large and dynamic collections of linguistic resources and tools – in a Grid environment. ALPE can be used to build linguistic processing chains involving the annotation formats and tools integrated into a hierarchical structure. A Grid network will reduce the processing times and will allow the users a greater flexibility regarding the selection and usage of processing chains. The particularities, advantages and usage of integrating ALPE in a project involving the development and utilization of multiple linguistic resources is the other main topic of this paper.*

Keywords: *Linguistic resources, Grid environment, Processing architectures*

1. Introduction

One of the latest developments in Natural Language Processing, and one which promises to have a significant impact for future linguistic processing systems, is the emerging of linguistic annotation meta-systems, which make use of existing processing tools and implement some sort of processing path, pipelined or otherwise. The newly emerged linguistic processing (LP) meta-systems make use of existing modules in building LP chains, use existing linguistic resources, and allow the user to add/build new ones and also compare and choose between different available modules. The two most prominent systems of this type are GATE[1] and IBM's UIMA[2].

GATE [3, 4] is a versatile environment for building and deploying NLP[3] software and resources, allowing for the integration of a large amount of built-ins in new processing pipelines that receive as input a single document or corpora. The user can configure new

[1] General Architecture for Text Engineering: http://gate.ac.uk/
[2] Unstructured Information Management Architecture: http://www.research.ibm.com/UIMA/
[3] Natural Language Processing

architectures by selecting form a repository pool the desired modules, as parts of a processing chain. The configured chain of processes may be put to work on an input file and the result is an output file, XML annotated.

UIMA [6] is a product of IBM research and offers the same general functionalities as GATE, but once a processing module is integrated in UIMA it can be used in any further chains without any modifications (GATE requires wrappers to be written to allow two new modules to be connected in a chain). Also, UIMA allows the user to work with various annotation formats and perform various additional operations on annotated corpora. Since the appearance of UIMA, the GATE developers have made available a module that allows GATE and UIMA processing modules to be interchangeable, basically merging the “pool” of modules available.

ALPE is another system, currently in development, that offers another approach to the task of developing a LP meta-system, one that plans to offer more accessibility and flexibility than existing systems. ALPE is based on the hierarchy of annotation schemas described in [1]. In this model, XML annotation schemas are nodes in a directed acyclic graph, and the hierarchical links are subsumption relations between schemas. In [2] is described how the graph may be augmented with processing power by marking edges linking parent nodes to daughter nodes with processors, each realizing an elementary NL processing step.

ALPE can offer several advantages over existent systems with a similar scope, as it is able to identify the format of annotated corpora, then to automatically compute and run the processing steps required to bring an input file to the required output format. The planned development of ALPE as a Grid processing resource will further differentiate ALPE from the existing systems and will accentuate some of its design benefits, most importantly the end user experience.

In the last years the computational Grids [11, 12] have become an important research area in large-scale scientific and engineering research. The computational Grids offer a set of services that allow a widely distributed collection of resources to be tied together into a relatively seamless computing framework, teams of researchers can collaborate to solve problems that they could not have attempted before. Unfortunately, after years of experience in this area, the task of building Grid applications still remains extremely difficult, mainly because there are few tools available to support developers.

Many of the big ideas behind the Grid have been around long before the name Grid appeared, but there are five areas [13, 14] where Grid developers are spending their time and effort:

- *Resource sharing* on a global scale: Sharing is the very essence of the Grid.
- *Secure Access:* There must be a high level of trust between resource providers and users, who often don't know each other. Sharing resources is fundamentally in conflict with the conservative security policies being applied at individual computer centers and on individual PCs. So getting Grid security right is crucial.
- *Resource use*: Demand for Grid resources should be balanced, so that computers everywhere are used more efficiently.

- *The death of distance:* For Grids to work, we need to ensure that distance makes no difference to efficient access to computer resources.
- *Open standards:* Open standards are needed to ensure that everyone can contribute constructively to Grid development, and that industry will be prepared to invest in developing commercial Grid services and infrastructure.

Section two of this paper presents the theoretical foundation of the implemented system, as well as the general functionalities offered by ALPE. Section three shows the benefits of a Grid integration of ALPE, as well as the current plans and stage of development. The conclusions, as well as the further planned developments are described in section four.

2. ALPE

2.1 ALPE Type Hierarchies

A direct acyclic graph (DAG) is described in [1], and it configures the metadata of linguistic annotation in a hierarchy of XML schemas. Nodes of the graph are distinct XML annotation schemas, while edges are hierarchical relations between schemas. Users' interactions with the graph can modify it from an initial trivial shape, which includes just one empty annotation schema, up to a huge graph accommodating a diversity of annotation needs. If there is an oriented edge linking a node A with a node B in the hierarchy (we will say also that B is a descendant of A) then the following conditions hold simultaneously:

- any tag-name of A is also in B;
- any attribute in the list of attributes of a tag-name in A is also in the list of attributes of the same tag-name of B;

As such, a hierarchical relation between a node A and one descendant B describes B as an annotation schema which is more informative than A. In general, either B has at least one tag-name which is not in A, and/or there is at least one tag-name in B such that at least one attribute in its list of attributes is not in the list of attributes of the homonymous tag-name in A. We will agree to use the term path in this DAG with its meaning from the support graph, i.e. a path between the nodes A and B in the graph is the sequence of adjacent edges, irrespective of their orientation, which links nodes A and B. As we will see later, the way this graph is being built triggers its property of being fully connected. This means that, if edges are seen undirected, there is always at least one path linking any two nodes.

Modern software engineering design uses interchangeable modules, which are interconnected in complex processing architectures. In NLP, this approach has proven advantages with respect to reusability, and language and application independence. In such a view, each module has inputs, outputs and accesses resources. In order for the modules to be truly inter-connectable, each of the module's inputs and outputs must observe the constraints of certain annotation schemas. Usually the language and, sometimes, application dependence, of a module is given by the specific set of resources it accesses. For instance, a POS-tagger, runs the same algorithms on different sets of language models in order to tag documents for POS in different languages. For the system builder, the real functionality of a module can be obscured in a black box, since is it fully determined by the triplet: input, output and resources. This is equivalent with saying that given a triplet of schemas, characterizing the input, the resources and the output, there should be a module which produces as output a file observing the restrictions of the output schema, whenever it receives as input a file observing the restrictions described by the input schema, and accesses resources observing the resources schema. This way, the hierarchy of annotation schemas becomes a graph of interconnecting modules. We will call a graph of annotation schemas on which processing modules have been marked on edges as being augmented with processing power (or simply, augmented).

Sometimes the existence of a process attached to an edge in the graph depends on the existence of adequate resources. For instance, one may have access to an automatic tagger, but it will not be able to apply it for a language L because of the lack of a language model (a resource) adequate for that language. This way, in a repository of resources and instruments dedicated to NLP, the maximal graph of annotation schemas hosted can have different instantiations for different languages, depending on the existence (or absence) of adequate resources. A more detailed example is presented in section three.

A single edge in the graph can have multiple processing modules attached to it, if those modules observe the same restrictions regarding their input and output formats.

2.2 Building and using ALPE Hierarchies

The augmented hierarchy as described above can be built in our model in an entirely automated fashion. For this, three hierarchy building operations are used: initialize-graph, classify-file and integrate-process. They are described below.

The **initialize-hierarchy** operation receives no input and outputs a trivial hierarchy formed by a ROOT node (representing the empty annotation schema). Once the graph is initialised, its nodes and edges (having just blank labels, for the moment) are contributed by classifying documents in the hierarchy.

The **classify-file** operation takes an existing hierarchy and a document marked with metadata observing a certain schema and classifies the schema of the document with respect to the hierarchy. The operation results in an updated hierarchy and the location of the input schema as a node within the hierarchy. If the input document fully complies with a schema described by a node of the hierarchy, the latter remains unchanged and the output indicates this found node; otherwise a new node corresponding to the annotation schema of the input document is inserted in the proper place within the hierarchy and this node is returned.

Integrate-process is an operation which attaches a process (a new tool) to one edge of a hierarchy (or creates a new edge, if the output of the tool is not a format present in the hierarchy). Integrated process stores information about the additional resources required by it, as well as availability and cost issues.

Any path between two nodes in the augmented graph can be seen as a processing flow starting from an initial node and ending in the other node. The term “flow” comes easily if we imagine that the information actually “flows” through the edges of the graph, while also producing changes in the input files. The model described is able to compute these flows and use them on input files corresponding to a schema (node) in the augmented graph to produce files corresponding to another node in the graph.

2.3 The Implemented System

The ALPE system implements the model described previously. ALPE offers the following functionalities:

- the user can generate a new hierarchy;
- the user can input an annotated file and ALPE will classify it in the existing hierarchy;
- the user can input a linguistic processing tool and some required data (see next section) and it will be added to an existing hierarchy, and will be usable in later computed flows;
- the user can input an annotated file and specify a required format (either selecting from the existing hierarchy, or providing a new schema specification) and ALPE will compute processing flows between the two formats. The user then has the choice as to which of the computed flows to be executed by ALPE, which will output the file with the required format.

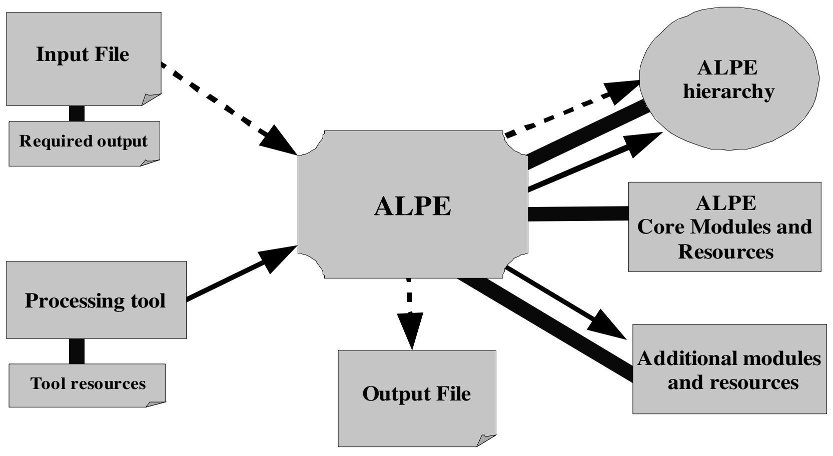


Figure 1: The ALPE architecture and general functionalities

In figure 1, the existence of a thick line between two components denotes the obligatory presence of both connected components. Basically, ALPE requires an ALPE hierarchy, the core modules and resources and the additional modules and resources. If the user inputs a file to be processed, he/she has to specify the required output and APLE will possibly input some changes in the available hierarchy, as well as produce the output file. If the user inputs a new processing tool, ALPE will input the changes implied in the hierarchy, will add the tool to the existing additional modules, as well as add the tool's resources to those available.

As base for any new ALPE hierarchy a core hierarchy is offered, comprised of 12 annotation schemas ranging from basic xml format to the full XCES [7] linguistic annotation specification[4]. The intermediate formats are designed to conform to specific requirements for document annotation, such as tokenization, tokenization with POS-tagging and NP-chunking, and others. The purpose of the core hierarchy is to offer both a starting point to any new hierarchy as well as an anchor for any new linguistic annotation formats included.

ALPE includes 11 core modules, used in any ALPE hierarchy (the hierarchy augmented with processing power, as described) but not attached to any edge. The core ALPE modules perform:

- language identification for input documents;
- format identification and classification for an annotated document;
- simplification of an annotated document to a format in the hierarchy;
- merging of multiple annotated versions of the same text;
- creation/development of an ALPE hierarchy;
- integration of a new tool in the hierarchy.

These are the ALPE core modules required in the current state of the system; further developments may add additional modules. These core modules are used in any ALPE hierarchy and are not replaceable by user tools. They ensure that any ALPE hierarchy is able to perform according to the specified features.

Since the flow computation process may produce two or more flows for a single user task, a selection can be made. Each computed flow is characterized by a set of features. These features include properties such as flow length (defined as number of processing steps involved) and flow weight (number of intermediate formats produced if computing the flow). Other features are the cost of the flow (the actual financial cost, if one or more modules involved require payment), the estimated precision of the flow (computed using the performance measure specified when adding a new tool to the hierarchy) and the estimated time of computation. The user can then select and run the flow most suitable to his/her needs. The user will be able to specify some default value for the selection, such that flow computation, selection and execution can be performed automatically.

[4] http://www.cs.vassar.edu/XCES/dtd/xcesAna.dtd

3. ALPE as a Computational Grid

3.1 Grid Networks

In spite of a number of advances in Grid computing, resource management and application scheduling in such environments continues to be a challenging and complex undertaking [10]. This is due to geographic distribution of Grid resources owned by different organizations with different usage policies, cost models and varying load and availability patterns with time. The Grid service providers (resource owners) and Grid service consumers (resource users) have different goals, objectives, strategies, and requirements. To address these resource management challenges, a distributed computational economy has been recognized as an effective metaphor for the management of Grid resources as it: enables the regulation of supply and demand for resources, provides economic incentive for Grid service providers, and motivates the Grid service consumers to trade-off between deadline, budget, and the required level of quality-of-service. These factors also promote Grid services to become valuable economic commodities.

Our scope was to integrate the ALPE system in a Grid environment and to try to use all Grid computing benefits from this. *Grid computing* [8] is a phrase in distributed computing which can have several meanings [8]:

- A local computer cluster which is like a “Grid” because it is composed of multiple nodes.
- This computer can offer online computation or storage as a metered commercial service, known as utility computing, computing on demand, or cloud computing.
- It can permit the creation of a “virtual supercomputer” by using spare computing resources within an organization.
- Also, it can create a “virtual supercomputer” by using a network of geographically dispersed computers. Volunteer computing, which generally focuses on scientific, mathematical, and academic problems, is the most common application of this technology.

These varying definitions cover the spectrum of “distributed computing”, and sometimes the two terms are used as synonyms.

Functionally, one can also speak of several types of Grids:

- *Computational Grids* (including CPU Scavenging Grids) which are focuses primarily on computationally-intensive operations.
- *Data Grids* or the controlled sharing and management of large amounts of distributed data.
- *Equipment Grids* which have a primary piece of equipment e.g. a telescope, and where the surrounding Grid is used to control the equipment remotely and to analyze the data produced.

Usually, a *computational Grid* consists of a set of resources, such as computers, networks, on-line instruments, data servers or sensors that are tied together by a set of common services which allow the users of the resources to view the collection as a seamless computing/information environment. The Grid services include [9]:

- security services which support user authentication, authorization and privacy
- information services, which allow users to see what resources (machines, software, other services) are available for use,
- job submission services, which allow a user to submit a job to any compute resource that the user is authorized to use,
- co-scheduling services, which allow multiple resources to be scheduled concurrently,
- user support services, which provide users access to “trouble ticket” systems that span the resources of an entire Grid.

Our ALPE system can be adapted as a system working in a computational Grid, focused on providing linguistic services. In ALPE, all GRID services were implemented except the security services.

3.2 Globus Toolkit

The key to the success of Grid computing is the development of the “middleware”, the software that organizes and integrates the disparate computational facilities belonging to a Grid. Its main role is to automate all the “machine to machine” (M2M) negotiations required to interlace the computing and storage resources and the network into a single, seamless computational “fabric”. The middleware is made of many software programs.

Practically all major Grid projects are being built on protocols and services provided by the Globus Toolkit[5], a “work-in-progress” software which is being developed by the Globus Alliance[6].

The toolkit provides a set of software tools to implement the basic services and capabilities required to construct a computational Grid, such as security, resource location, resource management, and communications.

The first prototype Grid service implementation was demonstrated on January 29, 2002, at a Globus Toolkit tutorial held at Argonne National Laboratory. Since then, the Globus Toolkit 3.0 and 3.2 offered an Open Grid Services Architecture (OGSA) implementation based on the Open Grid Services Infrastructure (OGSI), a precursor to WSRF (WS-Resource Framework). Currently, the Globus Toolkit 4.0 provides a set of OGSA capabilities based on WSRF. The Globus Toolkit 4.0 is an open source, community-driven software project, and it can be downloaded and used under the terms of an open source license.

3.3 Linguistic services offered thorough the Grid

Grid services beside classical computational systems come with two new characteristics:

[5] http://www.globus.org/toolkit/
[6] http://www.globus.org/alliance/

- *Computational power* is very high and it is very extensible relative to addition of new nodes;
- The possibility to share *information on a large scale* emerges and complex problem solving is enabled.

Initial our scope was to offer basic linguistic Grid services, and after that, based on these, we built complex linguistic Grid services. Basic Grid services we have implemented include lemmatization, POS, tokenizing, name entity recognition, WordNet (on English and on Romanian). As complex services we have already implemented a service for definition extraction from web (using Wikipedia or using Google engine). Next work will be focused on special services necessary in Question answering and in Textual Entailment. All of these services will form a global ALPE hierarchy which will serve as a repository of services and as a configuration environment for complex processing chains.

Grid services implementation

For writing and deploying a WSRF Web Service we must follow five steps [15]:

1. **Define the service's interface** with *WSDL*
2. **Implement the service** with *Java*.
3. **Define the deployment parameters** with *WSDD* and *JNDI*
4. **Compile everything and generate a GAR file** with *Ant*
5. **Deploy service** with *a GT4 tool*

Define the service's interface. The first step in writing a web service (including those that use WSRF to keep state) is to define the *service interface*. We need to specify what our service is going to provide to the outer world. At this point we're not concerned with the inner workings of that service (what algorithms it uses, other systems it interacts with, etc.). We just need to know which *operations* will be available to our users. In Web Services lingo, the service interface is usually called the *port type* (usually written *portType*). There is a special XML language which can be used to specify what operations a web service offers: the Web Service Description Language (WSDL). At this step we write a description of our NLPServices using WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="NLPServices" …>

<!--=====   T Y P E S   =======-->
<types>
<xsd:schema targetNamespace="http://www.globus.org/namespaces/uaic/fii
/NLPServices_instance"
xmlns:tns="http://www.globus.org/namespaces/uaic/fii/NLPServices_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <!-- REQUESTS AND RESPONSES -->

          <xsd:element name="lemma" type="xsd:string"/>
          <xsd:element name="addResponse">
                   <xsd:complexType/>
          </xsd:element>
          <!-- RESOURCE PROPERTIES -->
          <xsd:element name="Word" type="xsd:string"/>
          <xsd:element name="NLPResourceProperties">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
              <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
          </xsd:complexType>
          </xsd:element>
</xsd:schema>
</types>

<!--===  M E S S A G E S  ===-->
<message name="LemmaInputMessage">
          <part name="parameters" element="tns:lemma"/>
</message>
<message name="LemmaOutputMessage">
          <part name="parameters" element="tns:lemmaResponse"/>
</message>

<message name="GetValueRPOutputMessage">
          <part name="parameters" element="tns:getValueRPResponse"/>
</message>
</definitions>
```

Implement the service: After defining the service interface ("*what* the service does"), the next step is implementing that interface. The implementation is "*how* the service does what it says it does". The methods written in the service are the interface by which users can access and modify the resource values beneath. It can be thought of as a gateway or a “public face” to the service. The class with implementation of the NLP services can be broken down into different sections and methods as follows:

1. Implements the values and resource properties from the namespace interface;
2. Has methods for creating of resources;
3. Has (private) methods for retrieving of resources;
4. Have specific service operations – like *lemma, WordNet or Wikipedia* methods that obtain for a given word its lemma, synonyms or relations from Wikipedia.

```
package uaic.fii.nlp.impl;
import javax.xml.namespace.QName;
public interface NLPQNames {
      public static final String NS = "http://www.globus.org/
                          namespaces/uaic/fii /NLPServices_instance ";
       public static final QName RP_VALUE = new QName(NS, "Value");
       public static final QName RP_WORD = new QName(NS, "Word");
       public static final QName RESOURCE_PROPERTIES = new QName(NS,
                      "LemmaResourceProperties");
}
```

Configuring the deployment in WSDD: Up to this point, we have written the two most important parts of our state full Web service: the service interface (WSDL) and the service implementation (Java). This step makes our web service available to client connections. For that we must take all the loose pieces we have written up to this point and make them available through a *Web services container*. This step is called the *deployment* of the web service. One of the key components of the deployment phase is a file called the *deployment descriptor*. It's the file that tells the Web Services container how it should publish our web service (for example, telling it what our service's URI will be). The deployment descriptor is written in WSDD format (Web Service Deployment Descriptor). The deployment descriptor for our Web service looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="core/NLPService" provider="Handler" use="literal"
                   style="document">
     <parameter name="className" value="uaic.fii.services.NLPService"/>
<wsdlFile>share/schema/NLPService_instance/NLP_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Create a GAR file with Ant: At this point we have a service interface in WSDL, a service implementation in Java, and a deployment descriptor in WSDD telling the Web Services container how to present and to the outer world. Using those three files we will generate a *Grid Archive*, or *GAR file*. This GAR file is a single file which contains all the files and information the Web services container needs to *deploy* our service and make it available to the whole world. The creation of a GAR file is a pretty complex task which involves the following:

- Processing the WSDL file to add missing pieces (such as bindings);
- Creating the stub classes from the WSDL;
- Compiling the stubs classes;
- Compiling the service implementation;
- Organize all the files into a very specific directory structure.

This task is performed with Ant. Ant, an Apache Software Foundation[7] project, is a Java *build tool*. It allows programmers to forget about the individual steps involved in obtaining an executable from the source files, which will be taken care of by Ant.

Deploy the service into a Web Services container: The GAR file, as mentioned above, contains all the files and information the web server needs to deploy the web

[7] http://www.apache.org/

service. Deployment is done with a GT4 tool that, using Ant, unpacks the GAR file and copies the files within (WSDL, compiled stubs, compiled implementation, WSDD) into key locations in the GT4 directory tree.

4. Conclusions

ALPE will be used as a management tool for Grid services, in itself being adapted as a Grid service. Due to its particular functionalities, it will offer improved usability and access to linguistic tools and resources, factors especially important to large scale and multilingual research projects. Using ALPE as a Grid services management environment allows the creation of a global linguistic hierarchy, integrating a multitude of services targeting linguists and students alike.

In this paper we have shown how using the ALPE system in the context of a multilingual research project can give significant advantages. ALPE automatically configures complex processing chains involving several modules and documents in different languages. We show how the features brought by the addition of an ALPE type hierarchy to a complex project can contribute significantly to acquire multilinguality, distributivity, versioning of language resources, automatic annotation, management of IPR and cost issues, as well as managing diversity of annotation styles. The creation of a ALPE hierarchy as a management and processing environment for such a large scale multilingual project is already in progress, and this will offer a qualitative practical test of ALPE's capabilities. The adaptation of ALPE as a Grid service has the potential to further improve the qualitative evaluation, and will be the focus of a second evaluation stage.

One important further development of ALPE will be a web-service allowing users to build, configure and use ALPE hierarchies on the web, either as a limited password-protected resource or a global linguistic resources collection. This type of hierarchy is able to manage multilingual resources and resources which require a fee to be paid before usage. Each user will be able to contribute with his/her own tools and annotated resources, as well as to use processing chains adapted to his/her specifications, both in terms of input and output formats and cost and performance issues.

The development of ALPE with Grid services will bring significant benefits to the user. The increase in speed and processing power is significant, leading to an increase in ALPE's accessibility. Also, since Grid type networks already are employed in several NLP research projects, the developed tools and resources of those projects can be easily integrated into ALPE.

Acknowledgements: The authors would like to thank prof. Dan Cristea for his help and support at different stages of the development of the system. The work presented in this paper is partially financed by the CEEX GRAI (Grid Computing and Artificial Intelligence) project number 74, II03/31.07.2006.

References

[1]. D. Cristea, C. Butnariu, Hierarchical XML representation for heavily annotated corpora. In *Proceedings of the LREC 2004 Workshop on XML-Based Richly Annotated Corpora*, (2004), Lisbon, Portugal.

[2]. D. Cristea, C. Forăscu, I. Pistol., Requirements-Driven Automatic Configuration of Natural Language Applications. In Bernadette Sharp (Ed.): *Proceedings of the 3rd International Workshop on Natural Language Understanding and Cognitive Science - NLUCS 2006*, in conjunction with ICEIS 2006, Cyprus, Paphos, (2006). INSTICC Press, Portugal. ISBN: 972-8865-50-3.

[3]. H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan., GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the ACL (ACL'02).* (2002), Philadelphia, US.

[4]. H. Cunningham, V. Tablan, K. Bontcheva, M. Dimitrov., Language engineering tools for collaborative corpus annotation. *Proceedings of Corpus Linguistics* (2003), Lancaster, UK.

[5]. T. Dunning, Statistical identification of language, available at: http://ling.ohio-state.edu/~cbrew/papers/dunning94.ps (1994)

[6]. D. Ferrucci and A. Lally., UIMA: an architectural approach to unstructured information processing in the corporate research environment, *Natural Language Engineering* 10, No. 3-4, 327-348. (2004)

[7]. N. Ide, P. Bonhomme, Romary L., XCES: An XML-based Encoding Standard for Linguistic Corpora, *Proceedings of the Second International Language Resources and Evaluation Conference*. Paris: European Language Resources Association, (2000)

[8]. I. Foster and C. Kesselman, The grid: blueprint for a new computing infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (1999)

[9]. D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, N. Rey-Cenvaz, "Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications", *In Cluster Computing journal*, Volume 5, Number 3, Pp. 325-336. 2002

[10]. R. Buyya. "Economic-based Distributed Resource Management and Scheduling for Grid Computing", PhD Thesis, Monash University, Melbourne, Australia, 2002.

[11]. G. C. Fox, D. Gannon, "Computational Grids", *IEEE Comput Sci* Eng. Vol 3, No. 4, pp. 74-77, 2001.

[12]. D. Gannon, and A. Grimshaw, "Object-Based Approaches", *The Grid: Blueprint for a New Computing Infrastructure*, Ian Foster and Carl Kesselman (Eds.), pp. 205-236, Morgan-Kaufman, 1998.

[13]. I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure". Morgan Kaufmann Publishers. ISBN 1-55860-475-8.

[14]. F. Berman, A. J. G. Hey, G. C. Fox, "Grid Computing: Making The Global Infrastructure a Reality". Wiley. ISBN 0-470-85319-0.

[15]. The Globus Toolkit 4 Programmer's Tutorial: http://gdp.globus.org/gt4-tutorial/

Integrating Grid Services in a Web Decision Support System for Greenhouse Projects

Cristian Aflori[1], Marius Călin[2],
Feodor Filipov[2], Ciprian Chiruță[2]

[1]Technical University of Iasi
[2]The University of Agricultural Sciences and Veterinary Medicine of Iasi, Romania
caflori@cs.tuiasi.ro

Abstract. *Combining web services and grid paradigms provides the infrastructure for exposing functional capabilities for efficient, distributed, secure and reliable use. Therefore, modeling decision-making processes in fields like greenhouse projects as grid services is an innovative approach with many challenges and effective results. We propose a model for integrating grid services in a web decision support system for greenhouse projects. The model covers a large scale of phases: functional specifications, web decision support system architecture, grid services design, implementation, integration and final deployment. Several analytical aspects of the process of the grid services integration in the web decision support system are investigated.*

1. Introduction

The recent global diffusion of the Internet, the large number of computational resources and the wider availability of fast network connections have increased the importance of the effective distributed computing processes, but, at the same time, have created demanding challenges. The original concept of Grid as described in [1] is limited to computational resources similar to the metacomputing concept [2]. The extension of the concept implies a more generic understanding as a mechanism of controlled resource sharing enabling cross-organizational collaborative business or scientific processes.

In our days the application cycle, from planning to development and operation, is shorter than ever. Also, the initial specifications and performance requirements change very often by adding new functions or modifying the existing ones. Therefore, there is a strong demand for technologies that can support flexible construction and operation of a large variety of information systems at low cost and high reliability.

Based on the definition of Grid Types [3], there are different types of “Grids”:

- Cluster Grids or computing Clusters: aiming to replace large Shared Memory Computers, built from a set of compute resources typically connected using high speed networks;
- Distributed Enterprise Grids or Intra Grids: commercial or scientific environments for secure resource sharing across organizational boundaries;
- Utility Grid Services: similar to Intra Grids but, in addition, being able to be extended on demand with computational resources offered by a third party;
- Collaborative Business Grids: the key goal is to enable reliable, managed and secure sharing of resources across organizational boundaries.

This approach can be included in the Collaborative Business Grid category and it allows designing and integrating the Grid Services into a web system. The Business Grid is a fusion of Web services and Grid computing technologies, initially intended to enable effective use of distributed supercomputer for scientific computation. The Business Grid middleware key technologies are in the process of standardization within various standards bodies, including the Global Grid Forum (GGF) – Open Grid Forum (OGF) [4].

Open Grid Services Architecture (OGSA) is a proposed standard architecture for next-generation Grid systems [5]. It combines Web services technology used for application integration and Grid computing technology used for virtualizing and sharing distributed computing resources. OGSA defines a uniform infrastructure for Grid systems by using a set of existing Web services technologies, Web Services Resource Framework (WSRF) [6]. OGSA can benefit from advances in business application management and control. Furthermore, OGSA defines the functional component required for the virtualization of IT resources and autonomous control of the Grid system. The key element of the OGSA architecture is the Grid Service. Grid Services are extended web services that provide a programming environment for stateful services with asynchronous messaging feedback.

In this approach, the Globus toolkit was used as infrastructure for the Grid Services. The Globus toolkit is a community based, open architecture, open source set of services and software libraries that supports Grid applications [7]. The choice of the Globus toolkit was motivated by a large number of reasons:

- open source framework commonly used in the development of the Grid scientific applications as a result of a major effort for development and standardization in scientific and business research communities;
- flexible and effective way to develop distributed application based on the Service Oriented Architecture (SOA) and Open Grid Service Architecture (OGSA);
- high compatibility with object oriented environments;
- high portability on various flavours of operating systems;

The toolkit addresses issues of security, information discovery, resource management, data management, communication and portability. Globus toolkit mechanisms are in use at hundreds of sites and by dozens of major projects worldwide. The toolkit is a set of software tools to be used in developing Grid applications; it represents an implementation of the Open Grid Services Infrastructure (OGSI) [8].

2. Case Study

The proposed model is useful for developing a Decision Support System (DSS) aimed to be utilized in studying the suitability of land units for building greenhouses.

Locating a greenhouse is often dictated by a series of preconditions like water resources, market requirements and other factors. Moreover, a greenhouse is a protected space that requires long term monitoring of soil features, especially those ones that are easily modifiable. This is why a pedological study is necessary in order to decide if the respective land unit is suitable for starting such a project.

A pedological study regarding a greenhouse location comprises a terrain phase, a laboratory phase and a data processing phase. In the terrain phase, specific observations are made on the geology, lithology, hydrography, morphology and other soil features. Climate observations are also made. In the laboratory phase physical and chemical analyses are preformed on soil samples. The results are then compared with the observations made in the terrain phase. The resulted data must be processed in order to make a decision.

Official classifications of soil types divide greenhouse soils in classes, subclasses, groups and subgroups. Classes represent the highest grouping level. Situating a soil in a certain class is determined by different restricting factors, the most intense of them being considered as determinant. The following classes are defined for soils that are to be used for building greenhouses:

Class I – soils with *no* restrains or degradation risks;

Class II - soils with *low* restrains or degradation risks;

Class III - soils with *moderate* restrains or degradation risks;

Class IV - soils with *severe* restrains or degradation risks;

Class V - soils with *extremely severe* restrains or degradation risks.

In order to assess the different soil properties, the field data and the laboratory results must be framed in land suitability classes for greenhouses. Official tables, recommendations and other reference materials are available for this purpose, because of the great amount of data, an automation of this process would bring obvious advantages.

The general approach is to study several *soil profiles* from the respective land unit and to draw a global conclusion from the partial ones. There are several *horizons* in each soil profile, which is a relatively homogenous soil layer that must be identified and measured. A standard depth of about 50 – 100 cm is also designated.

The Decision Support System will receive as input the terrain and laboratory data. As output, it will produce a report which during the data processing was step by step enriched with partial conclusions regarding the land suitability from different points of view like *soil texture, edaphic volume, salinization, alkalization, terrain slope, Calcium Carbonate contents, nonuniformity degree, humidity excess, lateral drainage.* An overall conclusion is suggested at the end of the report.

The system uses a knowledge base that stores the existing official recommendations and classifications. These documents were processed to extract and systematize the useful information. The system is designed to accept future adding of new information when appropriate.

3. Web Decision Support System for Greenhouse Projects Architecture

The objective for the Decision Support System is to help making decisions on the soil suitability for greenhouse projects. The general system has three categories of components: experimental input data, standard indicators and soil properties, soil texture, edaphic volume, salinization, alkalization, terrain slope, Calcium Carbonate contents, nonuniformity degree, humidity excess, lateral drainage) and a set of algorithms that process the input data on the basis of existing standards and draw the conclusions regarding the land suitability for building a greenhouse.

The architecture of the Web Decision Support System is presented in the Figure 1:

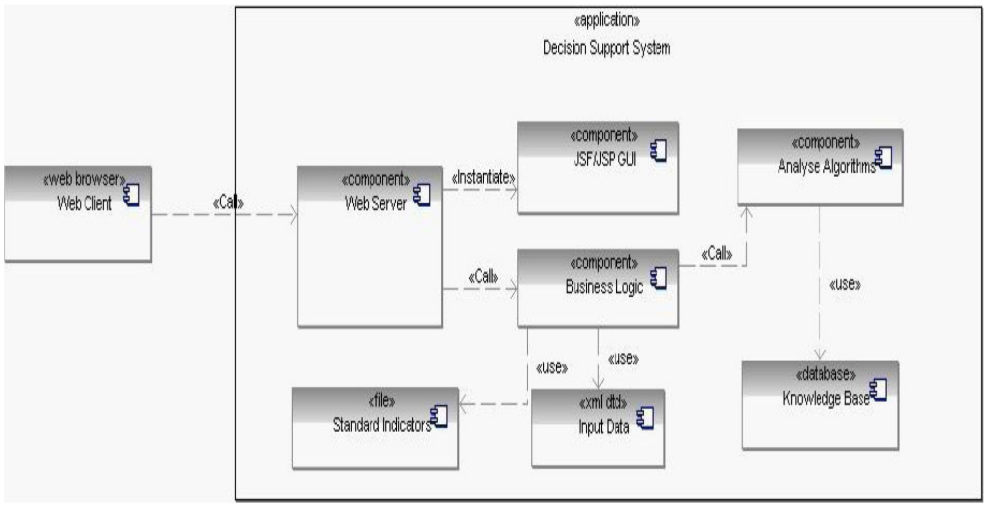


Fig. 1. The architecture of the Web Decision Support System for Greenhouse

The system has a web based architecture with a server application and web browser clients. The server application has several components:

- the Web server that is the interface between the browser clients and the server application;
- the Graphical User Interface, displayed in the client browser, use Java Server Pages and Java Server Faces (component based approach) technologies;
- the Knowledge Base contains the standard set of indicators and parameters used by the algorithms and loaded from the flat file Standard Indicators. It also contains the input data resulted from field work and laboratory analyses. These data are input by users through the GUI or from the XML Input Data file;
- the Business Logic component manage the communication between the Web Server and the rest of the server components;
- the Analyze Algorithms component implements the algorithms that process the experimental soil data and the standard indicators; there are now implemented three

algorithms: the Calcium Carbonate contents algorithm (class A), the edaphic volume algorithm (class B), the salinization algorithm (class C).

The web client has the possibility to call the following methods:

- UpdateKB – updates the Knowledge Base from the flat files Standard Indicators (Excel format), indicators needed for applying the analyze algorithms;
- LoadDataIn – optional method to load the input soil experimental data to be processed by analyze algorithms; usually, the users fill this data in the GUI forms;
- three algorithms: ExecuteAlgorithmCG (class A), ExecuteAlgorihtmVE (class B), ExecutaAlgorithmSal (class C).

The most important component of the system is the Analyze Algorithms module that applies the processing algorithms and output the conclusions. This is the major reason to integrate this component into a Grid system and to take advantages of all Grid features: distribute security and reliable computing, resource management, data management, communication and portability.

4. Design and Implementation of the Grid Services

Grid Services are extended web services that provide a programming environment for stateful services with asynchronous messaging feedback. The Analyze Algorithms module is re-designed in order to publish the algorithms as Grid Services.

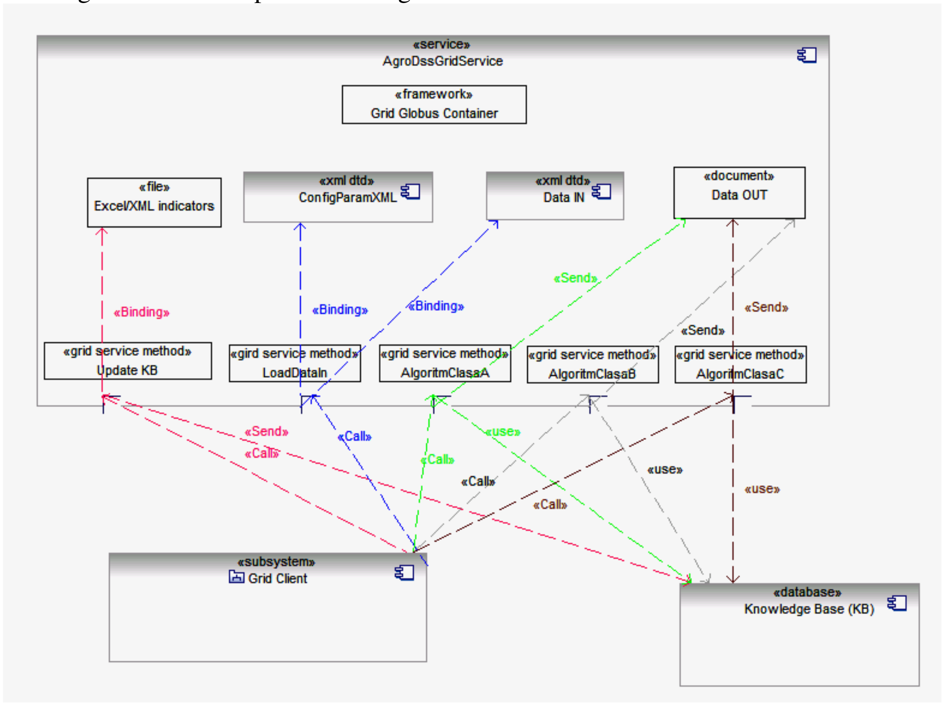


Fig. 2. The general structure of the AgroDss Grid Service

The AgroDss Grid Service has features specific to Grid Services: service registry, service creation, authorization, notification, manageability and concurrency.

The general structure of the AgroDss Grid Service is presented in the Figure 2 and there are some distinct entities: the Globus container where the Grid Service is deployed, the AgroDss Grid Service with all three components (Class A, B and C algorithms), the Grid client which can take various forms (from console entities to server application module and complex graphical user interfaces – GUI) and the Knowledge Base and the auxiliary configuration or data files. The diagram shows also the information flows for different Grid Service methods and the interaction between components: red is for updating the Knowledge Base, blue is for uploading the input experimental data, green is the flow for the Class A algorithm, black is the flow for the Class B algorithm and brown is the flow for the Class C algorithm.

The process of developing and installing a Grid Service has five steps [9]. The first step is defining the interface of the service in WSDL (Web Service Definition Language) format. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [10]. The WSDL file contains the operations, messages and types for the AgroDSS Web Service. Also, it contains the Resource Properties parameters because the Class A algorithm is a two-step algorithm and the second step uses the previous state of the algorithm. This is the reason for using the stateful feature of the Grid Service, mechanism implemented by the resources properties for stateful web services (WSRF) [6].

The following phase is to implement the interface with Java language. This coding phase implements all the details for each analyze algorithm, following the logical schema and using the input data and the knowledge base. The configuration and deployment parameters are defined in the next phase using WSDD (Web Service Deployment Descriptor), JNDI (Java Naming and Directory Interface) and namespace to package mapping.

The last phase is to compile and to generate the deployment archive in GAR (Globus ARchive) format and deploy the Grid Service in the Globus container. This can be done using a building tool like Apache Ant [11].

The AgroDss Grid Service is build and deployed in the Globus container following the five steps above. The next step is to create the Grid client in order to evaluate the AgroDss Grid Service. The client is in the console format and calls the LoadDataIn method for uploading the experimental input data, stored in the XML files.

At this moment there are two different approaches:

- the Web Decision Support System that use a web server (Apache Tomcat) and provide complex graphical user interfaces (GUI) at the browser client side;
- the AgroDss Grid Service that uses the Globus container: all the features are implemented inside the Grid Service but the client interaction with the service is difficult because of the complex user interfaces at the client side.

For this reason, the real challenge is to integrate the AgroDss Grid Service into the general infrastructure of the Web Decision Support System for Greenhouse from our case study.

5. Integrating the Grid Services into the System

The process of integration of the AgroDss Grid Service described in the previous paragraph into the general infrastructure of the Web Decision Support System for Greenhouse can be analyzed under different aspects:

- maintaining the usability and the portability of the Web Decision Support System: complex GUI at the client side but easy to use: the client must use the normal web browser to access the application (knowing only the URL of the web server and not all the security – authentication policies from the Grid applications);
- take advantage of the Grid features: distribute security and reliable computing, resource management, data management, communication and portability;
- keep the efficiency of the system: introducing the Grid Services must not affect the response time of the system;
- extensibility of the system: autonomous extending of the GUI features or of the analyze algorithms module;
- simple process of installation, configuration and deployment of the overall system.

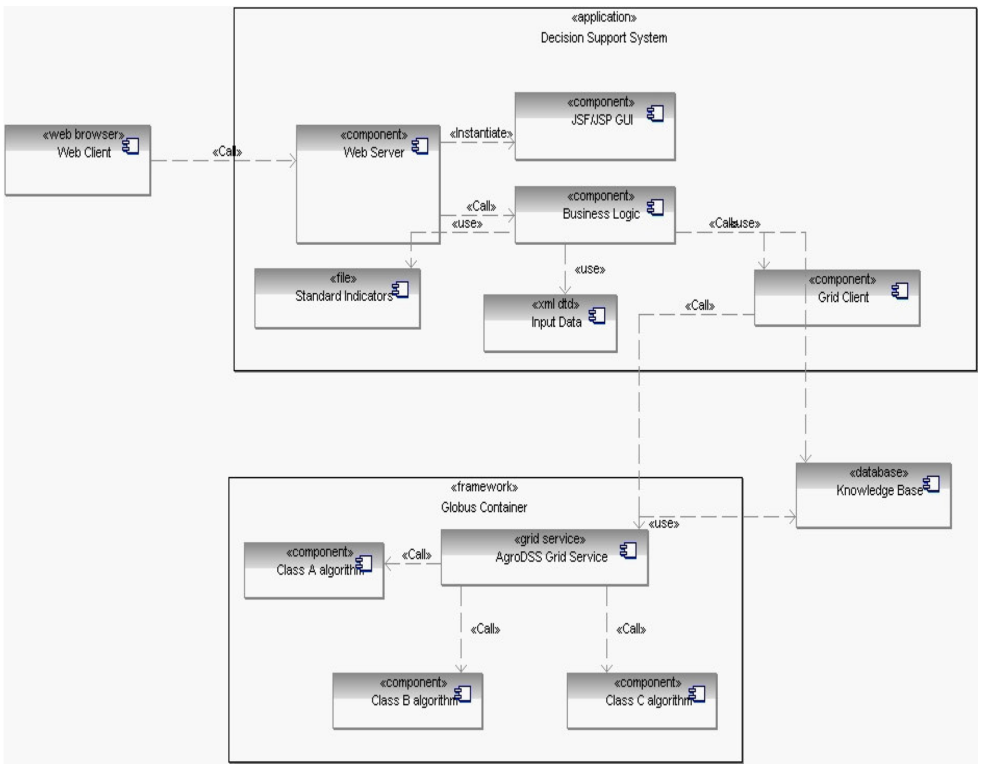


Fig. 3. The integration of the AgroDss Grid Service in the Web System

The difficulty of the design is to identify the components to be published as Grid Services in order to comply with the constraints described above. The solution presented in the Figure 3 shows a multi-layered architecture:

- the client browser that display the GUI (the same as in initial system);
- the web application server that contains in plus the Grid Client (manage the calls to the Grid Service);
- the Knowledge Base that is in a separate layer because it is used from both web application server and from the Grid Service;
- the AgroDss Grid Service that implements the analyze algorithms: class A, B and C (can be extended with other algorithm implementations).

From the implementation point of view, the architecture is composed from several components:

- a web server that provide support for J2EE technologies (Servlet, Java Server Pages, Java Server Faces), like Apache Tomcat version 6;
- the Globus container – the place where the AgroDss Grid Service is deployed;
- a light Data Base Management System like Derby for storing the Knowledge Base of the system.

The installation, the configuration and the deployment of the system has different aspects:

- the installation of the Apache Tomcat web server, of the Derby Data Base Management System and of the Globus container;
- the initial configuration of each entity by modifying the XML parameters file, running the initial database scripts, setting the parameters for the remote connections (database url and Grid Service url);
- the deployment of the AgroDss Grid Service inside the Globus container and the deployment of the Web DSS application inside the Tomcat container;
- the last step is to start the containers (Derby DBMS, Tomcat and Globus) and to launch the application from a web browser (ex: http://10.11.1.1:8080/DSSWeb).

After launching the application from the web browser, the user can choose one or more algorithms to execute and fill the specific input data for each algorithm. The user selections are passed to the client Grid that creates an instance of the endpoint specific to the AgroDss Grid Service. The client Grid invokes the "create Grid service" request on the AgroDss Grid Service factory and tries to create the AgroDss service instance. Each request involves mutual authentication of the Grid client based on a trusted authentication certificate, followed by authorization of the request. If the request is successful, the Grid Service instance is created with some initial lifetime. The AgroDss Grid Service uses its proxy credentials to start processing analyze algorithms steps. Also, the Grid Service method for Class A algorithm - ExecuteAlgorithmCG (the Calcium Carbonate contents algorithm) is a stateful web service method. This mean that the algorithm stores the first phase processed data in the Globus resource properties and based on the user choice, the Grid Service can use those data for proceeding with the further steps. After the AgroDss Grid Service finishes the processing, it sends the result to the Grid client that forward the answer to the Business Logic module in order to display in a proper format to the browser client side.

6. Conclusions and Future Work

The paper presents some aspects of integrating Grid Services in a Web Decision Support for Greenhouse Projects. The solution proposed is analyzed from different points of views: architecture and design issues, implementation and technological choices, application development, installation, configuration and deployment. We tackle various approaches from Web Decision Support System to complete Grid Services framework. It results that the most efficient solution based on a large number of constraints is integrating the AgroDss Grid Service into the Web System. Also, publishing the analyze algorithms module as Grid Services brings some major advantages: flexibility and independence in developing and deploying new Grid Service algorithms, major control to the Web System and intuitive graphical interface for effective usage of the system.

Future work is focused on the deeper evaluation and optimization of the implemented algorithms for pedological studies regarding greenhouse locations, but also on extending the Grid Services and Web Decision Support System with new algorithms.

Acknowledgment. The GRAI project runs under the CEEX grant no. 74 II03/31.07.2006, a research framework created by the Romanian Ministry of Education, Research and Youth.

References

[1]. I. Foster, C. Kesselmann, editors, The GRID: Blueprint for a New Computing Infratructure, Morgan Kaufmann, (1999)
[2]. C. Catlett an L. Smarr, Metacomputing, Communications ACM, 35(6): 44-52, June (1992)
[3]. Quocirca Insight Report, Grid Computing Update, November (2005)
[4]. www.gridforum.org
[5]. I. Foster et al, The Open Grid Services Architecture version 1.0, OGSA Working Group, Global Grid Forum, July 12, (2004)
[6]. Web Services Resource Framework (WSRF), OASIS Web Services Resource Framework TC, OASIS.
[7]. Globus Toolkit, (2001), www.globus.org
[8]. Foster I, et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, tech. report, Globus Project; (2002)
[9]. C.Aflori, M.Craus et al, Grid – technologies and applications, p.54, Ed. Politehnium, (2005)
[10]. http://www.w3.org/TR/wsdl
[11]. http://ant.apache.org/

PART III APPLICATIONS

Sequential and Distributed 3D Terrain Model Generation. Performance Analysis

Silviu Bejinariu, Ramona Luca

Institute for Computer Science, Romanian Academy,
Iaşi Branch, Romania
silviub@iit.tuiasi.ro, ramonad@iit.tuiasi.ro

Abstract. *The 3D model generation methods are based on the Delaunay triangulation algorithms. These methods are time and memory intensive and sometimes the datasets are too large for serial machines. This paper presents an implementation of the distributed Delaunay triangulation algorithm using the domain division method on grid architectures and a performance comparison of the sequential and distributed methods for large datasets.*

Keywords*: grid computing, triangulation, 3D model, GIS*

1. Introduction

The 3D terrain model generation requires a digital map of the studied area. This map is described either as contour lines or as isolated points. Usually, the map is prepared using a Geographic Information System (GIS) application [1].

The 3D model generation procedure uses this description to generate the 3D terrain model as a set of 3D triangles, using a Delaunay triangulation. Usually the triangulation is performed in 2D, considering only x and y coordinates of the known points. The 3rd coordinate is added after, considering that the terrain does not contain concavities in the vertical direction.

If the surface is described by isolated points, a classical Delaunay triangulation is applied. In the second case, the 3D model is created applying the Constrained Delaunay Triangulation over the contour lines set.

Depending on the terrain description accuracy (density of the description points) and the size of the studied area, the 3D model generation can be a very time consuming procedure. For this reason, a distributed version of this procedure is useful.

2. 3D Terrain Model Generation

2.1 Sequential Method

The 3D terrain model generation procedure is based on a Delaunay triangulation algorithm that receives as input the description of the domain as a set of points/lines with known coordinates.

Delaunay triangulation represents an important sub step in many computationally intensive applications, including pattern recognition, terrain modeling, and meshes generation. Delaunay triangulations and their duals, Voronoi diagrams are among the most widely studied structures in computational geometry [2, 3, and 9]. There are many implementations for the triangulation algorithms, published in free to use libraries dedicated to computational geometry.

The general structure of the triangulation procedure is described bellow:

```
triangulate()
{   create the initial supertriangle
    for (each vertex in the input set)
    {
          add_vertex(vertex)
    }
    for (each triangle)
    {     if (one or more vertices stem from supertriangle)
          {
            remove triangle
}   }     }
```

An initial triangulation is required in order to apply the above procedure. Usually, the initial triangulation is the convex hull of the vertices set, named „super-triangle”. The most important step of the triangulation procedure is the insertion of a new vertex in an existing triangulation that is based on the Delaunay property: for each of the generated triangles, the circumscribed circle does not contain another vertex of the input set (*all circumscribed circles are empty*).

Based on this property, a very simple method to insert vertices in an existing triangulation is described bellow [4, 5]:

```
add_vertex(vertex)
{   for (each triangle)
    {     if (vertex is inside triangle's circumscribed circle)
          {     store triangle's edges in edge buffer
                      remove triangle
    }     }
    remove all double edges from edge buffer, keeping only
           unique ones
    for (each edge in edge buffer)
    {
          form a new triangle between edge and vertex
}   }
```

This procedure can be optimized if some preprocessing procedures are applied to the initial set of points. One of the most used methods is to sort the input set by one coordinate.

2.2 Distributed Method

There are many serial algorithms for Delaunay triangulation. The best have been extensively analyzed and implemented as general-purpose libraries. These algorithms are time and memory intensive. For this reason the parallel implementations are important both for improving performance and for solving problems for which memory requirements are too large for serial machines.

This paragraph describes the parallel Delaunay triangulation algorithm [6] as a coarse parallel partitioner, switching to an efficient implementation of Dwyer's serial algorithm provided by the Triangle package at the leaves of the recursion tree.

We describe a simple version of the algorithm [6]:

```
Algorithm: ParallelDelaunay(P, B,  T)

Input:  P = a set of points in $R^2$,
        B = a set of Delaunay edges of P which is the
                             border of a region in $R^2$ containing P,
        T = a team of processors,
Output:  The set of Delaunay triangles of P which are contained within B.
Method:
1.  If |T| == 1 return SERIAL_DELAUNAY (P, B).
2.  Find the point q that is the median along the x axis of all internal
    points (that is, points in P that are not on the boundary B). Let L
    be the line $x = q_x$.
3.  Let P' = $\{(p_y - q_y, ||p - q||^2), (p_x, p_y) \in P\}$, derived from projecting
    the points P onto a 3D paraboloid centered at q, and then onto the
    vertical plane through the line L.
4.  Let H = LOWER_CONVEX_HULL (P'), H is a path of Delaunay edges of the
    set P. Let $P_H$ be the set of points on the path H and H' be the path H
    traversed in the opposite direction.
5.  Create the input for left and right sub-problems:
    $B^L$  = BORDER_MERGE (B, H)
    $B^R$  = BORDER_MERGE (B, H')
    $P^L$  = $\{ p \in P \mid p$ is left of L$\} \cup \{ p' \in P_H$, p' contributed to $B^L\}$
    $P^R$  = $\{ p \in P \mid p$ is right of L$\} \cup \{ p' \in P_H$, p' contributed to $B^R\}$
    $T^L$  =  subset of T of size $|T|\ |P^L| / (|P^L| + |P^R|)$
    $T^R$  = $T - T^L$
    Return ParallelDelaunay ($P^L$, $B^L$,  $T^L$) $\cup$ ParallelDelaunay ($P^R$, $B^R$, $T^R$)
```

The input domain (P) is recursively divided in sub-domains ($P^{L,}$ P^R) until the maximum number of available processors is reached. All sub-domains are processed in parallel using the sequential algorithm (**SERIAL_DELAUNAY**). The obtained triangles sets are merged to obtain the final 3D surface.

This algorithm contains the following important steps which must be detailed:

SERIAL_DELAUNAY – can be any sequential triangulation procedure. A fast procedure is recommended for this step.

LOWER_CONVEX_HULL - The lower half of the convex hull of the projected points is used to find a new path H that divides the problem into two smaller problems.

BORDER MERGE – this routine computes the borders for each sub-region, in the recursive call. It merges the old border B with the newly dividing path.

One of the most important problems is the method used for domain partitioning. It depends on the distribution of input vertices set.

The domain is divided only on the x axis direction in order to simplify the algorithm description. In practice, the domain is divided alternatively on both directions, x and y.

2.3 Implementation

The algorithms presented in the previous paragraph were implemented as a 2 applications package:

- a client application (AppGen3D) for 3D models visualization and
- a server application (AppGen3D_Server) in which the algorithms are implemented. This application is executed by AppGen3D using the MPI / LAM-MPI interface or as a Grid service depending on the compiling options [7, 8].

The applications were written in C++. The client application uses the OpenGL library for 3D scene rendering. The interface was developed using the wxWidgets library, a C++ framework providing GUI (Graphical User Interface) and other facilities for the most common operating systems (Windows, UNIX with GTK+, UNIX with Motif, and MacOS).

The server application uses an implementation of the sequential Delaunay triangulation from the WildMagic library. All the libraries used in the application are Open Source libraries under the GNU General Public License.

The input terrain description and the output 3D model are stored in 'shape' files, a standard file format for GIS applications [1].

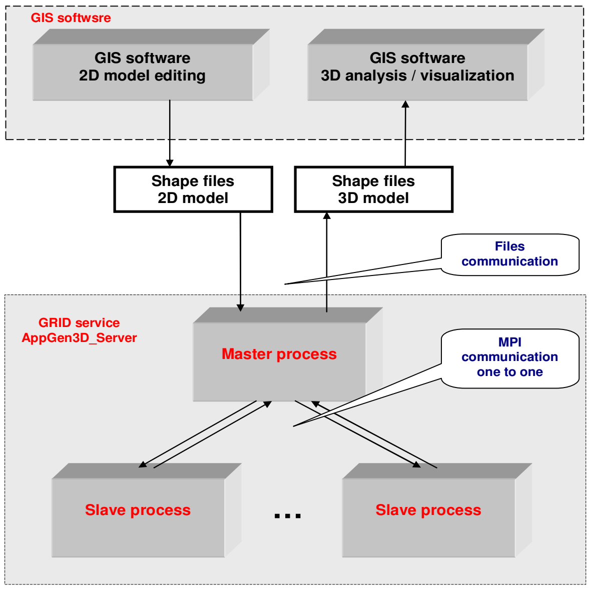

GIS softwsre
GIS software 2D model editing
GIS software 3D analysis / visualization
Shape files 2D model
Shape files 3D model
Files communication
GRID service AppGen3D_Server
Master process
MPI communication one to one
Slave process
...
Slave process

Fig. 1. App_Gen3D_Server distributed application structure

3. Performance Analysis

The AppGen3D_Server application has been tested on a large amount of data that allowed us to manage the error conditions and also to optimize the implementation.

3.1 Data Sets

The application uses for the input / output files the “shape” format which is a standard for GIS applications.

The main analysis was made on 5 different datasets (the attached number is equal to the number of surface description points) (Figure 2):

- *t_010000*, *t_020000*, and *t_081947* – the model was created using a Lidar laser scanner. The original model is t_081947; the other models were created by reducing the number of description points to 10000 / 20000. Due to the scanning method the vertices’ distribution is not uniform. The vertices’ density is greater in the center (near the scanner position).

- *a_096105* and *a_397286* – the input model was created by digitizing the contour lines on topographic maps. The original model is a_397286; the second model contains only a sub-area of it. In this case the vertices’ distribution is uniform.

File	Size	Type	Elements	Vertices
t_010000.shp	440.100	point 3D	10.000	10.000
t_020000.shp	880.100	point 3D	20.000	20.000
t_081947.shp	3.605.768	point 3D	81.947	81.947
a_096105.shp	3.195.344	polyline 3D	1.362	96.105
a_397286.shp	13.107.820	polyline 3D	4.475	397.286

Fig. 2. The properties of the input datasets

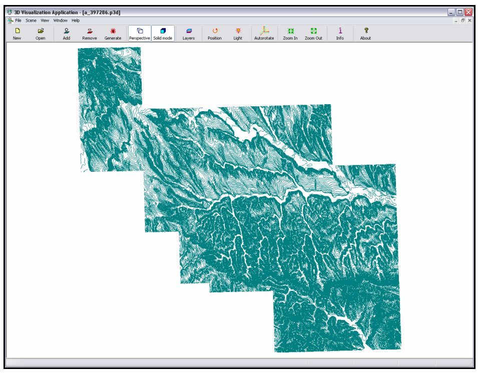


Fig. 3. File a_397286.shp

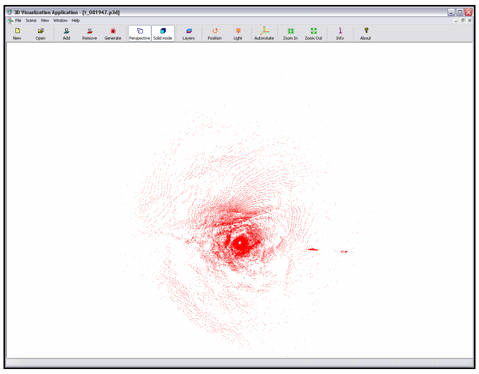


Fig. 4. File t_081947.shp

3.2 Testing Methodology

The AppGen3D_Server application performs the distributed processing of the input data set depending on the following parameters:

- minimum number of vertices for which a domain is a candidate for sub-division;
- maximum number of available processors; this parameter has a higher priority; the domain subdivision is stopped when this value is reached.

The testing conditions were:

- the maximum number of processors was 25 (1 master + 24 slaves).
- we performed the model generation using the following values for the minimum number of vertices for subdivision: 2000, 5000, 10000, 25000 and 50000.
- the sequential processing was simulated using the INT_MAX value as minimum number of vertices in the subdivision condition.

The output files are visualized using the client application as depicted bellow.

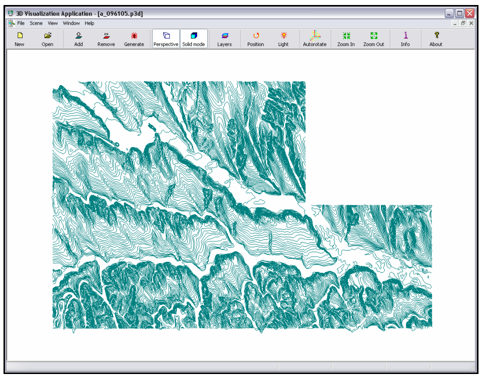

3D Visualization Application - [a_096105.p3d]
File Scene View Window Help
New Open Add Remove Generate Perspective Solid mode Layers Position Light Autorotate Zoom In Zoom Out Info About

Fig. 5. Input file a_096105

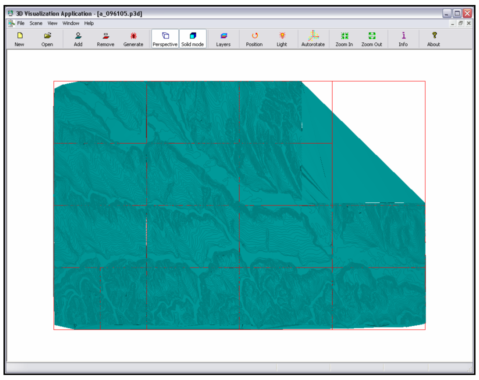

3D Visualization Application - [a_096105.p3d]
File Scene View Window Help
New Open Add Remove Generate Perspective Solid mode Layers Position Light Autorotate Zoom In Zoom Out Info About

Fig. 6. Output surface in the default position

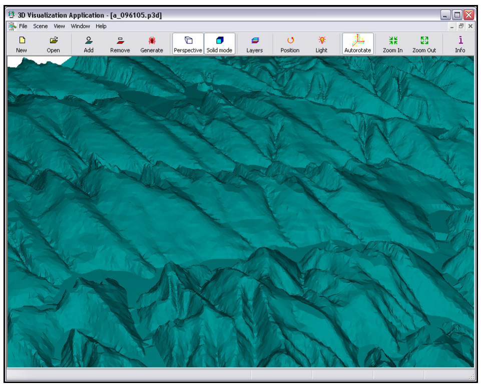

3D Visualization Application - [a_096105.p3d]
File Scene View Window Help
New Open Add Remove Generate Perspective Solid mode Layers Position Light Autorotate Zoom In Zoom Out Info

Fig. 7. Detail of the 3D output in "solid" mode

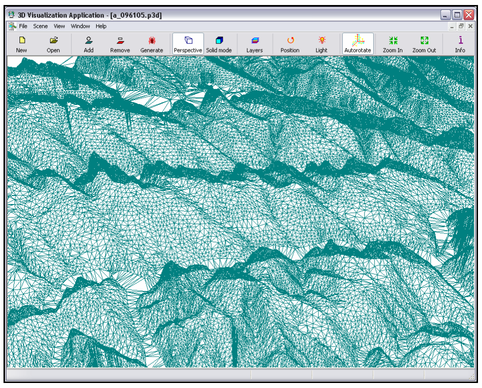

3D Visualization Application - [a_096105.p3d]
File Scene View Window Help
New Open Add Remove Generate Perspective Solid mode Layers Position Light Autorotate Zoom In Zoom Out Info

Fig. 8. Detail of the 3D output in „wire frame" mode

The same tests were performed on the following platforms:

- the frontend server of the GRAI grid using the Scientific Linux operating system and MPI for message passing, specified by *"mpi_linux_grid"*
- a Pentium 4, 3GHz, 1 GB RAM computer with disabled hyper-threading using the CentOS 5.0 operating system and Lam/MPI 7.1.4, specified by „*lam_linux_p4*".
- the same Pentium 4, 3GHz, 1 GB RAM computer with disabled hyper-threading, using WindowsXP SP2 and MPICH 1.2.5, specified by „*mpi_win32_p4*".
- a Dual Core, 1.6 GHz, 1Gb RAM computer using the WindowsXP SP2 operating system and MPICH 1.2.5, specified by „*mpi_win32_dual_core*".

3.3 Results on the "*Mpi_linux_grid*" Platform

The following table contains the processing time (in seconds) for all data sets on the "*mpi_linux_grid*" configuration.

vertices	d 0	d 2000	d 5000	d 10000	d 25000	d 50000
397286	***	840.21	833.80	664.50	724.78	1472.92
96105	2914.49	**78.14**	**79.61**	**80.13**	140.23	604.08
81947	2079.14	151.10	148.22	**135.14**	174.01	450.12
20000	70.92	**15.60**	**15.59**	19.09	64.69	67.37
10000	13.45	**7.24**	**8.31**	**8.43**	13.49	13.97

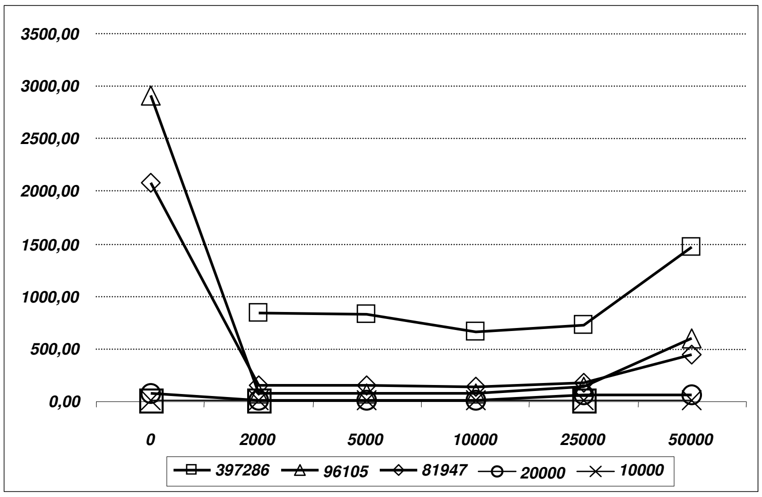


Fig. 9. Computing time in "*mpi_linux_grid*" configuration

If the 397286 vertices dataset (too large comparing with the other datasets), the sequential processing and the 50000 vertices division (which are equivalent for small datasets) columns are excluded, the following detailed chart is obtained:

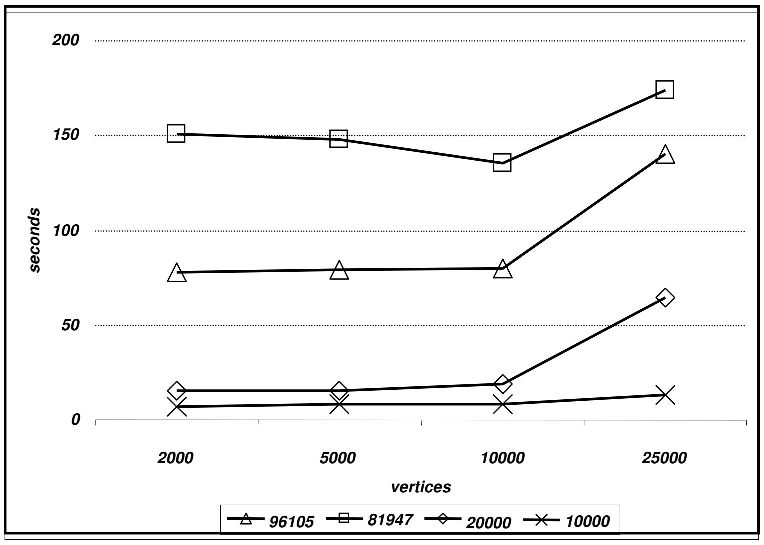


Fig. 10. Computing time in "*mpi_linux_grid*" configuration (detail)

We can observe that:

- the a_397286 data set was never processed using the **sequential** algorithm in an acceptable time (the execution was stopped after 150 minutes),
- except for the 10000 points dataset (distributed processing is not required), the optimal subdivision size is that of 10000 points,
- about 35 – 40% of the processing time is used by the master process for the output file creation. It's obvious that this function requires an optimized version,
- the execution time is increased for sub-domains larger than 10000 vertices (for the small datasets, it is equivalent to the sequential method),
- an unexpected conclusion: the processing time required by the 81947 vertices dataset is greater than the 96105 vertices processing time. This situation is explained by the following tables:

	Points	Triangles	Bytes received	Bytes sent	Process time (sec)	Total time(sec)
Total:	**96105**	**191932**	**2306584**	**13795336**	**max: 12.76**	**80.13**
			2.3 M	13.7 M		
Sub-division		191601				2.72
Extra-surface		331				6.57
Output creation:						62.48
Sub-domain 1	532	1042	12772	75028	0.02	3.12
Sub-domain 2	5164	10283	123940	740380	3.09	5.84
Sub-domain 3	9255	18473	222124	1330060	9.46	12.22
Sub-domain 4	6794	13547	163060	975388	5.37	8.60
Sub-domain 5	5367	10670	128812	768244	4.96	7.98

Sub-domain 6	5278	10508	126676	756580	4.82	7.99
Sub-domain 7	6209	12382	149020	891508	6.67	9.84
Sub-domain 8	5877	11720	141052	843844	3.62	6.73
Sub-domain 9	**9704**	19375	232900	1395004	**12.76**	15.87
Sub-domain 10	5389	10740	129340	773284	5.31	8.47
Sub-domain 11	8238	16440	197716	1183684	6.27	9.44
Sub-domain 12	5800	11562	139204	832468	3.95	7.01
Sub-domain 13	4517	9002	108412	648148	3.33	7.11
Sub-domain 14	7357	14678	176572	1056820	5.22	9.29
Sub-domain 15	4902	9768	117652	703300	2.81	7.21
Sub-domain 16	5722	11411	137332	821596	3.46	8.18

Fig. 11. Processing information for the 96105 vertices dataset using the 10000 vertices division

- the column “Process time” contains the duration (in seconds) of the triangulation procedure for each sub-domain
- the column “Total time” contains the complete duration of the associated process. It includes the communication time and the subdivision time spent by the master process.
- The spatial distribution of the 96105 vertices dataset is uniform (Figure 5). The dataset is divided in 16 sub-domains and the sub-domain size condition is reached before the maximum available processors condition.
- the largest sub-domain contains 9704 vertices and the triangulation time was 12.76”.
- the processors’ load is well balanced, excepting the first one (explained by Figure 5, there is an area in the upper-right corner without vertices).

	Points	Triangles	Bytes Received	Bytes Sent	Process time (sec)	Total time (sec)
Total:	**81947**	**163672**	**1966824**	**11752944**	**max:72.39**	**135.14**
			1.9 M	11.7 M		
Sub-division:		163234				3.86
Extra-surface:		438				9.1
Output creation:						55.68
Sub-domain 1	2500	4978	60004	358420	0.51	4.39
Sub-domain 2	862	1704	20692	122692	0.06	3.96
Sub-domain 3	247	470	5932	33844	0.01	3.96
Sub-domain 4	475	936	11404	67396	0.02	3.91
Sub-domain 5	2197	4355	52732	313564	0.40	4.70
Sub-domain 6	432	845	10372	60844	0.12	4.70
Sub-domain 7	1174	2320	28180	167044	0.08	4.71
Sub-domain 8	412	806	9892	58036	0.01	4.70
Sub-domain 9	789	1547	18940	111388	0.04	4.75
Sub-domain 10	447	870	10732	62644	0.01	4.72
Sub-domain 11	2531	5027	60748	361948	0.44	5.18
Sub-domain 12	387	753	9292	54220	0.01	4.76

Sub-domain 13	1878	3727	45076	268348	0.27	5.05
Sub-domain 14	1186	2347	28468	168988	0.12	5.09
Sub-domain 15	9619	19202	230860	1382548	8.05	12.98
Sub-domain 16	1298	2570	31156	185044	0.12	5.10
Sub-domain 17	738	1458	17716	104980	0.04	4.97
Sub-domain 18	1147	2271	27532	163516	0.20	5.78
Sub-domain 19	**20015**	39978	480364	2878420	**72.39**	77.99
Sub-domain 20	1589	3148	38140	226660	0.18	6.08
Sub-domain 21	500	976	12004	70276	0.02	6.04
Sub-domain 22	**11628**	23217	279076	1671628	12.81	19.01
Sub-domain 23	**18740**	37439	449764	2695612	65.47	71.88
Sub-domain 24	1156	2290	27748	164884	0.21	7.21

Fig. 12. Processing information for the 81947 vertices dataset using the 10000 vertices division

In the second case, we can observe that:

- the distribution of the vertices is not uniform (Figure 4),
- the maximum number of sub-domains is reached before obtaining the maximum size (10000) for all sub-domains,
- there are 3 sub-domains larger than 10000 vertices, two of these (sub-domains 19 and 23) have almost double size,
- the maximum execution time is 72.39” (6 times the maximum time for the previous dataset),
- the processors’ load is not balanced (the largest sub-domain contains 10 times more vertices than the smallest sub-domain).

3.4 Results on the Other Three Platforms

As we mentioned in paragraph 3.2, the same jobs were executed on other platforms, where the parallel execution was emulated using the MPI / LAM-MPI interface. The results are presented in the table bellow.

„*lam_linux_p4*” configuration						
vertices	**d 0**	**d 2000**	**d 5000**	**d 10000**	**d 25000**	**d 50000**
397286	***	1076.01	1015.82	985.21	940.51	1507.94
96105	884.80	90.36	89.56	102.99	174.37	323.15
81947	643.60	137.67	126.55	123.93	153.79	305.96
20000	39.59	10.56	12.95	18.20	39.77	39.75
10000	12.15	5.51	7.11	8.40	12.29	13.01
the „*mpi_win32_dual_core*” configuration						
397286	***	570.33	508.01	478.10	443.51	850.05
96105	581.10	35.81	35.66	39.08	81.66	190.26
81947	420.82	69.78	61.86	58.13	78.97	184.59
20000	25.13	5.81	6.62	8.01	24.98	25.06
10000	6.98	2.73	3.16	3.75	6.98	6.99

„*mpi_win32_p4*” configuration						
397286	***	542.73	501.35	486.69	465.76	740.05
96105	514.76	52.95	52.72	58.04	91.84	159.33
81947	370.55	77.10	72.96	71.13	83.92	153.83
20000	21.76	9.11	9.95	12.32	21.92	22.25
10000	7.29	4.34	4.88	5.47	7.20	7.23

Fig. 13. Processing times on the other three platforms (seconds)

The general conclusions are similar:

- the a_397286 data set was never sequentially processed in an acceptable time,
- the optimal subdivision size is that of 10000 points,
- the processing time is larger for non-uniform distributed datasets than for uniform datasets,
- there is also an unexpected conclusion: the distributed 3D model generation is faster on Windows machines than on Linux machines, including the 4 processors server of the local grid network. Figure 14 shows the processing times of the 397286 vertices dataset on all 4 platforms.

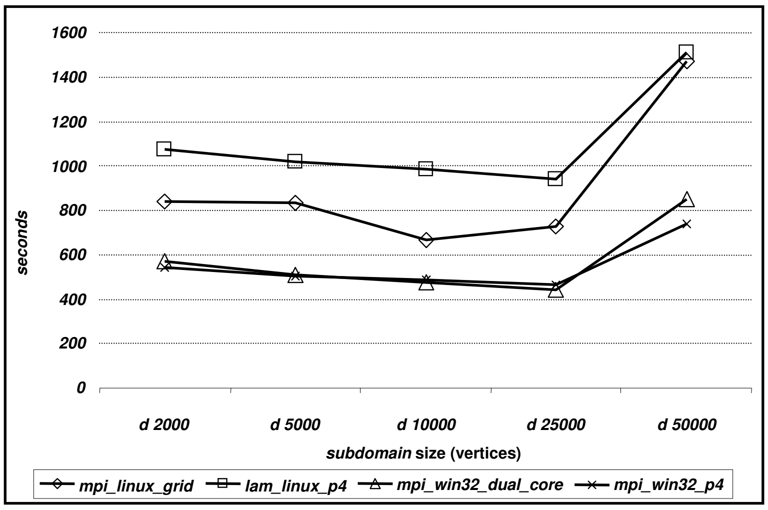


Fig. 14. Graphic representation for the 397286 vertices dataset processing time in all configurations

The 3D model generation was 1.5 times faster on the “mpi_win32_dual_core” configuration than on “mpi_linux_grid” configuration when the 10000 points sub-division size was used. This can be explained by the following:

- the Microsoft C++ compiler optimizes the generated code for his original operating system,
- for each configuration we used different versions of the MPI library,
- on Scientific Linux / CentOS operating systems the code was generated using the default release settings of the g++ compiler,

- the main time difference is created by the output shape file writing function as is presented in the following extras of the processing summaries:
  - *mpi_linux_grid* configuration

```
Shape 'a_397286_Gen' (output) created in 00:04:33.94
Shape 'a_397286' (397286 vertices) processed in 00:11:04.50
```

  - *mpi_win32_dual_core* configuration

```
Shape 'a_397286_Gen' (output) created in 00:01:23.13
Shape 'a_397286' (397286 vertices) processed in 00:07:58.10
```

  If we exclude the output creation time, the processing time is approximately the same 6'30".
- In Windows, the 3D model generation was the single active application (except for the default operating system services).

4. Conclusions and Future Work

This paper presents the results obtained using the Grid service implemented for the distributed 3D terrain model generation:

- since 3D generation algorithms are time and memory intensive, parallel implementations are important both for an improved performance and for solving problems for which memory requirements are too large for serial machines. For example, our 397286 vertices dataset was processed in about 8 minutes using the distributed method.
- the processing time is optimal for the 10000 vertices domain sub-division.
- the service was tested on various platforms using different vertices distributions in the test datasets.
- the service was optimized for the triangulation procedures, but there are also other functions which must be optimized (output shape file creation).

Acknowledgments. The research presented in this paper was supported by the CEEX Research Contract nr. 74, CEEX-II03/31.07.2006, "Academic Grid for Complex Applications".

References

[1]. John E. Harmon and Steven J. Anderson, „The Design and Implementation of Geographic Information System", Published by John Wiley & Sons, Inc., Hoboken, New Jersey, (2003)

[2]. L. Paul Chew. Constrained Delaunay Triangulations. Algorithmica 4(1): 97–108, Springer-Verlag, New York, LLC, (1989)

[3]. Cignoni, P., De Floriani, L., Pascucci, V., Rossignac, J., and Silva, C. T. Multiresolution modeling, visualization, and compression of volumetric data. IEEE Visualization,(2003)

[4]. J. R. Shewchuk, Lecture Notes on Delaunay Mesh Generation, Department of Electrical Engineering and Computer Science, University of California, (1999)

[5]. L. De Floriani, S. Bussi, P. Magillo, Triangle-Based Surface Models, in Intelligent Systems and Robotics, Editors: G.W. Zobrist, C.Y. Ho, Gordon, Breach Science Publishers (2000) 340-373

[6]. C. Hardwick, Implementation and Evaluation of an Eficient 2D Parallel Delaunay Triangulation Algorithm, Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, June (1997)

[7]. The LAM/MPI Team, Open Systems Lab, LAM/MPI User's Guide, Version 7.1.4, Indiana University

[8]. Mitică Craus, Cristian Amarandei, Bogdan Romanescu, Algoritmi şi limbaje pentru calcul paralel, Îndrumar de laborator, (2005)

[9]. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Applications. Springer-Verlag, Heidelberg, (2000)

Grid Based Visualization Using Sort-Last Parallel Rendering

Simona Aruştei, Alexandru Archip,

Cristian-Mihai Amarandei

Department of Computer Science and Engineering
Technical University of Iasi
{alexandru.archip,sarustei,camarand}@cs.tuiasi.ro,

Abstract. *Complex visualization of large data sets has become a mainstream requirement for many scientists and engineers. In this paper we present the architecture of a Grid framework that aims to facilitate the visualization of large data sets through the use of object space parallel rendering. In our approach the visualization framework is based on a service-oriented architecture and we make use of a data decomposition scheme that reduces the requirements for communication during processing. The main advantage of the proposed framework is its extendibility as its generic components can easily be customized to grid enable different types of parallel graphic applications. It represents a work in progress towards the development of a generic grid-based visualization framework.*

Keywords: *Grid Visualization, Parallel Rendering, Grid Services, GT4*

1. Introduction

The need to visualize immensely complex structures and ideas has pushed high performance visualization and graphics towards distributed computing that can be provisioned by Grid systems, even though parallel graphics processing offers truly unique opportunities to engage in real-time visualization. The need for real-time visualization of large data sets leads the way towards developing a Grid implementation of a parallel rendering pipeline for which the communication overhead is minimized and the data distribution scheme doesn't involve poor scalability.

In order to give such a solution for visualization of large data sets in a grid environment, we designed a framework based on grid services that employs parallel rendering based on a sort-last scheme. The main requirements we accounted for in designing the framework were extensibility, flexibility and portability. Thus we developed the framework based on generic components so it could be easily extended to provide the functionality of other parallel rendering schemes (like sort-first) or for large image processing.

2. Related Work

Parallel visualization and Grid systems have rapidly evolved in the past years. The graphics accelerator market and the availability of high-performance, low-cost networking technologies have enabled construction of inexpensive, high performance visualization clusters out of commodity components. Also, the development of distributed computing first into meta-computing and then into Grid computing provides solutions for the security, administration, scheduling, and data-transfer problems that accompany a geographically and administratively dispersed set of resources. The convergence of parallel visualization and Grid systems has made using distributed graphics pipelines on the Grid possible.

The gViz project [1] has Grid-enabled IRIS Explorer, using the XML-based skML to describe visualization pipelines. The gViz library enables steering and communication between simulation and visualization components, both running on Grid resources. Other work includes the Grid visualization kernel, a middleware extension that lets a scientific visualization's various components—data sources, simulation processes, and visualization clients—interconnect [2]. GVK can dynamically change this visualization pipeline without user knowledge, adapting to changing network conditions. GVK is implemented as input and output modules for numerous modular visualization environments, including OpenDX and AVS/Express.

RealityGrid [3] aims to facilitate computational studies of complex condensed-matter systems. RealityGrid applications are built in a three-component structure of simulation, visualization, and steering client. Scientists can interact with applications during runtime through the steering client and view a remote rendering of the output.

Most visualization systems developed within Grid systems support scientific visualization over sometimes great distances. However, they don't really provide for more tightly coupled, interactive, cluster-based systems in which nodes can be used individually. However, current visualization systems don't support interactive Grid-based OpenGL applications and they often require purpose-built applications. Chromium [4] is one of very few open source cluster solutions supporting OpenGL, but it isn't compatible with Grid middleware. The jgViz [5] approach uses Chromium to handle parallel graphics tasks and enables the automatic discovery and use of graphics accelerators in a commodity Grid environment across fast networking technologies.

Remote rendering systems typically lack the flexibility that the Grid aims to provide, so integrating visualization with Grid middleware is often desirable, a strong requirement being the implementation of dynamic pipeline scheduling [6]. When such a middleware is not available, the solution is to provide a service oriented framework for visualization.

In the remainder of the paper we describe such a visualization framework and it is organized as follows: section 3 presents the architecture of the proposed framework and gives a detailed description of its components and of the parallel rendering scheme adopted, followed by a case study implemented as proof of concept. Section 5 presents the conclusions and issues that we will address in future development of the framework.

3. Grid Visualization Framework

Our framework was designed in a service-oriented fashion and is based on three main components: the client component, the service component and the work execution component. Both the client component and the work execution units act as clients for the service component, thus allowing the communication of input and output data in a service-oriented manner, while the collaboration between the three components is notification-based. This modular design was adopted in order for the framework to be easily extended for different types of parallel rendering schemes and to allow an easy configuration for different application scenarios. In order to employ parallelism in the rendering pipeline we adopted a sort-last parallel rendering scheme. The parallel rendering pipeline and the components of the visualization framework are detailed in the following.

3.1. Sort-Last Parallel Rendering Scheme

For our visualization service we considered object space parallel rendering as, in contrast with image space sorting, this technique neither involves an extremely fine grained parallelism nor does it require inter-process communication - that can often be very data intensive - in order to solve the compositing stage. Within Object Space parallel visualization, each node (individual unit of the parallel system, typically a single machine or processor) is responsible for the rendering of its block of data, irrespective of whether it may actually be visible at that precise moment. Object Space parallelization is also known as Sort-Last [7], reflecting the late stage in the graphics pipeline at which the graphics primitives are sorted from object-space into the resultant image-space (Figure 1). Each node computes the values of the pixels for its associated sub-set and sends them to the compositing node which solves for the visibility of the pixels received from all processing nodes.

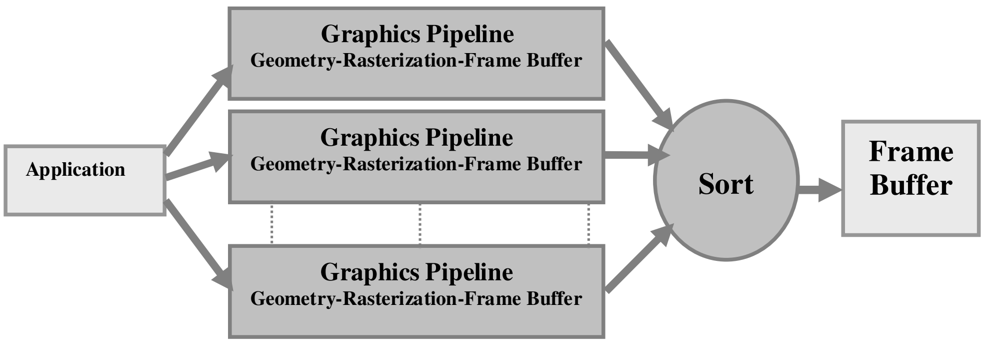

Application
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Sort
Frame Buffer

Fig. 1. Sort-Last Graphics Pipeline [7]

Object space parallelization is less prone to load imbalance than other parallel rendering schemes because it is not sensitive to the distribution of primitives within the image, since most of the computations are performed using the initial object-space mapping of primitives to processors. Even though it is very scalable, the main disadvantage of the Sort-Last parallelization of the graphics pipeline is that it usually requires an image composition network with very high bandwidth and processing capabilities to support transmission and composition of overlapping depth images.

3.2. Architecture of the Visualization Framework

As there is a need for a framework for visualization general enough to support further development and enhancement through either improvement of initial modules or addition of other parallel rendering techniques, we developed our visualization service based on three components that correspond to the three types of nodes involved in a parallel graphics pipeline (Figure 2): the client component (OpenGL Client), the coordinating component (Render Service) and the work execution component (Render Worker).

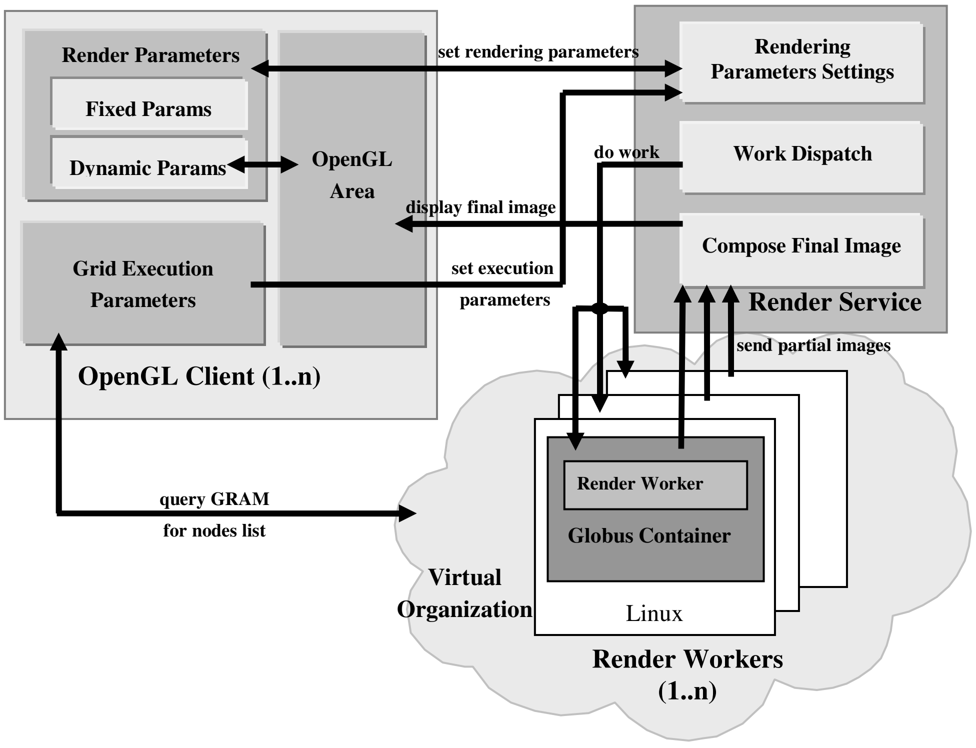


Fig. 2. Architecture of the Visualization Service

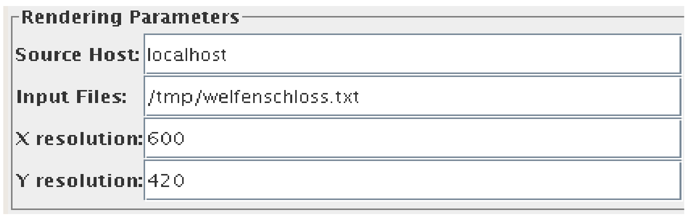
The rendering process is driven and displayed through the client component. This module allows the user to interact with the visualization system in two ways: first, by specifying or providing the input data to be rendered, and second by driving the visualization parameters through the interaction with an OpenGL visualization area.

The grid execution parameters and the visualization parameters are provided to the Render Service which controls the execution of the Render Workers that produce the images of the associated sub-sets of graphics data. The Render Service is also responsible with compositing the final image to be sent to the client, based on the partial results received from the workers. Both the client and the worker components act as clients for the Render Service. The three components of the visualization service will be discussed in more detail in the following.

The Client Component

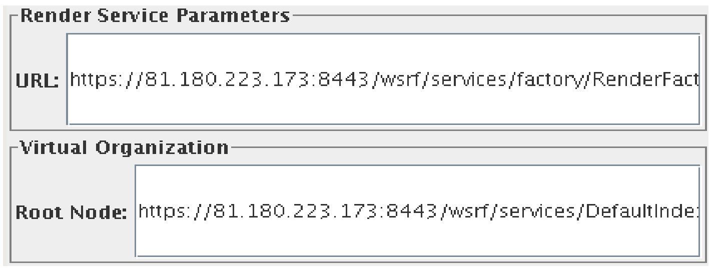
This component is a Java application that uses OpenGL through the means of a wrapper library (JOGL [9]) and allows the user to interact with the visualization system and to explore the results of the rendering. The interaction between the OpenGL Client component and the Render Service is described by two types of parameters: parameters that drive the execution of the visualization system on the grid and rendering parameters

The former parameters express the information related to the input data (the location of the files containing the graphics primitives to be rendered – Figure 3), the resolution and aspect of the resulting image and the configuration of the parallel rendering pipeline (the user can specify the number of rendering nodes needed and can specify or choose the exact nodes that will execute the rendering – Figure 4).


Rendering Parameters

Source Host:	localhost
Input Files:	/tmp/welfenschloss.txt
X resolution:	600
Y resolution:	420

Fig. 3. Setting input data set and OpenGL rendering area resolution


Render Service Parameters

URL: https://81.180.223.173:8443/wsrf/services/factory/RenderFact

Virtual Organization

Root Node: https://81.180.223.173:8443/wsrf/services/DefaultInde

Fig. 4. Setting URI of the Render Service and the full URL to the root node of the virtual organization.

The nodes capable of executing the Worker tasks are queried using GRAM. The user provides the full URL to a `DefaultIndexService` running in a GT4 container that is defined as the root node of the virtual organization. This URL is queried to get a list of all nodes in the VO which are listed in the client application (Figure 5). The user can choose one or more nodes that will execute the rendering work.

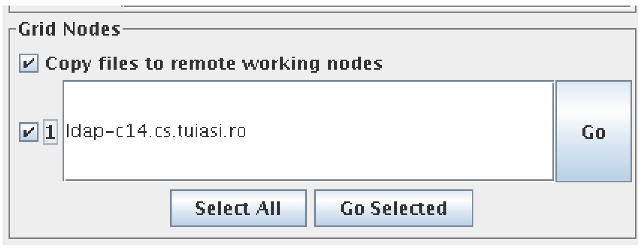


Fig. 5. Nodes within Virtual Organization capable of executing the rendering tasks

The latter type of parameters are needed to drive the rendering process by specifying modeling and visualization (rotation, translation, scaling etc) and projection transformations to be applied to the input data. These are dynamic parameters determined by the client application based on the user interaction with the OpenGL area. Any update on the rendering parameters triggers a new execution of the rendering pipeline on the working nodes.

The client is informed about the status of the framework (jobs states, file transferring, image completion) and can also receive error or warning messages related to the settings needed to be made on the working nodes in order for the jobs to execute successfully. Because the rendering on the working nodes is accomplished off-screen, the host running the client can also be configured to execute the job of a working node.

The Service Component

The Render Service component is a grid service developed using Globus Toolkit 4. It runs inside a grid service container and it administrates the execution of the Render Worker processes in the system. In the grid system there can be only one such component but there might be a need for multiple client applications to run at the same time. In order to solve this situation, the Render Service component was designed using the factory-instance pattern [10]. Using this pattern, when the client needs the creation of a new resource, it will contact a fabric service that will manage the instantiation and initialization of a new resource. Because multiple resources need to be managed at the same time they are assigned a unique key needed for their identification. Thus, the fabric service will return an *endpoint reference* information (EPR) associated to a WS-resource. The EPR will contain the URI of the service as well as the resource key so that the client can invoke the service operations through the means of an instance service.

For deploying specific image processing applications on the grid, like parallel rendering of Scalable Vector Graphics files [8], the work distribution can be managed by the client component. But, in order to make use of a completely service-oriented architecture, we employed the work dispatching through the service component. This is also desirable since, no matter the type of parallel rendering involved, the visualization framework must implement a composition stage. Thus the client will have at most the task of specifying the number and exact hosts that will execute the work, while the whole process of transferring input files, launching jobs, gathering results and creating the final image is handled by the service. The collaboration between the three components of the framework is notification-based. Notifications are delivered to the service each time a working node finishes the execution of a job (either successfully or not) and a notification is delivered to the client component when the service component completes the compositing stage and the image is ready to be transferred and displayed.

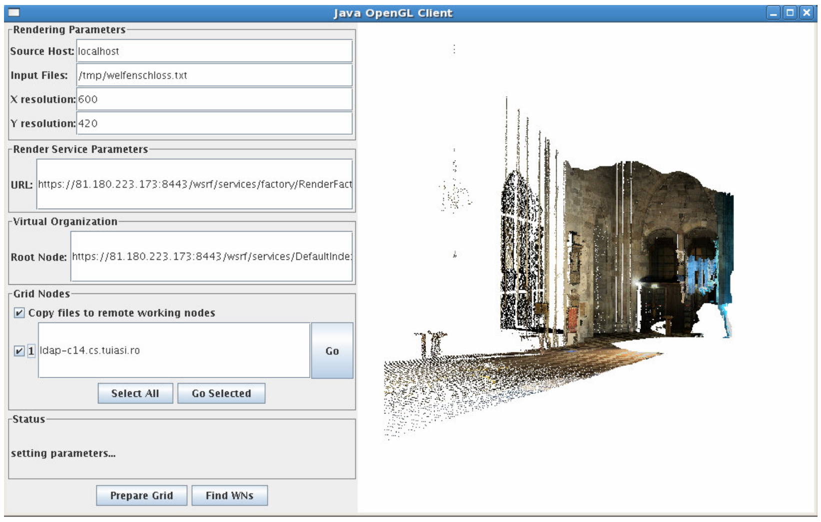
The initialization of the visualization framework is triggered by the client application which causes the service component to partition the data set according to the number of working nodes specified through the client application. The Render Service sends the rendering jobs to the working nodes through a worker dispatcher that uses gridFtp [11] and GRAM [12] to transfer the needed files and to launch the Render Worker components. After the working nodes have finished their rendering tasks, the Render Service begins the compositing stage of the final image by depth sorting the resulting pixels. When the final image is ready the client is notified that it can begin transferring it from a specific location.

The Worker Component

This component represents the application launched on a working node by the rendering service through GT4 GRAM. It was developed as a Java application as it acts as a client for the grid service. The Render Service pushes the files needed for the host machines to act as clients for the grid service and prepares the environment by executing a script on the host. Once launched, the Render Worker application renders its corresponding input data into an off-screen buffer. The contents of this buffer and the depth information associated with each pixel are then transferred by the working node to the Render Service which will accomplish the composition step. The working nodes transfer the partial results by accessing a method exposed by the service. In order for multiple instances of the client component to run in the same grid environment, the working nodes must contact the correct service instance with appropriate credentials. This is accomplished by transferring the corresponding EPR and proxy information to the workers and securing the service component to be accessed only with the credentials of the client creating a specific instance.

3.3 Application: Rendering a Point Cloud

We tested our visualization service by rendering a point cloud acquired with a range scanning device (Figure 6). The range scanners are capable of producing highly detailed point clouds, so, even though point primitives can be rendered simply and relatively fast, problems arise due to the often huge size of the datasets. Our test data is represented by a 3 million points scan (with color information) from the interior of St. Stephen's Cathedral in Vienna[8]. The functionality and correctness of the proposed framework was tested on an offline test bed with a Globus Toolkit 4 secured service container running on the front-end node.



Source Host: localhost
Input Files: /tmp/welfenschloss.txt
X resolution: 600
Y resolution: 420
URL: https://81.180.223.173:8443/wsrf/services/factory/RenderFact
Root Node: https://81.180.223.173:8443/wsrf/services/DefaultInde
[✓] Copy files to remote working nodes
[✓] 1 ldap-c14.cs.tuiasi.ro
setting parameters...

Fig. 6. Visualizing the St. Stephan's Cathedral point cloud using Sort-Last parallel rendering executed on the proposed visualization framework

4. Conclusions and Future Work

We have presented a visualization framework based on a service oriented architecture which enables a sort-last parallel rendering scheme in a Grid environment. Our framework was designed to satisfy at least the following requirements:

- *reusability and extendibility* - the code can be easily reused and extended for new types of graphics applications. This is accomplished as the framework is based on generic components that can be customized to implement different functionalities (like sort-first parallel rendering schemes or large image processing),

- *efficient exploitation of the infrastructure* - the framework allows the execution of both sequential and parallel codes, depending on the resources available in the grid environment,

- *portability* - the framework transparently deals with the heterogeneity of the resources from the grid environment - this requirement is accomplished by developing the framework in Java.

[8] Data provided by the Institute for Computer Graphics and Algorithms, TU Vienna, Austria.

With these considerations, our framework could be easily extended to supply the functionality of a configurable grid service that provides application developers with a high level programming model, hiding the complexity of dealing with web services and Grid technologies. Such an architectural design of the framework would allow custom functionality to be plugged into an adaptive grid service in a simple manner, while adding built-in functionality to the framework would allow the development of an extension dedicated to grid visualization

Further development of the visualization framework will also address the problem of monitoring and dynamic rescheduling of the graphic pipeline and providing Quality of Service support.

Acknowledgments. The research for this paper was supported by the Grant 74 CEEX-II03/31.07.2006.

References

[1]. K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, J. Wood, Visualization in Grid Computing Environments, Proc. IEEE Conf. Visualization (VIS 04), IEEE CS Press (2004) 155–162.
[2]. P. Heinzlreiter, D. Kranzlmuller, Visualization Services on the Grid: The Grid Visualization Kernel, Parallel Processing Letters, vol. 13, no. 2 (2003) 135–148.
[3]. J.M. Brooke, P.V. Coveney, J. Harting, S. Jha, S.M. Pickles, R.L. Pinning, A.R. Porter, Computational Steering in RealityGrid, Proc. UK e-Science All Hands Meeting (2003); www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf.
[4]. G. Humphreys, M. Houston, Y.-R. Ng, R. Frank, S. Ahern, P. Kirchner, J.T. Klosowski, Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, ACM Trans. Graphics, vol. 21, no. 3 (2002) 693–702.
[5]. A. J. Fewings, N. W. John, Distributed Graphics Pipelines on the Grid, IEEE Distributed Systems Online, vol. 8, no. 1 (2007) art. no. 0701-o1001.
[6]. J. Shalf, E.W. Bethel, The Grid and Future Visualization System Architectures, IEEE Computer Graphics and Applications, vol. 23, no. 2 (2003) 6–9.
[7]. S. Molnar, M. Cox, D. Ellsworth, H. Fuchs**,** A Sorting Classification of Parallel Rendering. IEEE Computer Graphics & Applications, 14(4):23.32 (1994).
[8]. B. Jacob, M. Brown, K. Fukui, N. Trivedi**,** Introduction to Grid Computing, IBM redbooks (December 2005) 197-220.
[9]. https://jogl.dev.java.net/
[10]. http://gdp.globus.org/gt4-tutorial/multiplehtml/ch05.html
[11]. http://www.globus.org/toolkit/docs/3.2/gridftp/
[12]. http://www.globus.org/toolkit/docs/3.2/gram/ws/

Remote Visualization Techniques for Distributed 3D Scenes

Dorian Gorgan, Rareş Barbantan

Computer Science Department,
Technical University of Cluj-Napoca
dorian.gorgan@cs.utcluj.ro, rares.barbantan@gmail.com

Abstract. *This paper proposes two solutions to the problem that appears when trying to reassemble the output of a distributed task in a Grid environment. Although the solutions can have a more general theoretical scope, discussions are made on experimental results on the reconstruction of a distributed 3D scene.*

1. Introduction

Suppose there is a need to simulate a complex 3D environment, where the 3D rendering is not the most expensive operation involved. This happens to be the case with the AOM model, on which the experiments are based. The computing tasks are assigned to several jobs and distributed over the Grid. The problem appears when trying to gather information from all running jobs and reconstruct the scene.

Due to the parallel and distributed nature of a Grid network, the scene needs to be constructed and constantly updated from several update messages that arrive simultaneously from a number of jobs running concurrently on the Grid. The scene not only needs to take into consideration messages from all jobs, but also needs to take into consideration the order in which messages arrive. If we add to that the problem of small bandwidth, high network traffic and NAT (Network Address Translation) firewalls then the time gained with distributing the task over the Grid might be lost when trying to reassemble the output and deliver it to the user.

The aim of this paper is to find the best solution that offers both an optimal execution time and greater flexibility in using the application. This paper is structured as follows: Section 2 makes a brief review of some of the related projects especially the Active Objects Model, Section 3 presents the proposed scenarios and solutions, based on the location of the user running the application: inside or outside the Grid, Section 4 presents and discuses some test results and finally Section 5 draws some conclusions and future development directions are discussed.

2. Related Work

Although the conclusions of this paper can apply to any Active Object Model [2], this paper is based on the theoretical model presented in [1] and further developed in [7], [8], and [9]. Basically, the AOModel is comprised of a set of finite entities, these being of two types: active and passive. The only active entities are the “aliobs” or “active objects”. They are entities with a well-defined private behavior. The execution of that behavior implies that certain actions are executed on other objects or the object in question itself. This means that an active object can modify the structure and behavior of another object while they are both executing, without the need to recompile the code, thus making the objects adaptable [14]. The communication between the threads implementing the active objects and even between the objects themselves is performed by exchanging messages.

Since communicating objects can be located on different machines, the messages exchanges travel through the Grid network with the help of a web-service. Whenever a message leaves the node, it is converted into text, using an XML type structure. This allows compatibility with external modules implemented on other platforms or programming languages. This approach is somehow similar to Oasis’s [14] ProActive [13] project.

The main structure of the application is comprised of several jobs running on different nodes on the Grid network, each job processing an active object. They all communicate to each other by exchanging messages though a web service. Whenever there is a change in the visual representation of an object, the corresponding job sends an update message. All these messages are then collected and the scene is reconstructed, updated and presented to the user. This paper focuses on the graphical rendering of the scene, as it is a relatively new area in grid networking, with research still in progress. See [3],[6].

3. Proposed Solutions

This section proposes several scenarios in which the problems mentioned about dealing with reassembling a distributed task may arise and also offers some solutions to those problems. The scenarios proposed differ mainly on either the location of the users accessing the application:

- *internal users* which are inside the Grid;
- *external users* which may be behind a firewall and access the Grid through the internet.

Or the type of interaction between the user and the application:

- *spectator* – the user just witnesses the execution of the model, cannot interfere with it;
- *actor* – the user interacts with the scene.

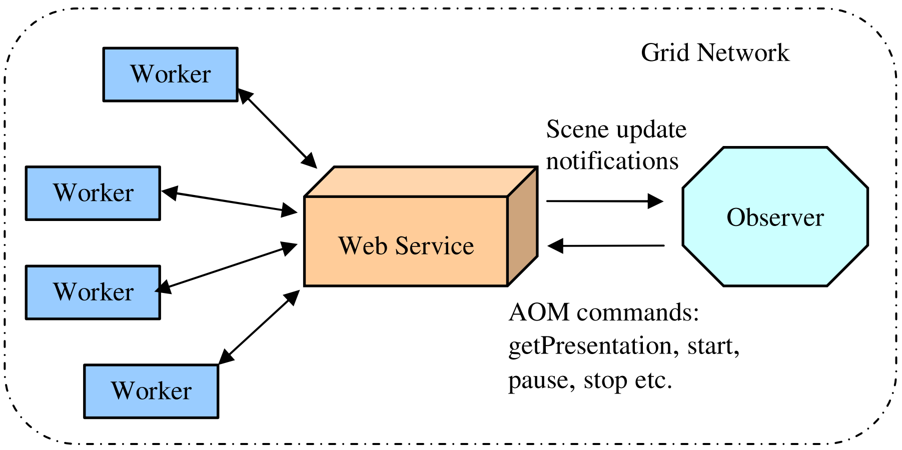

Grid Network
Worker
Worker
Worker
Worker
Web Service
Scene update notifications
Observer
AOM commands: getPresentation, start, pause, stop etc.

Fig. 1. The AOM application structure

3.1 Internal User

Suppose the user of the application has access to the Grid network. This means that the visualization client runs in the same network as the jobs running the computations. This means that it can access the web service directly and receive the update messages itself. This scenario is very convenient as it allows the client to benefit from one of the advantages of using Grid-specific web services: notifications.

At the beginning of the execution, the presentation client can register with the web service to be notified of any update received. Using these updates, the client can then update the scene at the same rate at which the messages are received, thus providing a very accurate representation of the model's execution.

This situation represents the present state of the AOM application and is presented in Figure 1. This paper deals with the problems appeared when trying to extend the application to suite the needs of a broader audience, one that has no knowledge of or access to a Grid network. This situation is presented in the next section.

3.2 External User

A more common situation is that of a user trying to use a Grid application from outside the Grid, using the internet. This means that there should be an intermediary, a web application, which can be accessed from outside the Grid and can be used to run jobs inside. The problem that appears in this case is that of typical web applications: there is no telling where the user is located. And the user can be behind a NAT firewall.

NAT involves re-writing the source and/or destination address of IP packets as they pass through a Router or firewall. Most systems using NAT do so in order to enable multiple hosts on a private network to access the Internet using a single public IP address. The problem with NAT is that Hosts behind NAT-enabled routers do not have true end-to-end connectivity and cannot participate in some Internet protocols. Services that require the initiation of TCP connections from the outside network, or stateless protocols such as those using UDP, can be disrupted. Unless the NAT router makes a specific effort to support such protocols, incoming packets cannot reach their destination. In Figure 2 one can easily observe that the communication between the user situated outside the Grid and the application running inside it is only one way.

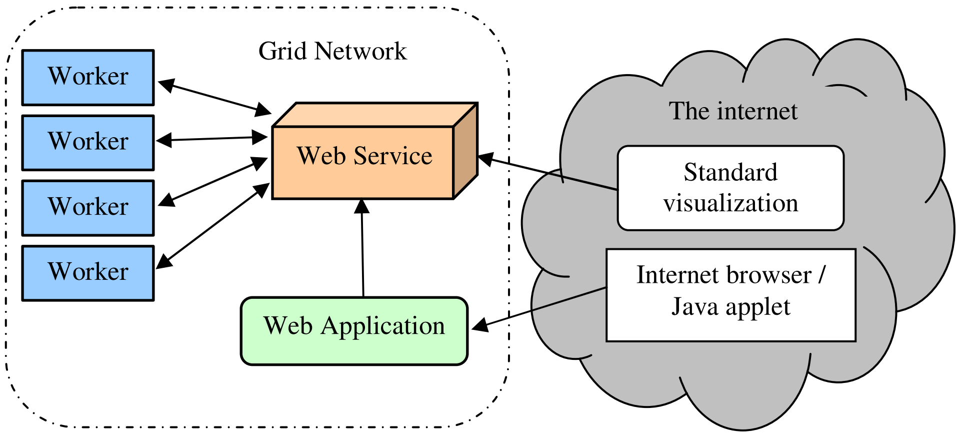


Fig. 2. The extended AOM application structure

This is also the case with web-service notifications. The web-service has no way of telling where the client is, so it cannot send any notification when an update message has arrived. The solution would be to have the client periodically interrogate the web-service about the updates. Using this approach still allows us to know about the updates, but we lose any information about the time interval at which these updates were received. Therefore, the spatial representation of the active model's execution would be just as accurate, but the temporal representation would not.

One solution to this new problem could be having a very small interval between interrogations, much smaller than the average time between update messages. This approach is not very feasible from three reasons:

- the rate at which update messages arrive is totally random, so there is no way to compute a realistic mean;
- the overhead and web-traffic of almost continuously interrogating the web-service would be too large;
- the traffic exchanged between the client and the application would be constant even if no update messages are being received.

Another, more feasible solution would be to incorporate the time of arrival in the update messages. That way, whenever the client interrogates the web-service it receives a list of update messages along with their time of arrival. Using this information, the temporal representation regains its consistence with the actual execution of the model.

The only disadvantage would be that pauses between message updates that are larger than the period between interrogations tend to be amplified in the representation.

3.3 “Spectator” User

A spectator user is that user which can observe the execution of the Active Object Model, but cannot interfere with it. This approach is called streaming and some solutions for the grid are already available [15]. This scenario is more a side effect than a request. Suppose the client using the application runs on a machine that is not equipped with a powerful graphics card. This makes 3D rendering very slow, or even impossible. For this reason, the reassembly and the drawing of the distributed scene is performed in the web-service, sending further to the client the rendered image.

The advantage of this solution is that it imposes no special requirements on the client machine, and has the same performance, no matter what the hardware configuration is. The downside of this approach lies in its lack of interaction, and in the high network traffic resulted when sending images through the internet instead of simple text messages.

The lack of interaction could of course be resolved with the help of an extra layer imposed in the client over the image, which would intercept user interface commands and transform them into AOM messages. This again is not recommended, as it would split implementation of AOM logic and would add extra unnecessary image processing computations. Another downside is that for the image needs to be updated from the server also for every change in the observer’s position not just for structural changes in the scene. This of course leads to a very low frame rate.

3.4 “Actor” User

In contrast to the *spectator*, the *actor* takes part in the execution of the Active Objects Model, interacting with the scene and the objects in it.

This is the most common situation as we want the user to be able to interact with the scene, but most important, to be able to explore it. Since it is rendered on the client, it is not just a picture, and the scene can be freely explored, without the need to exchange messages back and forth between the client and the web application. The message exchanging only needs to take place on the presence of update messages. This greatly reduces the network traffic when comparing with the previous case of a *spectator* user. The downside, as mentioned earlier, it that it imposes certain hardware requirements on the client accessing the application.

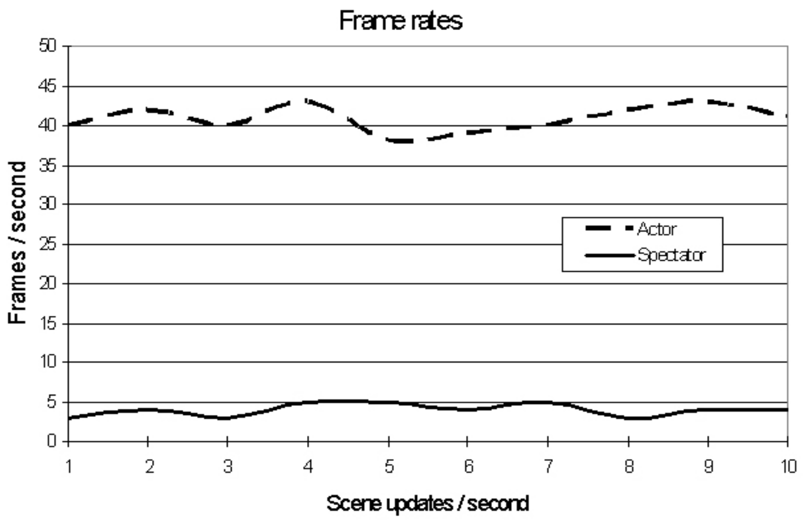


Fig. 3. Frame rates for external actor/spectator

4. Experiments and Results

When performing the experiments and measurements we are not interested in comparing a sequential execution of the active objects model with the distributed one, as the only difference would be in the increased number of update messages received in the case of a distributed application. This has already been proved in a previous work [4].

Also, since the cases where the user is inside/outside the Grid network, are totally different and latter presents some restrictions specified in the previous sections, a comparison between the two would serve no purpose. It is obvious that the configuration for an external client is less demanding, therefore more suitable for a web application.

The experiments focus on the external users and the solutions proposed to resolve the problems that appear when having this configuration. Thus a comparison is made between the *spectator* and *actor* configurations.

When trying to recompose and render a continuously changing distributed 3D scene in real time over the internet we need to take into consideration three aspects: the frame rate, the network traffic and the accuracy of the rendering.

4.1 Frame Rate

By frame rate we understand the rate at which the screen is repainted and it is not to be confused with the rate at which the scene is updated.

As one can see from Fig. 3, the frame rate (or the refresh rate) is almost constant and does not depend on the number of updates received from the Grid. In the case of the *actor* user, since the 3D scene is rendered on the client machine, the frame rate is much higher and is only limited by the computer's processor and graphics card. On the other hand, the *spectator* is not limited by hardware but by the network speed, which greatly reduces the number of frames transmitted due to the large size of an image (frame).

4.2 Network Traffic

The biggest problem of all web application, the limited bandwidth available, represents a major limitation for the performance of the AOM too.

Again, in the case of the *spectator* user, the rate at which images are sent over the network does not depend on the rate of messages received therefore the network traffic generated also remains constant and at a high value.

When update messages are transmitted and not already rendered images, then the traffic generated depends of course on the number and size of the messages sent. With every message sent, there is a communication overhead that increases the size of the actual message being sent. Therefore, as one can see from Fig.4, the solution of sending more messages at once is more efficient (external actor) than sending them one at a time (internal user), at least from the traffic's point of view as there is less overhead added. The overall efficiency is questionable, as sending more messages at once implies more processor time for handling the list of messages. The best solution in this case depends on the configuration of the environment in which the application runs and should be chosen the one that best deals with the more serious limitation: network bandwidth or computing power.

4.3 Rendering Accuracy

The Active Objects Model is represented as a continuously changing three dimensional scene of objects. The structural information about the scene is contained in the messages being sent between the workers, the web service and the observing client. Therefore any technique used to render the scene has as a result an exact replica of the AOM scene, from a structural point of view.

From the execution time's point of view, accuracy means being able to represent the changes that take place in the scene at the same pace as they did in the model.

Take for example another type of 3D application like a scientific simulation or why not, a video game. Being run on the same machine, the mathematical model and its presentation, messages from the mathematical model like objects' positions reach the presentation instantly. But from time to time, rendering the scene requires more time so the updates are not drawn as they occur and the simulation seems to freeze or run in slow motion.

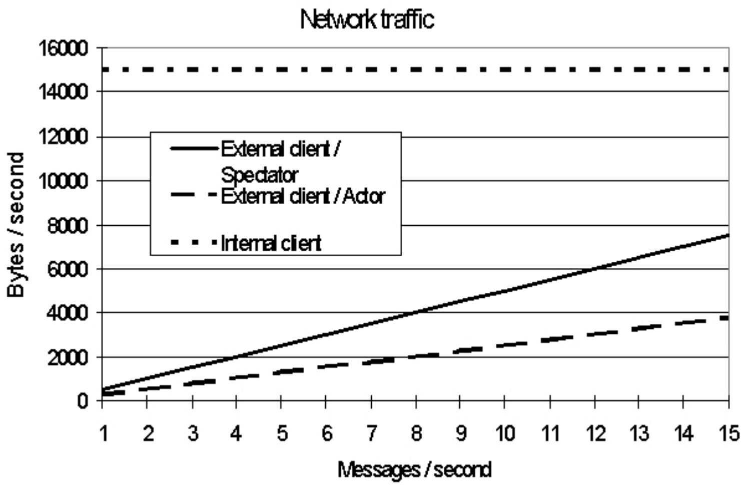


Fig. 4. Network traffic generated

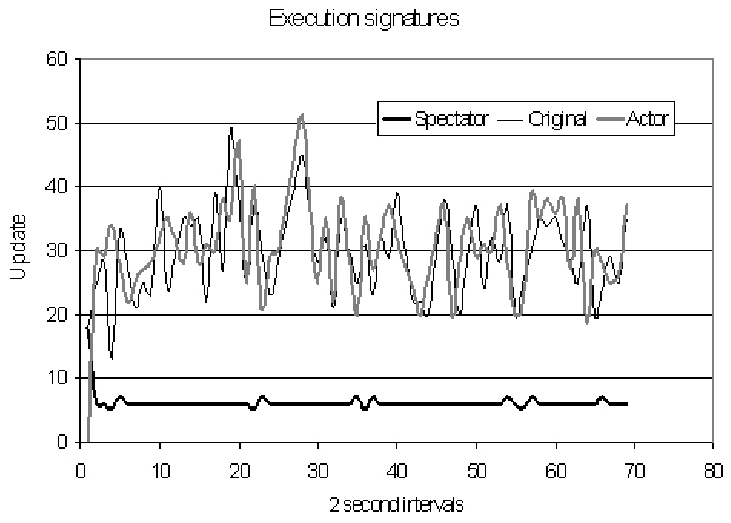


Fig. 5. Scene rendering accuracy

Considering that the mathematical computations run on different machines than the 3D rendering process then this problem manifests itself in a more obvious way. To make things worse, one of the solutions proposed groups messages together, for reducing the network traffic. To reduce this very effect, timestamps were added to messages so that the time intervals at which they arrive on the web service could be replicated on the client's side at rendering time.

In order to better see the accuracy of the solutions proposed, the execution of an Active Objects Model was "*recorded*" at the web service layer and then compared with the recorded execution in the client layer. The experiment was divided into time frames of 2 seconds in which the number of update messages was recorded. By putting together these numbers obtained at consecutive intervals, we get a sort of a *signature* of the execution. Fig.5 compares the signature obtained at the web service level with the two signatures obtained at the client side for an actor, respectively a spectator.

One can easily see that the spectator does a poor job of accurately simulating the execution of the model, and this was to be expected, as the images received from the web service represent only snapshots of what the scene looked like at the time of the request, no matter how many updates were performed on the scene.

One the other hand, having the timestamps of arrival at the web service, helps the actor do a pretty good reproduction of the model's execution. One could argue that by taking as reference the time of arrival of a message on the web service does not provide the real signature of the execution, as it does not take into consideration the delays inside the Grid. But this is a compromise we just have to make, as there is no way of assuring that the time on the machines that make up the Grid network and on which the jobs are executed is synchronized.

5. Conclusions

When comparing the two proposed solutions for dealing with external users to the AOM Grid application, the actor and the spectator, the first one is clearly the winner. It is the one that provides a more realistic representation of the active objects model's execution. It uses less bandwidth, and is much more user friendly as it allows the user to freely explore the scene in real time.

The only recommended case of using the spectator approach would be the case in which we want to enable the application to be used from a simple web page, for a quick view, or maybe for demonstration purposes.

Acknowledgments. The research described in this paper was supported by the Romanian Education and Research Ministry, under Contract 19CEEX-I03 through the MedioGrid project.

References

[1]. Active Object Model, http://users.utcluj.ro/~gorgan/res/aom/index.html
[2]. Active Object Model definition, http://c2.com/cgi/wiki?ActiveObjectModel
[3]. Ade J. Fewings and Nigel W. John, "Distributed Graphics Pipelines on the Grid," IEEE Distributed Systems Online, vol. 8, no. 1, 2007, art. no. 0701-1001.

http://dsonline.computer.org

[4]. Rares Barbantan, Dorian Gorgan, Active Objects Based Application over Grid Environment, GridCAD2006 Workshop at SYNASC06, IEEE Computer Press (2006), pp. 289-295.

[5]. Marc H. Brown, Marc A. Najork, Distributed Active Objects. Computer Networks and ISDN Systems, Volume 28, issues 7–11, pp. 1037.

[6]. Ken Brodlie, David Duce, Julian Gallop, Musbah Sagar, Jeremy Walton, Jason Wood, "Visualization in Grid Computing Environments" 15th IEEE Visualization 2004 (VIS'04), 2004, pp. 155-162.

[7]. Dorian Gorgan, Programming Control Structures in Active Objects Model, CONTI2004, Timisoara, 2004 and Transactions on Automatic Control and Computer Science, Vol.49 (63), 2004 No 3, ISSN 1224-600X, pp. 119-122.

[8]. Dorian Gorgan, Vasile Cornea, Interactivity in Active Objects Model. Proceedings of the IEEE-INES2004 Conference, 19-21 Sept. 2004, Cluj-Napoca, (ISBN 973-662-120-0), pp. 551-556.

[9]. Dorian Gorgan, David A. Duce, Multimedia Synchronization Through Interactive Active Objects. Proceedings of the EUROGRAPHICS'97 UK Conference, pp. 131-155, Norwich, UK, March 1997, and Scientific Commons, http://en.scientificcommons.org/dorian_gorgan.

[10]. Dorian Gorgan, David A. Duce, The Notion of Trajectory in Graphical User Interfaces. Research Report, Rutherford Appleton Laboratory, January 1997. (The first version of the paper presented at the DSV-IS workshop), and Scientific Commons, http://en.scientificcommons.org/dorian_gorgan.

[11]. Fabrice Huet, Distributed Objects and Components for the Grid: The open source platform Objectweb ProActive, http://www.cs.vu.nl/~kielmann/asci-a14/slides/proactive/1-ProActive.pdf

[12]. Mihaela Ordean, Dorian Gorgan, Distributed Active Object Model. Proceedings of the IEEE-INES2004 Conference, 19-21 Sept. 2004, Cluj-Napoca, (ISBN 973-662-120-0), pp. 557-560.

[13]. ProActive, A Comprehensive Solution for Grid Computing, http://www-sop.inria.fr/oasis/ProActive/release-doc/html/index.html

[14]. Project Oasis, http://ralyx.inria.fr/2006/Raweb/oasis/uid4.html

[15]. Sung Park, Lars Linsen, Oliver Kreylos, John D. Owens, Bernd Hamann. A Framework for Real-Time Volume Visualization of Streaming Scattered Data. Proceedings of Tenth International Fall Workshop on Vision, Modeling, and Visualization 2005, pp. 225-232.

[16]. Joseph W. Yoder's Adaptive Object-Model Pages, http://www.adaptiveobjectmodel.com/

Parallelization of Some Spatial Epidemic Models

Turnea Marius, Dragoş Arotăriţei, Radu Ciorap,
Ilea Mihai

University of Medicine and Pharmacy "Gr.T. Popa"
Iaşi
Faculty of Biomedical Engineering
dragos_aro@yahoo.com

Abstract. *The compartmental models using differential equations are basic models in epidemiology. The temporal evolution of spatial models for epidemic spreading is suitable for parallelization and GRID services are solutions for speeding the algorithms used in these models. We investigate several computational aspects of parallel algorithms used in cellular automata model and small world networks model. The four compartmental small world network model of disease propagation (SEIR) is parallelized.*

Keywords*: compartmental models, small world networks model, parallel algorithms, GRID*

1. Introduction

The distributed computing is an efficient solution for applications that require high computational effort, information retrieval from geographically distributed resources or both. Such applications use interconnected networks of computers or supercomputers, very large databases, software instruments for storage and retrieval, advanced devices and scientific instruments [1, 2].

GRID computing solution is used for implementation of few epidemiological models, but the applications are mainely focused on sharing very large data bases. Statistic models used in these applications are based on serial approach with no parallelization. Few applications used the GRID computational advantages [3]-[6]. eMicrob [3] built a GRID platform (eMicrob miniGrid) to provides secure access to heterogeneous data and expensive resources in different locations. A system for distributed cohort characterization is proposed in [4]. The system is applied to study the first episode psychosis [4]. GISE is a flexible service built on Globus 4 grid infrastructure and it has been tested in an epidemic monitoring and surveillance system [5].

To our knowledge, the GRID applications related to epidemiological models usually refer to data management across the GRID (breast cancer, mammography, etc.) and only few applications refer to code paralellization.

A parallel algorithm that is implemented on the GRID must be efficient that is the parallel version must be an improvement of serial version. An efficient parallelization depends on the problem that must be solved and the serial version of the algorithm that solves the problem. In what follow we describe the problem and the serial algorithms in order to identify an efficient parallelization of the serial algorithms used in epidimiological models.

Epidemiology is one of the standard methods used for evaluation of status of population health [6-9]. The epidemiologists developed mathematical models of epidemics. These models allow the prediction of how disease will occur. A common mathematical model for epidemiology uses differential equations [8]. An important part of these models use multi-stages (compartmental) approaches. These compartmental approaches that use ordinary differential equations (ODE) are suitable to be implemented on computer systems in order to simulate the temporal and spatial evolution of phenomena [8].

The mathematical models that use ODE can be solved sequentially by iterative methods, numerical methods, or parallelization of the solver algorithm that is based on Euler or Runge-Kutta methods. The efficiency of these algorithms depends of how much overhead is given by communication among processors and the load balancing of the tasks.

The maladies spectrum for a finite population can be sporadic [10], endemic (regular, with continuous apparition), epidemic (continuous increase of number of affected persons) or pandemic (many countries are affected). Many of these models are affected by seasonal variations (e.g. influenza that is more frequent in the winter) or had seasonal variation that are known as cycle of burst after a number of years. These templates are identified by seasonal patterns [11]. The most common model of seasonal variation is the periodic function based that use sinus or cosinus formulas.

The interest in mathematical models for epidemiology has grown exponentially in the last years. Some models involve aspects such as passive immunity, gradual loss of vaccine and disease-acquired immunity, stages of infection, vertical transmission, disease vectors, macroparasitic loads, age structure, social and sexual mixing groups, spatial spread, vaccination, quarantine, and chemotherapy [10]. Special models have been proposed for diseases such as measles, rubella, chickenpox, whooping cough, diphtheria, smallpox, malaria, onchocerciasis, filariasis, rabies, gonorrhea, herpes, syphilis, avian flu, and HIV/AIDS. The most common models are presented as a set of ordinary differential equation (ODE) or partially differential equations (PDE). During the past few years we noted an increased interest related to paralellization of ODE that are used to describe epidimiological models [11]-[16].

In the context of disease transmission, some of the studies focused on several forms of computer-generated networks that are defined in terms of how individuals are distributed in space (which may be geographical or social) and how connections are formed [17]-[21]. This complex process simulates the spatial spread of disease that happens within real populations. We can mention the epidemiological models that fall in this case: random networks, small-world networks (SWN), spatial networks, scale-free networks, exponential random graph models, lattices, and cellular automata (CA) [21].

Cellular automata (CA) are characterized by their discretization of space and time. The epidemiological model using cellular automata is a model that focuses on spatial spreading of a disease. Cellular automata consist of spatial grid with cells that are characterized by discrete time and state. At each discrete time, we perform an iteration in which cells are updated using certain rules. Corridors of spread in cellular automata might be considered to improve model with real situation when infected individuals may move toward other locations (via train, bus or car) and construct a new infection node.

The term small-world network refers to networks where over a regular lattice small number of shortcuts are introduced. A small-world structure is similar to situation when clusters of connected individuals (social groups) have contact with “nearby” groups and “far-off” groups via the sparse long-range links.

Lattices display high clustering but long path lengths take many steps to move an infected individual between two cells that are randomly selected. Small-world networks offer a means of moving between the rigid arrangement of lattices and the unstructured connections of network models. The high level of clustering means that most infection occurs locally, but short paths’ lengths mean that epidemic spread through the network is rapid and the disease is unlikely to be contained within small regions of the population.

2. The basic SEIR Model

The procedure to find out an adequate model that fit to one specific epidemic is a difficult operation. The mathematic model is a tradeoff between simplicity, accuracy and generality. A model should approximate what happens in the real world. A complex model might have a greater accuracy but it could be too difficult to be parameterized and understood. The most common models are compartmental models [8, 10].

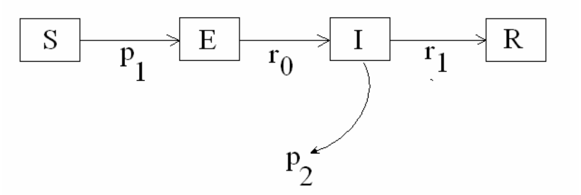


Fig. 1. The four compartmental SWN model of disease propagation

The four compartmental small world network model of disease propagation has four categories of populations: S - Susceptible (the fraction of susceptible individuals, those individuals able to contact the disease), E – Exposed (the fraction of exposed individuals, those individuals that have been infected but are not yet infectious), I – Infective (the fraction of individuals that are able to transmit the disease), and *R* – Recovered (the fraction of individuals who became immune). The compartmental model and transitions are showed in Fig. 1. Suppose the birth and death rate μ is constant. The equations of basic SEIR model are [20]:

$$dS/dt = \mu - \beta(t)SI - \mu S \quad (1)$$

$$dE/dt = \beta(t)SI - (\mu + \alpha)E \quad (2)$$

$$dI/dt = \alpha E - (\mu + \gamma)I \quad (3)$$

$$S + E + I + R = N \quad (4)$$

In the equations above $1/\alpha$ is the mean latent period for disease and $1/\gamma$ is the mean infection period [20]. The parameter $\beta(t)$ represents the force of infection (infection rate) and can be constant $\beta = \beta_0$ = constant or can be seasonal:

$$\beta(t) = \beta_0(1 + \beta_1 \cos(2\pi t)) \quad (5)$$

The paths of transmission (state transition graph) are depicted in Fig. 2. The infected individuals can create susceptible individuals to whom are linked with some probability. The immediate neighbors will become infected with probability p_1 meanwhile the long range links will become infected with probability p_2.

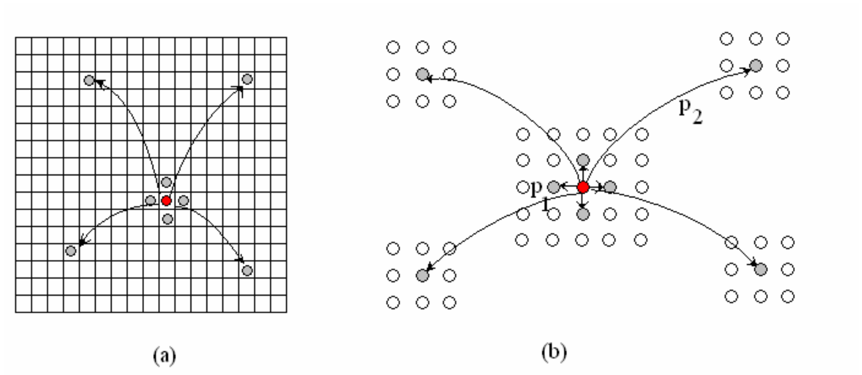


Fig. 2. The short-range and the long-range network links (a) grid locations (b) the local arrangement of nodes in small network

Exposed individuals will become infected with probability r_0 and finally, infected individuals will become immune (recovered) with probability r_1. The distribution can be truncated power-law form (6) or discrete exponentially decaying distribution (7). Our experiments use the form presented in (7).

$$f_X(x) = \frac{1}{C} e^{-x/\mu}, \quad C = \frac{1}{1 - e^{-1/\mu}} \tag{6}$$

$$p(n_2^i = e^k) = \frac{1}{k} e^{-\frac{k}{\mu}} \tag{7}$$

The corridors are considered only in four directions (*North*, *South*, *East* and *West*). These corridors allow the spread of disease without the links cell-by-cell. The extension with the other four intermediary points creates additional difficulties and these extensions will be considered for further research.

For each simulation we seed the model with one initial infection. We denote some variables inspired by [18]. For a population of N individuals we assume that there are no other births or deaths during simulation that is the population *N* is constant. The population is disposed on a grid with *L*×*H* rectangular area, so in this case *L*=*H*, *N*= L^2.

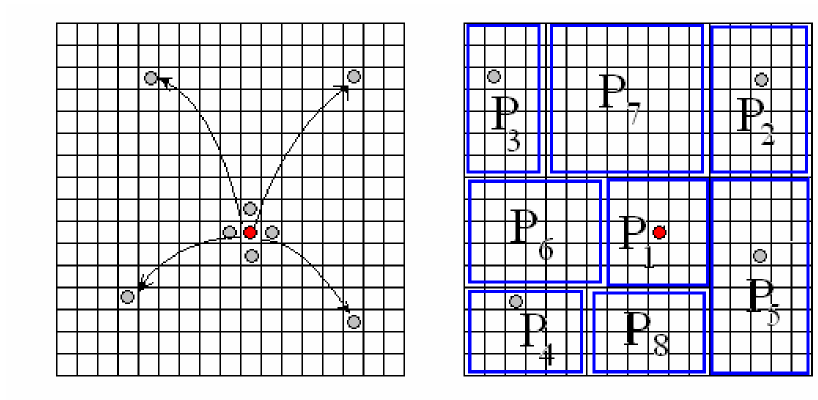


Fig. 3. The allocation of processors

Let *np* be the number of available processors. In the case of rectangular grid, the partition of the map among *np* processors is simple. Each processor has a rectangle of dimension *M= (int) N/np* for the first *np-1* processors, while processor *np* takes the remainder of the rectangle. An arbitrary partition based on heuristics can be taken also into account. The partition could have a different number of squares. The urban population has an increased probability to raise the number of infected people if a single infected individual is present in the city area. The heuristic of partition is based on trial to allocate the equal distribution of population of each processor and to allocate an entire city to one processor without splitting the location among processors (Fig. 3).

The algorithm is shortly presented below.

```
Partition of N cells in M rectangles (M << N)
Start the seed of disease in point S(i_s, j_s)
Allocate the processor P_1 to rectangle R_1 where S(i_s, j_s) ∈ R_1
Allocate the rectangles $M_i$ to processor $P_i$ for $1<i<np+1$
 Repeat until no area to be allocated
      Allocate the rectangles $M_k$ to processor $P_i$ for $1<i<np+1$, $np<k<M$
Allocate the boundaries of each area to corresponding processor
Start SEIR model

For step=1 to step = max_steps
      SEIR model and find the new E and I points (short range and long range)
      Identify the area and processor allocated to new E and I points
      Verify the boundaries and correct the processor allocation
End for

Collect all the data to Processor $P_1$
```

The heuristic of partitioning is based on trying to allocate an equal distribution of population for each processor and to allocate an entire city to one processor without splitting the location among processors (Fig. 3).

3. Experimental Results

The model has the parameters set to L= 1500, $N=L^2 = 2250000$, $n_1 = 4$, r_0 follow a geometric distribution $f_X(x) = (1-p)^{x-1}p$, $p_1 = 1/n1(0.27-\mu p_2)$, $r_0 = 0.135$, $r_1 = 0.25$, $\mu = 7$ values partially inspired from [13] with SEIR model for SARS. The parameter p_2 had been given values between [0, 0.07].

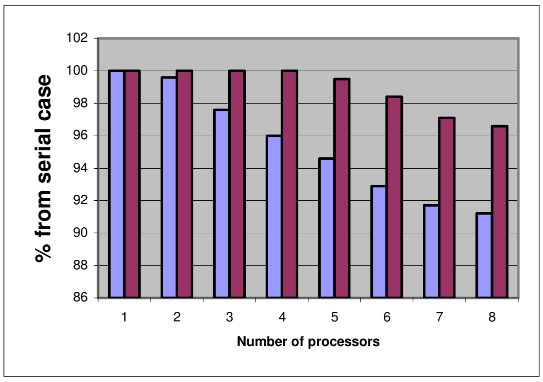


Fig. 4. The result of simulation (the best case and the unfavorable case)

We tested the simulation of the disease's spread in parallel implementation for 2-8 processors. The tests have been made under several restrictions: (1) the simulation is made for *T* days meanwhile the spread of disease doesn't cross the border of rectangles allocated to processors (2) the scalability of the algorithm is tested only for maximum 8 processors (3) we compared the parallel algorithm with the serial version that is run on a single processo.

The evolution after 122 days is presented (in percent time of the serial algorithm) as follows: 2 processors – 99.6%, 3 processors – 97.6%, 4 processors - 96%, 5 processors 94.6 %, 6 processors – 92.9% 7 processors – 91.7%, 8 processors – 91.2%. We must note that the performance of the parallel algorithms increases with the evolution of the disease but it's limited by the number of processors. Each new epidemic seed gets a processor allocation and this processor will compute all the operation from cells the neighborhood of the seed cell.

Modeling the corridors proves to be difficult. The main problem is that long range seeds for new cluster of infection, despite the corridor improvement have difficulties. The corridors are difficult to be modeled according to principle of cellular automaton and the realistic case. For these reasons, only preliminary results are reported in this paper regarding cellular automata model. The experiments use a city location, a medium size city (450000 inhabitants) and only vertical corridors have been taken into account. The results after 257 days are not very satisfactory so we have not continued in this direction.

4. Conclusions

For each simulation we seeded the model with only one initial infection. Two or more seeding points could be a realistic situation but this aspect will be extended in the further researches. The results of simulation are based on synthetic data. It is clear that in practice, some coefficients can have particular values that depend on estimation results in the first days of epidemics. We can operate with the limits of the parameters in the sense of pessimistic and optimistic case (lower and upper limits) but the intervals are usually very large. In some cases, some values could have as result the extinction of epidemic in very few days, which is an unrealistic case (or very less probable).

The communication among processors for a small number of processors could overcome the benefit of parallelization. The proposed algorithm can be useful for a large area (e.g. pandemic spread of diseases) and a fine granulation of partitions that must be allocated to processors. Also, the partition contour is very important to exploit the benefices of parallelism. Meanwhile, a laced optimal contour produces in increased of computational effort especially in the case of fine granulation: the processor must verify each cell on the border in order to avoid going in the partition allocated to other processor.

However, the model proved to have a good approximation for evolution of the epidemic disease meaning that by modifying the parameters including the probability of disease we can reach any cell from the lattice in a reasonable number of days and the model can cover a very large class of real epidemic spreads.

Acknowledgments. The Grant CEEX II03/31.07.2006 "Academic GRID for Complex Applications – GRAI" has supported the research for this paper.

References

[1]. Geoffrey Fox, and Anthony J.G. Hey (Eds), The Grid Computing: Making the Global Infrastructure a Reality. Wiley & Sons (2003)

[2]. Ian Foster I, et. al., The Physiology of the Grid: An Open Grid Service Architecture for Distributed System Integration. Global Grid Forum (2002)

[3]. Jianping Guo, et. al., eMicrob: A Grid-Based Spatial Epidemiology Application. Lecture Notes in Computer Science 3516 (2005) 472-475

[4]. John Ainsworth, Robert Harper, Ismael Juma and Iain Buchan, PsyGrid: Applying e-Science to Epidemiology, 19th IEEE International Symposium on Computer-Based Medical Systems CBMS 2006 (2006) 727-732

[5]. Eduardo Gallo et. al., GISE: A Data Access and Integration Service of Epidemiological Data for a Grid-Based Monitoring and Simulation System, 40th Annual Simulation Symposium ANSS (2007) 267-274

[6]. Roy M.Anderson, Robert M. May, and B. Anderson, Infectious Diseases of Humans. Dynamics and Control. Oxford University Press (1992)

[7]. D.J. Daley, J.Gani, Epidemic Modelling: An Introduction. Cambridge University Press (2001)

[8]. O. Diekmann, and J.A.P. Heesterbeek, Mathematical Epidemiology of Infectious Diseases: Model Building, Analysis and Interpretation. John Wiley & Sons (2000)

[9]. Johan Giesecke, Modern Infectious Disease Epidemiology. A Hodder Arnold Publication (2001)

[10]. Herbert W. Hethcote, The Mathematics of Infectious Diseases, SIAM REVIEW, 42-4 (2000) 599–653

[11]. Kevin Burrage, Parallel methods for systems of ordinary differential equations. Oxford University Press Inc., New York (1995)

[12]. Mantas J. Miguel, et. al., Parallelization of Implicit-Explicit Runge-Kutta Methods for Cluster of PCs, Lecture Notes in Computer Science 3648 (2005) 815-825

[13]. P.J. van der Howen, B.P. Sommeijer, Parallel ODE solvers, Proceedings of the 4th international conference on Supercomputing 18-3b (1990) 71-81

[14]. Dana Petcu, Mircea Dragan, Designing an ODE Solving Environment, LNCSE 10 (1999) 245-266

[15]. Matthias Korch and Thomas Rauber, Scalable Parallel RK Solvers for ODEs Derived by the Method of Lines, Euro-Par 2003. Parallel Processing. LNCS 2790 (2003) 830–839

[16]. Shuichi Ichikawa and Yoshikatsu Fujimura, Iterative Data Partitioning Scheme of Parallel PDE Solver for Heterogeneous Computing Cluster, Proc. Applied Informatics AI 2002 (2002) 364-369

[17]. Sangeeta Venkatachalam, and Armin R. Mikler, Towards Computational Epidemiology: Using Stochastic Cellular Automata in Modeling Spread of Diseases. Proceedings of the 4th Annual International Conference on Statistics, Mathematics and Related Fields, Honolulu, HI (2005) 1-16

[18]. Michael Small, and C.K. Tse, Clustering model for transmission of the SARS virus: application to epidemic control and risk assessment. Physica A: Statistical Mechanics and its Applications 351 (2005) 499-511

[19]. Matt J. Keeling, and Ken T. D. Eames, Networks and epidemic models. J. R. Soc. Interface 2 (2005) 295–307

[20]. J.L. Aron and I.B. Schwartz, Seasonality and period-doubling bifurcations in an epidemic model, Journal of Theoretical Biology, 110 (1984) 665-679

[21]. Padmavathi Patlolla, Vandana Gunupudi, Armin R. Mikler and Roy T. Jacob, Agent-Based Simulation Tools in Computational Epidemiology. Workshop Computational Epidemiology, Lecture Notes in Computer Science, 3473 (2006) 212-223

Issues Related to Distributed Implementations of Models for Large Economic Systems

Marius Zbancioc [1,2], Horia-Nicolai Teodorescu [1,2] ,

Laura Pistol [2]

[1] *Technical University "Gheorghe Asachi" of Iaşi*
[2] *Institute for Theoretical Informatics of the Romanian Academy*

zmarius@etc.tuiasi.ro, hteodor@etc.tuiasi.ro,
laura.pistol@iit.tuiasi.ro,

Abstract. *We analyze several implementation issues occurring in the modeling of a class of neuro-fuzzy models for large fuzzy economic systems. Because the computations for fuzzy logic inference are time consuming, because the decision making process for establishing the selling prices is iterative, moreover the network of companies in the market interact in an intricate way, the distribution of the computation may make sense. However, task partitioning is not trivial. Various implementations are exemplified and the experimental results of the implementations are contrasted.*

Keywords*: fuzzy systems, market model, fuzzy decision, dynamic behavior, distributed algorithm, modeling results, GRID environment.*

1. Introduction

The literature on economic modeling under uncertainty is too vast to be reviewed comprehensively; examples of papers dealing with statistical methods and complex feedback loops in the decisional process are [1-3]; belief degrees appear in [4-6]; fuzzy logic-based models and models involving nonlinear dynamics are treated in [7-13], etc. In several papers published between 1990 and 1995, it has been demonstrated that a fuzzy feedback system can become, in certain cases, a chaotic system, if the control strategy is not chosen sufficiently stable [8-13]. Recently, the dynamics of several fuzzy economic models involving decision loops has been and studied in [15-18].

The basic economic model is discussed in the papers [8-13] and [15]; it includes two management strategies classes in order to take the decision of the selling prices. The first strategies class, named "*max-benefit*", seeks only the profit maximization; the second strategies class, named "*comp-benefit*", aims to obtain similar profits as the concurrent firms, and are envy-based strategies. Price modification can be done with a fixed or fuzzy increment. Fuzzy price modifications were found to induce rapid marketplace stabilization. Conclusions regarding the way the system dynamics is affected by the strategy type and the increment type are presented in previous papers [18-21]. Because of algorithms' complexity and the high running times for large models, the necessity arose to develop a parallel/distributed processing algorithm.

Here, we discuss only a few computational issues related to the models described in [15-21]. We compare serial and parallel implementations under the rules based language FuzzyCLIPS and under the procedural languages (C++, Visual C), for Windows/DOS and for Linux operating systems.

2. Distributed Algorithm for Fuzzy Economic Models

The following parallel computing algorithm has been announced in a brief form in [20]. For a complete description of the fuzzy membership functions and of the fuzzy inference rules set, or other details concerning the management strategies applied to the prices modification, see the previous papers [15-21].

Essentially, there are two types of networks under consideration; the first has a graph equivalent to K_N, with all companies interconnected; the second has the graph $K_{N/q,N/q}$, where q corresponds to the number of companies in a group, when group organization is modeled. In either case, the algorithm has basically the complexity N^2, because each node or each group has to communicate with at least N/q nodes or groups of nodes, while the computations inside a node is proportional to *N*. Thus, at every time moment, there are $O(N^2)$ message transfers and $N \cdot O(N\ ) = O(N^2)$ computations in the network. Importantly, the constants multiplying N^2 are very large, because of the amount of computations required by the fuzzy inference (fuzzy systems). For small *N*, these constants dominate N^2.

In the implementations we performed up to now, the parallelization is trivial: each company or group of companies is assigned to a processor, while the central node (server) performs all communications. This parallelization is acceptable for small networks, but when the number of companies increases, passing all messages through the central node becomes a bottleneck. In fact, while computations are parallelized, the communications are still serial in this approach. For large *N*, the distribution of communication is required to increase the speed. Communication distribution requires that N/c processors (where *c* is a constant) are assigned as local communications nodes. These nodes should take care of communicating prices and benefits between *c* companies. Notice that in either one of these parallelization methods, the communicating nodes still have to perform $O(N^2)$ communications, and the strategy computing processors still perform $O(N^2)$ computations. Subsequently, the constants in the complexity expression are evaluated and detailed on the estimation of the complexity are provided.

Notice that the strategies used are essentially "one-to-one" adaptation; if the strategy would be "adapt each to the average market", the complexity decreases to $O(N)$.

Herein, the notations used are: SF1 - fuzzy systems used to estimate the firm benefits; SF2 - the fuzzy system for price increment computing, BC_k - the neuro-fuzzy networks of benefit processing block; b denotes the benefit, b_{med} is the average benefit; t stands for time etc.

We briefly review the parallel algorithms used.

1) *Initialization phase (in the central process only)*

The central process distributes, respectively receives information from other processes. For every firm from the parallelized economic model, the following are set:

- the lists (circular queues) with the initial prices,
- the descriptive fuzzy membership function for the „*price*" and „*benefit*",
- a line #*k* from the delayed matrix will be used by the computing nodes in the estimation of firm #k benefit,
- the type of strategy used to establish the new selling prices,
- the type of increment (require a SF2 component) and
- the fuzzy rules set (if these rules are not the same for all the firms in the model).

The following items are also defined: the stop condition for the system (maximal number of steps *P*, the reaching of stabilization requirement, the exceeding of the maximum or the minimum value accepted which correspond to an abnormal state, the repeating of a scenario by a pre-established number of times – at the entering in a loop)

2) *while step* ($p \geq 1$) *or (the stop conditions are not activated) do*

3) *Parallelization phase*

for k = 1 *to N*, each of the *N* firms is associated to a computing node in the GRID.

3a) According to the input parameters (strategy, increment, price vector etc.), the average benefit of firm #k at the moment of time t is computed, moreover, possibly the average benefit of the competitors, using 'delayed' prices: $b_{med\,k}[t]$ and $b_{med\ delayed\ k}[t]$. According to the strategy type, they might be estimated also the obtained benefits with an increased selling price ${}^{+}b_{med\,k}[t]$ or a decreased price ${}^{-}b_{med\,k}[t]$ (see fig. 2, also [21]).

3b) Each block BCk includes $N-1$ fuzzy systems SF1

```
for i = 1 to N-1
    compute the benefits $b_{k,i}[t]$, $b_{i,k\ delayed}[t]$, according to the prices $p_k[t]$, $p_i[t-\tau_{k,i}]$,
    using the fuzzy inference rules and defuzzification on the fuzzy system output.
endfor
```

3c) *Modify the selling price using the "decision block"*, using the three rules described in [18-21]. If the increment is fuzzy, we need a calling of the fuzzy system SF2 in order to compute the crisp value of the increment.

3d) Transmit the result $p_{k+1}[t]$ to the root process

4) *Data centralization* in the root process (at every iteration step - discrete moment of time)

5) *Decrementing the step* $p \leftarrow p-1$*, and return to phase* 2.

This parallelization of the economic model requires $N+1$ computing nodes - one for the central process and N for the process associated to every firm $\#k$, for $k=1\ldots N$. The computing time varies with each node and it is dependent on the type of strategy adopted by the firm and the type of increment.

3. Estimation of the Complexities of the Strategies

Because there are two strategies, each with two types of increment in the adaptation of the companies, we actually need to deal with four models, each with a different complexity. The complexity has been estimated as the number of calls of the modular fuzzy systems (SF1 – benefit calculus, SF2 – fuzzy increment) included in the distributed economic model. The time complexity is computed with respect to the number of companies in the network, N.

Constants taken into account in the complexity estimation are the number of rules defining the fuzzy decision process, R, the number of premises in the rules, N_P, the number of fuzzy attributes which describes the input fuzzy variables (*price*), N_A, the number of iterations (steps), P, (provided that number is pre-determined, not varying, for example varying according to the length of the transitory regime), and the number of samples for which the fuzzy membership functions are estimated, k, the same for all membership functions.

The complexity of the computations performed into a single node, *O*(SF1) and *O*(SF2), must be accounted for in the overall complexity computation.

TABLE 1. Estimation of the complexity of the algorithms for the four strategies [21]

Strategy type	Increment type	„best-case" complexity	„worst -case" complexity
„*max-benefit*"	fixed	3*N* ·*O*(SF1)	
„*max-benefit*"	fuzzy	4*N* ·*O*(SF1) + *O*(SF2)	
„*comp-benefit*"	fixed	2*N* ·*O*(SF1)	4*N* ·*O*(SF1)
„*comp-benefit*"	fuzzy	2*N* ·*O*(SF1)+*O*(SF2)	4N ·O(SF1)+O(SF2)

The complexity of computing the response of each fuzzy system is $O(SF1) = (R \cdot N_P + R + 2) \cdot k \approx R \cdot N_P \cdot k$, $R = N_A{}^{N_P}$. The fuzzy system for price increment computation uses a single-input ($N_P = 1$) single-output Mamdani system; therefore, the complexity of these systems is $O(SF2) = N_A \cdot k$. The complexity of the benefit block BC$_k$ computation is $N \cdot (4N \cdot O(SF1) + O(SF2))$. The overall complexity of the model is $P \cdot \left(4N^2 \cdot O(SF1) + N \cdot O(SF2)\right) \approx 4N^2 \cdot P \cdot R \cdot NP \cdot k$, where k, P, R, N_P, N_A are constants.

Notice in Table 1 that, for the firms with the strategy type based on *envy* „comp-benefit", the systems number vary depending on the firm situation on the marketplace:

- if the firm benefit exceeds the competitor's medium benefit, it will perform supplementary estimation on benefits, which will be obtained for a growth/decrease of the selling price
- if the firm benefit is lower than the competing firms, it will mimic the competitors behavior regarding the price; in that case, the complexity results about $2N \cdot O(SF1)$.

The complexity of the blocks BC$_k$ is given in Table 1. From the point of view of the computing time, the most expensive is the strategy „*max-benefit*" with a fuzzy increment – complexity, $4N \cdot O(SF1) + O(SF2)$.

4. The Distributed Algorithm Design

The order of the execution phases has been taken from the initial model developed in a dedicated AI language – FuzzyCLIPS, according to the rules priority levels. This organization can be found in the serial models made in C++, and in the distributed models designed for Linux.

The disadvantages of the implementation of the economic model developed in FuzzyCLIPS are:

- In case of using a large number of actors (firms) on the market the running time increases (almost exponentially - Fig. 3b). For the model with 10 firms, we determined a running time of about 15 minutes. Economic models with dozens of firms require too long running times to be practical.
- The rules that have been used have only two premises. A larger number of premises would lead to an exponential growth of the number of inference rules and to higher calculus time.
- The system does not allow the change of the fuzzy operators (allowing the use of the classic operators, Mamdani $\max(\mu_A, \mu_B)$ for reunion and $\min(\mu_A, \mu_B)$ for intersection), the change of fuzzy inference type or defuzzification operator

Parallel or distributed computation is needed to obtain acceptable running times whenever we deal with complex models with several sets of membership fuzzy functions, fuzzy rules with multiple premises, and involving large numbers of firms.

Solutions for the parallelization algorithms by assigning one agent computation per node and respectively by assigning one group of agents as one computation task have been proposed in [19]. Here, the parallelization of the algorithm is made at the level of the benefit computing block BC_k. Each computing node, excepting the central node, receives the task to perform a complete computation of the benefit for one company of the economic model, at every moment of time (computation cycle in the evaluation of the dynamics of the network). This parallelization solution has the advantage of using reduced communication times between the network GRID nodes in comparison to other solutions [19]. The communications are done only between the central node and working nodes. A functional unit BC_k requires between 2*N* and 4*N* calls of SF1 and a call of SF2 (see Table.1). The central node task is much simplified (data centralization, control of data transmission, verification of system stop condition).

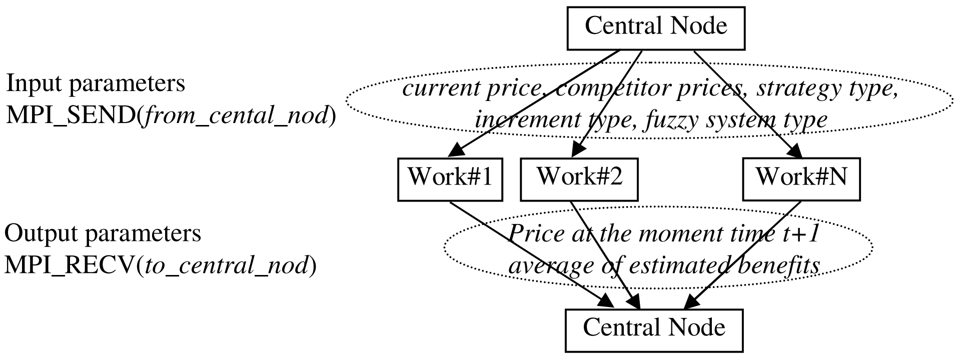

Central Node
Input parameters
MPI_SEND(*from_cental_nod*)
current price, competitor prices, strategy type, increment type, fuzzy system type
Work#1
Work#2
Work#N
Output parameters
MPI_RECV(*to_central_nod*)
Price at the moment time t+1
average of estimated benefits
Central Node

Fig. 2. Communication between central process and processing nodes (one to one)

The only obvious disadvantage which can be associated to the method is the lack of flexibility in reusing some output results of fuzzy systems constituent SF1, SF2. It is possible to compute several times the benefits for identical input parameters: $p_k[t]$, $p_i[t-\tau_{k,i}]$. The frequency of this kind of unfavorable situations is low, because every firm may have a different type of increment, various delays applied to the competitor's prices etc.

In the initialization phase, the central node transmits to the workers the set-up files "fuzzy.in#k", "fuzzy.inc" according to the received tasks. The dimensions of the set-up files are less than 1 kB, the sending is done once time at the system initialization, so the time for file sending/receiving will not affect at all the total running time of the distributed application.

Each "worker node" computes the average benefits $b_{med\,k}[t]$, $b_{med\,competitor\,k}[t]$, ${}^{-}b_{med\,k}[t]$, ${}^{+}b_{med\,k}[t]$ depending on the strategy and increment type. Based on the estimated benefits, the selling price decision is made. The flowchart of the price assignation is shown in Fig. 2.

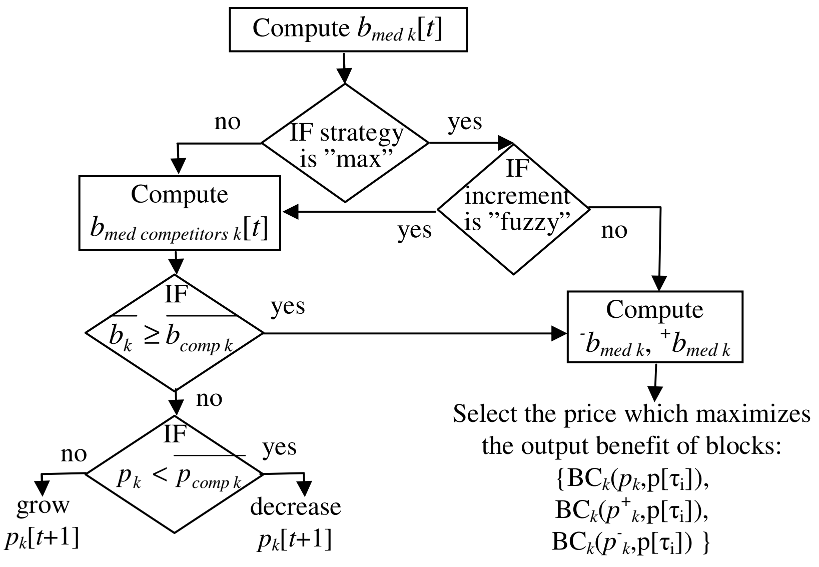

Compute $b_{med\,k}[t]$
IF strategy is "max"
no
yes
Compute $b_{med\ competitors\ k}[t]$
IF increment is "fuzzy"
yes
no
IF $\overline{b_k} \geq \overline{b_{comp\,k}}$
yes
Compute ${}^{-}b_{med\,k}$, ${}^{+}b_{med\,k}$
no
IF $p_k < \overline{p_{comp\,k}}$
no
yes
grow $p_k[t+1]$
decrease $p_k[t+1]$
Select the price which maximizes the output benefit of blocks: $\{BC_k(p_k, p[\tau_i]), BC_k(p^{+}_k, p[\tau_i]), BC_k(p^{-}_k, p[\tau_i])\}$

Fig. 3. Flowchart of the price decision module

5. Simulation and Results

The application for the simulation based on the economic model was run on several computers, for several firm numbers, N=2, 5, 10, 20, 30, 40, 44, 100, with all the firms having the same type of strategy; namely, the „*comp-benefit*" strategy with either fixed or fuzzy price increment, respectively „*max-benefit*" with the two increment types.

The hardware configuration of computers used in serial simulation were AMD Sempron(tm) 2200+, 256 MB RAM; Intel Celeron, 2.80 GHz, 512 MB RAM, Intel Pentium (R) D, 3.20 GHz, 1GB RAM.

For running the economic model application, the required setup files are:

- „fuzzy.inc" describes the rules, the variables and the membership functions used by the fuzzy increment system SF2;
- „fuzzy.in1", „fuzzy.in2",..., „fuzzy.inN", which contain various fuzzy description of the „price" and „benefit" concepts, and the fuzzy inference rules. We made the assumption that not all the firms from the economic model must use the same rules to establish the selling price, or the same descriptions for price and benefit variables;
- „firm.in" this set-up file contains the input parameters for each actor on the marketplace (input price, strategy type, increment type, description types of the fuzzy sets, the delays in finding the concurrent prices and the graph connections with other competitors).

The management activity of the firm is described by the delays vector τ_{ij}. Usually, the firm with small delays has an envy based strategy. The delays can describe the distance between the firms and the influence area of each firm.

For the group strategies, the following are given the ‘parent’ firms that coordinate the ‘child’ firms’ activity, possibly, the impact the central firm can have into the group firms (price decision may be affected by the located firm area, by the neighbors’ competition, by the client’s purchasing power in that area etc.)

The last file “firm.in” can be automatically generated, the executable (script) asks for the number of firms on the mini-market N, the percent of management strategy type and percent of the increment type.

1. FuzzyCLIPS application. The execution control program is made by IE (inference engine), which realizes the connections between rule base RB and the fact base. To have a global overview of the algorithm’s complexity were introduced the number of rules activated by the inference engine.

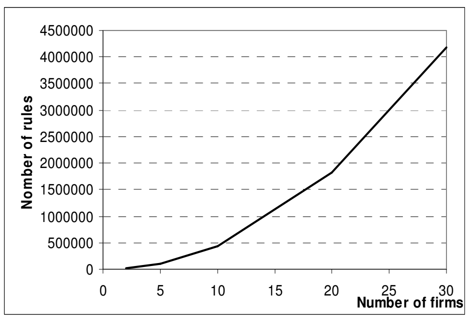
TABLE 2. Medium number of activated rules in FuzzyCLIPS application

Firm numbers / **Strategy**	**N=2**	**N=5**	**N=10**	**N=20**	**N=30**
„max-fix”	10837	90298	391275	1625345	3610629
„max-fuzzy”	15605	123008	525995	2170747	4937975
„comp-fix”	12023	97268	426165	1787739	4169889
„comp-fuzzy”	13443	101148	418371	1739117	3988945
Average	**12977**	**102930**	**440451**	**1830737**	**4176859**

TABLE 3. Average running times time (in seconds) of the FuzzyCLIPS application depending on the strategy type of the firms from the simulated economic model

Firm numbers / **Strategy**	**N=2**	**N=5**	**N=10**	**N=20**	**N=30**
„max-fix”	4.60	42.29	244.9	1885.5	7492.4
„max-fuzzy”	6.01	55.83	318.7	2534.3	9838.8
„comp-fix”	4.71	42.02	257.1	2154.2	8530.5
„comp-fuzzy”	4.97	44.44	252.6	2046.5	8615.2
Average	**5,1**	**46,2**	**268,4**	**2155,2**	**8619,3**

A growth of rules number leads to an exponential increase of running time (see Fig. 3b). The FuzzyCLIPS language uses specific data matching mechanisms – comparison of existent facts over the rules’ patterns (premises). The ReTe algorithm of the inference engine is a big time consumer, this fact being justified by the medium running times obtained for the simulated economic model to a firm number growth. Thus, for N=30 firm numbers, medium running times are about 2 hours (113 minutes) comparatively with running times of a few seconds (N=5), less than 5 minutes for an N=10 firm numbers. One simulation made for N=40 firm numbers took almost 7 hours (398 minutes).



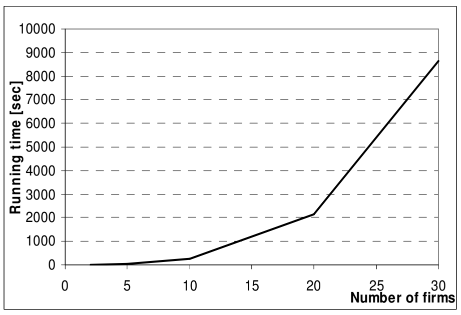


Fig. 4. Study for application complexity for FuzzyCLIPS implementation
a) Reported with rule numbers activated by IE b) the application medium running times

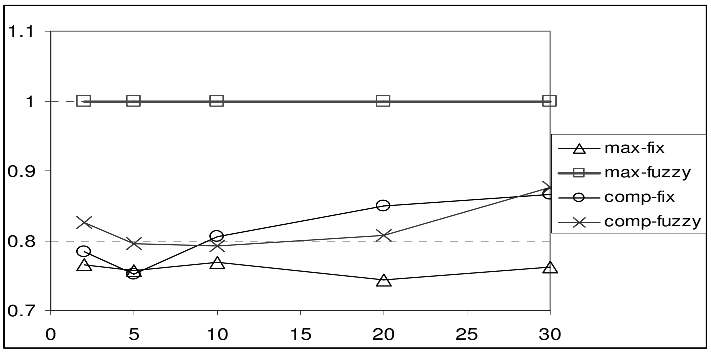


Fig. 5. Running time study for FuzzyCLIPS applications for the four strategies (normalized values of Table 3)

The running times shows that the maximization strategy „*max-fix*" with fix increment always obtains the smaller running times, and „*max-fuzzy*" the higher running times (see Fig. 4). This fact is not available for C++ runs, because of the designing conceptual differences of the algorithms on a rule-based language and a procedural language. Realizing a normalization for Table 3 values and graphically representing the obtained data it can be observed that the lowest running time („max-fix" strategy) is $\approx 75\%$ from the highest running times (for „max-fuzzy" strategy) – fact confirmed by the algorithm's complexity estimation (in Table 1).

For "envy"-based strategies, „comp-fix" and „comp-fuzzy" can not be clearly differentiated because in these economic models the firms tend to compare their benefits with the competitors and to take possession of their reactions. Because the set-up files for N=5, N=10 ... firms differ through associated values from the start prices and the delays in discovering the concurrent prices (stochastic selected values) and the prices and benefits evolutions will be distinct. The system's dynamics is influenced by the membership functions that describe the fuzzy variables „*price*" and „*benefit*", by the fuzzy inference rules, and by the delays matrix which reflects the management activity of the firms.

2. C (serial application). In this case, the algorithm is run on a single serial computer. The running times are significantly lower than for the FuzzyCLIPS application; while the running time is about 6-7 hours for an economic model with 40 firms under CLIPS, it is about 25 seconds under C. For the C version, we have implemented several handling procedure sets (libraries) of the fuzzy sets manipulation, a module for selecting the activated rules, and appropriate functions for the computation of the benefit under the allowed strategies. The application has been run on two operating systems (Windows and Linux). The running times are given in Tables 4 and 5.

TABLE 4. Average running time (in seconds) of the C variant application – serial variant under Windows

Firms number / **Strategy**	**N=5**	**N=10**	**N=100**
„max-fix”	0.788	3.488	505.1
„max-fuzzy”	1.078	4.865	742.6
„comp-fix”	0.816	3.823	521.5
„comp-fuzzy”	0.862	3.710	605.3
Average	**0,886**	**3,972**	**593,7**

TABLE 5. Average running time (in seconds) of the C application variant – serial variant under Linux

Firms number / **Strategy**	**N=2**	**N=5**	**N=10**	**N=20**	**N=30**	**N=40**	**N=100**
„max-fix”	0.04	0.38	1.68	5.76	12.33	22.5	135.5
„max-fuzzy”	0.057	0.52	2.19	7.80	16.59	29.3	182.1
„comp-fix”	0.04	0.38	1.77	5.95	12.585	22.8	141.1
„comp-fuzzy”	0.042	0.40	1.74	6.07	13.482	23.9	148.7
Average	**0.044**	**0.421**	**1.85**	**6.40**	**13.75**	**24.68**	**151.8**

The simulations performed using C under Linux compiler have 4-5 times lesser running time than the simulations that used the C under Windows compiler (see Tables 4 and 5).

After performing the simulations we obtained running times of ≈110 times less than FuzzyCLIPS application for 5 firms, respectively of **≈500** times less for N=30 firms, this ratio continues to grow proportional with *N*. The differences are justified by the inference engine mechanism (EI) from FuzzyCLIPS, which must set a hierarchy (execution order) of rules using various strategies (searching methods in the solutions spaces in depth, in width etc.). The agenda actualization – the list of the active rules (which have satisfied the premises) is made each time when new fact is introduced or is deleted from BF (Data-Base of Facts). The facts are assigned with a unique identifier on all running time programs, and it does not lose more time with the resets of the facts addresses.

TABLE 6. The ration between the medium running times of the FuzzyCLIPS application and the C application – serial variant under Linux

Firms number Strategy	N=2	N=5	N=10	N=20	N=30
„max-fix”	115	111	145	327	608
„max-fuzzy”	106	107	145	325	593
„comp-fix”	118	111	145	362	678
„comp-fuzzy”	118	111	144	337	639
Average	**114**	**110**	**145**	**337**	**629**

For *N* firms there are *N*x*N*-1 connections, complexity $O(N^2)$ - for a complete description of the economic model.. The estimation benefits process is iteratively made for all the simulations with *step*=200 iterative loops. We computed the necessary time to estimate the benefit of a firm #*i* reported to one single competitor #*j* (the run of one single fuzzy system). We suppose a total number of *steps* x *N* x *N*-1 fuzzy systems.

TABLE 7. Computing average time for the benefit (one single fuzzy system run)

Firms no.	**Fuzzy Systems No.**	**FuzzyCLIPS medium time**	**Time per Fuzzy System**	**Medium time C-Linux**	**Time per Fuzzy System**
2	400	5.07875	0.012697	0.04475	0.000111
5	4000	46.15036	0.011538	0.4215	0.000105
10	18000	268.377	0.01491	1.84925	0.000102
20	76000	2155.152	0.028357	6.401	0.000084
30	174000	8619.269	0.049536	13.746	0.000079
40	312000	0.076498	0.076498	24.685	0.000077

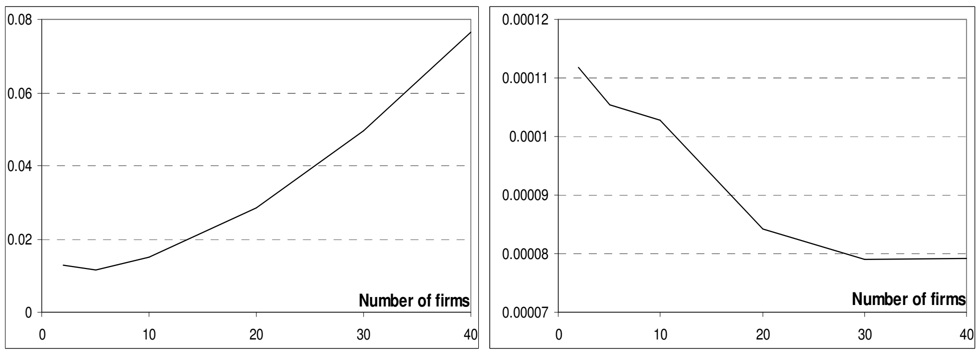


Fig. 6. Estimated time for one single fuzzy system run (the estimation of one benefit) a) in FuzzyCLIPS (depending on the number of rules activated by IE) b) in C-Linux

The necessary times to call one single fuzzy system (one benefit computing) are estimated in Table 7. It can be observed that in C application, the running time becomes stable when the firm numbers grow over 30. This fact is explained by the times used to read/write the set-up files or the output files, which become insignificant when compared to the times used to estimate the benefits and the selling prices. In FuzzyCLIPS the increase of the number of activated rules and of new facts from BF will lead to a bigger computational effort for the IE. The time necessary to compute a single benefit in FuzzyCLIPS application will increase along with *N* (see Fig. 5a).

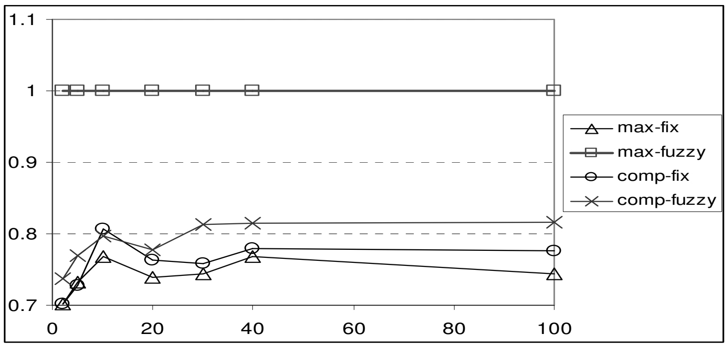


Fig. 7. Running time of the C serial (Linux) application reportedly to the running time per strategy (normalized values of Table 6)

TABLE 8. Comparing the running times for C-serial application (Linux) depend on the type's strategy of the economic model simulated (normalized values Table 6)

Firms number / **Strategy**	**N=2**	**N=5**	**N=10**	**N=20**	**N=30**	**N=40**	**N=100**
"max-fix"	0.702	0.732	0.768	0.739	0.743	0.768	0.744
"max-fuzzy"	1.000	1.000	1.000	1.000	1.000	1.000	1.000
"comp-fix"	0.702	0.728	0.807	0.763	0.759	0.779	0.775
"comp-fuzzy"	0.737	0.770	0.797	0.778	0.813	0.814	0.817

The C-serial results of the strategies complexity estimation are similar with the FuzzyCLIPS results (see Fig. 4), even if the implementations of the used algorithms in simulations, are conceptually different (one is designed in a rule-based language, the other one in a procedural language). The „*max-fuzzy*" strategy is the most time expensive. The „*max-fix*" strategy leads to smaller running times (≈70%). These results exemplify the theoretically estimated time complexities of the strategies (see Table 1).

3. C (distributed computing application). The models were tested on the "*frontend*" computing machine of the GRAI (IBM X3800, INTEL XEON parallel computer with 4 processors 3 GHz, RAM 8G, 4 Hard-disks 146GB SCSI Ultra320, Gigabit network card). This IBM server has the following features, according to the manufacturer: "64-bit four-processor, with up to 3.6 TB of high-speed internal storage; dual-core Intel® Xeon®

Processors MP; runs 32- and 64-bit applications simultaneously; three levels of memory protection; supports new PCI-Express I/O technology.”

The algorithm is distributed; the communication between the central node and the computing nodes is done by the MPI functions (Message Passing Interface). In order to compute the differences obtained between the running times of the distributed variant and the serial one, the economic model has been simulated on the *frontend* for both variants, with the same set-up files.

TABLE 9. Average runtime (in seconds) of the C-serial application, for Linux

Firms number / Strategy	N=2	N=5	N=10	N=20	N=30	N=40	N=44
“max-fix”	0.02	0.31	1.36	5.52	12.71	23.40	27.88
“max-fuzzy”	0.03	0.42	1.84	7.68	17.26	30.86	38.09
“comp-fix”	0.02	0.31	1.45	5.81	13.11	23.72	28.86
“comp-fuzzy”	0.028	0.33	1.42	6.05	13.88	24.88	30.08
Average	**0.0245**	**0.34**	**1.52**	**6.265**	**14.24**	**25.71**	**31.23**

TABLE 10. Average runtime (in seconds) of the C-distributed application variant

Firms number / Strategy	N=2	N=5	N=10	N=20	N=30	N=40	N=44
“max-fix”	0.02	0.105	0.790	4.408	10.759	20.33	21.92
“max-fuzzy”	0.03	0.140	0.936	6.153	13.858	25.13	30.66
“comp-fix”	0.02	0.125	0.731	4.485	10.970	20.19	26.05
“comp-fuzzy”	0.03	0.133	0.735	4.465	11.339	21.91	26.35
Average	**0.03**	**0.13**	**0.80**	**4.88**	**11.73**	**21.89**	**26.25**

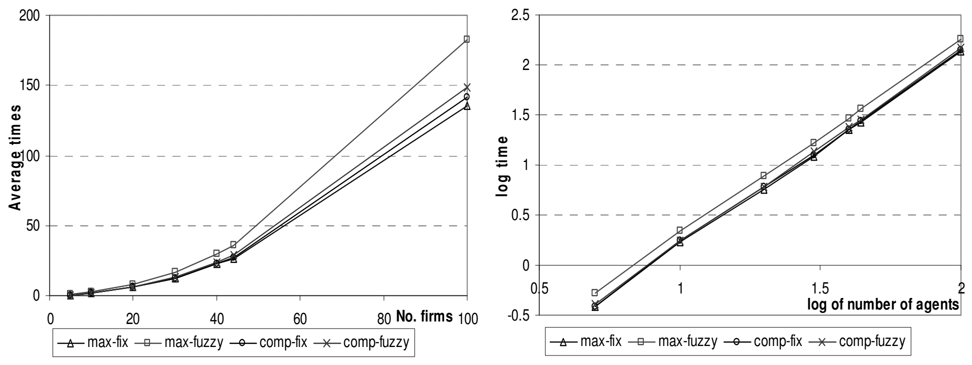


Fig. 8. Variation of the economic system running times for C-serial application (Linux)
a) logarithmic representation b) without logarithm

6. Conclusions

Simulations of the economic models with various strategies of the firms on the market have been performed and contrasted. We compared the running times of economic models simulated in rules-base language (FuzzyCLIPS) and procedural language (C++), in order to study the management strategy complexity. The simulation results show, as expected, that the language used to implement the algorithms has significant influence on the constants appearing in the expression of the complexity of the algorithms. Overall, for the market models tested, the „*max-fuzzy*" strategy proved to be the most time costly, while „*max-fix*" strategy leads to smaller running times (≈70%).

For all the applications we optimized the input data by reading them from the set-up files; this allows a large flexibility for the simulations. Each firm or group of firm of the economic models can develop its own management strategy, with specific rules of benefits sets, of price sets etc.

The simulations of the implemented distributed economic model were made for two operating systems, Windows and Linux, using different compilers, on computers with different configurations, to check how the running times are affected by the system performance – processor, memory capacity, etc.

As expected, the best running times have been obtained for the parallel computation of the economic model, even though we used a multiprocessor server with no computers network attached. After the GRID network completion, simulations will be performed to compare the efficiency of parallel and truly-distributed implementations of the models.

Acknowledgments. This research was partially supported by the project "Academic GRID for Complex Applications – GRAI", 74 CEEX-II03/31.07.2006, financed by the National Authority for Scientific Research.

References

[1]. P. S. Grassia, "Delay, Feedback and quenching in financial markets", The European Physical Journal B, 2000, vol. **17**, pp. 347-362.
[2]. R. D'Hulsta and G. J. Rodgers, "Models for the size distribution of businesses in a price driven market", The European Physical Journal B, 2001, vol. **21**, pp. 447-453.
[3]. T. Negishi, R. V. Ramachandran, and K. Mino (Eds.), "Economic theory, dynamics and markets. Essays in honor of Ryuzo Sato", Series: Research Monographs In Japan-U.S. Business and Economics, vol. **5**, Kluwer Academic Publishers, 2001.
[4]. J. Barkoulas and C. Baum, "Stochastic long memory in traded goods prices", Applied Economics Letters, 1998, vol. **5**, pp. 135-138.
[5]. E. Barucci, "Heterogeneous beliefs and learning in forward looking economic models", Journal of Evolutionary Economics, 1999, vol. **9**, 4, pp. 453-464.
[6]. E. Galic and L. Molgedey, "Beliefs and stochastic modelling of interest rate scenario risk", The European Physical Journal B, 2001, vol. **20**, pp. 511-515.

[7]. C. J. Thompson, "Chaos in economics and management", in R.L. Dewar, B.I. Henry (Editors): 'Nonlinear dynamics and chaos", Proc of Fourth Physics Summer School. World Scientific, Singapore, 1992, pp. 213-229

[8]. A. M. Gil Lafuente, J. Gil Aluja, H.N. Teodorescu, "Periodicity and chaos in economic fuzzy forecasting". Fuzzy Systems. Proc. ISKIT'92, Iizuka, 1992. pp. 85-93

[9]. H. N. Teodorescu., "Chaos in fuzzy systems and signals", Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks. Vol. 1, 1992, Iizuka, Japan, pp. 21-50

[10]. J. Gil Aluja, H. N. Teodorescu, A. M. Gil Aluja, and Al. P. Tacu, "Chaotic fuzzy models in economy", Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks. Vol. 1, 1992, Iizuka, Japan, pp. 153-156

[11]. J. Gil Aluja, H. N. Teodorescu, A. M. Gil Lafuente, and V. Belousov, "Chaos in recurrent economic control of enterprises", Proc. First European Congress on Fuzzy & Intelligent Technologies, Aachen 1993. Verlag Augustinus Buchhandlung, Aachen, ISBN 3-86073-176-9. vol. **1**, pp. 982-986

[12]. H. N. Teodorescu, "Non-linear systems, fuzzy systems, and neural networks", Proc. 3rd Conference on Fuzzy Logic, Neural Nets and Soft Computing, Iizuka, Japan, 1994

[13]. H. N. Teodorescu, T. Yamakawa, V. Belous, and St. Suceveanu, "Interpretation of neuro-fuzzy systems in models in management and creativity. Chaos generation", Fuzzy Economic Review (Spain). Nov. 1995, vol. **1**, pp. 25-42

[14]. T. Weishaupl et al. – "Towards the merger of grid and economy", in H. Jin, Y. Pan, and N. Xiao, editors. Grid and Cooperative Computing - GCC 2004, Wuhan, China, vol. 3252 of Lecture Notes in Computer Science, pp. 563-570. Springer, 2004

[15]. H. N. Teodorescu and M. Zbancioc, "The dynamics of fuzzy decision loops with application to models in economy", Memoriile Secțiilor Ştiințifice ale Academiei Române – Memoirs of the Sections of the Romanian Academy, MAR, Tome XXVI (2003) pp. 301-317

[16]. H. N. Teodorescu, M. Zbancioc, "Two fuzzy economic models with nonlinear dynamics", Proceedings of the Romanian Academy, The Publishing House, vol. **6**, No. 1, 2005, pp. 75-84

[17]. H. N. Teodorescu, M. Zbancioc, "Dynamics of fuzzy models for market players. The three companies case", F.S.A.I., Vol. 11, No. 1-3, 2005, pp. 73-107

[18]. H. N. Teodorescu, M. Zbancioc, "Dynamics of fuzzy models for market players", SOFA 2005, IEEE International Workshop on Soft Computing Applications, ISBN 963-219-001-7, 27-30 Aug, 2005, Szeged-Hungary and Arad, România, pp. 200-205

[19]. H. N. Teodorescu, M. Zbancioc, "Parallelizing and distributing computation for fuzzy economic models", CSCS16 - 16th Int. Conf. on Control Systems and Computer Science, 22-28 May, Bucureşti, România, 2007

[20]. M. Zbancioc, H. N. Teodorescu, "Parallelizing neuro-fuzzy economic models in a GRID environment", Proc. SACCS 2007, Iaşi, Romania, November 16-18, 2007, Editura Politehnium Iaşi, ISSN 1843-7257, pp. 447-452

[21]. Horia-Nicolai Teodorescu, Marius Zbancioc, Laura Pistol, "Parallelizing neuro-fuzzy economic models in a GRID environment", Studies in Informatics and Control, March 2008, Volume 17, Number 1, ISSN 1220-1776, Edited by National Institute for R&D in Informatics ICI Bucharest, pp. 5-16