Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

Study of the Grid infrastructure and middleware of the ALICE project at CERN and preparation for simulations of networked data processing at regional centers

Prague 27.4.2010

Čeněk Zach

Acknowledgements

I would like to thank RNDr. Dagmar Adamova CSc. for consultations and attentive help with writing of this work.

Title:	Study of the Grid infrastructure and middleware of the ALICE project at CERN and preparations for simulation of ALICE's grid system
Author:	Čeněk Zach
Field:	Nuclear Engineering
Specialization:	Experimental Nuclear Physics
Supervisor:	RNDr. Dagmar Adamová CSc., Nuclear Physics Institute, ASCR
Abstract:	In my work, I study the ALICE Computing model, focusing on its Grid aspects. Then, I proceed with learning how to use simulation tool MONARC2, to model performance of the data processing in regional centers. A brief test of MONARC2 basic functions is performed.
Key words:	ALICE Computing model, MONARC

Contents

1 Introduction

The ALICE Experiment at Large Hadron Collider (LHC) at CERN [1][2], as well as the other LHC experiments, is facing the challenge to process volumes of data from the particle collisions of the order of PetaBytes (PB) in one data taking year. To store, process and provide access to the data for thousands of scientists participating in the LHC experiments, the LHC Computing Grid (LCG) [3] has been built. Each of the LHC experiments including ALICE has developed its own computing model utilizing partly the LCG services and partly the experiment-specific Grid infrastructures.

Although from a view of a user, a Computing Grid behaves as a single supercomputer, it is a very complicated infrastructure of network structures, hardware resources and software frameworks. For the good performance of the Grid systems it is necessary to have some estimates of effects of possible changes in the infrastructure, e.g., network and hardware failures, network topology changes, hardware upgrades, Grid middleware updates and so on.

My work has been concerned in the first part with the study of the ALICE Computing model [4]. Then, I turned to study and testing of a software framework MONARC2 [5]. It is a tool/simulation framework whose aim is to provide a design and optimization tool for distributed computing systems. In particular, it is focused on providing realistic simulations of distributed computing systems, customized for specific physics data processing.

My work with the MONARC2 tool was limited to a study of its individual packages and their customization for the simulation of a specific model of a regional computing center, in preparation for development of a more complex framework to be completed in frame of my Diploma thesis.

2 ALICE Computing Model

The ALICE experiment is a dedicated heavy-ion (HI) experiment at the CERN LHC. During the HI data taking it will take data with a bandwidth of up to 1.25 GB/s . It also has its proton-proton (pp) physics program and has also been and will be taking pp data, with a bandwidth of up to 500 MB/s. The project is consuming a huge amounts of computing resources and this will increase with time. These resources are distributed at the computing facilities of the institutes and universities participating in the experiment. This scenario of decentralized offline computing centers has been foreseen in the MONARC model (Models of Networked Analysis at Regional Centers), the predecessor of the MONARC2.

2.1 The MONARC tiered network

MONARC suggests a distribution of computing resources in a hierarchical lay-out populated by so called Tier centers.

Tier-0 center is CERN, with tens of thousands of processing units and PetaBytes of storage. Here all raw data coming from the ALICE Data Acquisition system (DAQ) are safely stored and first pass reconstruction is done.

Tier-1 centers are the major computing centers outside CERN, providing safe storage of copies of raw data. Also, these are places where the reconstructed data are stored and multiple passes of reconstruction and scheduled analysis are taking place. Tier-1s are responsible for long term storage of data produced at Tier-1 and Tier-2 centers.

Tier-2s are smaller regional computing centers. They are used for centrally managed productions of Monte Carlo simulations of collision events in the detector and for processing of end user analysis jobs.

MONARC considers also Tier-3 centers - universities and institutes departmental computing centers intended for local analysis, and Tier-4 centers, which are end user machines, but these are not relevant in the Grid systems of the LHC experiments.

2.2 Computing resources requirements

According to the LHC and experiments planning, for one Standard Data Taking Year (STDY) it is foreseen 7 months of pp running, 1 month of HI running and 4 months for a technical stop, for maintenance and upgrades. This amounts for 10 Ms for pp and 1 Ms for HI combined with the data bandwidth of up to 500MB/s for pp and 1.25 GB/s for HI. Altogether, ALICE should be prepared to process data of the order of PB/SDTY.

After more detailed analysis taking into account estimated sizes of raw and reconstructed data objects per 1 event, expected time needed for data processing, estimates concerning the requirements of the necessary Monte Carlo simulations and volumes of calibration and conditions data, it is possible to elaborate expected requirements on the hardware resources needed for data processing and storage.

The current estimates [6] for the ALICE needs of computing resources for year 2012 are listed in Table 1 (CAF stands for CERN Analysis Facility, a special computer center inside CERN dedicated to prompt raw data reconstruction and for user analysis):

	T0	CAF	T1	T2
CPU (KHEP06)	55.6	10.1	146.1	140.2
Disk (TB)	8741	395	13122	11401
MSS (TB)	9645	-	23632	-

Table 1: The current estimates for the ALICE needs of computing resources for year 2012

For comparison, one core of a modern processor might represent a performance of about 10 HEP06 specs, so altogether the requirements concerning CPU represent approximately 33 thousands of processing units and 35.6 PB of disk space. A very important fact is that almost half of the resources is and will be provided by the Tier-2 centers, which demonstrates the great importance of these centers in the ALICE computing model.

Unfortunately, ever since the beginning of the ALICE Grid computing, the resources available for ALICE have been substantially lower than requested, especially concerning the disk capacity, and this requires a very complicated procedures concerning decisions which data should be permanently stored.

According to the ALICE Computing Technical Design Report [4], all raw data produced by the ALICE detectors will be stored permanently for the lifetime of the experiment. This includes 2 copies of raw data, one at CERN and one at a Tier-1 center.

Then there are reconstruction passes, analysis objects, Monte Carlo data, calibration, alignment and condition data. The policy as to which data will be stored on tapes (permanent storage) and which will go only to disk (transient storage) is not yet fully specified. In the beginning of 2010, more than half of the ALICE storage capacity was full with the data sets from the cosmic and calibration data taking and from the first collisions period in November and December 2009. Thus, the need of a new strategy concerning the data storage is evident.

2.3 Network

As was already mentioned, data from the ALICE detector will be recorded at a rate up to 1.25 GB/s during HI runs and up to 500 MB/s during pp runs. It is therefor sufficient to have a 10 Gb/s link between the ALICE detector site and CERN.The ALICE DAQ system, which collects data from the 18 ALICE subdetectors and performs the event building, is providing a large disk buffer on the detector site, which is supposed to be able to store an equivalent of one day of pp data taking. At the CERN storage facility, the transfer of data from disk to tape is performed at a rate of up to 100MB/s maximum; not all the data recorded on disk (e.g., data from subdetectors calibration runs) will be stored permanently.

The data traffic outside CERN and the in/out- coming traffic to/from CERN is quite complicated.

Tier-0 → Tier-1s All the raw data permanently stored at CERN are replicated to Tier-1 centers. The data produced during the processing passes of raw data (Event Summary Data - ESD) are stored at CERN and Tier-1s, so they are replicated from the site they were produced. The data produced at Tier-1s during the MC production cycles are copied to CERN. The replicas of data objects produced during the scheduled analysis productions (Analysis Object Data - AOD) at Tier-1s can be copied to CERN.

Tier-1s-Tier-2s → Tier-0 This traffic includes transfer of ESDs, both from the raw data processing and the MC simulations, and AODs from the production site to another site in the system.

Currently, the network links of the capacity of 10Gb/s between CERN and Tier-1s and links of capacity 1Gb/s between Tier-1s and Tier-2s are available and this capacity is considered sufficient at the moment.

2.4 The global picture

Most of the processes which data objects are subject to are fully automatic. For instance, concerning the raw data, the conditions data gathering, the transfer from the detector site to CERN storage facility, the registration of data objects in the ALICE Grid catalog, the first pass of the reconstruction and registration and storing of its output, this all is done automatically, all processes being amply monitored. Taking another example, the same is true concerning the launch of the MC simulation productions and transfer, storage and registration of its output. All the data objects available for processing are accessible only with the help of tools provided by the ALICE Grid middleware AliEn [7], i.e., the software framework for the management of processes on the Grid. Some of the services and tools are briefly described in the following section.

The ALICE Grid system has been in development ever since 2002 and represents a quite complicated structure these days. Altogether, it includes about 100 computing sites and 60 distributed storage facilities [8][9]. The centers are spread all over the world including next to Europe and North America also Asia, Africa and South America (see Figure 1 or [10]). For such a complex system to be functional and deliver a steady performance, it must be extensively monitored and there must be some predictions concerning the behavior of its components in cases of heavy loads or disturbances of all kinds.

One possible framework for the simulation of behavior of such a complex grid system components is provided just by the MONARC/MONARC2 project, study and testing of which is a part of this work.

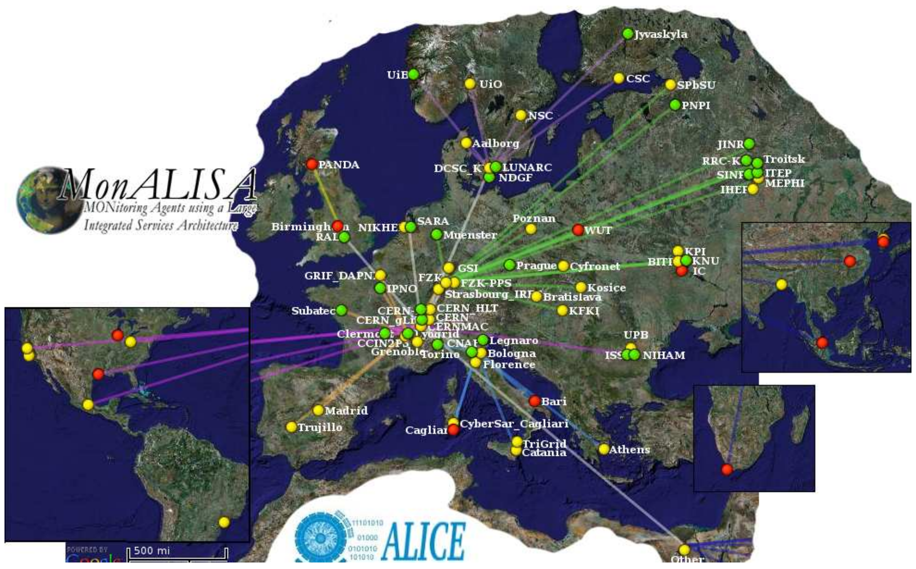

Figure 1: Map of ALICE Tier centers around the world

3 Middleware

In order to correctly simulate the behavior of a Grid system, one has to incorporate elements/services of this system-specific middleware into the simulation framework. The framework should implement as realistically as possible the system-specific management of, e.g., submissions, registrations and processing of jobs, data transfers, and reliability of the individual elements. WLCG uses the gLite middleware [11]. The ALICE Grid project has developed its own middleware, AliEn, which provides some project-specific tools and services not provided by gLite. A short description follows of the services essential for the simulations.

3.1 Computing Element

Computing element is an entry point to a grid site. It authenticates users and submits jobs to worker nodes, aggregates and publishes information from other nodes. It includes generic interface to the local cluster called Grid Gate (GG), Local Resource Management System (LRMS) and collection of Worker Nodes (WN). For simulation, important factors apart from the CPU power and reliability are: percentage of jobs failed during normal operation of CE, during degraded state of CE, percentage of time CE is down, etc.

3.2 Storage Element

Storage element provides storage for working data and access to the data via a secure transfer protocol. Important variables apart from available storage space, read/write speeds and bandwidth concern reliability against overload, percentage of failed transfers from/to SE and percentage of lost/corrupted files.

3.3 Workload Management System

WMS is a middleware component/grid service, that receives job submission requests from users, assigns them to an appropriate CE, monitors their status and retrieves their output. Currently, gLite WMS with gLite-CE is used together with the CREAM-CE, which uses direct job submission not requiring an additional WMS service.

The important factors for a WMS are its stability against overload and the percentage of successful job submissions.

3.4 VOBOX

At each of the ALICE sites, a dedicated server called VOBOX is installed on which some ALICE specific services are running. They include, e.g., the ClusterMonitor, which transfers the monitoring informations to the ALICE Grid global monitoring system MonALISA [12], and AliEn Computing Element, which is a service different from the WLCG CE and its purpose is to submit so called JobAgents, described later in the text, to the site local WLCG CE.

4 MONARC2

MONARC (MOdels of Networked Analysis at Regional Centers) is a tool developed to provide realistic simulations of large distributed computing systems. The simulation framework is not intended as detailed simulator for basic components, but it aims at correct description of the performance and limitations based on realistic mathematical models. It is written in Java and has a discrete event process oriented simulation approach, which is well suited to describe concurrent running programs, network traffic as well as all the stochastic arrival patterns. It has a complex GUI to simulation engine which allows dynamic change of parameters and monitoring of simulation results.

MONARC2 is an extended version of the original MONARC simulator.

MONARC is based on a model of interconnected regional centers. Each has its own CPU farm, database servers, mass storage units, local area network, job scheduler and a waiting queue. With such structure it is possible to build a wide range of computing models. The diagram representing the most important classes of MONARC2 is shown in Figure 2.

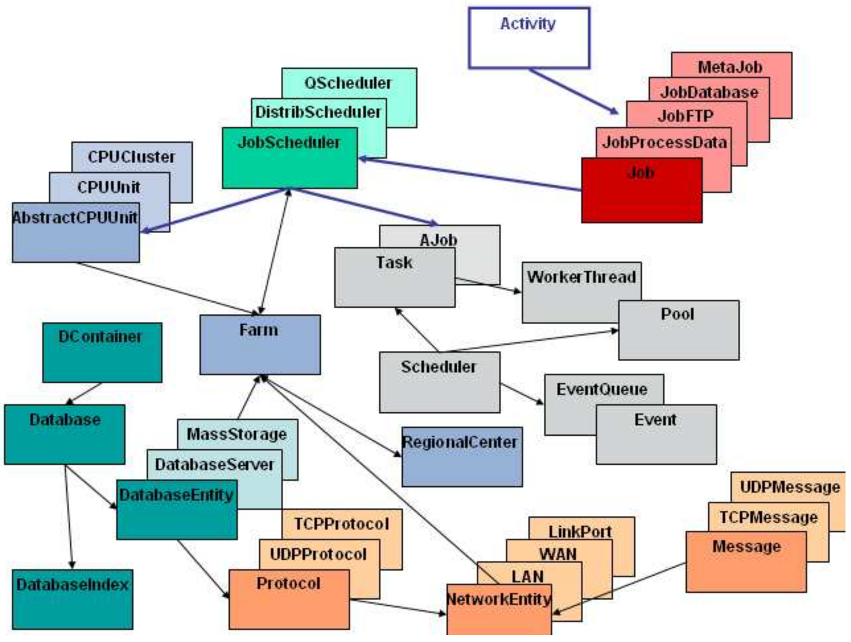

Figure 2: Most important classes of MONARC2

4.1 Simulation Engine

The simulation engine uses threaded objects (Active Objects) to simulate time dependent processes, which concurrently compete for shared resources. There is a base class Task, which must be inherited by all entities that require a time dependent behavior - such as the running jobs, the database servers or the network. Every Task is run by one thread. The creation and destruction of threads is expensive, so a pool of worker threads is used. If a Task is created, its assigned to a worker thread and when it finishes, the worker thread returns back to the pool. If there is not enough worker threads, more are created. Communication between Tasks is done through simulation events defined in Event class. An event is created by a Task and is sent to a Task. It contains:

- time the event was created
- IDs of source and destination Tasks
- type of event - ENULL, SEND or HOLD_DONE
- tag (in case of the type SEND) - integer that represents what should be done when message is received
- auxiliary data - for example, job that should be executed

The whole simulation is managed by an object called Scheduler. It keeps several data structures to manage what is happening:

- a vector with all the tasks that are currently alive
- a pool of worker threads
- a priority queue with the future events
- a priority queue with deferred events, that already happened, but could not be processed

Every existing Task is in one of the following states: Created, Ready, Running, Waiting or Finished. The scheduler goes through the following algorithm:

1. Look at each simulation Task and:

   a) If the task is in the Created state, assign it to worker thread and change its state to Ready

   b) If the task is in the Ready state, (re)start its execution
   c) If the task is in the Finished state, remove it

2. Wait until all the tasks that were running are in the Waiting state or finish their execution

3. Process the events

   a) Take from the future queue the event(s) with the minimum time stamp, the simulation time advances, becoming equal to that time stamp
   b) For each event taken from the queue, look for the destination task. If it is in the waiting state, deliver the event to the task, else, put the event into the deferred queue

4.2 Job Model

Jobs are described by class called Job, which defines job that does nothing, so it must be extended to actually do anything. A job has several attributes:

cpuPower - the CPU power needed by the job

memory - memory needed by the job

processingTime - the time needed to process the job, given defined CPU power

allocatedCPU - CPU allocated to the job

ti,tf - initial and finish time

nrDataUnits - number of data units processed by the job

dataUnitName - name of data units processed by the job

CpuID - ID of CPU where the job must run, if it doesn't matter, it's set to -1

nextJobs - a vector of jobs that will be processed after this one finishes (optional)

schedulePriority - priority in the regional center

runningPriority - the priority on the CPU, where the job is running

4.2.1 Execution of Jobs

Jobs are time dependent entities, so they are executed using threads. An Active Object called AJob (active job) is inherited from the Task class. The job scheduler has a pool of free active jobs. When it schedules a job, it sends an event to one of the free active jobs with tag TAG_START_JOB and the job itself is appended as auxiliary data. The active job then executes run() method of the job. When the job is finished, the active job starts waiting for another event.

An estimate of time for how long should a job be running is calculated as

$$time = job.processingTime * job.cpuPower/allocatedCPU$$

Every time a job is added or finishes on a CPU, all the other jobs on the same CPU receive event with tag TAG_CPU_CHANGED and their execution times are recalculated (see Figure 3). Also, jobs on the same CPU can have different priorities. If N jobs share the same CPU, then cpuPower assigned to job i is calculated as

$$allocatedCPU_i = totalCpuPower * P_i/(P_1 + P_2 + ... + P_N)$$

P_i is priority of i-th job.

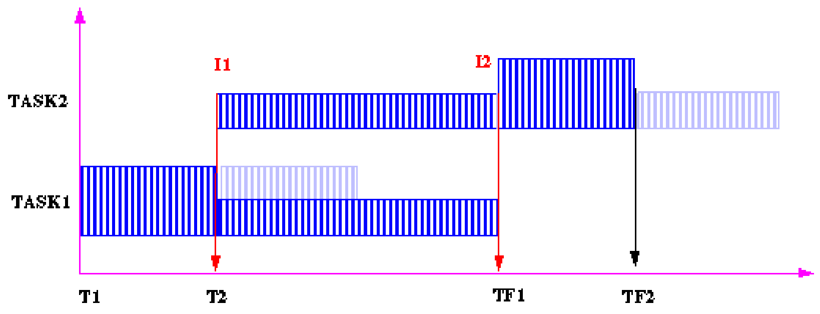

Figure 3: Jobs execution time is recalculated every time another job enters or leaves the same CPU

4.2.2 Job Implementations

The run() method is the one that should be overridden by the class extending job. Following Job classes are implemented:

JobProcessData - simulates the processing of specified data

JobCreateDB - creates specified database

JobFTP - handles the transport of one message from one place to another

JobProcessMM - reads data from a database server and processes it

JobDatabase - basic job for working with databases

4.2.3 Activities and Job Scheduling

Jobs are created and submitted using the Activity class, which has the mechanism for sending the jobs to a regional center. This class must be extended by the user (who has to create the actual jobs). Each regional center can have multiple activities which submits jobs to it. Each Activity contains a vector of jobs and Double objects, which serve as waiting time between submissions of jobs. Activity class is also an extension of the Task class, its RUN() method takes jobs (or Doubles) from the vector and:

- if object is a job, submits it
- if object is a Double, it sleeps for specified amount of time using simHold() method

When the Activity has no more jobs to submit, it sends event with the tag TAG_NO_JOBS, to announce that it finished. Each regional center has a job scheduler, which decides on new jobs, if they should be executed or put to the waiting queue. It's also possible for jobScheduler to send a job to another regional center. There is a basic jobScheduler class, which can be extended to implement another scheduling algorithm. It contains these important data fields:

activeJobs - a vector with free active jobs

jobQueue - a vector with jobs that are waiting for resources

activeJobsRunning - the number of active jobs that are currently running

4.3 Data Model

Two main entities are implemented to simulate the databases: the database server, which stores data on disks, and the mass storage center, which stores data on tapes. All the database entities have link ports and the interaction between jobs and those entities is done through the network implementation.

The smallest data unit is data container, which emulates a database file. Data containers are grouped into databases, which reside on database server or mass storage center. All the data containers, databases and database entities are managed by database index defined globally within the project.

4.3.1 Data Container

Data container is defined by:

size - size of container in MB

containerType - type of data the container holds

containerName - name of the container

data - data stored by the container

db - database the container belongs to

accessed - how many times was container accessed

lastUsed - when was the last access - these last two fields are used when determining which data should be moved to the mass storage center

lock - whether the data is read-only

place - whether data is on disk, tape or on transfer

4.3.2 Database

Database is defined by:

databaseName - name of the database

containers - hashtable of containers in the database

4.3.3 Database server

Database server is defined by: readSpeed, writeSpeed, readLatency, writeLatency and diskSize

4.3.4 Mass Storage Center

Mass storage center is defined by: nrDrivers, usedDrivers, tapeSize, mountTime, searchSpeed, readSpeed, writeSpeed, tapePerSilo, nrSilo, tapeDrives, totalSize.

4.3.5 Database index

Database index is used to find any data within the simulation model. It is defined by:

mapAddress - hashtable mapping the dbEntities vs. addresses

mapContainers - hashtable mapping the dbEntities vs. vectors with their containers

mapDatabases - hashtable mapping the dbEntities vs. databases

4.4 Network Model

Every entity involved in the simulation has a network link port, which is an entity that receives and sends messages. Link port is defined by unique IP address or by the unit it's attached to, also by maximum bandwidth in Mbps, etc. Link ports are connected to LANs, LANs to WLANs. WANs are connected with routers.

It is practically impossible to simulate the network at a packet level, so interrupt scheme is adopted. When a message transfer starts between two endpoints, interrupt event is sent to all messages that share at least one part of the way. Then arrival time is calculated for all of them using the minimum speed value of all the components on the way of each message (similarly to jobs sharing one CPU).

5 Using MONARC

Using MONARC can be (from my point of view) divided into three parts.

- Writing configuration file, which defines individual regional centers, their resources and network topology, which connects their individual parts and them together
- Extending available classes of jobs, schedulers, etc. to achieve a needed behavior
- Defining the output using available or custom clients

I decided to test the behavior of the default job scheduler, so I created a small model of our local cluster called Sunrise.

5.1 Main Configuration File

The main configuration file contains all information about topology and resources layout. The following code is one section of my configuration file s showing settings of CPUs of our cluster and definition of one data type. My model cluster has 10 CPU units, each of power 8000 SI95 (the units are basically arbitrary, but the same units must be used for CPU and data unit definition). Also, each CPU unit has 32 GB RAM, 100 Mb/s connection to LAN1 and ignores page size. First CPU unit is assigned address 192.168.20.101, next one *.102, etc. It is also possible to define varying processing time for data units, using diversity of available distributions.

```
[cpuSunrise]
from = 1
to = 10
cpu_power = 8000.0
memory = 32000.0
page_size = -1.0
link_node = 192.168.20.101
link_node_max_speed = 100.0
link_node_connect = LAN1

[esd]
cpu_power = 15.0
memory = 15.0
processing_time_average = 15
processing_time_distribution = monarc.distribution.FixedDistribution
```

5.2 Extending Classes

The user has at his/her disposal many implemented classes, that can be used right away. There is however one class, which user must extend in order to simulate anything - the Activity class. This class is used to submit work to its center and it is entirely the users responsibility, what he wants to do. The following code is a simple example of pushJobs() method, which is practically the only one that needs to be overridden to extend the Activity class. It submits *numJobs* jobs, where each job has to process *number_of_data_units* data units. It always waits for *time_to_wait* seconds after submitting each job.

number_of_data_units is a random number from 0 to 99, *time_to_wait* is also random number from 0 to (*delay*-1). *numJobs* and *delay* are parameters read from another configuration file - user can have as many as he needs of those.

```
public void pushJobs() {

    int i;

    Date now = new Date();
    random = new Random(now.getTime());

    for (i = 0; i < numJobs; i++) {

        int number_of_data_units = random.nextInt(100);
        int time_to_wait = random.nextInt(delay);

        JobProcessData job = new JobProcessData(10, 10, "raw", number_of_data_units);

        addJob(job);
        addTime(time_to_wait);
    }
    start();
}
```

5.3 Defining Output

The output is defined in a form of XML file. Many output formats are available, including text and graphics. Practically everything that goes on in the simulation can be monitored using clients. The most used clients are implemented. If the user wants a special monitoring, he/she has to extend the appropriate class. Monitoring can be real-time, which is useful for long simulations.

The following code is an example of output definition I used to get the graph presented later in this text. In this code, the monitored entity is the whole computing farm and 3 clients are used as parameters. These monitor

CPU utilization, total number running and waiting jobs and total CPU and memory load.

```
<client name="out1" class="monarc.output.GraphicClient.FarmCPUUtilization"/>
<client name="out2" class="monarc.output.GraphicClient.FarmJobs"/>
<client name="out3" class="monarc.output.GraphicClient.FarmIntern"/>

<declare farm="sunrise" cluster="farm" class="monarc.output.parameters.OutputParameters">
        <parameter client="out1"/>
        <parameter client="out2"/>
        <parameter client="out3"/>
</declare>
```

5.4 Test of Job Scheduler

I was changing the load put on the cluster using *delay* parameter - I always submitted the same amount of jobs. First I have put model cluster under a light load. Default job scheduler did not distributed to jobs evenly, but in order. First CPU unit was under load almost all the time while the last one executed just a few jobs (see Figures 4 and 5).

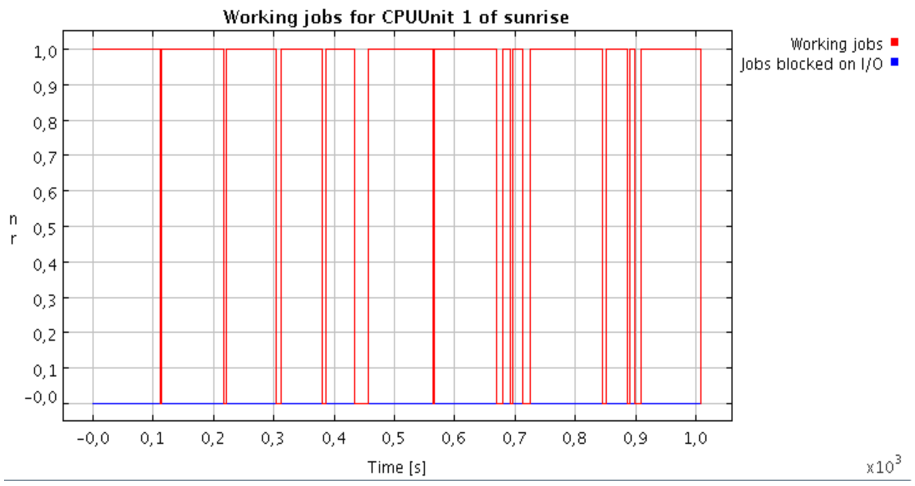

Figure 4: Jobs on first CPU unit with cluster on light load

It can be seen from Figure 6, that with cluster under the light load there were no waiting jobs throughout the whole time. This changed when I put model cluster under medium load. Then all the CPU units were under same load eventually (but again first one came first). Figures 7 and 8 show, how jobs were arriving at the farm and when the load reached the full capacity of the farm.

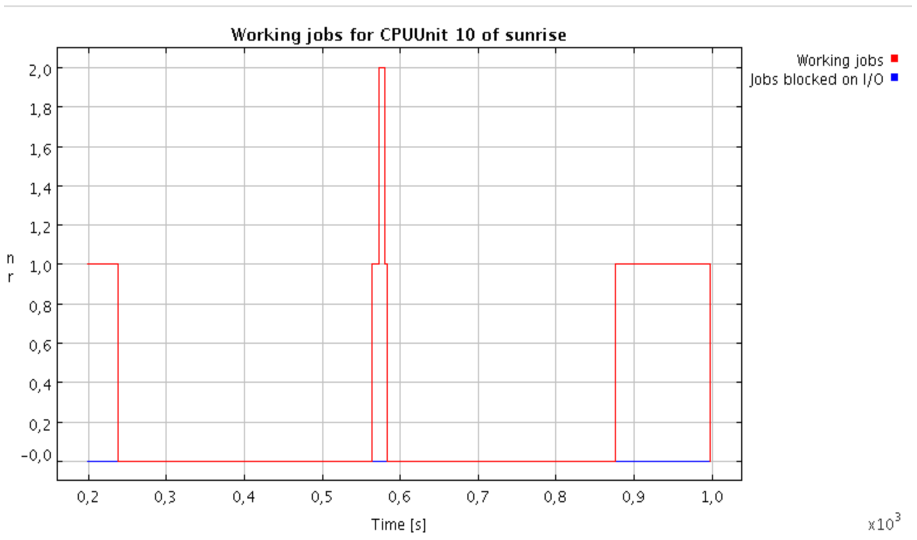

Figure 5: Jobs on last CPU unit with cluster on light load

The most interesting part is when the load becomes heavy. The default job scheduler finds itself with 100 jobs in a very short time, so it fills the whole cluster with as many jobs as memory can support. Then it lets the memory usage drop to approximately 50% and after that it starts submitting new jobs and keeps memory usage around 60%, until the end (as can be seen from Figure 9). The same can be seen from Figure 10, where many jobs are running in the beginning, but their numbers drop and stay stabilized after a while.

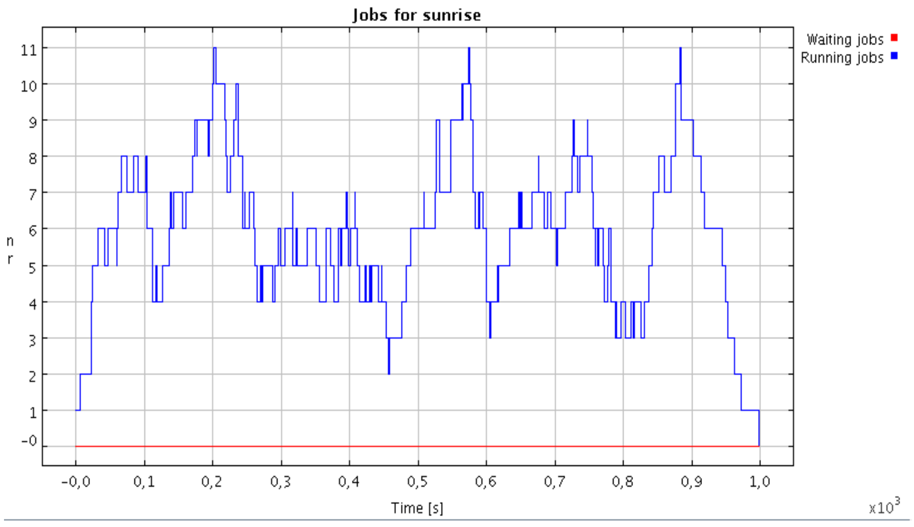

Figure 6: Jobs on the whole farm with cluster on light load

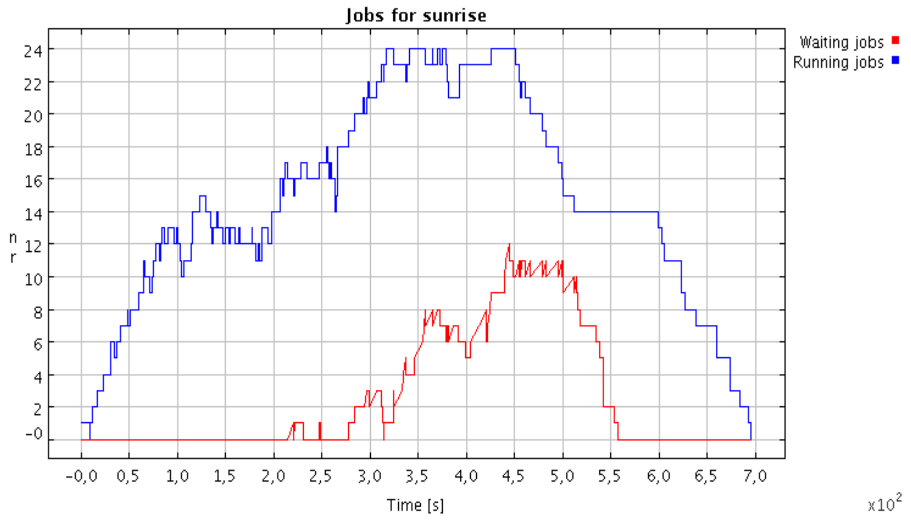

Figure 7: Jobs on the whole farm with cluster on medium load

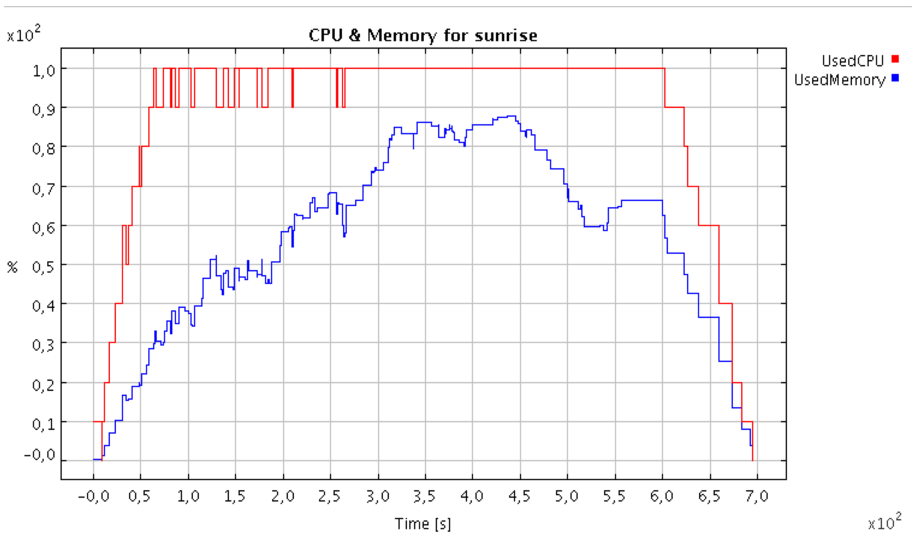

Figure 8: Load on the whole farm with cluster on medium load

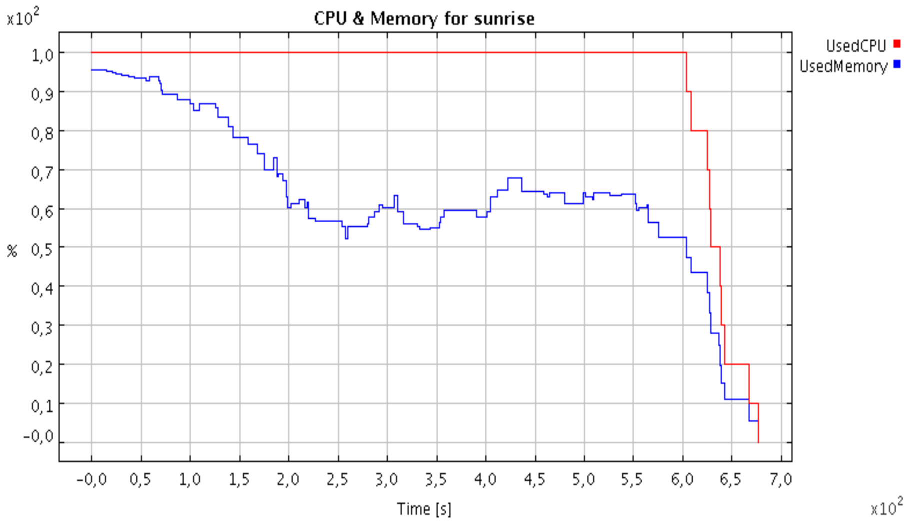

Figure 9: Load on the whole farm with cluster on heavy load

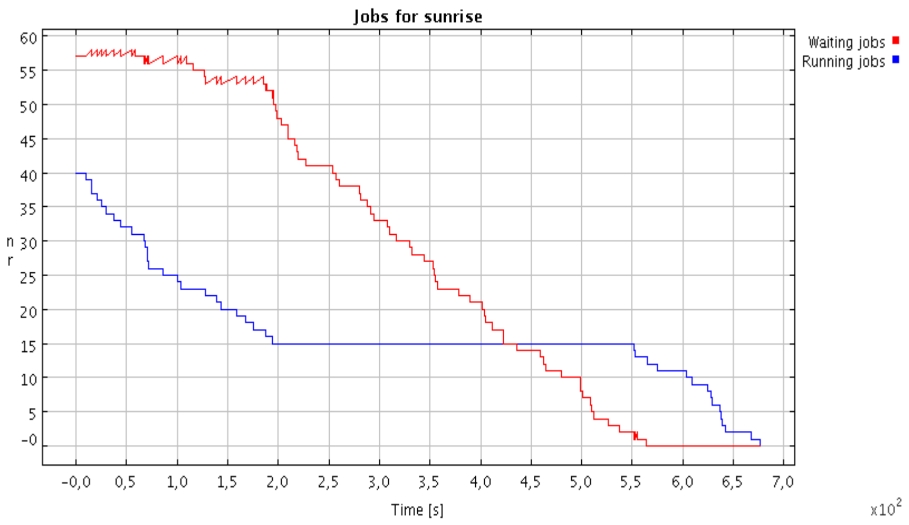

Figure 10: Jobs on the whole farm with cluster on heavy load

6 Summary

MONARC simulation tool offers an efficient way to deal with distributed computing systems. Users can define resource layout, network bandwidth, CPU power, storage capacity in configuration files and any wanted behavior of user submission, job scheduler, etc. by extending existing classes. Output is available in both text and graphical form and is highly customizable.

In my present work, I have first studied the Computing model of the ALICE experiment, focusing in particular on the aspects of the processing of data in the ALICE distributed computing environment, the ALICE Grid. This includes using the services and resources available from LCG and also ALICE-specific structure: the AliEn framework.

A very important part of the ALICE distributed computing system are the Tier-2 centers, which provide almost half of the computing and disk resources. With over 90 Tier-2 centers spread all over the world, it is necessary to master a system of job submission and data transfers which would ensure an efficient usage of the available resources with as few bottlenecks as possible. To elaborate such a system, realistic simulations must be performed of the behavior of the Tier-2 centers under loads of jobs.

The second part of my work was concerned with the study and testing of the MONARC2 simulation framework and tools. The aim of MONARC2 and its predecessor MONARC is to provide a design and optimization tool for distributed computing systems and is in particular customized for LHC-specific Physics data processing.

In my work, I decided to test the behavior of the MONARC2 default job scheduler in frame of a small model of our local cluster called Sunrise. To perform this, I put together an appropriate configuration file, then wrote extensions of corresponding classes available in MONARC2 class index and defined my preferable form of output.

I tested the developed model using different job submission schemes and obtained results favoring the use of so called medium load of jobs on the center. However, the model used in this work was extremely simple and has not incorporated any special features of the job submission schema of the ALICE Grid system and ignored entirely the network and storage systems features. These aspects will be incorporated in the model in frame of my Diploma thesis with the aim to obtain an optimal schema of the job submission management to achieve as stable processing of a maximum number of jobs at each center of the ALICE system as possible.

References

[1] ALICE Colaboration. Alice: Physics performance report volume I. *J. Phys.*, G30(1517), 2004.
`http://aliceinfo.cern.ch/Collaboration/index.html`.

[2] ALICE Colaboration. Alice: Physics performance report volume II. *J. Phys.*, G32(1295), 2006.
`http://aliceinfo.cern.ch/Collaboration/index.html`.

[3] LHC Computing Grid. `http://lcg.web.cern.ch/LCG/`.

[4] P. Cortese et al. Alice computing: Technical design report. Technical report, CERN, 2005.
`http://aliceinfo.cern.ch/Collaboration/Documents/TDR/Computing.html`.

[5] C.M. Dobre and C. Stratan. Monarc simulation framework. Proc. of the RoEduNet International Conference, Timisoara, Romania.
`http://monarc.cacr.caltech.edu:8081/www_monarc/monarc.htm`, May 2004.

[6] Y. Schutz. New site resources profile requirements and pledges. `http://indico.cern.ch/conferenceDisplay.py?confId=66646&view=cdsagenda`.

[7] P. Saiz et al. Alien - alice environment on the grid. *Nucl. Instrum. Meth.*, A502:437–440, 2003.
`http://alien.cern.ch/twiki/bin/view/AliEn/Home`.

[8] Status of Sites services.
`http://alimonitor.cern.ch/stats?page=services_status`.

[9] Status of SEs.
`http://alimonitor.cern.ch/stats?page=SE/table`.

[10] Map of ALICE sites.
`http://alimonitor.cern.ch/map.jsp`.

[11] gLite. `http://glite.web.cern.ch/glite/`.

[12] C. Grigoras et al. MonALISA: An Agent Based, Dynamic Service System to Monitor, Control and Optimize Distributed Systems. Proc. of the CHEP'07 Conference, Victoria, Canada, September 2007.