Multi-threaded, discrete event simulation of distributed computing system

I.C. Legrand[1,2] for the MONARC Collaboration

[1] California Institute of Technology
[2] CERN

Abstract

The LHC experiments have envisaged unprecedented complex computing systems, for which is necessary to provide a realistic description and modelling of data access patterns and of many jobs running concurrently on large scale distributed systems and exchanging very large amounts of data.
A process oriented approach for discrete event simulation is well suited to describe various activities running concurrently as well the stochastic arrival patterns, specific for such type of simulation. Threaded objects or "Active Objects" can provide a natural way to map the specific behavior of distributed data processing into the simulation program.

The simulation tool developed within MONARC[1] is based on Java(TM) [2] technology which provides adequate tools for developing a flexible and distributed process oriented simulation. Proper graphics tools, and ways to analyze data interactively, are essential in any simulation project.
The design elements, status and features of the MONARC simulation tool are presented. The program allows realistic modelling of complex data access patterns by multiple concurrent users in large scale computing systems in a wide range of possible architectures, from centralized to highly distributed. Comparison between queueing theory and realistic client-server measurements are also presented.

1. Introduction

The aim of this paper is to describe the simulation program, being developed by the MONARC project, as a design and optimization tool for large scale distributed computing system for future LHC experiments. The goals are to provide a realistic simulation of distributed computing systems, customized for specific physics data processing and to offer a flexible and dynamic environment to evaluate the performance of a range of possible data processing architectures.
An Object Oriented design, which allows an easy and direct mapping of the logical components into the simulation program and provides the interaction mechanism, offers the best solution for such a large scale system and also copes with systems which may scale and change dynamically. A discrete event, process oriented simulation approach, developed in Java(TM) was used for this modelling project. A complex Graphical User Interface (GUI) to the simulation engine, which allows to dynamically change parameters, load user's defined time response functions for different components and to monitor and analyze simulation results, provides a powerful development tool for evaluating and designing large scale distributed processing systems.

2. Design Considerations of the simulation program

The simulation and modelling task for MONARC requires the description of complex data processing programs, running on large scale distributed systems and exchanging very large amounts of data. A process oriented approach for discrete event simulation is well suited to describe concurrent running programs as well as all the stochastic arrival patterns, characteristic for such type of simulations. Threaded objects or “Active Objects” (having an execution thread, program counter, stack, mutual exclusion mechanism...) offer much great flexibility in simulating the complex behaviour of distributed data processing programs.

The MONARC simulation tool is built completely with Java$^{(TM)}$ technology which provides adequate tools for developing a flexible and distributed process oriented simulation. Java has build-in multi-thread support for concurrent processing, which can be used for simulation purposes by providing a dedicated scheduling mechanism. Java also offers good support for graphics and it is easy to interface the graphics part with the simulation code. Adequate and flexible graphics tools, and ways to analyze data interactively, are essential in any simulation project.

Currently, many groups involved in Computer System Simulation are moving towards Java. As an example, a well known project, Ptolemy II [3], is a complete new redesign of the Ptolemy simulation environment in Java. The reasons for which we decided to write a new “simulation engine” for process oriented, discrete event simulation were, first, a dedicated core for the simulation engine can be more efficient, and second, at the time we started this project, no such Java based simulation frame was yet available in a sufficient stable form. However the modular structure of this simulation package does not exclude the possibility to be interfaced with the engines of other general simulation tools.

3. The components models

Building a general simulation tool requires the abstraction from the real system of all the components and their time dependent interaction. This logical model has to be equivalent to the simulated system in all important respects. This simulation frame allows one to introduce any time dependent response function for the interacting components. Response functions may also be dependent on the previous states of the component allowing to describe correctly highly non-linear processes. The major components used in this simulation project are described below.

3.1 Data Model

It is foreseen that all HEP experiments will use an Object Database Management System (ODBMS) to handle the large amounts of data in the LHC era. Our data model follows the Objectivity architecture and the basic object data design used in HEP. The model should provide a realistic mapping of an ODBMS, and at the same time allow an efficient way to describe very large database systems with a huge number of objects.

The atomic unit object is the “Data Container”, which emulates a database file containing a set of objects of a certain type. In the simulation, data objects are assumed to be stored in such “data container” files in a sequential order. In this way the number of objects used in the simulation to model large number of real objects is dramatically reduced, and the searching algorithms are simple and fast. Random access patterns, necessary for realistic modelling of data access, are simulated by creating pseudo-random sequence of indexes.

Clustering factors for certain types of objects, when accessed from different programs, are simulated using practically the same scheme to generate a vector of intervals.

A Database unit is a collection of containers and performs an efficient search for type and object index range. The Database server simulation provides the client server mechanism to access objects from a database. It implements response time functions based on data parameters (page size, object size, access is from a new container, etc.), and hardware load (how many other requests are in process at the same time). In this model it is also assumed that the Database servers control the data transfers from/to mass storage system. Different policies for storage management may be used in the simulation. Database servers register with a database catalogue (Index) used by any client (user program) to address the proper server for each particular request. A schematic representation of how data access model is implemented into the simulation program is presented in Figure 1.

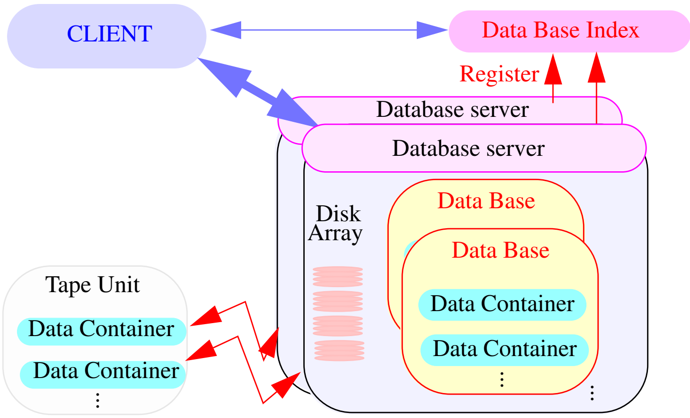


Figure 1:

A schematic diagram of data model based on ODBMS architecture

This modelling scheme provides an efficient way to handle a very large number of objects and in the same time an automatic storage management. It allows to emulate different clustering schemes of the data for different types of data access patterns, as well as to simulate the ordered data access when following the associations between the data objects, even if the objects reside in databases located in different database servers.

3.2 Multitasking Data Processing Model

Multitasking operating systems share resources such as CPU, memory and I/O between concurrently running tasks by scheduling their use for very short time intervals. However, simulating in detail how the tasks are scheduled in the real systems would be too complex and time consuming, and thus it is not suitable for our purpose. Our model for multitasking processing is based on an "interrupt" driven mechanism implemented in the simulation engine. An interrupt method implemented in the "Active" object which is the base class for all running jobs, is a key part for the multitasking model. The way it works is shown schematically in Figure 2.

When a first job starts, the time it takes is evaluated and this "Active" object enters into a wait state for this amount of time and allows to be interrupted. If a new job starts on the same hardware it will interrupt the first one. Both will share the same CPU power and the time to complete for each of them is computed assuming that they share the CPU equally. Both active jobs will enter into a wait state and are listeners to interrupts. When a job is finished it also creates an interrupt to re-distribute the resources for the remaining ones. This model is in fact assuming that resource sharing is done continuously between any discrete events in the simulation time (e.g. new job submission, job completion) while on

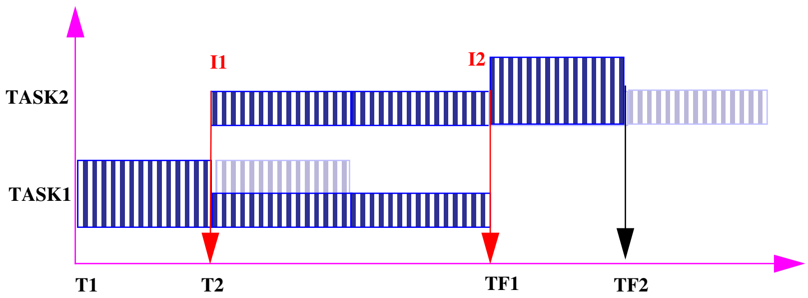


Figure 2:

Modelling multitasking processing based on an "interrupt" scheme

real machines it is done in a discrete way but with a very small time interval. This provides an accurate and efficient model for multiprocessing tasks.

3.3 LAN/WAN Networking Model

Accurate and efficient simulation of networking part is also a major requirement for the MONARC simulation project. The simulation program should offer the possibility to simulate data traffic for different protocols on both LAN and WAN. This has to be done for very large amounts of data and without precise knowledge of the network topology (as in the case of long distance connections). It is practically impossible to simulate the networking part at a packet level for such large amounts of data user's defined time dependent functions are used to evaluate the effective bandwidth.

The approach used to simulate the data traffic is again based on an "interrupt" scheme (Figure 3). When a message transfer starts between two end points in the network, the time to completion is calculated.

This is done using the minimum speed value of all the components in between, which can

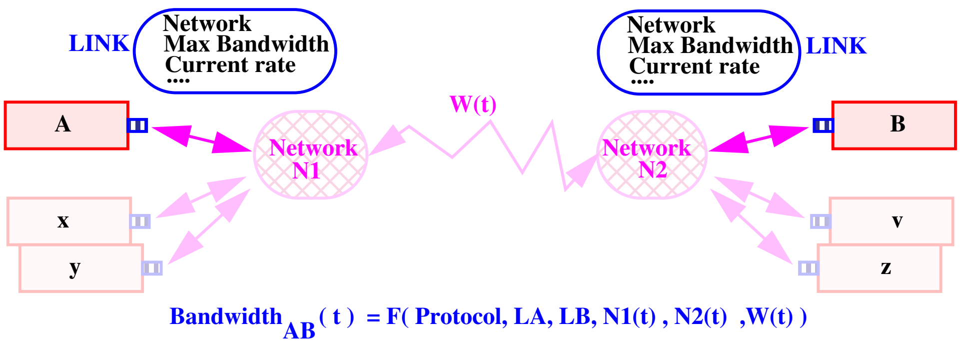


Figure 3: The Networking simulation model

be time dependent, and related with the protocol used. The time to complete is used to generate a wait statement which allows to be interrupted in the simulation. If a new message is initiated during this time an interrupt is generated for the LAN/WAN object. The speed for each transfer affected by the new one is re-computed, assuming that they are running in parallel and share the bandwidth with weights depending on the protocol. With this new speed the time to complete for all the messages affected is re-evaluated and inserted into the priority queue for future events. This approach requires an estimate of the data transfer speed for each component. For a long distance connection an "effective speed" between two points has to be used. This value can be fully time dependent.

This approach for data transfer can provide an effective and accurate way to describe many large and small data transfers occurring in parallel on the same network. This model cannot describe speed variation in the traffic during one transfer if no other transfer starts or finishes. This is a consequence of the fact that we have only discrete events in time. However, by using smaller packages for data transfer or artificially generating additional interrupts for LAN/WAN objects the time interval for which the network speed is considered constant can be reduced. As before, this model assumes that the data transfer between time events is done in a continuous way utilizing a certain part of the available bandwidth.

3.4 Arrival Patterns

A flexible mechanism to define the stochastic process of submitting jobs is necessary. This is done using the “dynamic loadable modules” feature in Java which provide the support to include (threaded) objects into running code. These objects are used to describe the behavior of a “User” or a “Group of Users”. It should be able to describe both batch and interactive sessions, and also to use any time dependent distribution describing how jobs are submitted. An “Activity” object is a base class for all these processes for which current experience should be used to estimate the time dependent patterns and correlations.

In order to provide a high flexibility in modelling all these activities, the user can provide very simple sections of Java code, to override the “RUN” method of the “Activity” class. Any number of such “Activities” can be dynamically loaded via the GUI into the “Regional Centre” object, simulating the “Users” using the computing facilities.

3.5 Regional Centre Model

“Regional Centre” is a complex, composite object containing a number of data servers and processing nodes, all connected to a LAN (Figure 4).

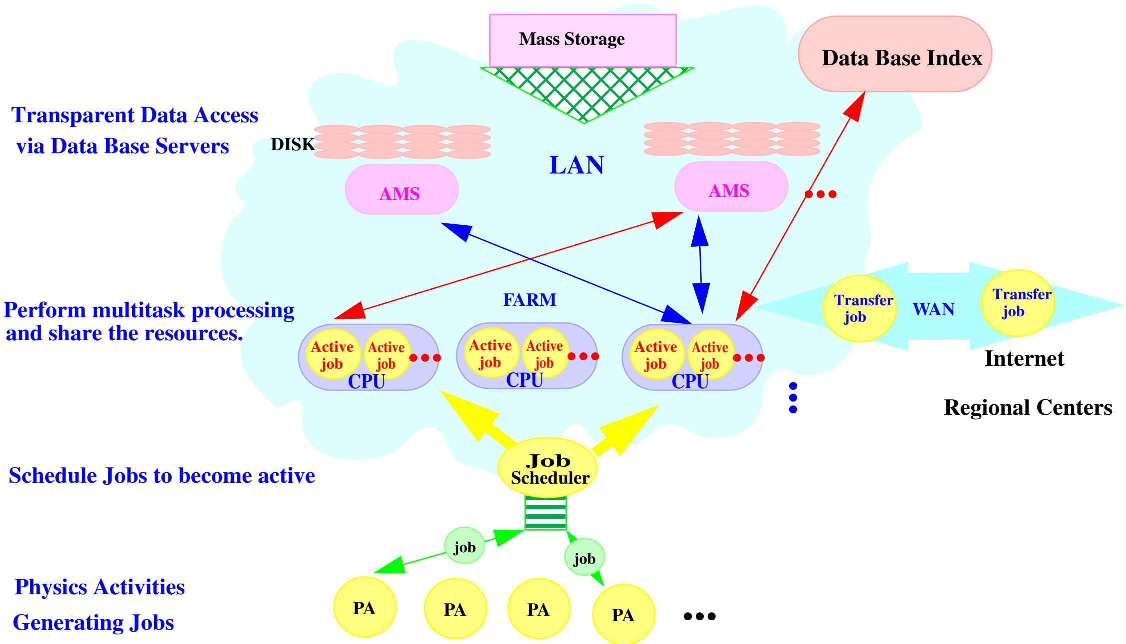


Figure 4: A schematic view of a Regional Centre object as a composite object

Optionally, it may contain a Mass Storage unit and can be connected to other Regional Centres. Any regional centre can instantiate dynamically a set of “Users” or “Activity” Objects which are used to generate data processing jobs based on different scenarios. Inside a Regional Centre different job scheduling policies may used to distribute the jobs to processing nodes.

With this structure it is now possible to build a wide range of computing models, from the very centralized (with reconstruction and most analyses at CERN) to the distributed systems, with an almost arbitrary level of complication (CERN and multiple regional centres, each with different hardware configuration and possibly different sets of data replicated)

4. The Graphical User Interface

An adequate set of GUIs to define different input scenarios, and to analyze the results, are essential for the simulation tools. The aim in designing these GUIs was to provide a simple but flexible way to define the parameters for simulations and the presentation of results. In Figure 5 the frames used to define the system configuration are presented.

The number of regional centres considered can be easily changed through the main window of the simulation program. The “Global Parameters” menu allows to change the (mean) values and their statistical distributions for quantities which are common in all Regional Centres. The hardware cost estimates for the components of the system and can also be obtained. From the Regional Center frame, which appears when the name of the centre is selected in the main window, the user may select which parameters to be graphically presented (CPU usage, memory load, load on the network, efficiency, Database servers load...). For all these results basic mathematical tools are available to easily compute integrated values, mean values, integrated mean values.

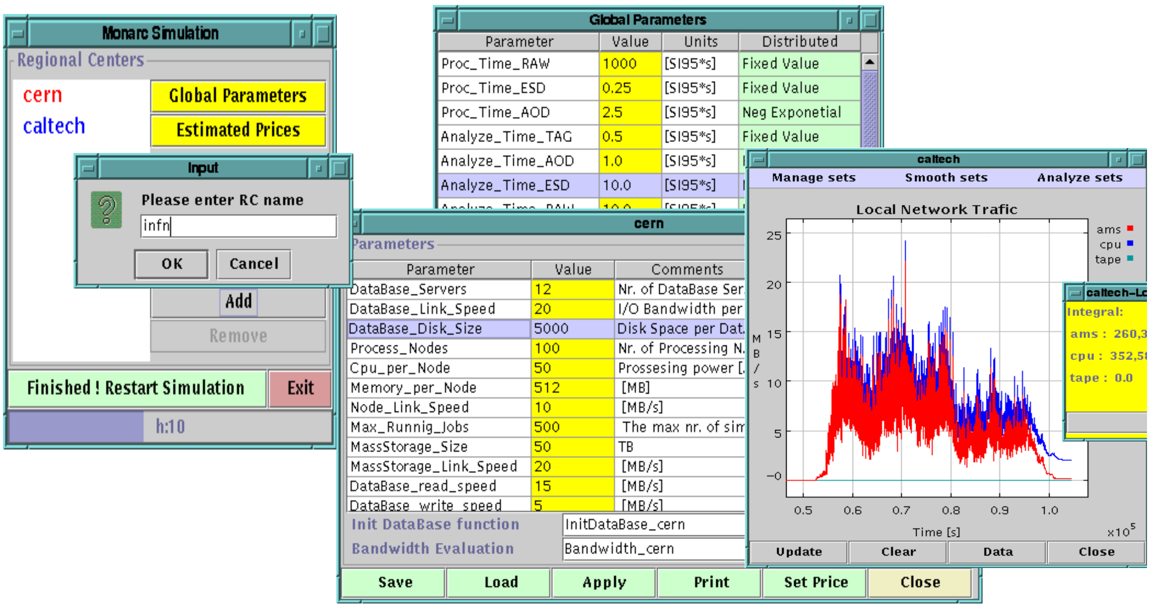


Figure 5:   The GUI Frames used to define the system configuration and monitor output results

5. Presenting and Publishing the Results

To facilitate publishing (storing) the simulation results, as well as all the configuration files used to generate those results, an automatic procedure to publish results either locally, or on a Web server, has been developed.

This Web Page offers and automatic repository for the Monarc Simulation Program. It allows to publish the configuration files, java source code and the results (tables and graphic output) for different simulation runs. The aim of this page is to provide an easy way to share ideas and results for developing regional center models. A schematic view of how this publishing mechanism is implemented is presented in Figure 6. This procedure to automatically publish configuration files, java sources, graphical results is fully implemented in java. When the user decides to “publish” a run, the simulation program as a client tries to find one of several dedicated servers. More than one server is used to make this service more reliable in case one system is down. The Server implements the Remote Method Invocation [2] mechanism and provide to each interested client the functionality to transfer files and automatically to update the content of this Web Page [4].

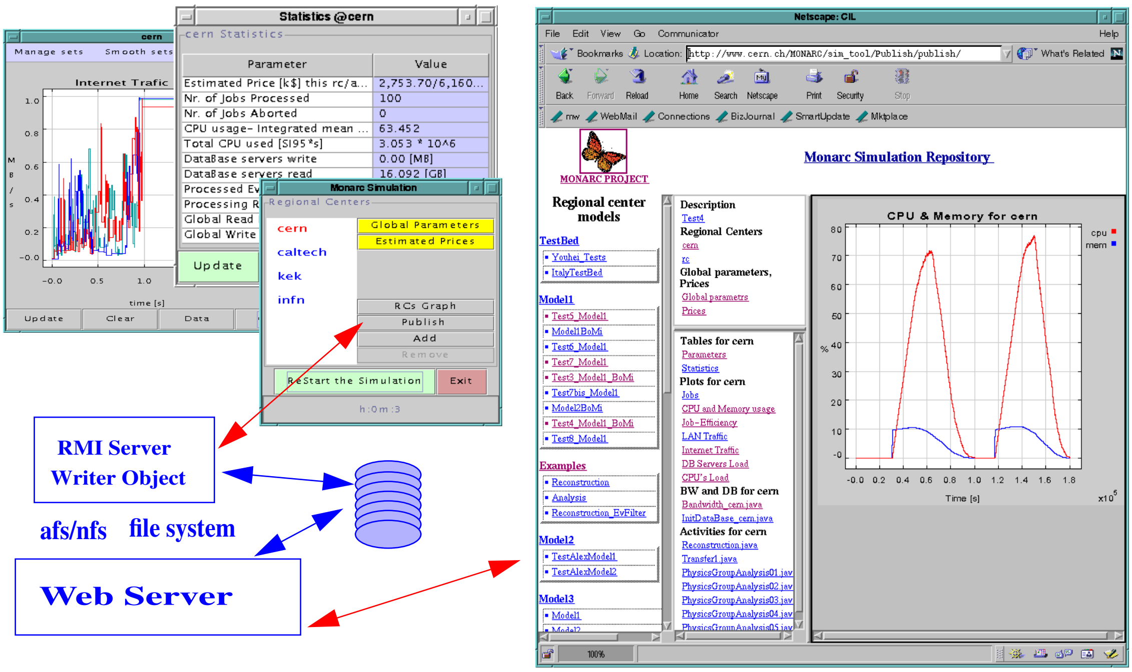


Figure 6: Publishing the simulation results on a web server

6. Testing and Evaluating the simulation program

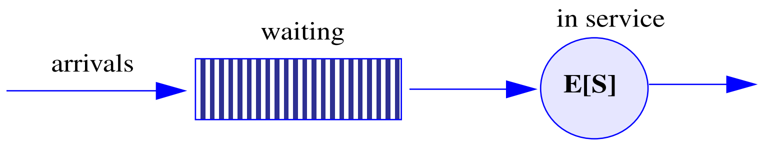
A number of tests have been performed to verify and test the simulation program. The full description of the validation tests for this program are presented in an other paper [6].

A few basic comparisons tests of the simulation program results with queueing theory and parallel client server data access measurements in real systems are presented.

6.1 Queuing Theory

6.1.1 M | M | 1 Model

This model [5] consists of queueing station where jobs arrive with a negative exponential inter-arrival time distribution with rate λ.

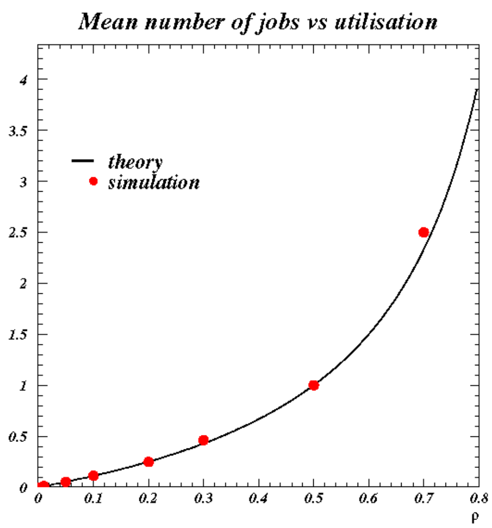


Furthermore, the job time service requirements are also negative exponentially distributed with mean $E[S] = \frac{1}{\mu}$. Simplest queueing model M/M/1 theory gives the formula for mean number of jobs in the system and mean response time:

$$E[N] = \frac{\rho}{1-\rho} \qquad \text{and} \qquad E[R] = \frac{E[S]}{(1-\rho)}$$

where E[N] is the mean number of jobs in the system, E[R] - mean response time of the system, E[S] - mean serve time of the system, utilization $\rho = \frac{\lambda}{\mu}$, and λ is mean job arrival rate, μ is mean job service rate, $E[S] = \frac{1}{\mu}$ is mean service time. This case can be described in the simulation program as a Database server acting as a queuing station for data requests coming from clients with the same time distribution. The data size for each request is also distributed as a negative exponential (Marcovian process).

Results for different input rate shown in the figure 7, for $\lambda = 500$, $\mu = 1000$, so $\rho = 0.5$, and mean number of jobs in the system is 1.0, which is equal to the value obtained from the simulation.

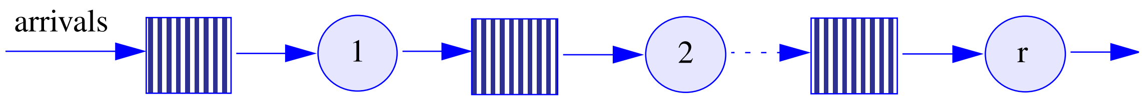


Arrival rate	E[N], sim	E[N] theory	E[R] sim	E[R] theory
1.0	0.001018	0.001001	0.001007	0.001001
10.0	0.001018	0.001010	0.010171	0.010101
50.0	0.001034	0.001053	0.052077	0.052632
100.0	0.001137	0.001111	0.112971	0.111111
200.0	0.001232	0.00125	0.246945	0.25
300.0	0.001538	0.001429	0.461039	0.428571
500.0	0.00199	0.0020	1.00087	1.0
700.0	0.003580	0.003333	2.497969	2.333333

Figure 7: Simulation results for the M|M|1 model

6.1.2 M | M | 1 network queue model

This type of queueing model consists of chain of M | M |1 queues [5].



In this case mean total number of jobs in the system and mean total response time of the network are defined by:

$$E[N] = \sum_{i=1}^{r} E[N_i] = \sum_{i=1}^{r} \frac{\rho_i}{1-\rho_i} \qquad \text{and} \qquad E[R] = \sum_{i=1}^{r} E[R_i] = \sum_{i=1}^{r} \frac{E[Si]}{(1-\rho i)}$$

where utilization for each stage is $\rho_i = \frac{\lambda}{\mu_i}$.

Simulating this model can easily be done by creating a sequence of jobs. This is similar with an Analysis job which will sequentially process AOD, ESD and RAW records for each event. As we have different record size for different types of data, we obtain different service time (or service rate) assuming the speed to read pages from the disk is constant. But according to Burk's theorem the departure process from a stable single server M/M/1 queue with arrival and service rates is a Poisson process. So we can apply the same formula as for M/M/1 case for each stage of process and sum mean number of job and mean response time in each stage. In the Figure 8, a comparison between the theoretical predictions and the values obtained with the simulation program is presented.

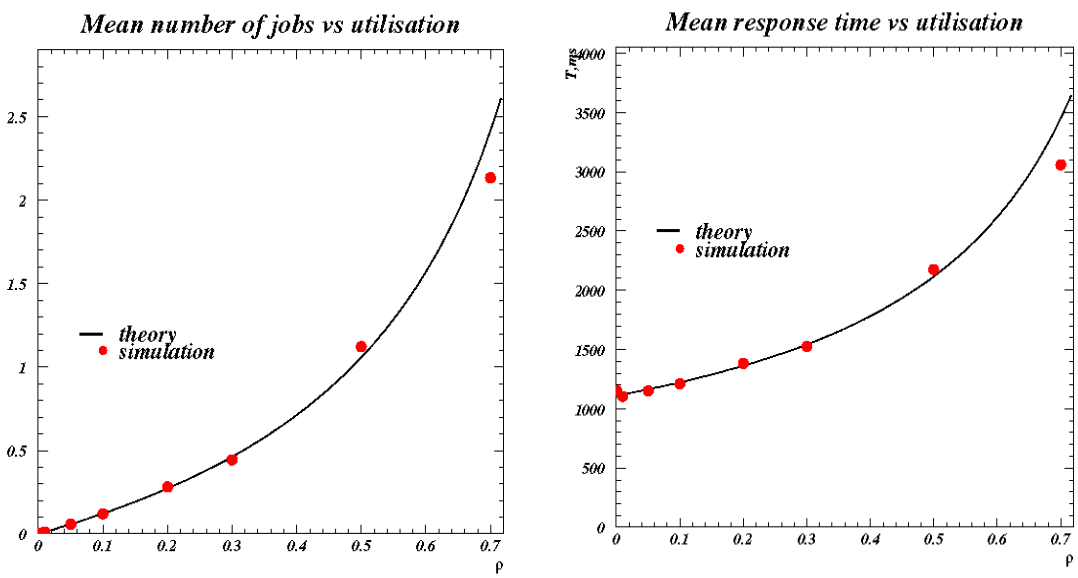


Figure 8: Simulation results for the M/M/1 queue model

6.2 Concurrent Database access

Database server and their interaction with clients is a key part which need to be properly described by the simulation program. Different measurements were done do evaluate

Objectivity's performance and to understand the basic logical transaction protocol. parameters for the simulation program were tuned using a single client measurements. The way simulation program describes multiple concurrent clients requesting data from the same database server is compared with measured values. In Figure 9, results using Objectivity 5.1 on a local area network are presented.

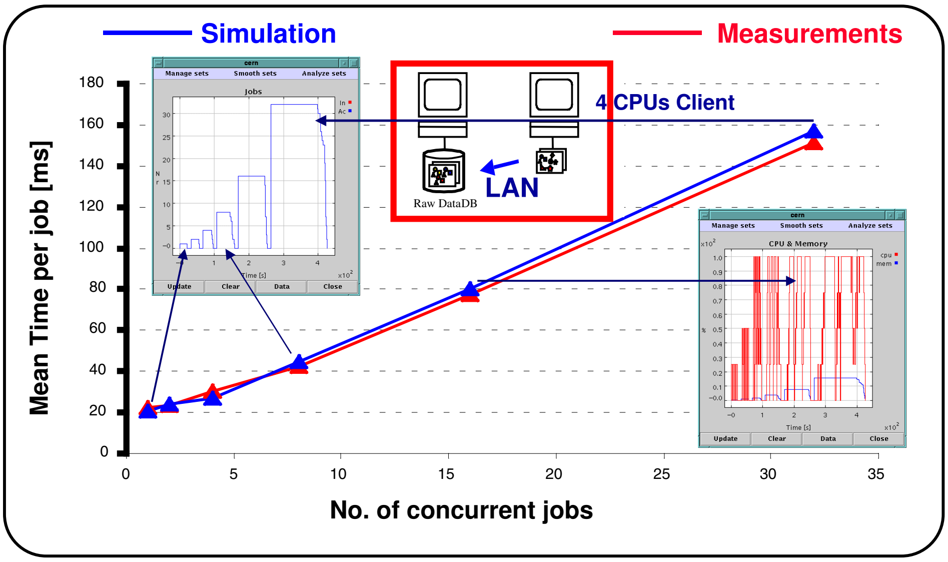


Figure 9: Simulation results compared with concurrent client/server measurements on LAN.

A similar setup, but on wide area network (CERN - CNAF) is presented in Figure 10.

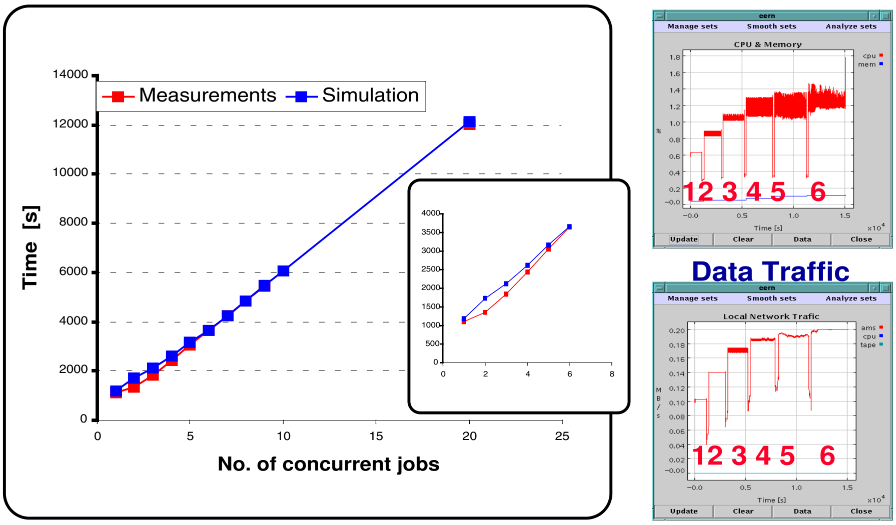


Figure 10: Simulation results compared with concurrent client/server measurements on WAN.

A substantial number of such measurements [6] were performed using different data types and network configuration to test and validate this simulation program. For all these test measurements we obtained a good agreement with the simulation performed to model them.

7. An Example

In three regional centers (symbolically named CERN, CALTECH, INFN) similar physics analysis jobs are done, but data replication is different as indicated in Figure 11.

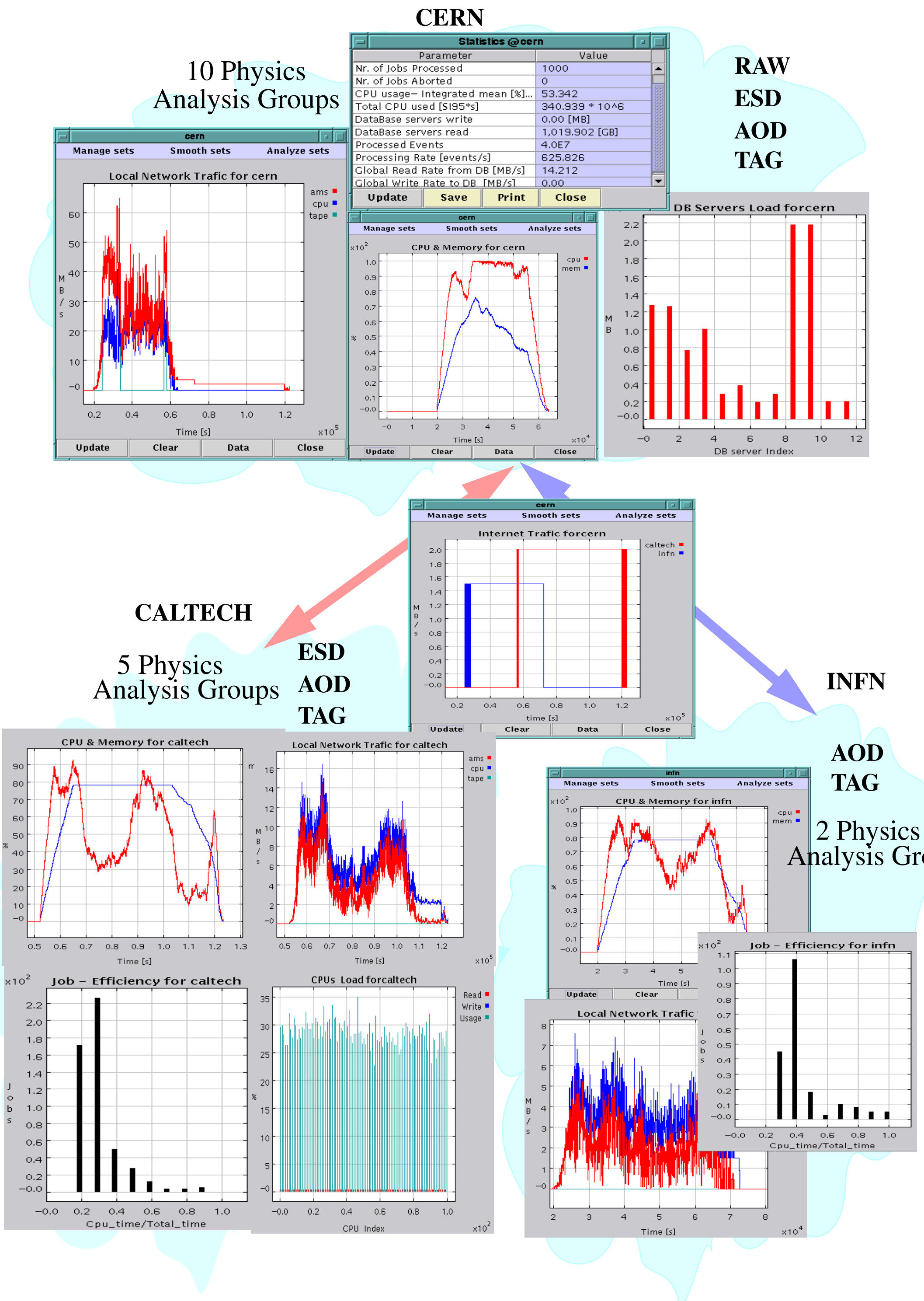


Figure 11: Schematic view of a distributed data processing simulation job

One physics analysis group is assumed to submit 100 jobs per day and is analyzing $4*10^6$ events. For 2% of the events ESD records are requested and for 0.5% of the events RAW data are used. At CERN we assume 10 physics analysis groups, at CALTECH 5, and 2 at INFN. In each center the activity starts in the morning and more jobs are submitted in the first part of the day. When a job needs data which are not available locally, the transfer is done from “CERN”. Typical parameters used to monitor such simulation jobs and how the resources are used in the system are presented schematically in Figure 11.

8. Summary

A CPU and code-efficient simulation approach to the problem of simulation of distributed computing systems has been developed and tested within the MONARC Collaboration. It provides a transparent way to map the distributed data processing, data transport and analysis tasks onto the simulation frame, and can describe dynamically even very complex computing models.
The Java programming environment, used extensively to build the MONARC simulation tool, is very well suited for developing a flexible and distributed process oriented simulation, equipped with adequate graphical and statistical tools.
This simulation program is still under development to include more sophisticated methods to optimize the utilization of resources in very large scale distributed computing systems.

References

1 Monarc simulation program
http://www.cern.ch/MONARC/sim_tool/
2 Sun Microsystems
http://www.sun.con and http://www.javasoft.com
3 PTOLEMY II Heterogeneous concurrent modeling and design in Java
http://ptolemy.eecs.berkeley.edu
4. Monarc simulation repository
http://www.cern.ch/MONARC/sim_tool/Publish/publish/
5 B. R. Haverkort, Performance of Computer Communication Systems
John Wiley & Sons Ltd., ISBN 0-471-97228-2
6 Y.Morita et al, Validation of the MONARC simulation tools
to be presented at CHEP 2000