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

Simulations of complex distributed computing systems in High Energy Physics

Prague 6.5.2011

Čeněk Zach

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ______________ ____________________

podpis

Poděkování

Chtěl bych poděkovat paní RNDr. Dagmar Adamové, CSc. za konzultace a nesmírnou pomoc při psaní této práce. Také bych chtěl poděkovat Dr. Latchezaru Betevovi, Ph.D a Prof. Iosifu Legrandovi, Ph.D za konzultace a cenné rady.

Název práce:	Simulace komplexních distribuovaných výpočetních systémů ve fyzice vysokých energií
Autor:	Čeněk Zach
Obor:	Jaderné Inženýrství
Zaměření:	Experimentální Jaderná Fyzika
Druh práce:	Diplomová práce
Vedoucí práce:	RNDr. Dagmar Adamová, CSc., Ústav jaderné fyziky, AV ČR

Abstrakt:

Ve své práci jsem studoval výpočetní model experimentu ALICE a simulační nástroj MONARC2. Rozšířil jsem balík MONARC2, aby obsáhl některé podstatné části z výpočetního modelu experimentu ALICE, jako je například JobAgent nebo Alien Resource Broker. Použil jsem rozšířený balík k simulacím 6 výpočetních center z Evropy. V simulacích jsem sledoval efekty požadavků na umístění dat a procenta analyzačních úloh na trvání úloh a jejich efektivitu. Také jsem provedl několik benchmark simulací balíku MONARC2.

Klíčová slova: Výpočty ve fyzice vysokých energií, výpočetní Grid, AliEn, LCG, MONARC

Title:	Simulations of complex distributed computing systems in High Energy Physics
Author:	Čeněk Zach
Field:	Nuclear Engineering
Specialization:	Experimental Nuclear Physics
Supervisor:	RNDr. Dagmar Adamová, CSc., Nuclear Physics Institute, ASCR

Abstract:

In my work I studied the Computing Model of the ALICE experiment and the MONARC2 simulation tool. I extended MONARC2 package to incorporate some important parts of the ALICE Computing Model like JobAgent or Alien Resource Broker. I used the extended package to simulate a model of 6 computing centers from Europe. In the simulations I examined effects of the requirements on data location and the percentage of analysis jobs on the duration of these jobs and their efficiency. I also performed a few benchmark simulations of the MONARC2 package.

Key words: Computing in High Energy Physics, Computing Grid, LCG, MONARC, AliEn

Contents

Motto:

"There are three essential components to a Particle Physics experiment: the accelerator, the detectors and the computing. . . .

At the collision point, energy is turned into mass. . . . Occasionally this leads to new particles to be produced that are not stable in our Universe today. . . . The decay products of these particles are electronically registered by the detectors. The computing turns a huge flow of digital data coming out of the detectors into useful physical information about the collisions and stores that information, which is then processed in search of those rare events where new particles are produced.

This is the function of the Worldwide LHC Computing Grid. It stretches around the Globe, collecting resources in hundreds of data centers in over 30 countries. . . . At the end of the day, the net product of this huge enterprise is pure knowledge."

– Robert Aymar, the Director General of CERN 2004 - 2008

1 Introduction

The ALICE Experiment at Large Hadron Collider (LHC) at CERN [1], [2], [3], [4], 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) [5] has been built. Each of the LHC experiments including ALICE has developed its own computing model utilizing partly the LCG resources and 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 networks, hardware resources and software frameworks. For a 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. During the data taking, the data flow is huge and sustained over long periods of time and a good performance of the Grid system is absolutely essential. During this period, any latencies can cause problems with data storage or processing or even lead to a loss of the precious data.

The highly dynamic behavior of Grid components makes it difficult to perform detailed analysis of their behavior. A practical and acceptable approach to the description of components of a Grid system is the evaluation by simulation (adopted for instance in [6]).

My work has been concerned in its first part with the study of the ALICE Computing model [7]. Then, I proceeded to simulate behavior of a Grid system components using a specific simulation package/framework MONARC2 [8]. It is a discrete-event simulator which allows modelling of large distributed systems. The original intention of the project was modelling and simulation of High Energy Physics (HEP) experiments' computing infrastructures. MONARC2 is based on Java technology with a built-in multi-thread support for concurrent processing. It offers an Interrupt-driven scheme for discrete event simulation, which is convenient for description of the events arrival patterns and concurrent running tasks sharing resources, which are typical for the HEP computing projects.

As a first step in my work with the MONARC2 toolset, I studied and tested its features and performance. I gained experience with its individual packages and their customization for the simulation of a specific model of a regional computing site. After this introductory part I performed, as a validation study of MONARC2, a simulation to reproduce a month-average performance of the Storage Element (SE) at one of the ALICE Tier-2 sites [9], and namely the average traffic (see [10]). Finally, I considered a system consisting of 6 selected ALICE Tier-2 sites. The computing and storage resources of each site and the corresponding network system were parameterized using information from the available monitoring systems. Using MONARC2, I investigated the job processing performance and the data transfer speed in dependence on job processing conditions.

At the beginning of my work with MONARC2, I intended also to make simulations of dependence of the job submission performance at an ALICE Tier-2 site on specific types of LCG Computing Elements(CE, see later in the text). This issue was however investigated intensively by the ALICE CERN Grid team and by the time I started to modify some of the MONARC2 packages for the purpose of my simulations, the CREAM CE [11] demonstrated its advantages and was adopted as the most suitable CE for ALICE. This is the reason I turned to the study of the job processing performance, which is a long standing and permanently topical issue in the Grid computing for ALICE, as well as for the other LHC experiments.

In section 2, I will briefly describe the Computing model of the ALICE experiment. In section 3, I will discuss individual elements of the Grid middleware, which will help to explain my implementation of the simulated system into the MONARC2 framework. In section 4, I will describe in detail the architecture and functionality of the MONARC2 toolset. Then in section 5 I will introduce the changes I made to the MONARC2 package. In section 6, I will describe in detail the simulation of the job processing performance with MONARC2 and the results. Section 7 concerns benchmarking of the MONARC2 package and section 8 contains a short summary and conclusions.

2 ALICE Computing Model

The ALICE experiment is a dedicated heavy-ion (HI) experiment at the CERN LHC. During the HI data taking it can take data with a bandwidth of up to 2.5 GB/s . It also has its proton-proton (pp) physics program and so it also takes 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 conceived in the MONARC model (Models of Networked Analysis at Regional Centers) using the MONARC simulation tool.

2.1 The MONARC tiered network

MONARC suggests a distribution of computing resources in a hierarchical layout 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) is safely stored and first pass reconstruction is made.

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 is stored and multiple passes of reconstruction and scheduled analysis are taking place. Tier-1s are responsible for a 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) there are 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 (megaseconds) for pp and 1 Ms for HI combined with the data bandwidth of up to 500MB/s for pp and 2.5 GB/s for HI. Altogether, ALICE should be prepared to process data of the order of PB/SDTY.

As a validation of these estimates, during the data taking in 2010, with 7 months of the pp collisions and 1 month of the Pb-Pb collisions, ALICE recorded about 2.35 PB of raw data out of which 0.9 PB came from the Pb-Pb collisions (this number does not include replicas of raw data files).

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 [12] 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. Tn stands for Tier-n and MSS are mass storage systems (tapes)):

	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 [7], 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 and which will go only to disks 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 is recorded at a rate up to 2.5 GB/s during HI runs and up to 500 MB/s during pp runs. 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 LHC data spill. 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 reconstruction 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 and Tier-2s can be copied to CERN.

Tier-1s ↔ Tier-2s ↔ Tier-0 This traffic includes transfers of ESDs, both from the raw data reconstruction 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 1-10 Gb/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 and the conditions data gathering, the transfer from the detector site to the 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, all this 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 with the help of tools provided by the ALICE Grid middleware AliEn [13], 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 78 computing sites and 54 distributed disk storage facilities [14][15]. 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 [16]). 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.

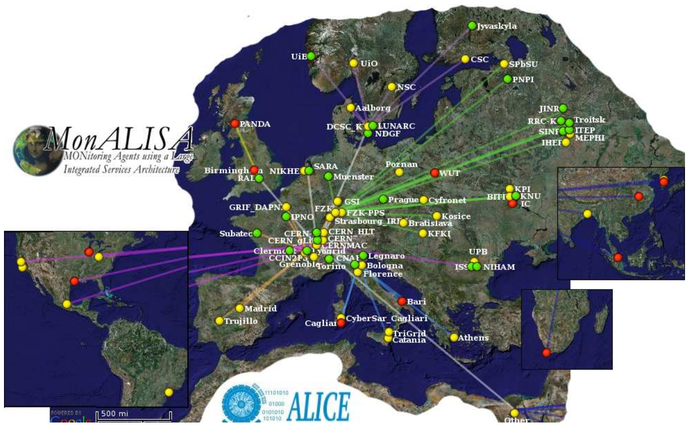

Figure 1: Map of ALICE Tier centers around the world

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 the system-specific management of, e.g., submission, registration and processing of jobs, data transfers, and reliability of the individual elements as realistically as needed to get correct results. WLCG uses the gLite middleware [17]. The ALICE Grid project has developed its own middleware, AliEn [13], 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 (CE) 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 (SE) provides storage place and access for data. 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, the gLite-CE of the gLite WMS has been deprecated in the ALICE job submission management and the priority has been given to 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 information to the ALICE Grid global monitoring system MonALISA [18], 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.

3.5 JobAgent Model

As mentioned above, one of the services running on the VOBOX is AliEn Computing Element. It is responsible for submitting JobAgents to the local batch system. JobAgents then take place of ordinary jobs that wait in the local queue. When assigned to a Worker Node they gather information about, among others, available resources, installed software, free space and close SEs and send these information to the AliEn central services. There this information is matched against jobs waiting in the Central Task Queue and permission to get the first matching job is sent back to the JobAgent. JobAgent then fetches the job directly from the Central Task Queue and runs it. After this job finishes, JobAgent again gathers info and the cycle is repeated. This holds until the time to live (TTL) given to the JobAgent expires and the JobAgent then dies. This is called the ”Pull model” of the job submission management (see. Figure 2).

An opposite alternative - the ”Push model” - relies on central monitoring system, which must hold all the static and dynamic information about all the resources. These are used to ”push” jobs whenever there are free resources available. This model does not scale well, so it is not widely used.

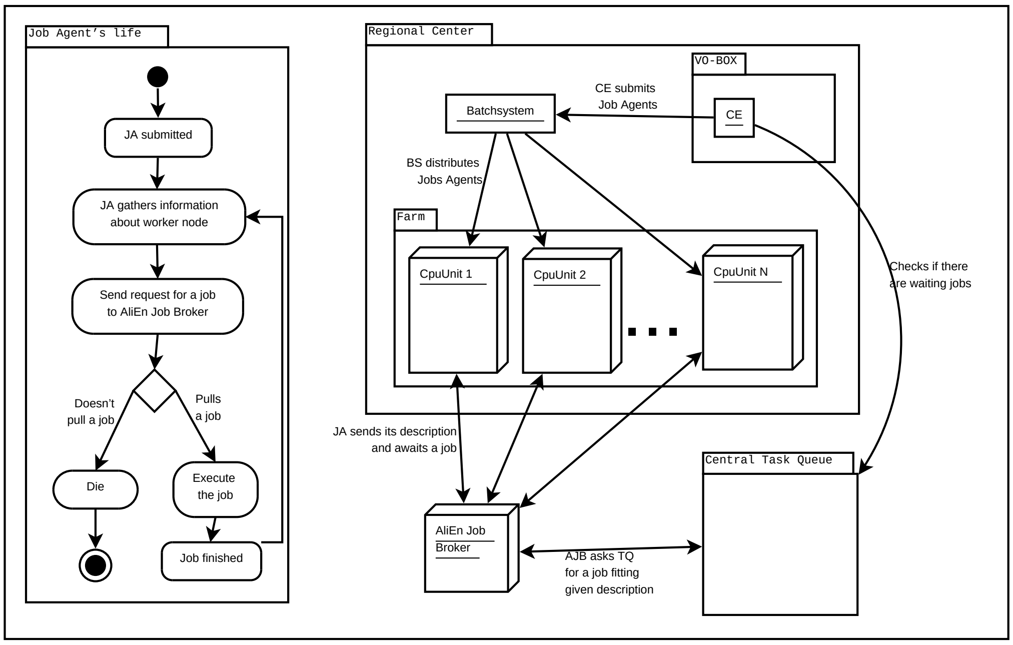
Job Agent's life
JA submitted
JA gathers information about worker node
Send request for a job to AliEn Job Broker
Doesn't pull a job
Pulls a job
Die
Execute the job
Job finished
Regional Center
VO-BOX
CE
CE submits Job Agents
Batchsystem
BS distributes Jobs Agents
Farm
CpuUnit 1
CpuUnit 2
CpuUnit N
Checks if there are waiting jobs
JA sends its description and awaits a job
Central Task Queue
AliEn Job Broker
AJB asks TQ for a job fitting given description

Figure 2: Simplified JobAgent model for the purpose of my simulations.

3.6 Job Status

One of important aspects of a job in AliEn is its status. When a job is created by a user and submitted to the Central Task Queue it is assigned the status INSERTED and if everything goes well continues to WAITING. Then if the AliEn Job Broker selects this job for a JobAgent, the job's status changes to ASSIGNED. The JobAgent then fetches the job from the Central Task Queue and attempts to run it - the job's status continues to STARTED and, if execution proceeds normally, to RUNNING. The job then does its work and when finished changes its status to SAVING, if the saving process completes normally, the next status is SAVED. Then there is a process of a validation of the results which must finish successfully to bring the job to the final status - DONE. If any of the status transitions fails (for example the job gets assigned to a JobAgent, but the JobAgent does not fetch it), the job ends with an error status (there are multiple error statuses in AliEn, cf. Figure 3).

It also happens that a job stops the communication with the AliEn central services. Its status is then changed to ZOMBIE after some time and if it does not recover within a given time limit, it gets EXPIRED or KILLED.See Figure 3 for the discussed schema.

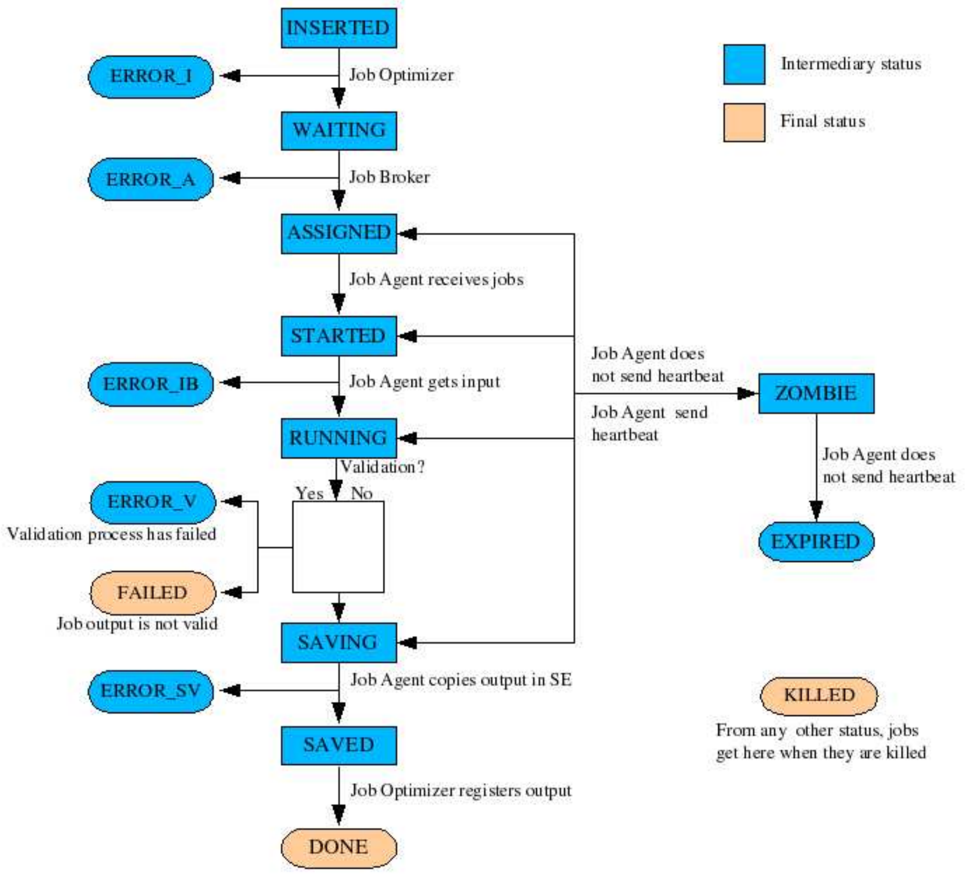

Figure 3: Possible status for AliEn jobs.

4 MONARC2

MONARC (MOdels of Networked Analysis at Regional Centers) is a tool developed with the aim to provide realistic simulations of large distributed computing systems. The simulation framework is not intended as a detailed simulator for basic components, but it aims at correct description of the performance and limitations based on realistic mathematical models. MONARC is based on Java technology with a built-in multi-thread support for concurrent processing. It offers an Interrupt-driven scheme for discrete event simulation, which is well suited to describe concurrent running programs, network traffic and other concurrent running tasks sharing resources, that are common in the HEP computing projects. It has a complex GUI to simulation engine which allows dynamic change of parameters and monitoring of simulation results.

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.

MONARC2 is an extended version of the original MONARC simulator. The diagram representing the most important classes of MONARC2 is shown in Figure 4.

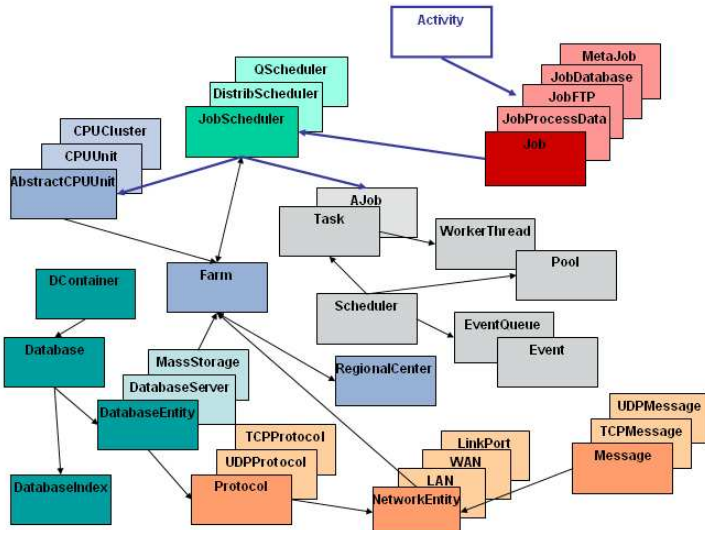

Figure 4: Most important classes of MONARC2

4.1 Simulation Engine

The simulation engine of MONARC2 uses threaded objects (Active Objects) to simulate time dependent processes, which concurrently compete for shared resources. There is a base class Task, from which all entities that implement a time dependent behavior – such as the running jobs, the database servers or the network entities – must inherit. 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, it gets assigned 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 mediated by a Scheduler using events defined in Event class. The Scheduler is managing and monitoring the whole simulation with the help of several data structures:

- 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 or Finished state

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 Modelling

Jobs in MONARC2 are described by a base class called Job, which defines a job that does nothing. The class must be extended to describe an active job. The class however contains a number of methods that can be used to define a job's behavior. For example, the class has the following methods:

send/waitFor Message: the method used to communicate via CPU is linkport with any other job or service on the network

get/upload Data: the method for moving data from/to the database server

processOnCPU: this method simulates processing on the CPU; the necessary processing time of the job is given by the parameter processingTime

The Job class defines many fields (attributes) like the running and scheduling priorities, the initial and final time, the required memory. Not all of the mentioned attributes are necessary for correct functioning of a particular simulated system.

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 the tag TAG_START_JOB and the job itself is appended as auxiliary data. The active job then executes the 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

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

Where *job.processingTime* is expected processing time on the CPU with power *job.cpuPower*. *job.allocatedCPU* is the CPU power allocated to the job when running on assigned Worker Node.

Every time a job is added or finishes on a CPU, all the other jobs on the same CPU receive an event with tag TAG_CPU_CHANGED and their execution times are recalculated (see Figure 5). Also, jobs on the same CPU can have different priorities. If N jobs share the same CPU, then cpuPower assigned to the job i is calculated as

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

Where P_i is priority of i-th job.

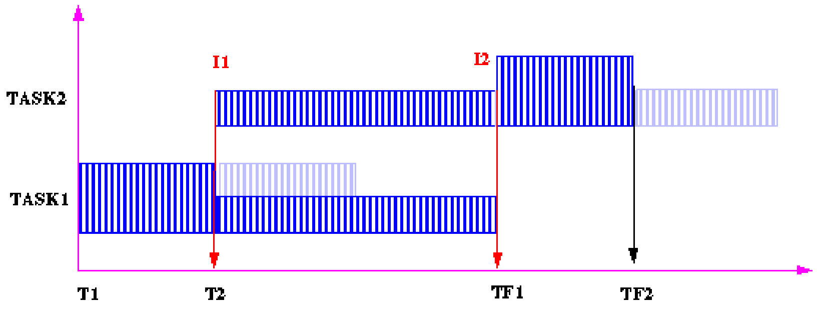

Figure 5: Jobs execution time is recalculated every time another job enters or leaves the same CPU

4.2.2 Job Implementations

The run() method is the one that should be overridden by the class extending job. Using methods mentioned above, some simple types of job are already implemented. These include:

JobProcessData - simulates the processing of specified data

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

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 submit jobs to it. Each Activity contains a vector of jobs and Double objects, which serve as the 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 the object is a job, submits it
- if the 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 is also possible for job scheduler to send a job to another regional center. There is the jobScheduler class, which acts as an implementation of the local batch system. If custom batch system behavior is needed, this class should be extended. Main configuration file accepts the name of the new class as the job_scheduler parameter in the global section.

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. Both are subclasses of the class called DatabaseEntity, which should be used in case of the definition of another storage device. All the database entities have linkports and the interaction between jobs and those entities is done through the network. For simple requests, that do not have impact on the bandwidth share on the network, it is better to use task event to communicate.

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

4.3.1 Data Container

Data container is the actual ”data holder”, so it has defined attributes like name, size (in MB), type of data it holds and database it belongs to, and also attributes saying how many times it was accessed, the last time it was accessed, whether it is locked (meaning it has not been entirely written yet) and where it resides - on disk, tape or transfer.

4.3.2 Database

Database as the attribute is not essentially needed in the simulation, but can help in ordering data. In the simulation it is represented by an object holding hash table of all the containers contained in it.

4.3.3 Database server

Database server is an extension of DatabaseEntity class, implementing general behavior of the hard-disk-based database server. It is capable of writing, reading and getting (read and remove) data containers. A few of its attributes are total space, used space and read/write speed/latency. Database servers have also implemented the capability to move the least used data containers to the closest usable mass storage system.

4.3.4 Mass Storage Center

Mass storage system operates in similar manner as database server, but has some additional attributes like number of silos and tapes per silo, mount time, tape size and number of drives. These are used to correctly describe the MSSs functions.

4.3.5 Database index

Database index is used to find any data within the simulation model. As already mentioned, it is a global object held by the static class MainSim. Information is stored in three hashtables:

mapAddress - the hashtable mapping the dbEntities vs. addresses

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

mapDatabases - the hashtable mapping the dbEntities vs. databases

4.4 Network Model

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, the 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).

Network in the simulation is represented by several entities: Linkport, LAN, WAN and Router.

4.4.1 Linkport

Every entity involved in the simulation has a network linkport, which is an entity that receives and sends messages. Linkport is defined by a unique IP address and by the maximum bandwidth in Mbps. It must also belong to a regional center - the name of the regional center is a required parameter of the constructor. Linkports are connected to LANs.

4.4.2 LAN

Function of LAN - the abbreviation for Local Area Network - is self-explanatory. It is described by a unique name and the maximum speed. It also must belong to a regional center as required by the constructor. LAN in the simulation keeps trace of all the messages passing through it and sets the correct speed for them. LANs are connected to WANs.

4.4.3 WAN

The function of WAN - abbreviation for the Wide Area Network - is also self-explanatory. Its definition and function is practically identical to LAN, but on the higher level. WANs are connected to Routers.

4.4.4 Router

Routers in the simulation serve as connection points for WANs. They are defined by a unique name and latency. The simulation supports three different routing mechanisms:

EstimAvailableBandwidth: the route with the greatest available bandwidth will be chosen

EstimUsedSpeed: the route with the smallest used speed will be chosen

EstimNumMessages: the route loaded with the fewest messages will be chosen

4.5 Using MONARC2

Using MONARC2 can be (from my point of view) divided into three parts - writing configuration files, extending classes and defining the output.

4.5.1 Configuration files

There is one main configuration file. This file contains the general simulation settings (for example which classes should be used in the simulation or which algorithm should be used for the network routing) and then the definition of the resource layout - including CPU clusters, Storage Elements, networks and other possible parameters. An example is shown of a simple configuration file with two CPU clusters with VOBOX, one of them with Storage Element connected through a router.

```
[global]
queue_type = vector
max_simultaneous_threads = -1
routing_type = default
load_estimator = network.AvailableBandwidth
resource_broker_address = 10.7.2.1
regional0 = CpuCluster1
regional1 = CpuCluster2
scale = 0.25

[CpuCluster1]
latitude = 15.1
longitude = 14.5
initial_pool_size = 100
lan0 = LAN1
lan0_max_speed = 1000.0
lan0_connect = WAN1
wan0 = WAN1
wan0_max_speed = 1000.0
wan0_connect0 = Router
router0 = Router
router0_latency = 0
cpu_unit0 = CPU1
database_server0 = SE1
activity0 = Activity
vobox_LAN = LAN1
vobox_linkport_address = 10.1.1.1
vobox_linkport_speed = 100.0
vobox_ce_port = 20
vobox_ce_check_interval = 600

[CPU1]
from = 0
to = 124
cpu_power = 16000.0
memory = 32000.0
cores_per_cpu = 8
page_size = -1.0
link_node = 10.1.0.1
link_node_max_speed = 1000.0
link_node_connect = LAN1

[SE1]
read_speed = 3000.0
write_speed = 1500.0
read_latency = 0.003
write_latency = 0.003
```

```
disk_size = 2000000000000.0
address = 10.1.2.1
link_port_speed = 450.0
lan_to_connect = LAN1

[CpuCluster2]
latitude = 15.1
longitude = 14.5
initial_pool_size = 100
lan0 = LAN2
lan0_max_speed = 1000.0
lan0_connect = WAN2
wan0 = WAN2
wan0_max_speed = 1000.0
wan0_connect0 = Router
cpu_unit0 = CPU2
vobox_LAN = LAN2
vobox_linkport_address = 10.2.1.1
vobox_linkport_speed = 100.0
vobox_ce_port = 20
vobox_ce_check_interval = 600

[CPU2]
from = 0
to = 62
cpu_power = 16000.0
memory = 32000.0
cores_per_cpu = 8
page_size = -1.0
link_node = 10.2.0.1
link_node_max_speed = 1000.0
link_node_connect = LAN2

[Activity]
class_name = Activity
```

A note regarding units: in general, whatever units can be chosen, but the same ones must be used throughout the whole simulation. In this manner the units for the network speed were chosen by authors to be Mbit/s as these units are used in the output classes to label the axis. The memory and disk sizes are supposed to be in MB. For CPU power however the units were not specified, so the user must decide upon them on his/her own (and use the same units when defining jobs).

With additional classes I introduced in the code I also created another configuration file called other.conf (the name of the main configuration file can be defined by the user as it is given to the simulation as an argument). This file contains the setting for JobAgents, CentralTaskQueue, ComputingElement and for events used for the communication substituting the network functionality. An example follows:

```
[job_agent]
connection_time_out = 600
```

```
delay_gather_info = 0.1
scheduling_priority = 10.0
running_priority = 10.0
cpuUtilization = 0.15
lifeTime = 200000
minLifeTimeToAsk = 0
maxJobs = -1
timeToAbortJob = 1800
dataLocation = ANYWHERE

[computing_element]
connection_time_out = 600
check_interval = 600

[event]
travel_time = 0.1

[task_queue]
delay_find_job = 0.1
maxTimeAssigned = 900
maxTimeRunning = 150000
maxConcurentSearches = 8
```

4.5.2 Extending classes

The user has at his/her disposal many implemented classes, that can be used right away. There is however one class, which the user must extend in order to simulate anything - the Activity class. This class is used to submit work to its center and the user must tailor this class to accomplish the desired functionality. The following code is a simple example of the pushJobs() method, which is practically the only one that needs to be overridden to extend the Activity class. It submits a *numJobs* of 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 - the user can have as many configuration files as he needs.

```
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();
    }
```

As mentioned above many other classes can be extended or replaced to get the wanted behavior.

4.5.3 Defining output

The output is defined in the form of an XML file. Many output formats are available, including text and graphics. The graphic format is in a form of graphs and has an advantage of being immediately usable, but lacks precision as it is averaged on the fly. The text format on the other hand needs some processing to actually see the results, but is much better for data keeping, more precise simulations and an eventual debugging.

Practically everything that goes on in the simulation can be monitored using clients. Clients are extensions of the SimClient class resp. the FileClientI interface and each of them is run by its own thread. They connect to the monitored object and wait for it to report changes, which are then logged.

The most useful clients are already implemented. If the user wants a special monitoring, he/she has to extend the appropriate class. Monitoring can also be real-time, which is useful when one needs immediate results.

The following code is an example of the graphics output definition. In this code, the monitored entity is the whole computing farm and 3 clients are used as parameters. These monitor the CPU utilization, the total number running and waiting jobs and the 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="Prague" cluster="farm" class="monarc.output.parameters.OutputParameters">
        <parameter client="out1"/>
        <parameter client="out2"/>
        <parameter client="out3"/>
</declare>
```

To present an example of the text output definition, I show the following output which was defined using the FileComplex class. This logs every information it is given in text files sorted in the hierarchy of directories that matches the objects hierarchy in the simulation. This client can thus be used to monitor anything:

```
<client name="client1" class="monarc.output.FileClient.FileComplex.FileClient" save="."/>
<client name="client2" class="monarc.output.FileClient.FileComplex.FileClient" save="."/>

<declare farm="Prague" cluster="farm" class="monarc.output.parameters.OutputParameters">
    <parameter client="client1"/>
</declare>

<declare farm="Prague" cluster="linkport" node="10.1.2.1"
        class="monarc.output.parameters.OutputParameters">
    <parameter client="client2"/>
</declare>
```

5 My Changes to MONARC2

In order for the simulation to reflect ALICE Computing model, I had to implement several new classes and introduce them into the MONARC2 package. These were VOBOX, (AliEn) ComputingElement, JobAgent, (AliEn) ResourceBroker and CentralTaskQueue. I also modified some of the existing classes, for example CPUUnit or JobScheduler. I also introduced a Scale factor to deal with performance issues.

MONARC2 package offers a friendly way to use custom classes. If you want to use your own CPUUnit, you can set it in the configuration file. I was however forced to do some changes directly in the package, to implement VOBOX, Resource Broker and Central Task Queue, so I decided it would be simpler if I modified them directly.

I also slightly modified some classes of the network package.

5.1 VOBOX

VOBOX, as mentioned above, is a server running some specific AliEn services. For the purpose of my simulation it was only necessary to have it running the AliEn Computing Element. My implementation of the VOBOX into the MONARC2 package is represented by a simple class containing ComputingElement and a linkport.

5.2 ComputingElement

ComputingElement (CE) class is responsible for submitting JobAgents to the local batch system. This is a simplification - in reality the procedure has two steps: 1. the submission of a JobAgent to the WLCG Computing Element installed at the considered site and 2. the passing of the JobAgent by the local WLCG CE into the local batch queue.

Since CE needs to perform some actions (unlike VOBOX, which in the simulation does nothing) it is not derived from the class Object (a base java class) but from the class Task. It is periodically checking the local batch system for and if the number of waiting JobAgents drops below the maximum amount of running jobs at the regional center, the CE starts submitting new JobAgents. It does not submit more JobAgents than the number of jobs waiting in the Central Task Queue. If there are no more jobs in the Central Task Queue, it ends.

5.3 ResourceBroker and CentralTaskQueue

The ResourceBroker is responsible for matching requests from JobAgents with the jobs waiting in the Central Task Queue. Since the Resource Broker is also a central service as well as the Central Task Queue, I merged them - the ResourceBroker class contains a Vector taskQueue which holds waiting jobs.

It is also derived from the Task class. The ResourceBroker communicates with Activities, to accept jobs (this is due to the merging with the Central Task Queue), and JobAgents. Communication is implemented using events instead of the network messages. This is mainly a performance issue. The more ongoing transfers are in the system, the slower the simulation is. Thus only transfers that are important for simulation results or transfers whose influence is not negligible are implemented using network. ResourceBroker ends when there are no jobs in the Central Task Queue (i.e. in the vector taskQueue) and no jobs are incoming (no future events with incoming jobs are scheduled in the simulation scheduler).

5.4 JobAgent

JobAgents in the simulation behave almost in the same way as in reality, except that in reality the AliEn Resource Broker returns info about the assigned job and the JobAgent fetches the payload (assigned job) directly from the Central Task Queue. In the simulation the payload is returned from the ResourceBroker.

The JobAgent class is derived from the Job class, so for the local batch system it is just another job.

To trace the progress of jobs in the simulation I implemented the field jobStatus. The simulation is monitoring the times when a status of a job changes. A job can obtain all of the states as in reality except for INSERTED (after the job is received by the Central Task Queue, it proceeds directly to the status WAITING). One extra status FIRST_MATCHED was introduced. Job enters this status while waiting in the Central Task Queue when it is first matched against a request from a JobAgent. When it is not necessary to simulate user submission time, it is simpler to submit all jobs at once. This makes it hard to compare waiting times in the queue. In these cases the FIRST_MATCHED status can help.

5.5 CPUUnit

The original CPUUnit class implements 1 core CPU. I rewrote it to implement a multi-core CPU, with load sharing resembling reality. I also rewrote some other methods of the original CPUUnit class.

5.6 Job Scheduler

The reason to modify this class was mainly to make it aware of payloads being run by JobAgents (JA). In the simulation the JAs are submitted in the same manner as normal jobs, because the JobScheduler class takes care of assigning an ajob (the class from the engine package derived from the Task class, thus owning a thread which is used to execute the job) and doing other things, so it was simpler this way. I also rewrote some other methods to comply with the needs of the simulation.

5.7 Scale factor

I hit some performance issues while simulating larger systems. In my case the simulation slows down exponentially with the growing number of network messages. To overcome this problem I implemented a Scale factor. It is set in the main configuration file and using it, MONARC2 automatically scales down the defined model - number of CPUUnits, speeds of network elements, number of jobs and so on. Most results are then exactly the same as they would be using the original setup. For example number of finished jobs will be different and because the Scale factor is not used to modify the results, it must be up-scaled back by the user.

Since the slowdown caused by the growing number of network messages is exponential, using the Scale factor leads to a better statistics with the same simulation time.

5.8 Network package

There is one missing feature in the MONARC2 network package: it does not offer a way to set an external load on a network entity. Usually computing centers or experiments do not have dedicated connections between them. Sharing with others has important impact on the bandwidth they have available - the more connections you have the bigger share you get. As MONARC2 does not offer this, there are two ways (I could think about) to accomplish this effect in the simulation.

You can create extra network entities, connect them to an existing considered network entity and transfer ”real” messages over it, thus creating an extra load on this part of the network. But as the model gets more complicated, it gets slower, mainly because of the increasing number of network messages. This method would thus have a negative effect on the pace of the simulation.

The Second option is to modify the source code and implement the feature yourself. This is, however, quite tricky as the source code of the network package is not publicly available. Luckily the classes implementing network entities can be extended and then used in the simulation instead of the original ones. One such way - the one I used - is to override methods for adding and removing messages from a network entity and recalculate total speed before the original the method is called, so that the speed of the entity will be set to the share you should have according to the ratio of yours-to-all connections.

Unfortunately, this has a flaw. Imagine a situation when you have for example 1 Gbit/s link with 600 external connections. Suppose you have 400 connections using this link, so in the simulation you are given (meaning: speed of link is changed to) 400 Mbit/s. Now what if 200 of your connections have bottlenecks elsewhere and are able to go maximally 1 kbit/s maximum each. This leaves the rest of your messages with 200 Mbit/s extra bandwidth which would in reality be distributed over all capable messages - even the external ones. This flaw can lead to higher transfer speeds of some messages, if this feature is needed. If this workaround is not activated, it has no effect on the simulation at all.

6 Simulation of ALICE Tier-2 Centers

The objective of this work was to use and subsequently to modify the MONARC2 tool, so that it could be utilized to simulate a part of the ALICE computing infrastructure.

6.1 Goals of Simulation

One goal of the presented simulations was to find the effect of different conditions for the location of data processed by the jobs at a considered regional center on the processing efficiency. The regional centers considered were Tier-2s which are dedicated to processing of simulation and analysis jobs.

The most recent policy in the workload management in the ALICE Grid system is, that the jobs are sent where their data is. This gives the network traffic a break from what would arise from jobs downloading their data from distant Storage Elements. On the other hand, this prolongs the time the jobs spend in the Central Task Queue waiting for specific free slots. There has been some arguments that today's networks are capable of handling the situation without this constraint and that the average overall time to finish jobs would be actually shorter.

6.2 Parameters of Centers

The simulated system resembles a real situation. Since I have some experience with working at the Prague Tier-2 center, the computing farm Golias [9], and I also had the access to the logs from the local batch system, which is very important for the evaluation of the behavior of the processed jobs, I considered a system of the Prague site plus 5 other close sites. For the resource layout see Figure 6. Each center is composed of a number of Worker Nodes (WN), a Storage Element (SE) and a VOBOX. The information on given computer centers and networks connecting them was extracted from MonALISA and simplified for the purposes of this simulation. Each center has a unique configuration. I will write it down for Golias only and the configuration files for the other centers I can provide on demand.

6.2.1 Worker Nodes

For ALICE, Golias has 125 worker nodes, each with 8 cores of the power 2000 and 32GB of memory. Every worker node has a linkport (which is practically a network card) with the speed 1 Gbit/s and is connected to

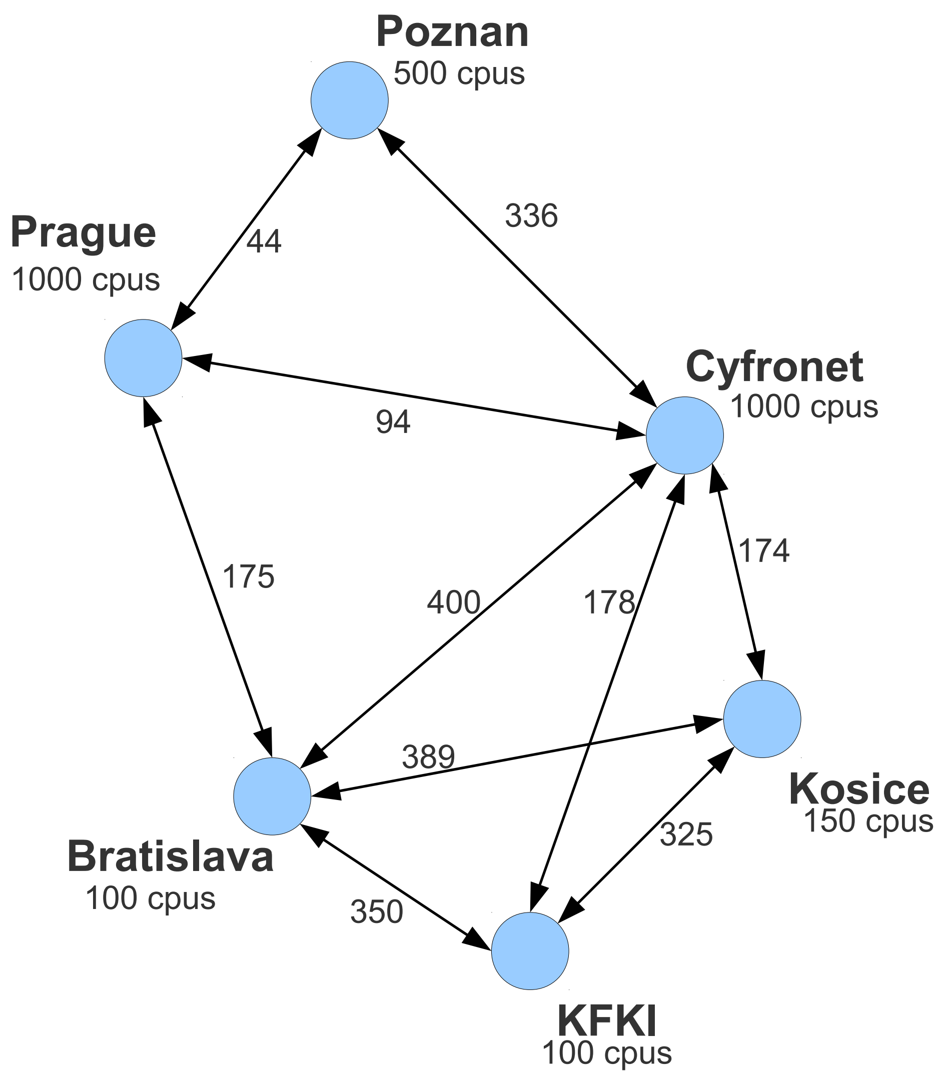

Figure 6: The layout of 6 centers from central Europe. Speeds of links are given in Mbit/s.

the Local Area Network (LAN). There are no failures of worker nodes nor of individual cpu cores nor of memory.

6.2.2 Storage Element

Golias has 3 SEs with excessive capacity about 2 EB - because the space consumption is of no importance in this simulation. The same goes for the read and write speed which are 3 Gb/s and 1.5 Gb/s, respectively. The limiting factor then is the network connection, which is 450 Mbit/s for each SE. Storage elements are also connected to LAN and do not fail.

6.2.3 Network

The top level of network can be seen from Figure 6. Each center has a Gbit LAN connected to its own router. The external load of 100 messages is put on each LAN. The network also does not fail.

6.2.4 Jobs

As mentioned earlier, there are 2 types of jobs in the simulation. One of them is the simulation job, also often called production job, which only needs to download the input data once upon the start and then uploads to an SE the output data when finishing. These jobs are from obvious reasons called CPU intensive. In my simulation, the download of the input data for the production jobs was neglected. The distribution of the processing time of jobs on a CPU with the power 2000 was calculated as follows:

$$ProcessingTime = (3500 * randomNormal + 18000) seconds$$

The second type of a job is the analysis job. These jobs process streamed data from a storage element and at the end upload results. Unfortunately it is quite difficult to obtain a good profile of an analysis job. I was able to obtain some approximate data from the local batch system of the computing center Golias, but these apply for JobAgents that can represent several jobs, so the profile I have is quite a wild estimate, which however was adequate enough for the purpose of this simulation. The distribution of the processing time per MB of data to analyze on a cpu with the power 2000 follows:

$$ProcessingTimePerMB = (4 * randomNormal + 5) seconds$$

$$AmountOfDataToAnalyse = (6400 * e^{-0.0015*randomInteger(5000)} + 10)MB$$

The analysis job then uploads its results of the size of square root of the size of the analyzed data. The memory requirements of both types of jobs are not important.

Implementation The implementation was done in the way, that there is only one data type in the simulation called 'uniform' which represents a file of the size 10 MB. When a job is created, it is decided how many and which files will the job analyze. When the job is executed, it downloads all the files. Then it calculates how much time it needs to process all the data. If the time is less or equal to the time it took to download the data, the job ends and the efficiency is calculated as

$$job\_efficiency = time\_to\_process/time\_to\_download$$

Otherwise the efficiency is 1 and the job continues until the total running time is equal to the time needed to process the data.

For the simulation it looks like the job was subsequently downloading 10 MB files, than sometimes it waits a bit and then ends. Thus, for one job there are at least two transfers (i.e. network messages) – it has to download at least 1 input file and upload results.

6.3 Simulated situations

There are two main simulations. The first where the jobs are sent only to the center that has their data on its SE (labeled 'local'). In the simulations of the second type (labeled 'anywhere') jobs are sent to the first free job slot. The 'anywhere' one was done with 20,50 and 75% of analysis jobs. The 'local' one with only 20% and served as reference point.
Measured variables (plots and averages):

I have performed two main simulations. In the first type of the simulation, the jobs were sent only to the center that has their data on its SE (labeled 'local'). In the simulations of the second type (labeled 'anywhere'), the jobs were sent to the first free job slot. The 'anywhere' simulations were done for 20, 50 and 75% of analysis jobs of all the concurrently running jobs. The 'local' simulation was done with only 20% of the analysis jobs and served as a reference point. The measured variables (plots and averages) were as follows:

- FIRST_MATCHED $\rightarrow$ ASSIGNED

- RUNNING → SAVING
- Transfer speed
- Job efficiency (CPU time/wall time)

Please note, that since the production jobs in the simulation did not download any data, all the results are given for **analysis jobs only**.

6.4 Results

Here I will present a set of figures and a short description of individual measurements.

6.4.1 Jobs using only local data (the 'local' simulation), 20% of jobs are analysis

Figure 7 shows the waiting time in the Central Task Queue from the moment the job was matched against a request from a JobAgent for the first time. In average, jobs waited 1.46 minutes. Figure 8 shows the time it took the jobs to finish their work - stream and process the data. The average working time was 1.84 hours. Figure 9 shows the transfer speed of network messages. The average transfer speed was 400 kB/s. Finally, Figure 10 shows the job efficiency with the average of 69%.

I will skip these comments for the simulations with jobs using data stored 'anywhere' and only present figures with the explanation in the figures captions. Also the FIRST_MATCHED → ASSIGNED figure will not be shown. In fact the only condition for matchmaking jobs was the data location which in this case was not 'local', but 'anywhere', so consequently this value was for all the jobs 0.

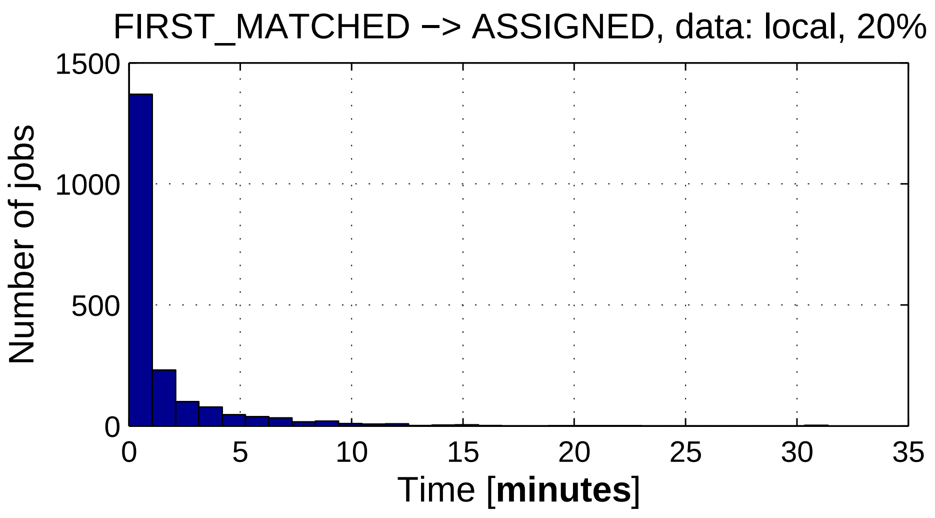

Figure 7: The FIRST_MATCHED → ASSIGNED plot for 'local' simulation with 20% of analysis jobs. Average is 1.46 minutes.

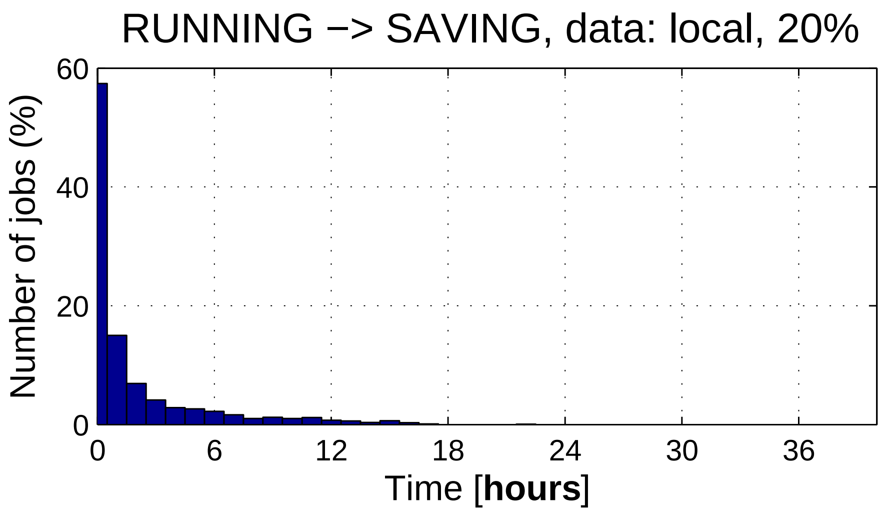

Figure 8: The RUNNING → SAVING plot for 'local' simulation with 20% of analysis jobs. Average is 1.84 hours.

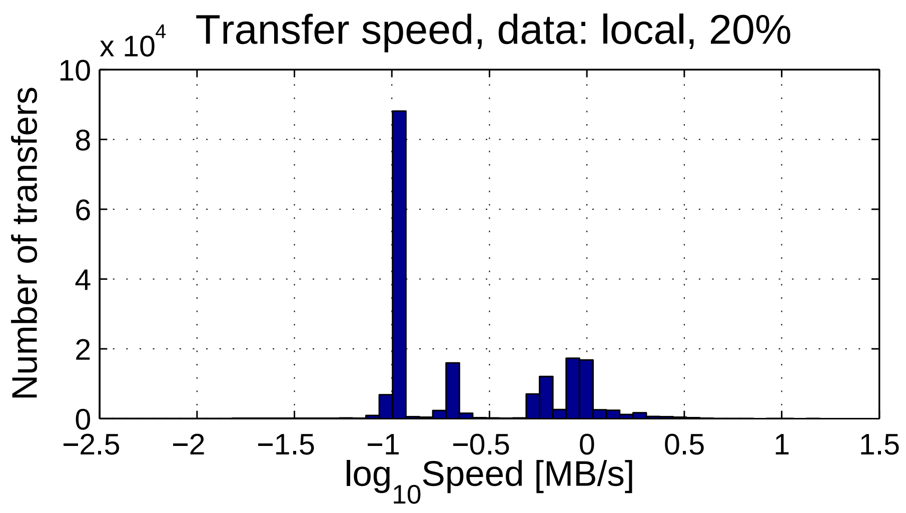

Figure 9: The transfer speed of network messages for 'local' simulation with 20% of analysis jobs. Average is 0.40 MB/s.

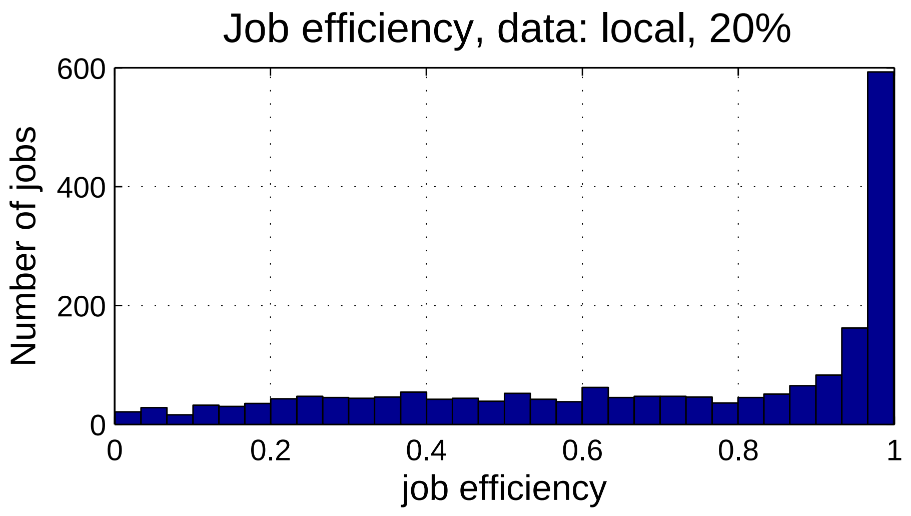

Figure 10: The job efficiency (cpu_time/wall_time) for 'local' simulation with 20% of analysis jobs. Average is 69%.

6.4.2 Jobs using data stored anywhere (the 'anywhere' simulation), 20% of jobs are analysis

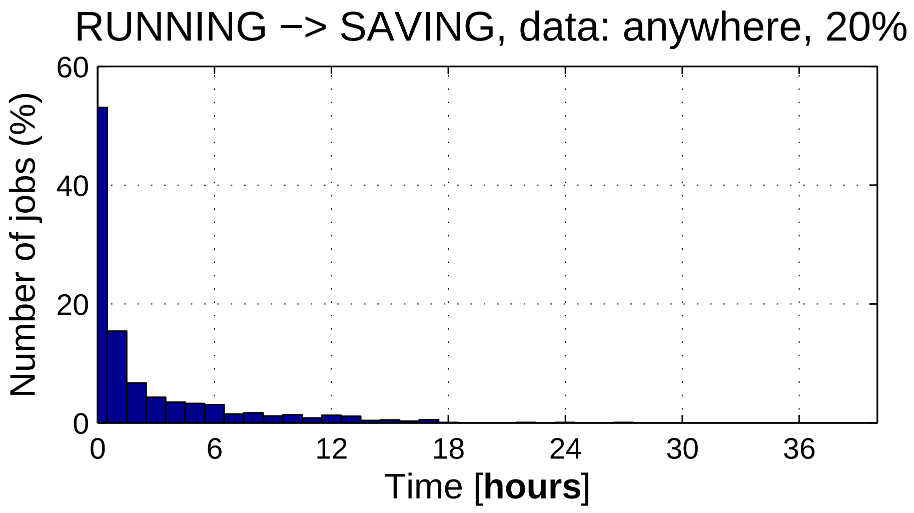

Figure 11: The RUNNING → SAVING plot for 'anywhere' simulation with 20% of analysis jobs. Average is 2.16 hours.

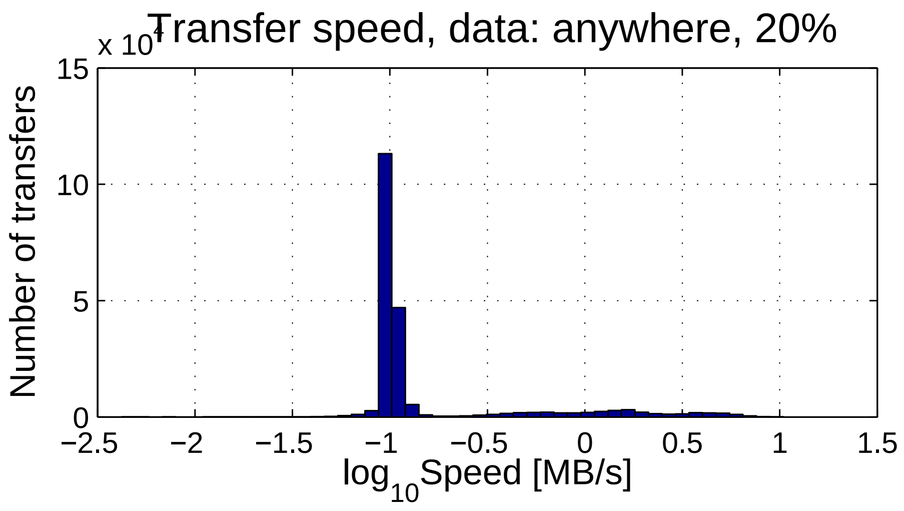

Figure 12: The transfer speed of network messages for 'anywhere' simulation with 20% of analysis jobs. Average is 0.42 MB/s.

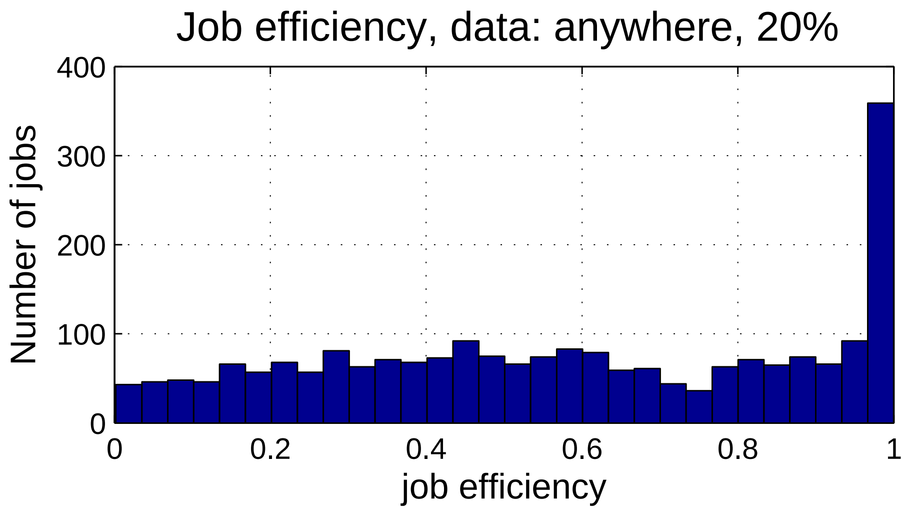

Figure 13: The job efficiency (cpu_time/wall_time) for 'anywhere' simulation with 20% of analysis jobs. Average is 58%.

6.4.3 Anywhere simulation, 50% of jobs are analysis

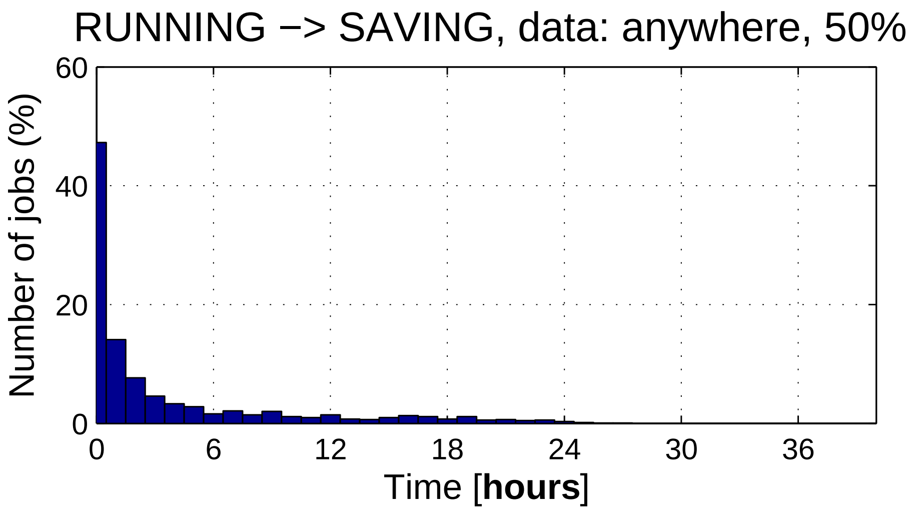

Figure 14: The RUNNING → SAVING plot for 'anywhere' simulation with 50% of analysis jobs. Average is 3.52 hours.

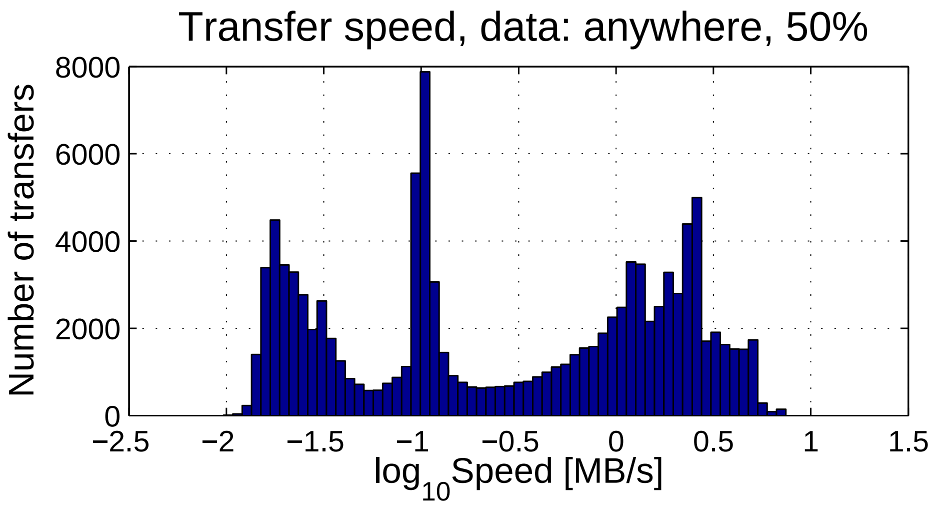

Figure 15: The transfer speed of network messages for 'anywhere' simulation with 50% of analysis jobs. Average is 0.99 MB/s.

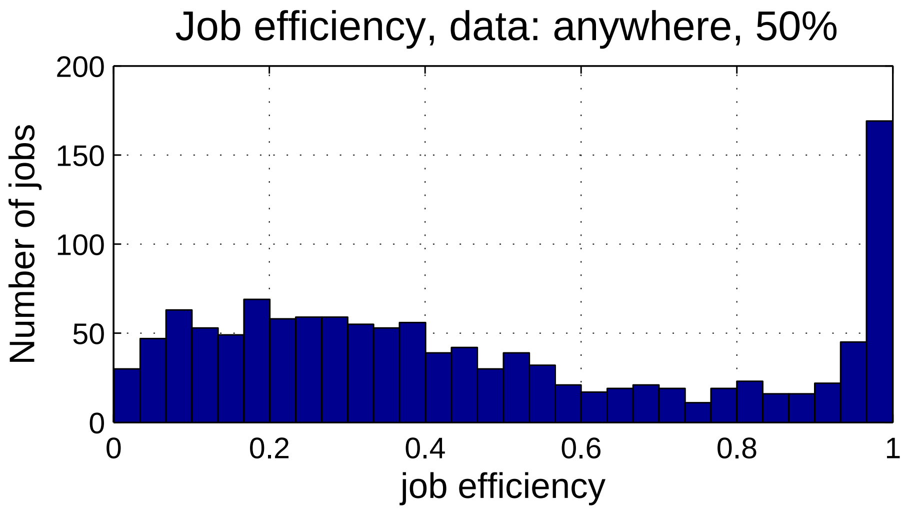

Figure 16: The job efficiency (cpu_time/wall_time) for 'anywhere' simulation with 50% of analysis jobs. Average is 47%.

6.4.4 Anywhere simulation, 75% of jobs are analysis

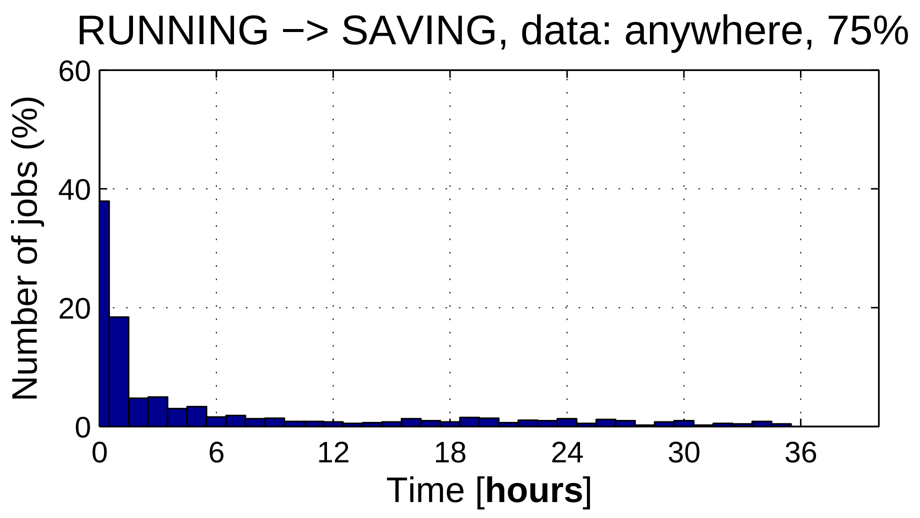

Figure 17: The RUNNING → SAVING plot for 'anywhere' simulation with 75% of analysis jobs. Average is 5.88 hours.

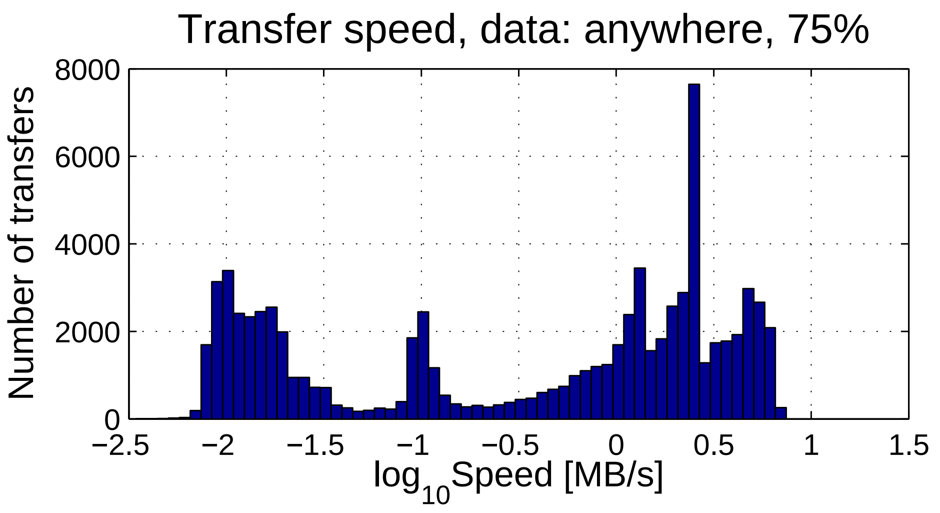

Figure 18: The transfer speed of network messages for 'anywhere' simulation with 75% of analysis jobs. Average is 1.49 MB/s.

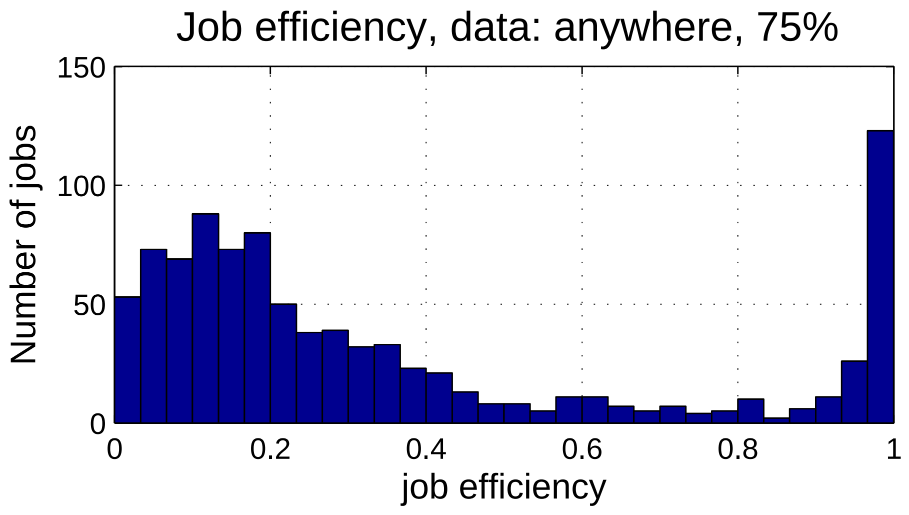

Figure 19: The job efficiency (cpu_time/wall_time) for 'anywhere' simulation with 75% of analysis jobs. Average is 36%.

6.4.5 Results overview

All the averaged results can be seen in Table 2.

Average	Local, 20%	Any, 20%	Any, 50%	Any, 75%
FM $\rightarrow$ ASSGN [minutes]	1.46	0	0	0
RUN $\rightarrow$ SAV [hours]	1.84	2.16	3.52	5.88
Efficiency	69%	58%	47%	36%
Transfer speed [MB/s]	0.40	0.42	0.99	1.49

Table 2: Overview of the simulation results

6.5 Discussion

The simulation I have performed using MONARC2 predicts, that, in this case, using data stored at distant Storage Elements and adding on the percentage of analysis jobs running at the site:

- the average efficiency of job processing will go down
- the average job processing time will grow several times
- the average transfer speed will grow

If there was not the last observation (see the last row in the Table 2), the results would match general expectations. However it is unexpected that the average speed is rising.

When you look at Figure 18 it can be virtually separated at the value -0.5 in two parts. The transfers on the left are very slow and they are effectively blocking the corresponding jobs. This leaves quite a lot of bandwidth for other transfers which can be seen on the right side of the figure. These transfers are boosted so their corresponding jobs get processed much faster. For one slow transfer there can be hundreds of fast ones. This is the main reason, for the rising transfer speed. The 'external load modification' flaw (see Section 5.8) also adds on this effect.

The effect of slow against fast transfers is not reflected in the job efficiency nor the job duration. This is due to the fact that one job can represent many transfers (see Section 6.2.4). Some of them might be slow, some of them are fast. When one job downloads 1000 files (10 MB each) at 10 MB/s and finishes in 15 minutes and another job downloads 100 files at 0.01 MB/s (28 hours), the average duration will be around 14 hours, but the average transfer speed will be 9 MB/s. Also, in the case of job efficiency, even if a job gets the data instantly, the efficiency cannot be higher than 100%.

7 Benchmarking MONARC2

In MONARC2, the network is handled by classes from the network package. It works well for small systems, but as the network environment becomes more complicated simulation slows down exponentially. Since the network package of MONARC2 is a part of the simulator core, for which the source code is not publicly available, I can only guess what is behind this slowdown. It is possible that algorithm used to recalculate the speed of individual messages creates in more complicated situations cascades of recalculations - each message could trigger recalculation of all other messages that share some part of the way. Even thousands of changes of 'used bandwidth' were reported with a single time stamp in the output of a single linkport.

With the simulations from Section 6 I touched the limits of the MONARC2 package. Using the same level of detail for more complicated models would require an enormous amounts of time - my model was scaled down 4 times and still required several days to finish. The bottleneck seems to be the network package of MONARC2, which has difficulties with managing many messages coming from many different sources. This is why I did a benchmarking of MONARC2 to evaluate its limits.

7.1 Benchmark simulations

First I should make clear, that these simulations were done with the original MONARC2 package (v2.1.12).

I used a symmetrical model of 6 identical regional centers connected in a ring topology with 1 Gbit/s links (see Figure 20). Each center had:

- 501 CPUs per center
  - each CPU had a 1 Gbit/s linkport
- 1 SE per center with a 2 Gbit linkport
- the memory and processing power were not important

Using this model I did 3 different simulations:

Simulation 1. (CPU): only processing jobs (processOnCpu method), there were no network transfers in this simulation

Simulation 2. (local): All jobs were "download" jobs, they downloaded some data from the **local** Storage Element (SE) and ended - there was no traffic between the regional centers

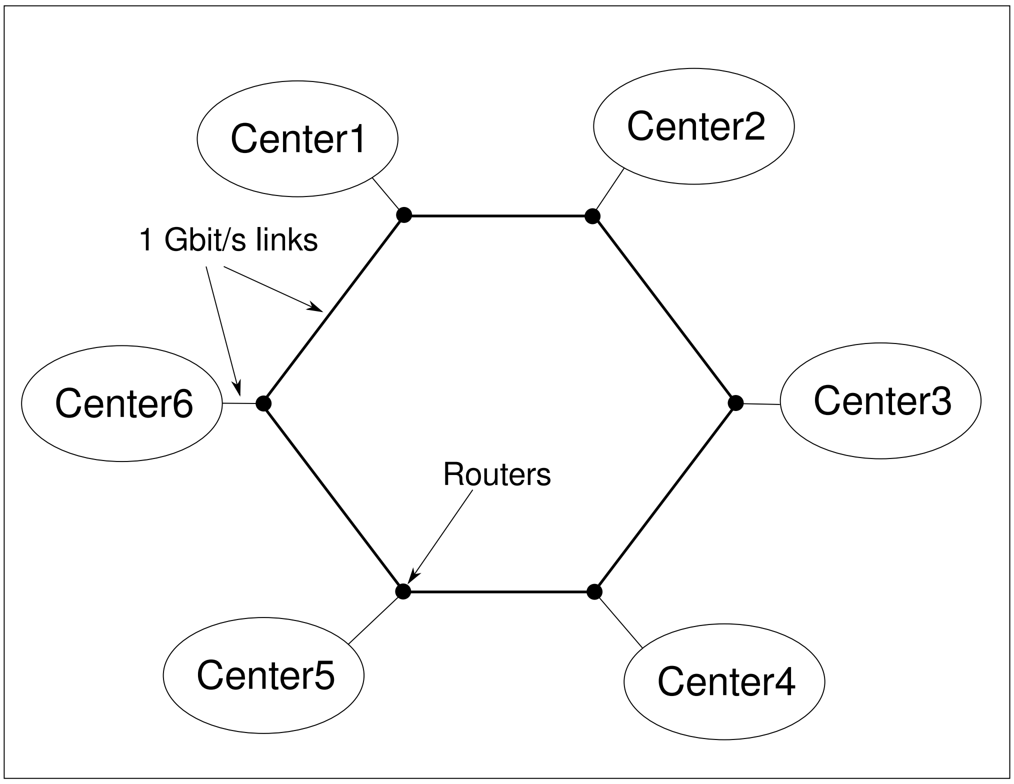

Figure 20: The symmetrical layout of 6 regional centers connected in the ring topology with 1 Gbit/s links.

Simulation 3. (global): All jobs were ”download” jobs, they downloaded some data from a **random** SE and ended

10 jobs per second were submitted to each regional center.

7.2 Specifications of the machine used for the described simulations

CPU: Intel Xeon 5130, 4 cores, 2 GHz each, hyper-threading off

Memory: 4 GB

OS: Linux 2.6.9-89.0.23.ELsmp, redhat based

Java: sun jre 1.6.0_21

No other significant processes were running on the machine during my simulations (at least at any time I checked).

7.3 Results

$$\epsilon = \frac{\Delta simulation\_time}{\Delta real\_time} \tag{1}$$

The 'CPU' simulation reached maximum of 3000 running jobs at the time of 50 seconds and crashed at 490 seconds when the machine ran out of memory and failed to create any new threads. The ϵ however never dropped under 10^{-1} (see Figures 21, 23 and 24).

The 'local' and 'global' simulations crashed after approximately 70 seconds for the same reason (note that they use 2 threads per job - one for the job itself and one for the transfer). Here the 'global' simulation got even to 10^{-4} - see Figures 22, 23 and 24.

After having discussed my findings with the developers of MONARC2 we concluded, that for more complicated models it would be more appropriate to use the class CPUCluster instead of the class CPUUnit. CPUCluster is practically the same as CPUUnit only with an enormous memory, many virtual CPUs and only one linkport (instead of hundreds). This would very likely lead to faster simulations. MONARC2 also offers the class JobOpt, which represents an optimized job, that can handle multiple jobs and transfers with a single thread.

Unfortunately, due to time constraints I have not performed this additional benchmarking simulation test.

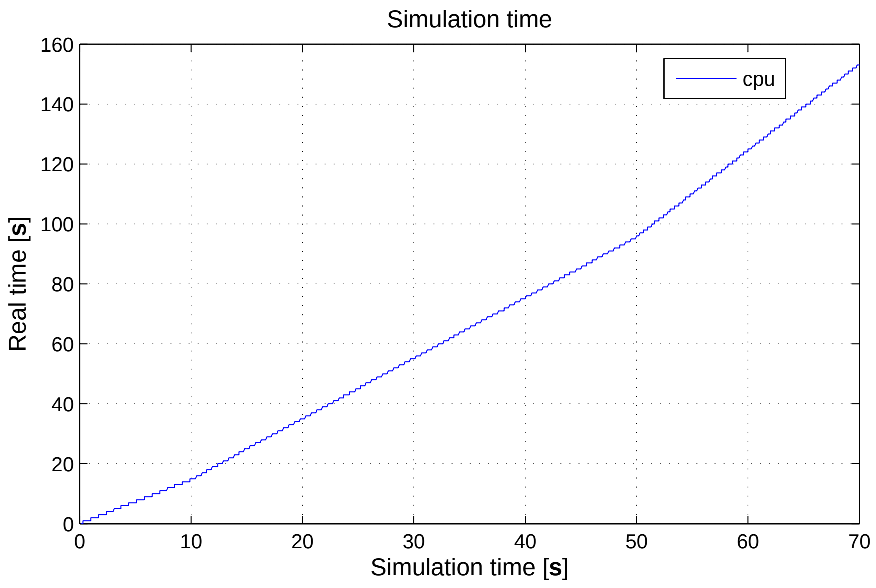

Figure 21: The simulation_time vs real_time plot for the 'CPU' simulation.

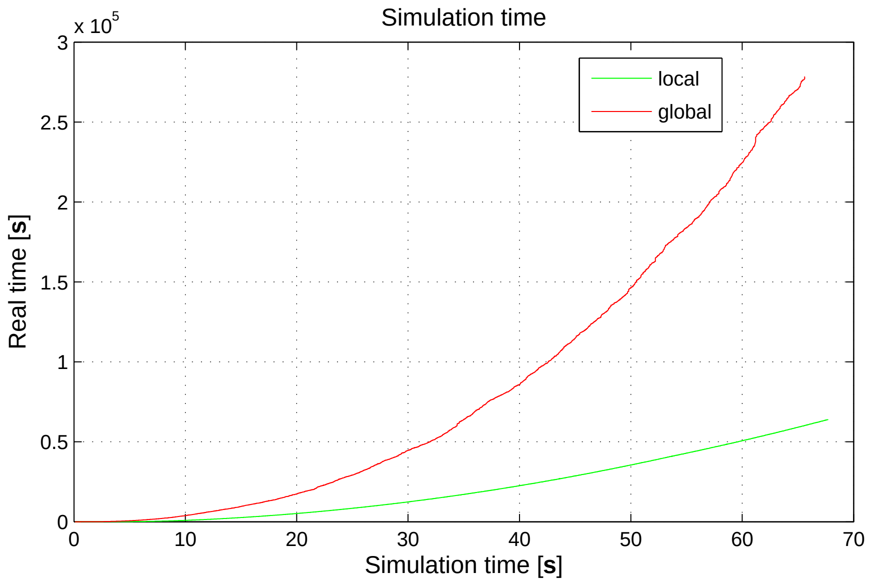

Figure 22: The simulation_time vs real_time plot for the 'local' and the 'global' simulation.

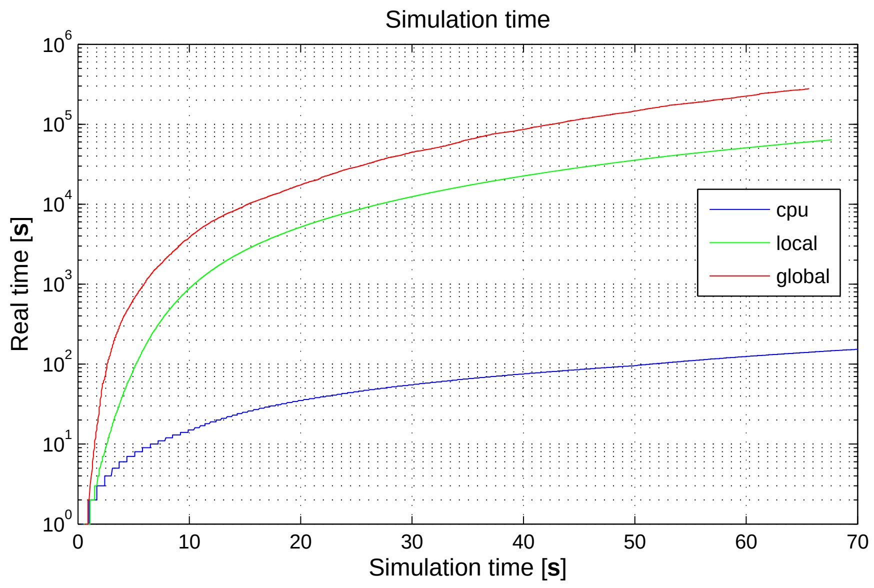

Figure 23: The simulation_time vs real_time plot for all the simulations.

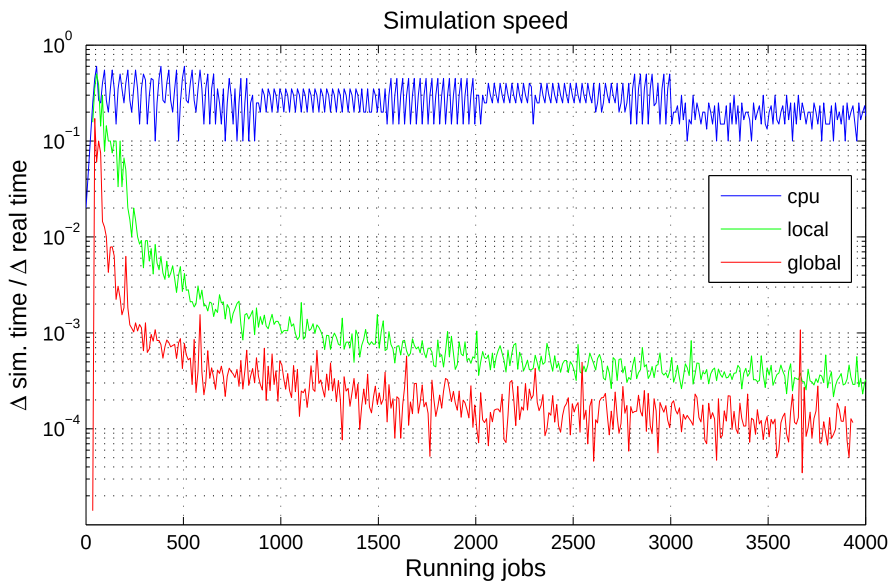

Figure 24: The ϵ vs running jobs plot for all the simulations.

8 Summary

In my present work, I 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 the ALICE-specific infrastructure: the AliEn framework. A very important part of the ALICE distributed computing system is represented by the Tier-2 centers, which provide almost half of the computing and disk resources.

With about 80 computing sites of the ALICE distributed system spread all over the world, it is necessary to master a machinery of job processing and data storage and transfers, which would ensure an efficient usage of the available resources with as few bottlenecks as possible. When building such a complicated system, it is essential to have a possibility to use results of reasonably realistic simulations of the behavior of various components of the system under big load, which should go along with the real stress-tests of the system. Since the basic part of the ALICE computing potential is concentrated in the Tier-2s, simulations of their behavior under heavy loads of jobs are of basic importance.

The second part of my work was concerned with the study and extension of the MONARC2 simulation framework and tools for the purpose of simulations of the ALICE Tier-2 centers. The aim of MONARC2 and its predecessor MONARC is to provide a design and optimization tool for building of large distributed computing systems. The simulator is in particular customized for the LHC-specific HEP data processing.

I successfully extended the MONARC2 simulator package to include needed parts of the ALICE Computing Model like JobAgent or AliEn Resource Broker. Then I created a model of six ALICE Tier-2 sites which included the Prague computing center Golias and additional 5 ALICE regional centers from Central Europe. With this model I ran simulations to predict effects of different requirements on the location of data processed by jobs on the jobs' duration, efficiency and other important characteristics.

As a result, my simulations have shown that the average time needed for the processing of jobs grows up and the average efficiency of the jobs processing drops down when the processed data files are stored at distant centers. A similar effect was produced when I was changing the ratio between the number of the simulation and analysis jobs: when the percentage of the analysis jobs goes up, the average processing time grows and the average efficiency drops.

While the mentioned observations were something one would in general expect, there was also a surprising result of the average data transfer speed growing as a result of having the data stored on distant centers and raising the percentage of analysis jobs. I have discussed possible reasons for this behavior.

These presented results are specific to the chosen model. For other models of computing centers and layouts, the results might not have been the same.

Using my specific model, I ran into difficulties with the performance of the MONARC2 simulator. Therefore, I completed a couple of benchmark simulations to test the limits of the MONARC2 package.

9 Conclusion

In the real situation analysis of data consists basically of two tasks: to get the data and to process the data. The 'getting the data' part is done by the network and the processing is done by computers. To have an optimal performance means to have these two parts in balance.

When you have faster computers, they will wait for the network to transfer the data. When you have faster network, it will not be used to its full potential. Sometimes it is worth using only a part of the CPU resources (when the network is slower), so that you would get some results faster and the energy consumption would be reduced, while the total time of processing all the data would be the same.

In simple situations analytic solutions can be found, but in the case of the complex systems like the LHC Computing Grid, the simulations are the only way how to estimate or predict the behavior of such systems. The first months of the LHC operations characterized by the envisaged enormous volumes of PetaBytes of the recorded raw data have clearly shown the importance of the smooth functioning of the data processing machinery. Any failures of or bottlenecks in this system is a threat of loosing the precious data or running into problems with the data processing, this time loosing the basic Physics results.

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] CERN. `http://www.cern.ch`.

[4] LHC: The Large Hadron Collider. `http://lhc.web.cern.ch/lhc/`.

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

[6] C. M. Dobre and V. Cristea.
A Simulation Model for Large Scale Distributed Systems.
`http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4430426`, 2008.

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

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

[9] Web page of the Golias farm. `http://www.particle.cz/farm/`.

[10] C. Zach, L. Betev, and D. Adamova. *Simulation of the job processing performance at an ALICE Tier-2 site with MONARC.* accepted for publication in the Proceedings of the 18th International Conference on Computing in High Energy and Nuclear Physics (CHEP) 2010 (to appear in Journal of Physics: Conference Series, 2011).

[11] CREAM CE.
`http://grid.pd.infn.it/cream/`.

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

[13] 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`.

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

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

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

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

[18] 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.