

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Setting up a HTC/Beowulf cluster for distributed radiation transport simulations

Fernando Joaquim Leite Pereira

Report of Project

Master in Informatics and Computing Engineering

Supervisor: Prof. Jaime E. Villate

Local supervisors: Christian Theis and Eduard Feldbaumer

2008, July

Setting up a HTC/Beowulf cluster for distributed radiation transport simulations

Fernando Joaquim Leite Pereira

Report of project
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Pedro Alexandre Ferreira do Souto

External Examiner: António Amorim

Internal Examiner: Jaime Villate

7th July, 2008

Resumo

Nos últimos anos, o progresso nas tecnologias de comunicação bem como a redução do custo de plataformas de uso comum tem permitido o desenvolvimento de clusters cada vez mais poderosos e a sua prosperidade nos mais diversos campos de aplicação. Em particular, os clusters Beowulf têm tido um enorme sucesso devido ao seu baixo custo, elevada performance e alta flexibilidade inerente à possibilidade de utilização de computadores pessoais. Contudo, para além da estrutura de hardware, o maior desafio na instalação de um cluster é definir o software que permita obter a maior eficiência na utilização dos recursos.

No contexto deste projecto, o grupo de protecção de radiação (RP) do CERN pretende instalar um cluster para executar simulações de transporte de partículas através do software de cálculo Monte-Carlo FLUKA, utilizando os seus próprios computadores. No entanto, não havendo possibilidade de modificar o software ao nível do código fonte para explicitamente suportar paralelismo, o trabalho desenvolvido nesta tese concentra-se em optimizar a distribuição da execução de processos de FLUKA pelo cluster.

Com este objectivo, um Job Managament System (JMS) existente designado Condor foi utilizado para criar um ambiente de High-Throughput-Computing (HTC) que distribuisse as tarefas (neste caso os processos de simulação) pelo cluster com base na actual carga e desempenho dos CPUs e também na prioridade das tarefas. De forma a facilitar a utilização do cluster e oferecer aos utilizadores uma interface de mais alto nível, foi desenvolvido um conjunto de aplicações que incluem duas interfaces visuais e interactivas: por um lado uma aplicação para linha de comandos, e por outro lado um site web contendo um valioso conjunto de funcionalidades.

Abstract

In the last years the advancements in network capacity and the decreasing price of commodity platforms has permitted the development of more and more powerful clusters and their prosperity among the most various fields of applications. Particularly, Beowulf clusters became very popular because of their low cost, high performance and high flexibility obtained through the usage of personal computers. But beyond the hardware structure, the real challenge in setting up a cluster system is to provide a software layer which efficiently takes advantage of the resources.

In the context of this project, the Radiation Protection group at CERN wants to set up a cluster to run particle transport simulations with the Monte-Carlo code FLUKA, by using their own existing machines. Having no possibility to change the application's source code to explicitly support parallelism, this thesis focuses on optimizing the distribution of FLUKA processes across the cluster.

For this purpose, an existing job management system named Condor was used to create a High-Throughput-Computing environment and manage jobs taking into account the current CPU load, the job priorities and the CPU performance. To facilitate the usage and provide a high-level interface to the cluster, a set of applications was developed including two front-ends: on the one hand a terminal based program and on the other hand a full-featured website.

Acknowledgements

I would like to express my total gratitude to all the extraordinary people which made possible the development of this project in such circumstances.

I feel pleased to have integrated the RP group at CERN, whose elements unconditionally helped and supported the various steps on the project. They would be too many to reference separately in this short part, but their contribution is present all over this work.

I would especially like to express my large gratefulness towards my local supervisors Christian Theis and Eduard Feldbaumer for their friendliness and absolutely outstanding support. Thanks to them, work had always been an object of personal interest, reflection, and constructive discussion. It was a really comforting to know they were always available for debate as well for having a good time. I also owe them the physics knowledge they transmitted during these months, as well time and patience to put up with me and answer my constant questions.

Furthermore I would like to thank to Prof. Jaime Villate, as my supervisor at FEUP. In conjunction with Prof. Baptista from CIC, they are the ground of my physics knowledge and definitely account for my personal affection in physics subjects.

As a last point I would like to acknowledge all my family and friends who in some way made it possible for me to be here. I must thank Daniel for his support, for our odd physics discussions and to put up with me for the second time in a foreign country. I'm sure the unforgettable Erasmus DF-Sweden experience wouldn't have been the same without him. I must not forget Ivo, Joaquim and Bruna as well, for their presence in some of the best moments I've ever had. And finally to my parents, brother and sister, my deep thank for helping me and providing their unconditional support in all situations of my life.

Preface

This is always a big step when it comes the time a student is on the finishing line of his studies and has to choose what will he do in the next months. This can likely be the end of the academic world and the transition for the professional one. Many things have to be taken in account, starting with the choice of making a dissertation or a graduation project in a real company, and ending with the selection of the thesis subject or the project to be done.

In this case, I feel lucky I hadn't much doubts about what I would like to do. My erasmus experience and passion for physics guided me to a great international company: CERN. Luckily enough, the project proposal really fitted my profile as using informatics, by means of a cluster, to increase performance of a physics system. And that's what really fascinates me in this world of technologies: to push limits further to support human progress.

Contents

1	Introduction	1
1.1	The CERN organization	1
1.1.1	Historical facts	2
1.1.2	Current Projects	3
1.2	Clusters at CERN	4
1.3	Document structure	5
2	Simulations performed at RP group	7
2.1	CERN Radiation Protection group	7
2.2	Radiation transport simulations	8
2.3	The previous simulation system	9
2.3.1	The original problem	9
2.3.2	A first try	10
2.4	The project	11
3	Linux clusters analysis	13
3.1	Cluster systems	13
3.1.1	Cluster Types	14
3.1.2	Distributed computing	15
3.2	Cluster software review - Job Management Systems	15
3.2.1	Job Management System's general architecture	16
3.3	Comparative analysis of available JMS	16
3.3.1	Selection Criteria	18
3.3.2	Result analysis	19
4	Setting up the Condor cluster	21
4.1	Architecture	21
4.1.1	Condor execution modes	22
4.1.2	Condor cluster requirements	23

4.1.2.1	Shared file system	23
4.1.2.2	Centralized authentication system	24
4.1.3	Physical architecture	25
4.2	System specification	27
4.2.1	Condor jobs handling	27
4.2.2	A policy for simulation jobs	28
4.2.3	State transitions	30
4.3	Implementation of the Condor configuration	31
4.3.1	Centralized Condor global configuration	32
4.3.2	Defining new attributes for jobs and machines	32
4.3.3	Implementing Condor non-blocking suspension	33
4.3.4	Implementing job priorities behavior in Condor	34
4.3.5	Implementing job distribution behavior	34
4.3.5.1	Load management	35
4.3.5.2	Ranking resources	36
4.3.6	Implementing machine-dependent behavior	37
4.3.7	Implementing fail-safe and global optimization rules	38
4.4	Tests and analysis of results	39
4.4.1	Priority behavior and distribution tests	40
4.4.2	Load control and distribution tests	42
4.4.2.1	Running 1 local and 1 Condor jobs	42
4.4.2.2	Running 2 local jobs	43
4.4.2.3	Extensive test	44
5	Development of supplementary software tools	45
5.1	High-level architecture and specification	45
5.1.1	Logical structure	46
5.1.2	Coflu-Toolkit requirements	47
5.1.3	Coflu_submit requirements	47
5.1.4	Coflu-Web requirements	48
5.2	COFLU-Toolkit architecture	49
5.2.1	Simulation structure (coflu_inputs)	50
5.2.2	coflu_submit interface architecture	51
5.3	Coflu-Toolkit implementation	52
5.3.1	Error handling	52
5.3.2	coflu_submit interactive mode	53
5.4	Coflu-Web architecture	54

5.4.1	Horizontal decomposition	54
5.4.2	Vertical decomposition	55
5.4.3	Physical architecture	57
5.5	Coflu-Web Implementation	57
5.5.1	Project structure	58
5.5.2	Authentication and remote execution	58
5.5.3	Three-tier data validation	59
5.5.4	Using AJAX to import configuration files	60
5.6	Tests and result analysis	61
5.6.1	coflu_submit execution example	62
5.6.2	Coflu-Web execution example	63
6	Summary and conclusions	67
	Bibliography	71
	Glossary	74
A	Relevant Condor configuration	77
B	Condor policy test outputs	81
B.1	Priority behavior and resources ranking	81
B.1.1	Without preemption	81
B.1.2	With preemption	82
B.2	CPU Load management	85
C	Coflu-Toolkit implementation	89
C.1	Shared configuration parser	89
C.2	Interactive mode functions (coflu_submit)	90
D	Coflu-Web	91
D.1	User interface	91
D.2	Relevant implementation	95
D.2.1	sshlib.php source	95
D.2.2	AJAX server to import submission files	97

List of Figures

1.1	The world's first web server	3
1.2	The CERN accelerator complex	4
3.1	JMS general architecture	17
4.1	Machine roles in Condor	22
4.2	System physical architecture	26
4.3	Condor main daemons	27
4.4	Condor job state diagram	28
4.5	Preempt after suspension	30
4.6	System state diagram	31
4.7	Slots in a dual core machine	33
4.8	Load management algorithm	35
4.9	Desktop CPU load and interference with heavy processes	38
4.10	Load control - machine fills up with local jobs	42
4.11	Load control - machine waits until gets free	43
4.12	Load control - extensive test	44
5.1	System's usage profiles and application dependencies	46
5.2	COFLU-Toolkit	50
5.3	Simulation file structure	51
5.4	coflu_submit architecture	52
5.5	coflu_submit interactive mode	53
5.6	Coflu-Web architecture: Horizontal decomposition	54
5.7	Coflu-Web: Vertical decomposition	56
5.8	Physical architecture	57
5.9	File structure and module template	58
5.10	Remote execution dependencies	59
5.11	Coflu-Web: simulation configurations	60
5.12	PEAR/HTML_AJAX in Coflu-Web	61

5.13	Main simulation directory	62
5.14	Submission page debugging data	63
5.15	Coflu-Web - Submitting simulation	64
5.16	Job submission results in debugging mode	64
5.17	job-status	65
D.1	Coflu-Web Home	91
D.2	Coflu-Web Submission	92
D.3	Coflu-Web Status	93
D.4	Coflu-Web Administration	94

List of Tables

3.1	Comparison between HTC and HPC	14
3.2	HTC comparison table	19

Chapter 1

Introduction

The extreme demand of computational power of some applications is always pushing the performance limits of systems, leading to the creation of new architectures, new logics and new algorithms. But the constant development of communication technologies and distributed systems is changing the definition of supercomputers. Clusters of computers can offer such performance no single super-computer could.

With the advent of personal-computer based clusters (Beowulf clusters), the mirage of having a supercomputer at a fraction of the price and increased flexibility became an eminent reality. From the 2 computer cluster to the thousand globally connected grid nodes, clusters are being used everywhere supporting not only scientific but all kind of purposes.

1.1 The CERN organization

The European Organization for Nuclear Research (CERN) is the world's largest particle physics laboratory and it's located at the Franco-Swiss border, northwest of Geneva [1]. The name comes from the french acronym for “Conseil Européen pour la Recherche Nucléaire”, a body formed in 1952 with the purpose of establishing a world-class fundamental physics research organization.

The convention establishing CERN was signed on 29 September 1954 and the organization was given the current title, although the CERN acronym was maintained. Starting with 12 initial signatories of the convention, CERN currently has 20 member states, including Portugal since 1985¹. The key ideas of this convention still apply and can be summarized [2] as:

- Research: Seeking and finding answers to questions about the Universe

¹the same year when the portuguese author was born

- Technology: Advancing the frontiers of technology
- Collaboration: Bringing nations together through science
- Education: Training the scientists of tomorrow

1.1.1 Historical facts

Many experiments have been carried at CERN, and some great discoveries have been achieved [3].

The first accelerator was the Synchrocyclotron (SC), built in 1957, which provided beams for CERN's first particle and nuclear physics experiments. It was used by ISOLDE facility and was only decommissioned in 1990, after 33 years of service.

In 1959 the Proton Synchrotron (PS) was set up and became the world's highest energy particle accelerator for brief period. Since 1970 is being used for as pre-accelerator for other more powerful accelerators or directly to experiments.

In 1968, due to progresses in transistor technology, Georges Charpak revolutionized particle detection. Using a large number of parallel detector wires connected to an amplifier, his system was performing a thousand times better than previous detectors.

Having started its construction in 1965, the first proton-proton collider with 300 meters diameter, came into operation in 1971.

In 1973 the discovery of neutral currents was publicly announced, confirming the Glashow/Salam/Weinberg theory which unified electromagnetism and the weak interactions. In 1979 the three physicists received the Nobel prize in physics.

In 1976 the Super Proton Synchrotron (SPS) was commissioned. Measuring 7 km in circumference, it was a giant ring crossing the Franco-Swiss border for the first time. It accelerates particle beams up to 450 GeV/c nowadays. Its main achievement was the discovery of the W and Z particles in 1983 by colliding protons and anti-protons, which awarded Carlo Rubbia and Simon van der Meer the Nobel Prize in physics in 1984.

In 1989 the Large Electron-Positron (LEP) started its operation. With its 27 Km underground tunnel, it's the world's largest particle collider ever built. Its four enormous detectors provided a deep study of electroweak interactions and the proof of three, and only three, generations of particles of matter.

1990 was a great year in the history of IT: Tim Berners-Lee invented the World Wide Web. Planned to be a way to share information between scientists, it's probably the today's most used Internet service on the planet. In the www project he defined the URL, the http and then htm, and implemented the first web browser and server(Figure 1.1 on page 3).



Figure 1.1: The world's first web server

1.1.2 Current Projects

The current large scale project at CERN is the very well known Large Hadron Collider (LHC) [4]. Large because of its 27 Km (using the tunnel from LEP) and Hadron because protons and ions are to be collided. It will be used to recreate the conditions just after the Big Bang by colliding two particle beams at very high energy (about 7TeV per proton) which makes them to travel at more than 99.9% of the speed of the light. The project started back to the 1980s and in December 1994 the CERN council approved its construction, at a total cost of about 6 billion CHF.

Along with the tunnel, 4 main detectors are being installed to run different and complementary experiments: ATLAS, CMS, ALICE and LHCb. The particle beams will be accelerated sequentially by previous accelerators in the chain, starting in PSB then PS, SPS and finally LHC (Figure 1.2 on page 4).

The beams of particles are formed by 2808 bunches of 10^{11} protons. So, even with a very low probability of collision (about 1 collisions in 50 billion particles), as particles make 11000 rounds per second in the ring, some 600 million collisions per second are expected on average. This scenario explains the dimension and sensitivity of the detectors, and the outrageous amount of data they produce. For instance ATLAS [4,5], one of the main detectors, is 46x25x25 meters, weighs 7000 tones, produces about 70 Terabytes (TB) per second, and is considered the most complex equipment ever assembled on Earth.

The main objectives of the LHC and its detectors, very briefly, are:

- To explain the origin of mass, by searching for the Higgs Boson;

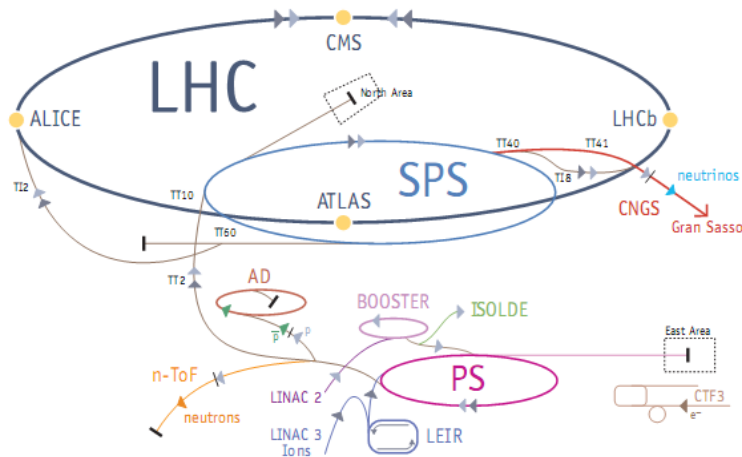


Figure 1.2: The CERN accelerator complex

- To unify fundamental forces, which would strengthen the Standard Model;
- To search for super-symmetric particles, which could form the dark matter;
- To investigate why matter has been preferred over antimatter, if in the Big Bang equal amounts were produced.

1.2 Clusters at CERN

It's easy to understand that 70TB of information per second is not easy to handle with a common server. Even if only a small amount of this information has to be saved, it must be filtered, which already requires huge computational power. Indeed, CERN is one of the world's leaders of computational power demand, but it's also the place where great effort is put into IT research, particularly for some grid projects.

At CERN, clusters are more than just a convenient way to speed up the calculation of results, but a mission critical piece of the experiments. If data could not be analyzed it would be worth nothing. All the main experiments depend on cluster systems, sometimes several, depending on the kind of task. For the LHC project a grid platform - the LHC Computing Grid (LCG) [6] - was installed which aims to integrate thousands of computers from hundreds of data-centers to analyze the huge amounts of data generated at the LHC. Additionally, some specific experiment simulations (like for Atlas and CMS) were performed over computing clusters [7–9].

While the main cluster systems are principally used for “live” data, i.e., they will handle data acquired (and/or generated) by the experiments, there are a number of other computer applications, for example particle transport simulations, which require a big amount of computational power as well and might still be driven the old way. Such simu-

lations constitute today's state-of-the-art means for a wide range of applications, spanning from medical physics to radiation protection. CERN's radiation protection group (RP) has been using such calculations for various tasks but up to now they used to be executed on individual computers. The goal of this project was then to help the RP group to increase efficiency of their simulations by improving the utilization of their computing resources.

1.3 Document structure

The project includes an initial investigation of the problem to define the cluster's mission and to select cluster technologies. After that, the project advanced to the development and implementation stage, which was performed in two phases. These topics are structured as follows:

A deep problem analysis is presented in chapter 2. It includes the simulation software, its working environment, how people use it and what can be improved. This information will be essential to define the system's objectives.

In the chapter 3 is given an introduction to clustering systems and explained which kind of them would fit in this project. A comparison between some specific software packages is performed, regarding the existing requirements, to select the most appropriate.

Chapter 4 presents in detail the installation of the cluster. It includes the design of the system's architecture, the definition of requirements and the most important decisions and algorithms which contributed for the configuration of the cluster's management software. Additionally, this chapter includes a section showing the working behavior of the cluster and its performance analysis.

Chapter 5 includes the details about the development of additional tools and front-ends, which provide a high-level interface for the user. Initially the solution is globally structured, after what each component is analysed, designed and implemented. At the end some results and respective analysis is presented.

Chapter 6 concludes this report with an overview of this project, including its contribution to the RP group and future possible developments.

Chapter 2

Simulations performed at RP group

In this project there is a well-defined central element: the simulation software. Therefore it's required to analyze in detail the behavior of this software, its specificities, constraints and requirements so a new system can be designed optimally. Besides the simulation software, there is a whole system environment where the simulations currently run: hardware, operating systems, communications and installed software. These factors also introduce restrictions which mustn't be neglected.

2.1 CERN Radiation Protection group

At CERN there is an specific division to account for Safety: the Safety Commission [10,11]. This division consists of specific groups: Fire Brigade, Integrated Safety & Environment, Radiation Protection, General Safety and Medical Service.

In the context of a nuclear research organization, there are a number of challenges regarding radiation protection. Therefore a dedicated group for Radiation Protection [12] (RP) was established with the objective to assess the hazards connected with radiation and radioactivity, to ensure human safety on site and assist all those working at CERN in protecting themselves from such hazards [13]. To accomplish this objective the group carries out several activities, already from the design phase of an accelerator and during its whole life cycle. Among them, it's this group's responsibility to:

- Advice in operation of current and in the design of new accelerators;
- Design shielding of workplaces mitigating effects of beamlosses;
- Estimate induced radioactivity both in equipment, air and water, and monitoring.

For these tasks, Monte-Carlo Simulations are widely used in the RP group at CERN.

Generally, simulations in particle transport started in late 1960's and became of great importance because of its support in:

- Radiation therapy in cancer treatment
- Simulation of the properties of radiation detectors
- Design of particle sources and accelerator
- Design of shields in high intensity radiation areas.

Notably, these characteristics relate to the RP group activities.

2.2 Radiation transport simulations

For the particle transport problems there are existing two popular simulation software packages: MCNPX [14], developed at Los Alamos National Laboratory (U.S. Department of Defense), and FLUKA [15] developed by a collaboration between CERN and INFN (Istituto Nazionale de Fisica Nucleare, Italy). Therefore the RP group has a much more intrinsic connection to the FLUKA project.

FLUKA is a tool for the calculation of particle transport and interactions with matter [16]. In other words, it calculates the path of inserted particles, as well all the events that may occur, which means collisions, new particles, heat, etc. Since it started being developed back in 1962-1967, the implementation of the physical models has been improved continuously [17]. Currently, FLUKA is at its third generation, simulating interactions of about 60 different particles with high accuracy and handling very complex geometries.

From a technical point of view, FLUKA is programmed in Fortran Code and already counts about 470.000 lines of code, handles data in double precision, supports additional user routines and floating point exceptions. It's compiled with g77 and available under Linux and "Digital-Unix".

FLUKA increases its efficiency by using combinatorial Geometry and a careful choice of algorithms. Nevertheless, its execution time is heavily dependent on the simulation conditions:

- Complexity of the geometry;
- Number and type of particles;
- Number of cycles required;

- Events occurred in the simulation, like interactions also with newly created particles.

These calculations are computationally very intensive and, depending on the previous factors, they can take up to several days, or even weeks, to achieve good results. So, even the first three factors are known to directly influence the length of the simulation, it's very hard to predict how much time it will consume since the last factor is not predictable. And because of the Monte-Carlo principle of using a random original state, each cycle usually has a slightly different duration.

There are also some key specificities of FLUKA, regarding its execution. FLUKA is run through a shell script with a major role in the simulation execution:

1. Creates a directory structure with the necessary files to the simulation to be run;
2. Manages to run the simulation a number of cycles the user defines;
3. Runs the user provided FLUKA executable per each cycle;
4. Redirects standard input, standard output and standard error stream between the FLUKA executable and cycle dependent files;
5. Handles file dependencies between execution cycles (e.g.: the resulting seed from one cycle is the starting seed for the next one);
6. Moves important files to the main directory and removes the created directory structure.

This execution properties introduces some constraints to a clustering approach, since most execution platforms do not support some of them. For instance the execution through a shell script, the stream redirection and the file dependencies between sequential runs generally have poor support in cluster platforms.

FLUKA is also a closed-source software which excludes any approach to parallelize or adjust to a specific execution environment.

2.3 The previous simulation system

2.3.1 The original problem

Starting with the previous system itself, it was composed by common independent personal computers, which usually have 1GigaByte RAM, operate at about 2GHz and run Scientific Linux CERN 4 (SLC4), a specific CERN Linux distribution based on Red-Hat Enterprise Linux 4 (RHEL4). Most of these computers were assigned to one or two users, who started individual simulation jobs manually, as a common unix processes.

Mainly because of the long nature of simulations, this approach to distribution, by statically assign different machines to users, had many drawbacks. But besides being long, the users' execution profile of simulations significantly increases the complexity of this problem, as:

- Users submit neither the same number of simulations nor computationally equivalent ones;
- Users submit simulations with very different frequencies;
- Simulations don't show up at a constant rate, but mainly in bursts;
- Many projects require several simulations, prepared almost at the same time;
- Some projects have priority over others.

As a general result, the more powerful computers got rapidly occupied (with only one or two simulations) for a long time period while many others remained completely free. This situation introduces the main problem of the system: **inefficiency**. But still other significant drawbacks can be mentioned:

- Inflexibility: machines were statically assigned a user neither regarding its computational needs nor the ongoing projects. Access to other machines would therefore not make sense since all personal files were only on their own computer;
- Lack of fault-tolerance: An error in one simulation would crash the process, meaning it would need to be restarted manually. Even worse, a failure of one machine would cause the previously mentioned problem, plus progress loss of **all running simulations**. Although not common, the latter could be responsible for the loss of weeks of computational power.

2.3.2 A first try

To mitigate the problem the group had tried a smart yet simple approach: to provide users easy access to all the machines in the group. So, using the NIS centralized credential system and a shared file system (NFS) a user could login on another machine (using his own credentials) and start a simulation job on it. Additionally, users were transparently copying files to the NFS server which introduced some fault tolerance.

Still, this attempt was not very successful, mostly because:

- Users didn't have a way to know the available machines;

- The additional effort and time needed simply didn't make it worth or, at least, discouraged its usage;
- Even starting simulations in different computers, the granularity of the problem was still at the scale of a simulation job, which is commonly a rather long task. This would again narrow efficiency, as the increase of resources wouldn't mean the increase of performance, unless more simulations were submitted. For instance, if there were 10 simulations of 7 days each and 20 computers available, it would need 7 days to complete as 10 computers would be idle all the time.

At this step the group realized that some kind of automatization of this task was needed. But the task of selecting a system which would automatically handle jobs from this simulation software became not as straightforward as expected. The software has such characteristics that place severe constraints on the approach to the problem.

2.4 The project

Clusters can increase performance to help solve complex problems, but undoubtedly they increase the complexity of the system. Beyond that, clusters are highly dependent on the service they will provide, so they must be carefully planned. In cluster design, there are 4 steps to guide the system architect [18], which were adopted and adapted to the current project plan.

- Determine the overall mission for the cluster;
- Select a general architecture for the cluster;
- Select the operating system, cluster software and other system software to be used;
- Select the hardware for the cluster.

As briefly stated in the introduction, it will not be possible to plan a system from scratch; instead the current infrastructure must be preserved, meaning that the new system will be limited to a software platform which must perfectly fit this infrastructure and work towards our main objective: optimize global system efficiency.

Also, because we're unable to change the simulation software on the source code level to run it in parallel, our cluster will have to be managed by a software working as a scheduler, even if it has support for more powerful execution environments.

The main objectives were, therefore, to set up a fully functional scheduling system, tailored to FLUKA and its actual usage profile, providing a friendly, fast and ease-of-use interface for both simulations and the cluster management.

Compiling all the project's information and meetings with supervisors and real users, one can define that the aim of this project is to implement a system which:

- Optimizes distribution of simulation jobs through all the available computers in the cluster;
- Considers jobs characteristics:
 - Priorities between simulation projects;
 - Time limitation;
- Considers machines characteristics:
 - CPU speed;
 - Available CPU time;
 - Available memory;
 - Common desktops (prioritizing users' processes) or dedicated servers;
- Enforces fair share between all users;
- The underlying cluster layer is transparent to the user, but still provides him sufficient information and control;
- Administrative tasks over the cluster don't require in depth system knowledge;
- Allow for easy cluster expandability;
- Provides fault tolerance and security.

As proposed in the project guidelines and refined at the first project meetings, the system will be developed in two phases which were partitioned in the following master tasks:

1. Getting to know the current simulation system, including computing resources and the simulation software;
2. Investigate about the available cluster platforms and choose the most appropriate;
3. Plan a configuration policy for job execution;
4. Implement cluster configuration;
5. Beginning of second phase of project - Design of tools and interfaces for the cluster;
6. Develop cluster tools and interfaces.

The production system is expected to grow up to 30 machines, so performance issues are widely taken in account.

Chapter 3

Linux clusters analysis

In computing, there are three basic approaches to increase performance: a better algorithm, a faster processor or divide the calculation among multiple computers. While in many situations the first two approaches are no longer viable, parallelization opens a new window to performance needs to the most domains.

3.1 Cluster systems

By definition, a cluster is a group of computers that work together [18]. It has three basic elements:

- A collection of individual computers;
- A network connecting those computers;
- A software that enables a computer to share work among the other computers via the network.

In this project a Beowulf Cluster is to be set up, but is important to stress that a number of variants exist as well. Beowulf clusters are probably the best-known type of multicomputer because they're constructed using *commodity off-the-shelf* (COTS) computers and hardware. This advantage specially reflects both on the availability of components and the price-performance ratio. On the other hand, commercial clusters often use proprietary hardware and software. Compared to commodity clusters, the software is often tightly integrated in the system and there's a better match between CPU and network, however they usually cost from 20 to 50 times more.

	HPC	HTC
Metric	FLOPS	FLOPS extracted
Ownership	Centralized	Distributed
Idle cycles	Lost	Captured
Optimizes	Response Time	Throughput
Memory	Tightly-Coupled	Distributed
Designed to run	1 job	1.000 jobs

Table 3.1: Comparison between HTC and HPC

3.1.1 Cluster Types

Clusters have also evolved in different ways depending on their purpose. One can differentiate between High-Performance-Computing (HPC), High-Throughput-Computing (HTC), High-Availability (HA) clusters and Load-Balancing (LB) clusters. The concept between HPC and HTC is often mixed up and, actually, there's always some overlap between all of these classifications. The main characteristic that distinguishes between HPC and HTC is the fact of the latter being designed for better global performance over a long period of time, instead of the best performance for a short period of time.

Table 3.1 on page 14 shows a comparison between both concepts. High Performance Computing is especially used for very complex problems, where maximum processing power is desired for a specific problem in order to get it solved in the minimum time. Because of their great success in some very well known problems (NUG30 [19], Kasparov defeated by Deep Blue [20]) as well as the performance records breaking (IBM reached 1PetaFLOP [21]) HPC tends to have higher impact.

High Throughput Computing is specially designed for systems where a large number of jobs are concurrently executed. Therefore they focus on the global system's performance (usually jobs per unit of time) instead of a single job performance.

High-Availability clusters, also called *failover clusters*, rely on redundancy to provide fault-tolerant systems, usually required for mission-critical systems. In such systems, there are several "mirror" systems whose only objective is to take the place of the master system whenever detected problems in its responsiveness, by means of a power cut-off, hardware or software failure, etc.

Load-Balancing clusters specialize in distribute independent tasks to increase each one's performance. A good example are web servers, where the queries are usually spread over the computers in the cluster.

3.1.2 Distributed computing

“Distributed” and “Parallel” are both terms commonly used in cluster systems. Yet, there’s a substantial difference between both, principally related to the architecture. Usually, parallel computing refers to tightly coupled sets of computation having a homogeneous architecture. A simple example is a dual core CPU which is running two threads *in parallel*. Distributed computing is more correctly used for describing clusters, as the term implies multiple computers or multiple locations, typically forming a heterogeneous system. Also, distributed computing is more likely to be executed asynchronously rather than parallel computing.

Still, clusters are just one type of distributed computing. Referring carefully to how distributed computing was presented, one can think of other “multiple computers or multiple locations, typically forming a heterogeneous system” than clusters. It’s the case of the very well-know “*grid*” and *peer-to-peer* projects, and some more sophisticated clusters, like *Federated clusters* and *Constellations*.

The idea behind the grid is to provide computing power as commodity, using LANs or the Internet. This is a state-of-the-art subject, which has received much attention from the scientific community, because it has the potential to provide unprecedented computational power by combining many different and heterogeneous computing sources, especially clusters.

3.2 Cluster software review - Job Management Systems

There is a number of software packages which would be able to distribute processes in a cluster. However, regarding to the requirements of this project (Section 2.4) and all its surrounding environments, it is required a software which can be fully personalized and optimized by defining rules. There are many convenient and simple systems which fail in this point. For instance, openMosix automatically and transparently migrates processes running on the local machine to others in the cluster by means of patching the Linux kernel, however it does not allow for scheduling profile set up, neither allows for FLUKA-process-migration since it directly manipulates I/O.

By the previous overview of cluster types, one can easily deduce that the one which better matches the RP group specific needs is HTC. Still, there is a more specific definition for HTC software which handles jobs and distributes them through a cluster of computers: Job Management System (JMF).

The main purpose of a Job Management System (JMS) is to efficiently schedule and monitor jobs in parallel and distributed computing environments, also known as workload

management, load sharing, or load management [22]. The JMS's objectives are the ability to leverage unused computing resources without compromising local performance, to work on very heterogeneous systems, and to allow cluster owners to define cluster usage policies [23], which dictate their advantage on this project, among other cluster software packages.

3.2.1 Job Management System's general architecture

In order to be successful in such objectives, it's necessary a JMS to perform some important tasks [22]:

- monitor all available resources;
- accept jobs submitted by users together with resource requirements for each job;
- perform centralized job scheduling that matches all available resources with all submitted jobs according to the predefined policies;
- allocate resources and initiate job execution;
- monitor all jobs and collect accounting information.

To execute those tasks, generally all JMS are based on a distributed architecture, composed by the following functional units:

1. Queue Manager (also known as User Server): the unit to which the user can submit their jobs to the JMS with the information about the resources to allocate;
2. Job Scheduler: the unit which performs job scheduling based on the job properties, available resources and administrative policies;
3. Resource Manager: monitors the available resources on an execution host, and dispatches jobs.

It is usual for the scheduler to maintain a database of all the available resources in the cluster. After a job have been assigned to an execution host, the entry of the claimed resource is removed from this database and the control and monitoring of job is delegated to the user's queue manager. This distributed behavior is extremely important to avoid a potential bottleneck on the central node, contributing to a very expansible system.

3.3 Comparative analysis of available JMS

There are a number of available JMS software packages, both commercial and public domain, which could be used for this project. Three representative JMSs, which are probably the most widely used ones, are analysed and compared among them:

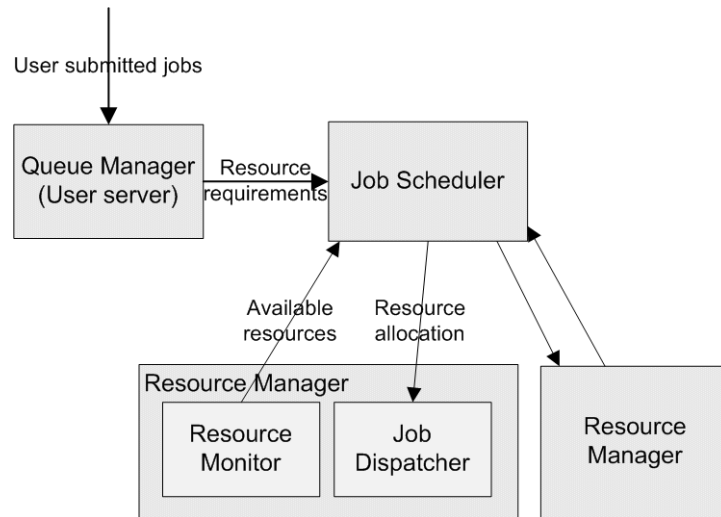


Figure 3.1: JMS general architecture

- Portable Batch System (PBS);
- Sun Grid Engine (SGE);
- Condor.

PBS is a system initially developed by Veridian Systems in the early 1990s. Its purpose was to substitute the Network Queuing System (NQS) from NASA Ames Research Center. It's currently available in both commercial and open-source versions: PBSPPro, acquired in 2003 by Altair Engineering, and OpenPBS which is the original and unsupported version. Yet, some open-PBS based projects are being developed, like Torque and OSCAR, which implements some additional features and integrates with other systems.

SGE is an open-source package from Sun Microsystems Company. It evolved from the Distributed Queuing System (DQS) developed at Florida State University, and is particularly known by its well developed GUI which enables complete management of the cluster. There is also a commercial version of SGE, called CODINE, which is also gaining some popularity.

Condor is also an open-source project, developed at the University of Wisconsin. It's designed specifically for High Throughput Computing and CPU harvesting and so was one of the first systems to take advantage of idle CPU cycles and to support process checkpointing and migration.

Since the project should be implemented over an open source platform, the previous candidate software packages were already filtered taking such parameter in account. If not, there would be at least another strong candidate, the Load Sharing Facility (LSF): a JMS evolved from the Utopia system developed at the University of Toronto which is one of the most widely used JMS [22].

3.3.1 Selection Criteria

In order to select the most appropriate software package, a set of criteria was defined to allow direct comparison.

These criteria are organized in 4 groups and represent the factors which influence the execution of the simulations.

Group 1 contains parameters regarding evaluate the usability of the platform itself, while a software product. Group 2 contains the constraints of the simulations. Group 3 and group 4 specifies parameters which should be present (not necessarily compulsory) but would allow for performance improvements for the specific simulation environment.

1. General criteria:

- (a) Platforms supported. Linux is necessary for FLUKA execution, but Windows support would useful if some windows software is to be run on the cluster later on.
- (b) User Interface. Monitoring GUI would be useful.
- (c) Support/Documentation. It should be as complete and current as possible.

2. Job Support

- (a) User defined job attributes. We must be able to define priorities and length information on each job.
- (b) FLUKA specificities. Please refer to 2.2 on page 9

3. Scheduling

- (a) Multiple queues. In order to provide expandability and ease-of-use, it's desired one queue per each main submission machine.
- (b) Job control. The level of control a user has over a submitted job, for instance to kill, suspend or change the execution node.
- (c) User defined scheduling policy. The administrator should be able to define in which cases, which resources of node can be used. For example, define that nodes should not be used when keyboard is used. This is a very important feature which allows scheduling optimization to increase throughput.
- (d) Fair share. The system should be able to track users' resource history to ensure fair cluster usage among them.

4. Resource management

	OpenPBS	SGE	Condor
1.a) Platforms supported	Linux	Linux	Linux & Windows
1.b) User Interface	Command line & limited GUI	Powerful GUI	Command line and web tools
1.c) Support/Documentation	No/Poor	Very good	Very good
2.a) New job attributes	No	Yes	Yes, using language ClassAds
2.b) FLUKA specificities	Yes	Yes	Yes, using Vanilla universe
3.a) Multiple queues	Yes	Not explicitly possible	Yes
3.b) Job control	Yes	Yes	Yes
3.c) User defined policy	Poor / good if using Maui	No	Yes, fully customizable
3.d) Fair share	Only available through Maui	Yes	Yes
4.a) Node configuration	No	Yes	Yes, specific configuration file
4.b) Fault tolerance	Low	Job migration	Checkpointing, Job migration
4.c) CPU Harvesting	No	Yes, defining sensors	Very good, fully customizable
4.d) Security	Authentication	Authetication	Authentication & encryption

Table 3.2: HTC comparison table

- (a) Node configuration. The administrator should be able to define node specific configurations, depending on the machine resources (e.g. between desktop and server computers).
- (b) Fault tolerance. The ability of the system to prevent or recover from a failure. For instance move a job from a machine which suffered a power cutoff.
- (c) CPU Harvesting. The process of exploiting non-dedicated computers (e.g. desktop computers) when they are not used. The importance of this feature goes beyond performance reasons, as a considerable part of resources are desktop computers whose performance should not be affected.
- (d) Security. Each cluster user is responsible for his jobs, and should never be able to modify other users' jobs.

3.3.2 Result analysis

The most important characteristics of the software packages for this project are selected and compared in Table 3.2 on page 19.

Characteristics from Group 1 advance SGE because of its powerful GUI and very good support and documentation from Sun. Right after comes Condor, whose Windows support is an advantage for possible sequent projects. OpenPBS didn't perform well in this test, and the lack of support and stagnation of development will definitely contribute to a bad ranking.

In Group 2 both Condor and SGE fulfill the requirements for the simulation software, still Condor has a slight advantage because of its extensible ClassAds mechanism. PBS suffers from another significant drawback by having no support for user defined job attributes.

The third group reveals good scheduling performance from Condor and OpenPBS when integrated with Maui scheduler [24]. Maui is an external scheduler which can be used in conjunction with a number of other resource managers to extend their functionality. It provides very advanced scheduling policies and thus allows OpenPBS to stand up to Condor, which has a very good built-in scheduler. Unexpectedly, SGE performed bad at this test, losing its advantage over Condor, since it does not support custom algorithms, advanced policies, neither preemption.

From group 4 one can check that both SGE and Condor fitted the requested profile, by providing good support for node configuration which allowed CPU harvesting to be also available. Again, OpenPBS fell down at this point because of its limitations in nodes configuration.

Conclusions Every system has its advantages and drawbacks and, even inside such a specific field, one can notice that each system is intended for slightly different objectives. SGE system tends to be popular because of its graphical interface and great support/-documentation, allowing for easy installation without requiring in-depth knowledge of the system. OpenPBS with the Maui scheduler and Condor are specially intended for HTC, thus allowing for full customization of the scheduling policy. SGE was not developed with such intention, and so does not allow for so sophisticated policies as the previous systems.

Being a very well known system, OpenPBS has some external modules which increase its potential; however the lack of support introduces a great obstacle to its adoption. In turn, Condor has been continuously developed and supported by the Condor group at University of Wisconsin. It is extremely customizable, designed for CPU harvesting and the additional modules make it one of the best software packages for implementing HTC over an existing platform of desktop computers.

Chapter 4

Setting up the Condor cluster

Setting up a Condor cluster is a process which may take from one hour to several weeks, depending on the system to be implemented and the amount of customization it requires.

For Condor, each resource has an owner, who has absolute power over his own machine, and so he's free to define a local policy. On the other hand, a user who submits a job can freely specify its requirements and surely wants his job to get as much processing cycles as possible. The role of the administrator is to configure Condor to maintain the equilibrium between both sides and so achieve the maximum output from the system.

4.1 Architecture

A Condor pool is comprised of a single main server, the *central manager*, and a number of other machines that may join it. Depending on the local configuration, each machine can play the role of:

- **Central Manager:** The machine (only one per pool) responsible for collecting data and to negotiate jobs with the available resources. Condor uses two separate daemons for these tasks so, in special cases, one machine for each daemon can be used. As single daemons, they should be installed on reliable machines with a good network connection to all other machines;
- **Execute:** The machines can be configured to provide execution resources to the pool. It can be any machine, including the central manager, and each may specify its own execution profile;
- **Submit:** Any machine can be configured as a job submission point to the cluster. Because each submitted job creates an image process in the submission machine, this machine should have a fair amount of memory, depending on the number of jobs to be run through it;

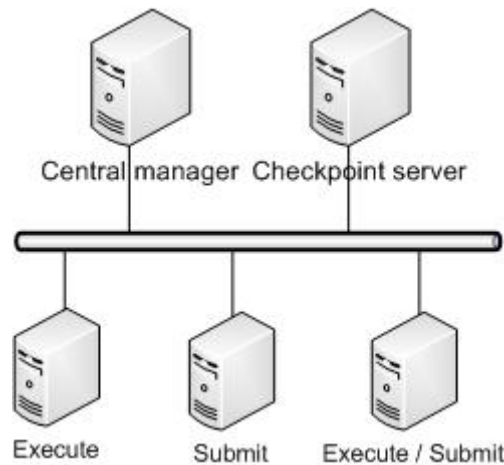


Figure 4.1: Machine roles in Condor

- **Checkpoint Server:** One machine (and only one) in the pool can be configured to store all the checkpoint files from every job in the pool. Therefore it needs a large disk space and good network connection.

Figure 4.1 on page 22 shows a usual configuration of a Condor system, where the Central Manager and the Checkpoint server are dedicated machines (but they do not need to), and all other machines are Execute, Submit or both just as needed.

4.1.1 Condor execution modes

In Table 3.2 on page 19 we checked that Condor supports FLUKA processes through the Vanilla universe. Universes in Condor are execution modes which are designed for a specific application group. Currently Condor has 8 built-in universes: Standard, Vanilla, MPI, Grid, Java, Scheduler, Local, Parallel and VM.

The Standard universe provides some of the best functionalities in Condor, including *checkpointing* and *Remote System Calls*. Checkpointing is the process of creating an image of the current job state and save it into hard-disk, so whenever the job needs to be moved from one machine to another (e.g. a failure in the machine, better resources) it can restart from the point it just left off. Remote System Calls allows the program to execute the system calls (e.g. accessing to a file or networking) on the machine it was submitted. Therefore the program behaves like if was executing in this machine but just using other machine's processing power.

The Standard universe can be used with a wide range of applications; however they must be relinked with the Condor libraries and meet a few restrictions. For instance, programs must not create sub-processes, nor use pipes, semaphores, shared memory, kernel-level threads, etc. Whenever our program does not meet the previous restrictions, one

have to use the Vanilla universe. This is the most flexible Condor universe, allowing any UNIX process to be executed, but does not support the useful functionalities from the Standard universe.

In our simulation system, there is no choice. The FLUKA execution, as analyzed in 2.2 on page 9, falls into the Standard universe restrictions. Therefore, the Vanilla universe must be used and, as consequence, we won't be able to create checkpoints nor to execute remote system calls.

4.1.2 Condor cluster requirements

Without checkpointing, the checkpoint server is useless and so the system machines are limited to three roles: Central Manager, Execute and Submit. However if, on one hand, the system gets simpler, on the other hand we need to implement a mechanism which provides the running machines access to the files located at the submission machine.

4.1.2.1 Shared file system

In order to handle the remote files access problem, Condor itself has an internal mechanism to copy files to the execution node and move others back to the submission one. These mechanisms are generally known as "file stage in/out. Despite integrated into condor, its "File Transfer Mechanism" has several drawbacks:

- The files that are to be transferred, other than the executable and the job submission file, must be explicitly set;
- Problems in communication may prevent the transfer of the result files back to the submission machine, and therefore their loss.
- Requires enough free hard drive to allocate copied files;
- Causes high network load while transferring the files.

In the context of this project, this option is not reasonable because each simulation might need several auxiliary files.

The solution is, therefore, to provide both computers transparent access to the same directory, which can be achieved through a shared file system. Indeed, Condor documentation refers this as the preferred method for running jobs running in the Vanilla Universe [25]. Using a shared file system, one can setup every computer in the cluster to mount the shared directories in the same path, creating the same directory structure in all the machines. This is especially useful as users may use full paths within the job configuration, as long as the path refers to a shared location.

For Linux, both the Network File System (NFS) and the Andrew File System (AFS) are very popular systems. NFS was originally developed by Sun Microsystems in 1984 and became the first widely used network file system. This is now commonly built-in in most Linux distributions and its features and simplicity of usage perfectly fit this project's needs. AFS is a distributed network file system which has some advantages over traditional network file systems as it concerns with security and scalability. However, these features are not necessary to our system and actually AFS does not currently have a way to authenticate itself to AFS [26], which means processes would have to use other method of output writing.

4.1.2.2 Centralized authentication system

In order to a shared file system to operate correctly, one must not forget security issues. Linux controls access to files by checking the authenticated user identifier (UID) against the file permissions he's accessing. On the other hand, user account operations (including creation) mostly deal with a more human-readable identifier: the username. This means users will have a different UID on each machine even creating the account with the same username, unless the UID is explicitly defined.

For a shared file system like NFS, this can turn to be a serious problem. Since the same username maps to a different UIDs on other machines, users aren't recognized as the same and therefore won't be allowed to do the same operations over the files. For instance, the owner of a file won't be recognized as such on the other machines, having no control over file permissions and probably no write permission.

Even preserving consistency when setting up credentials, with the correct UID for each user on each machine, the expansion of the cluster would get seriously compromised. The addition of a new cluster node would require the correct set up (including UID) of all the system users; and the addition of a new user would require the configuration of every cluster node, which is an unacceptable situation even for a small system.

The solution is therefore to install a centralized authentication system, where an authentication server holds all the user accounts' information. There are a number of advantages in using such approach:

- The same credentials are used each time the user logs in, no matter the physical location;
- All the account management operation are executed only once, like adding, removing and changing user accounts;
- Little client configuration;

- Data is guaranteed to be consistent and securely stored in one (or more) server(s).

In the context of this project, the requirements for such a system were:

- To integrate with Linux, in such way this mechanism was completely transparent to the user and generally to all software applications, especially Condor;
- To provide fault-tolerance, since authentication is a vital service for the network computers;
- To be of simple usage, regarding mainly installation and administrative tasks.

There are a few systems which do the required tasks. For this system we might consider the Lightweight Directory Access Protocol (LDAP) and the Network Information Service (NIS) [27], which was originally called Yellow-Pages service and is currently integrated into most Linux distributions.

From these systems, NIS actually fits the requirements, and provides good solutions to the last two. It allows for the configuration of a secondary NIS server, which takes the place of the primary server in the case of unavailability, and allows for data synchronization as well. Account management is as simple as if they were locally registered, and new accounts are just created on the server.

4.1.3 Physical architecture

Now that all the required services are gathered, they must be assigned to the available machines. This task must be done carefully to avoid inefficient load distribution, and take into account the role of the services, characteristics of the machines and to meet each service constraints. Summarizing:

- NFS server must be located in a computer with high storage capacity;
- NIS primary and NIS secondary servers must be installed on different machines;
- User's machines should not be used for server services neither significantly changed, as these machines are still used as desktop by its owner.
- Each user's machine should be Submitter, so users can submit jobs from its own machine and the amount of memory the submitter requires is only dependent on user's jobs.

From the perspective of the load each service represents, it's possible to combine services so computers can be more efficient used. NIS is an extremely lightweight service so it

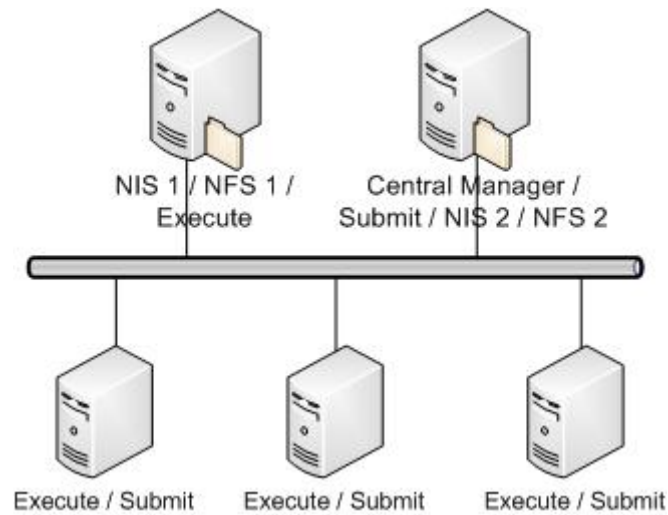


Figure 4.2: System physical architecture

can easily cohabit with other services. NFS introduces quite a heavy load for the hard drive (I/O), yet it leaves CPU almost idle. Condor Central manager is not such a specific service, mainly requiring CPU and memory, while Condor Submit mostly requires memory. Condor Execute depends on the job it's running but, in the current system, the simulation jobs consume as much CPU as available and also a large amount of memory, usually above 100 Mega-bytes.

Regarding the desktop computers, there aren't many changes. They will still exist but running additional software: NIS and NFS clients, Condor Execute and Condor Submit.

On the other hand, server processes have to be organized into dedicated computers, supporting uninterruptedly the network. At least two servers are needed (let's assume Server1 and Server2) due to NIS. Since this service is responsible for a very light load on the machines, some other processes can be installed in conjunction with it. So, regarding the other vital services, a NFS main directory was configured in Server1, and Condor Central Manager in Server2.

This system configuration meets the requirements but can still be improved in order to optimize load balance. Since NFS is a quite heavy process and Server2 processes do not introduce heavy load on I/O, a secondary NFS directory server can be set up on Server2 to handle some client's data. In turn, Server1, which alike Server2 is quite a powerful machine, was not assigned any high CPU demanding process. To take advantage of it's free computing power, Server1 was also designated to be a job execution machine, although in half mode¹. Figure 4.2 on page 26 shows the two servers (top), the common desktop

¹Running Condor in half mode means it will only take advantage of one out of two CPU cores. Details how it was achieved can be found in the implementation section .

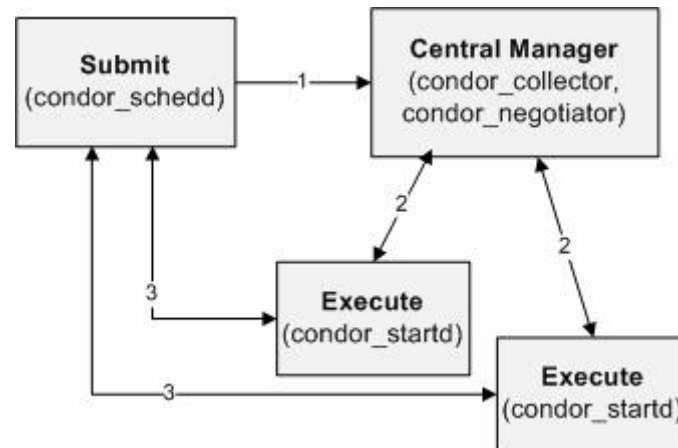


Figure 4.3: Condor main daemons

computers (bottom) and their running services.

4.2 System specification

Since the role of each computer is defined, the very next step is to install Condor and configure it in order to meet the simulations' requirements and work towards the best performance of the cluster. For the system to work as expected, a policy for the system must be modeled upon the objectives defined at 2.4 on page 11. This policy will afterwards be translated to a series of rules implemented into Condor's configuration.

4.2.1 Condor jobs handling

For the policy to be designed, it's important to know the states a job can go through while under the control of Condor.

Condor is based on a set of distributed agents - the condor daemons (Figure 4.3 on page 27). After a job being submitted into Condor, the scheduler daemon (condor_schedd) advertises it to the Central Manager (4.3 - information flow 1). The collector daemon (condor_collector) stores both jobs and resources' information in an internal database so it can be used by the negotiator daemon (condor_negotiator), which tries to match jobs and resources, based on their specification and, of course, the defined policy. After a job had been successfully assigned to a machine (2), the start daemon (condor_startd) handles the job and both entries (the job and the machine resource advertises) are removed from the collector database and the control of the job is delegated to the scheduler (3). From this point the job the scheduler monitors the job progress and though it the user can remove the job from the system, or ask the negotiator to re-match it. On the other hand, the start daemon may pause, continue or preempt the job based on the policy.

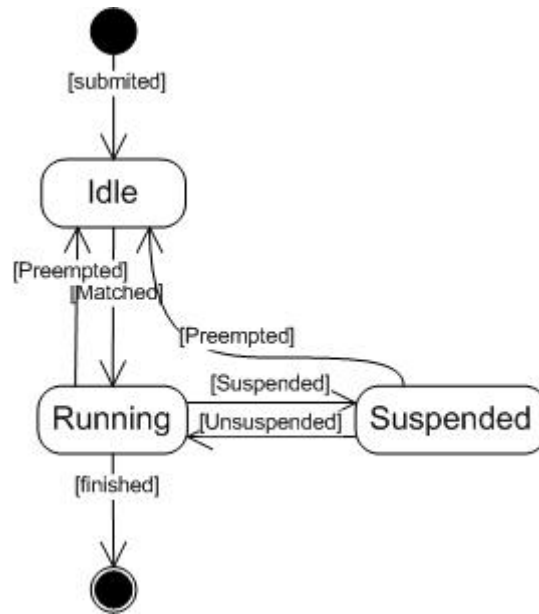


Figure 4.4: Condor job state diagram

Having a job preempted or explicitly request Condor to re-match it, will make the job to leave the resource and wait until the condor negotiator provides it a new resource.

So, from the job point of view, a simplified model for Condor jobs has three states:

- Idle, when the job is in the queue waiting for a resource to match it.
- Running, when the job is actually getting progress in the execute machine.
- Suspended, when the start daemon paused the job, but it remains in the execute machine.

So, both Running and Suspended states only exist when the job is matched and actually occupying a resource. A state diagram including all transition events is shown in Figure 4.4 on page 28.

4.2.2 A policy for simulation jobs

Since in Condor's Vanilla universe the jobs cannot checkpoint, the cluster should avoid preempting the jobs frequently, since they wouldn't be able to resume progress.

Setting up Condor to efficiently manage resources of a cluster without checkpointing is the biggest challenge of this project, and inserts a complex trade-off: should jobs always keep their resources or should they leave them (losing some progress) in order to search for a better one?

Consulting the RP group, a policy which optimizes the overall performance of the simulations jobs was defined as follows:

1. Definition of new attributes

- (a) Characteristics of the jobs
 - i. Jobs are marked as High, Normal or Low priority.
 - ii. Jobs are marked as Long or Normal time length.
- (b) Characteristics of the machines
 - i. Nodes are defined as Desktop or Server.

2. Behavior regarding job distribution

- (a) Prioritize local user processes as:
 - i. Condor jobs should only start when local processes leave enough CPU time available
 - ii. Condor jobs may be *suspended* in order to make room for new local processes.
 - iii. *Suspended* jobs may continue when CPU gets available again.
- (b) Rank available machines to select which will run the job:
 - i. Rank by Condor load on the machine, to avoid substitution or, at least, substitute the lower priority jobs.
 - ii. Rank by machine performance.

3. Behavior regarding job priorities

- (a) Only administrators authenticated as "condor" can submit high priority jobs.
- (b) Higher priority jobs may occupy other jobs' resources (whenever machine selected - point 2(b)i) as:
 - i. A Long job *preempts* the previous one.
 - ii. The previous job is *suspended* in the remaining cases.
 - iii. The previous job is *unsuspended* when the higher priority frees the resource.

4. Behavior regarding machine characteristics

- (a) Server nodes will run jobs with default process priority (nice = 0).
- (b) Desktop nodes will run jobs with lower process priority (nice = 15).

5. Fail-safe behavior and global optimizations

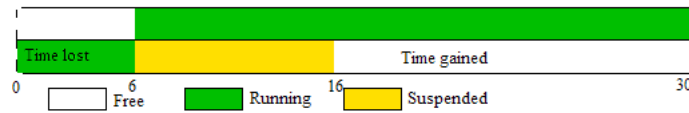


Figure 4.5: Preempt after suspension

- (a) Jobs not marked as Long are limited to 7 days processing time, after what they are removed from the system.
- (b) Jobs suspended for more than 10 hours are *preempted*.

This policy relies on suspension to avoid lose current job's progress. Nevertheless, when jobs are rather long, it may not worth to continue suspended because (1) they, much probably, will remain suspended a long amount of time and (2) other resources may have become free. The rules 3(b)i and 5b tries to address this situation, when it's known the job is Long (being marked as Long) and when the job is experienced Long as it's using the resource for more than 10 hours.

The reason behind 10 hours limit is related to the usual long duration of the simulation: if a simulation ran for 10 hours it isn't for sure a small one and will probably take more than one day, so it's worth to lose some progress. As illustrated in Figure 4.5 on page 30, the job that started in first place is suspended because a 24 hour job arrived; but after 10 hours it gets preempted and will be able to execute on another machine. Because simulations are composed by cycles, the duration of the cycle represents the highest limit for the time loss. This value depends greatly on the simulation nature but nevertheless is a fraction of its total time.

4.2.3 State transitions

Summarizing, the system is intended to behave as explained in the following job state diagram (Figure 4.6 on page 31), whose transitions can be described as:

1. When a job is submitted to the queue, the scheduler agent automatically sets the job state as "Idle";
2. One job starts running when it's *matched* to a machine. This occurs when the system has enough CPU free (policy rule 2(a)i) and its priority specific slot is available (policy rule 3b).
3. One job is *preempted* if when the new job is long (policy rule 3(b)i).
4. One job is suspended when a higher priority claims its resources (policy rule 3b) or when CPU load has increased due to local processes activity (policy rule 2(a)ii)

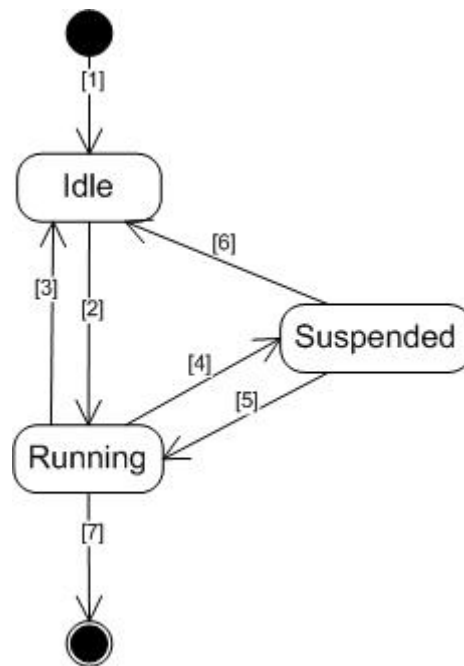


Figure 4.6: System state diagram

5. One job is unsuspended if a job freed its resources (policy rule 3(b)iii) or CPU load decreased until acceptable levels (policy rule 2(a)iii).
6. One job is preempted if it's suspended for too long (policy rule 5b).
7. One job may be removed from the system as long as it finishes or if it's running for too long (policy rule 5a).

4.3 Implementation of the Condor configuration

After the installation, which sets up a default configuration, Condor can be customized through a set of configuration files whose syntax is based on key=value pairs which support macro substitution. There are several configuration files, which are loaded sequentially:

1. global configuration file,
2. local configuration file
3. global root-owned configuration file
4. local root owned configuration file
5. specific environment variables prefixed with condor.

If the same definition appears in more than one file, only the latest will remain, and so root configurations and then local configurations have priority.

The global configuration file, which is located in `etc/` inside condor's directory, specifies all the common parameters that control the behavior of the Condor daemons. Then, machine specific parameters are loaded from the local configuration file.

4.3.1 Centralized Condor global configuration

Being the global configuration file, it's important to ensure its integrity and consistency among every machine in the cluster. Also, it should be easy to propagate changes made to the file.

Since NFS server is configured on the Central Manager, which also needs the same configuration file, a simple yet complete approach is to share the global configuration file's directory and mount it in the exact same location on each Condor's node.

Sharing the same file, consistency and changes propagation are automatically met. Integrity and security can be enforced by sharing the directory as "read-only", thus all modifications must be performed as root on the Central Manager.

4.3.2 Defining new attributes for jobs and machines

Condor supports the inclusion of additional attributes into the job description structure. For that the attribute should be passed to `condor_submit` executable as a parameter² or inside the job configuration file in the form "+attribute = value". For priorities, since a job can be identified as a High priority when submitted by 'condor' user, only a flag needs to be set to differentiate between Normal and Low priority. Regarding time length, the same rule applies to differentiate Long jobs from others. So, the job submit file should contain the following lines:

```
+IsNormalPrioJob = True #or False
+IsLongJob = True #or False
```

For the machines, to include an additional attribute to the machine description structure, it must be explicitly registered into the agent. So besides creating a definition in the local configuration file, the variable name must be added to `STARTD_EXPRS`, which can be done in the global configuration, as:

```
STARTD_EXPRS = IsDesktop ##IsDesktop is defined in the local conf.
```

²the complete reference for `condor_submit` can be found at http://www.cs.wisc.edu/condor/manual/v7.0/condor_submit.html

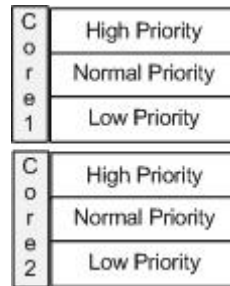


Figure 4.7: Slots in a dual core machine

4.3.3 Implementing Condor non-blocking suspension

By the default behavior, Condor does not suspend jobs, instead preempts them when jobs that receive a higher priority are submitted. When suspension occurs, the resources remain claimed, and so they cannot be reused, which would prevent the implementation of the policy rule 3b. Yet, Condor allows for a configuration where more resources can be allocated.

By default, Condor sets up one Slot (an execution resource) per each CPU core found on the machine. So a dual core machine will likely advertise two Slots for execute Condor jobs. The solution is, therefore, to setup Condor to create one Slot for each level of priority, and allow only one job to run at a time.

So for a dual core system to execute two jobs simultaneously within this configuration, 6 slots must be advertised, as in Figure 4.7 on page 33.

The number of slots can be defined by lying Condor about the number of CPUs the machine has, by explicitly set the NUM_CPUS value. But since this value depends on the number of cores, which is a machine specific property, it can't be defined the global configuration and moreover, it should be detected automatically to avoid user interference on underlying configurations. In order to achieve this, the local configuration file has to be loaded from a program ³ which calculates the number of slots and introduces the NUM_CPUS definition in the configuration. The program is a simple shell script, which detects and parses the number of CPU cores, and outputs the usual local configuration file, as:

```
#!/bin/sh
A='/bin/dmesg | grep "Initializing CPU#" -c'; A='expr $A '*' '3'
echo "NUM_CPUS = $A"
cat /usr/condor/local/condor_config.local
```

The order of the expressions still preserve the highest precedence of the local configuration file.

³This feature is described in section 3.3.1.4 of the condor's administration manual

4.3.4 Implementing job priorities behavior in Condor

Now that each node provides the necessary means for job priorities to work, it's possible to create the rules that implement policy point 3, by controlling the job execution through the slots. For this, it's necessary to define Condor's START, SUSPEND and CONTINUE expressions.

A job should be allowed to start on a High Priority slot when the submitter was "condor":

```
SLOTH1_START = (Owner == "condor")
SLOTH2_START = (Owner == "condor")
```

A job should start on a Normal Priority slot when the High Priority slot is not running jobs and the job is marked as normal priority:

```
SLOTN1_START = (slot1_Activity!="Busy") && (TARGET.IsNormalPrioJob==TRUE)
SLOTN2_START = (slot4_Activity!="Busy") && (TARGET.IsNormalPrioJob==TRUE)
```

A job should start on a Low Priority slot if it's not marked as normal priority and both Normal and High Priority slots are not running jobs:

```
SLOTL1_START = (slot1_Activity!="Busy") && (slot2_Activity!="Busy") && (TARGET.
    IsNormalPrioJob!=TRUE)
SLOTL2_START = (slot4_Activity!="Busy") && (slot5_Activity!="Busy") && (TARGET.
    IsNormalPrioJob!=TRUE)
```

Regarding suspension, a High Priority job cannot be suspended because of priorities, but a Normal priority is suspended while the High Priority slot is running a job, and a Low priority is suspended while any of the higher priority slots run a job:

```
SUSPEND_PRIO = ( ( (SlotID==2) && (slot1_Activity == "Busy") ) ||
    ( (SlotID==5) && (slot4_Activity == "Busy") ) ||
    ( (SlotID==3) && ( (slot1_Activity == "Busy") || (slot2_Activity == "Busy"
        ) ) ) ||
    ( (SlotID==6) && ( (slot4_Activity == "Busy") || (slot5_Activity == "Busy"
        ) ) ) )
```

To continue a job, the slot must check if the higher priority slots are already free:

```
CONTINUE_PRIO = ( ( (SlotID==2) && (slot1_Activity != "Busy") ) ||
    ( (SlotID==5) && (slot4_Activity != "Busy") ) ||
    ( (SlotID==3) && ( (slot1_Activity != "Busy") && (slot2_Activity != "Busy"
        ) ) ) ||
    ( (SlotID==6) && ( (slot4_Activity != "Busy") && (slot5_Activity != "Busy"
        ) ) ) )
```

4.3.5 Implementing job distribution behavior

This is one of the most important, if not the most important, implemented behavior for Condor, since it defines which machine will execute each job. Therefore it's responsible for the quality of the distribution and the global efficiency of the cluster.

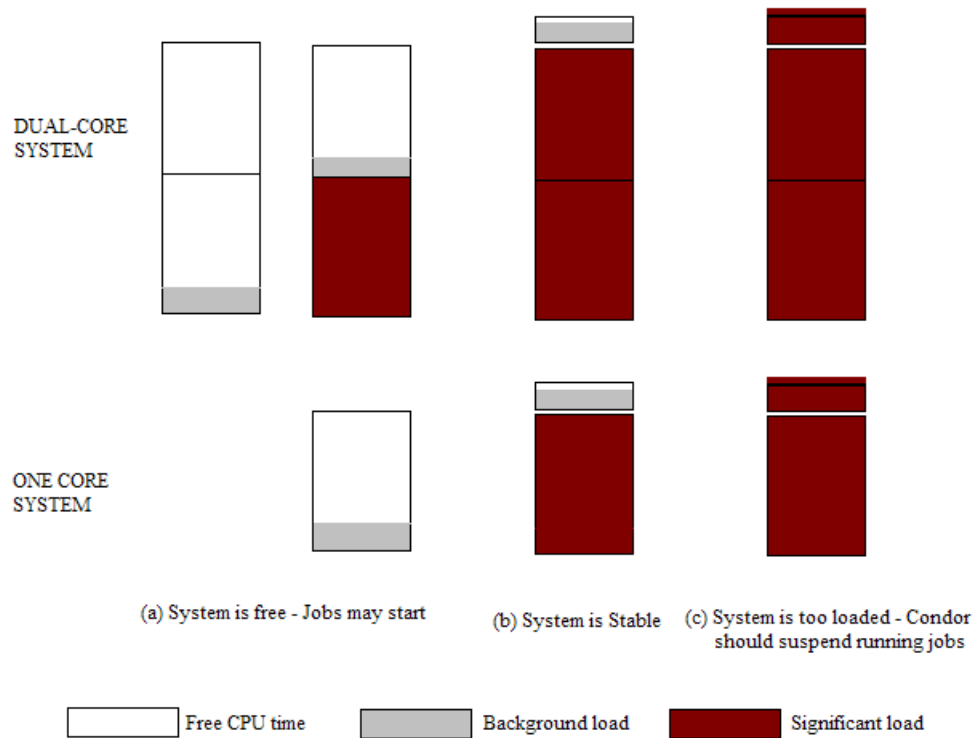


Figure 4.8: Load management algorithm

4.3.5.1 Load management

Because the simulation processes completely occupies one CPU core, it was defined that a Condor job would only run when a core was free. Since there are always some local processes running one must consider some background load, and thus allow Condor to start a job when the load is 100% - background load.

The same way, when the system is fully loaded with simulation jobs, the system will report about 100% of CPU time for each simulation plus the additional background load. So, a dual-core system is allowed to be loaded until 200% plus the background load. Above this value, it means the system is too loaded and one Condor job will have to be suspended. A schematic illustration is given in Figure 4.8 on page 35.

In order to simplify load handling, several macros (based on internal Condor variables) were defined:

```

RealCpus          = ( TotalSlots / 3 )
NonCondorLoadAvg  = ( TotalLoadAvg - TotalCondorLoadAvg )
AvailableCPU      = ( $(RealCpus) - TotalLoadAvg )
BackgroundLoad    = 0.25
HighestLoad       = ( $(RealCpus) + $(BackgroundLoad) )
CPUIidle          = ( TotalLoadAvg <= $(BackgroundLoad) )
CPUNotBusy        = ( $(AvailableCPU) > ( 1 - $(BackgroundLoad) ) )

```

```
CPUBusy          = ( TotalLoadAvg > $(HighestLoad) )
```

So in this case, the CPUNotBusy macro will have to be AND'ed to the START and CONTINUE Condor expressions, and CPUBusy OR'ed to the SUSPEND expression:

```
CERN_START = $(CPUNotBusy) && $(START_PRIO)
CERN_SUSPEND = ( $(CPUBusy) || $(SUSPEND_PRIO))
CERN_CONTINUE = $(CPUNotBusy) && $(CONTINUE_PRIO)
```

The dual-core question Still there is a detail when considering dual-cores. If Condor has 2 jobs in the queue and checks a machine has two available slots AND the CPU load is reporting the system is free, it may assign both jobs to the machine but one core is already fully loaded. And there is no way to start only one job since each slot behaves as an independent resource and is announced as such, making it impossible for the negotiator to realize both slots belong to the same machine. The possibility is to handle this situation after the jobs have been assigned, and to suspend one of them. Again, as both slots operate in parallel they will take the same action, suspending the jobs when the load gets too high and then resuming them when the load returns to the low level.

The solution was to make the slots behave differently by detecting its own index and introduce a load delay factor δ for the second group of slots (i.e. slots virtually assigned to core2).

The algorithm acts by suspending the jobs on the first group of slots right after the load passed beyond the upper limit and only suspend the second group when the total load of condor surpassed δ , as:

```
SUSPEND_LOAD = $(CPUBusy) && ( ( SlotID < 4 ) || ( CondorLoadAvg > 0.60 ) )
CERN_SUSPEND = ( $(SUSPEND_LOAD) || $(SUSPEND_PRIO))
```

This rule forces the second slot to wait until the other job to stabilize, decreasing also the total load average (TotalLoadAvg), which after a certain time makes CPUBusy variable to evaluate to false and therefore avoid suspension due to load control. Also, δ mustn't be set too high so the job can suspend in case the user fill the system with local jobs. For that to be possible $1 - \delta$ must be greater than BackgroundLoad, which means less than 0.75. Through some tests, $\delta = 0.60$ was empirically found to be very stable for this system.

4.3.5.2 Ranking resources

After Condor have collected all the resources able to run a job, it must somehow select one to which the current job will be assigned.

Regarding the policy in point 2(b)i, one can define that Condor should avoid suspension by giving priority to machines that are not running jobs, followed by machines running

low priority jobs, then by machines running normal priority and finally machines running normal priority and suspending a low priority.

A simple approach is to generate a ranking value depending on the claimed slots weighted by its priority. Defining 0.5 for normal priority slots and 0.25 for low priority slots the expression will evaluate to one value of the set [0, 0.25, 0.5, 0.75]. This expression depends on the slot group we are, so group 1 must consider slot 2 and slot 3 states, and group 2 must consider slot 5 and slot 6. Slots 1 and 4 are not considered since there are no higher priority slots. Detecting if the resource is used can be done by checking if the job owner is defined, and so the expressions can be as follows:

```
CORE1_JOBLOAD = ( 0.5*(slot2_RemoteOwner!=UNDEFINED) + 0.25* (slot3_RemoteOwner
!=UNDEFINED) )
CORE2_JOBLOAD = ( 0.5*(slot5_RemoteOwner!=UNDEFINED) + 0.25* (slot6_RemoteOwner
!=UNDEFINED) )
SLOT_AVAILABILITY = ( 1 - ( (SlotID < 4) * $(CORE1_JOBLOAD) + (SlotID > 3) * $(
CORE2_JOBLOAD) ) )
#Ranks machines considering "neighbor" slots availability
CERN_NEGOTIATOR_PRE_JOB_RANK = ( RemoteOwner == UNDEFINED ) * $(SLOT_AVAILABILITY
)
```

Still, the previous ranking may generate a draw between resources in the same conditions. This may be fairly solved by ranking the machine by its computational power. Condor has a built-in Linkpack benchmarking mechanism to measure the system's floating point performance, whose index can be used to resolve the tie. One may set the benchmarks to occur regularly, regarding the simulation times - in this case a value of 4 hours was defined - and include the performance index variable in the post ranking expression:

```
RunBenchmarks = (LastBenchmark == 0 ) || ($(BenchmarkTimer) >= (4 * $(HOUR)))
CERN_NEGOTIATOR_POST_JOB_RANK = KFlops
```

4.3.6 Implementing machine-dependent behavior

In the cluster, resources can be both Desktop machines daily used by people and Servers which are dedicated to system services and heavy computation processes. Whilst Server resources may run Condor jobs the usual way as they have more or less a constant load, in a Desktop system this situation is much different.

Because of user interaction, Desktop machines' load is highly unstable and therefore, since Condor cannot suspend or continue jobs so fast, users' processes will probably compete against Condor jobs. Even if Condor suspended and unsuspended the jobs, these operations would consume such an amount of CPU time which would event worse the problem. Competing processes may affect system responsiveness and therefore users may feel the system has a degraded performance due to Condor. In Figure 4.9 on page 38 is

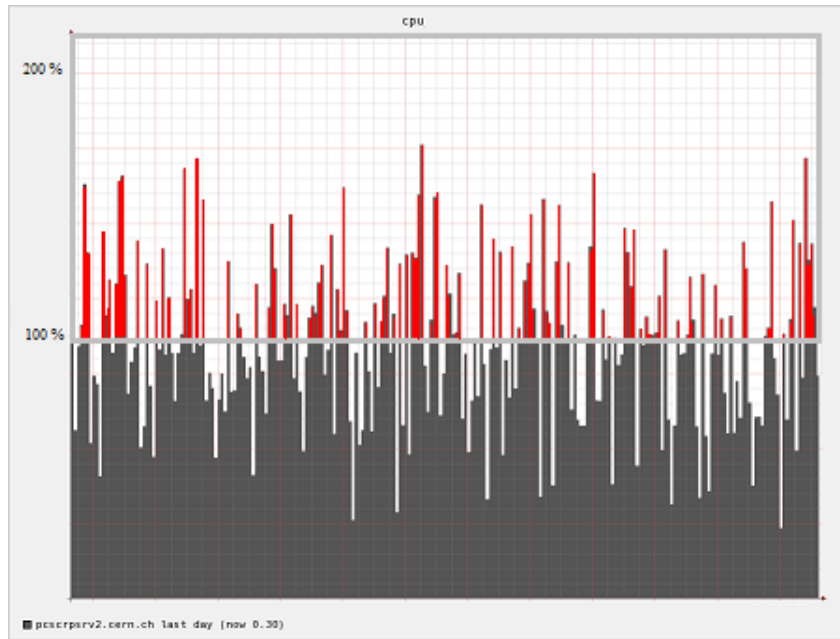


Figure 4.9: Desktop CPU load and interference with heavy processes

shown an exemplifying CPU load graph over time, whose red lines represents excessive load if a heavy process was running, affecting users' processes performance.

To solve the situation in an affective way, the load control must be handled at the execution machine for fast reaction. This can be implemented by assigning Condor jobs a lower operating system priority and therefore, as CPU share is managed in real-time, user jobs will keep their responsiveness since they have a higher priority. So, for this system it was defined that desktop machines would run Condor jobs “reniced” to 15.

Condor can be told the nice priority it should run the jobs, but we must define this behavior only for Desktop machines. Therefore each machine will have to have a flag on its local configuration file informing Condor whether they are a desktop or a server machine. This flag - 'IsDesktop' in this case - is afterwards used in the process priority expression, as:

```
JOB_RENICE_INCREMENT = ( 0 + 15 * $(IsDesktop) )
```

4.3.7 Implementing fail-safe and global optimization rules

These rules are intended to keep the system well functioning and improve the overall performance, even in the case some unwanted situations arise.

One of the main optimizations is to allow condor to preempt jobs after they have been too much time suspended, as demonstrated in 5b. To implement this rule, one must consider only the time the simulation had been suspended, which can be accessed through the job attribute *TotalTimeClaimedSuspended*. So when this variable surpasses a certain

predefined value the PREEMPTION expression should evaluate to true, as:

```
CERN_PREEMPT = ( TotalTimeClaimedSuspended > $(CERN_MAX_SUSPENDED_TIME) )
```

But this is not the only case. When a long higher priority job occupies the resources of other job the latter should be preempted to avoid suspension for a long time. A simple approach to implement this behavior is to allow the job to be suspended normally, and then preempt it if it's a long one, which can be accessed through the state of the slots in the same group, like:

```
CERN_PREEMPT = ( SlotID == 2 ) && ( slot1_IsLongJob != TRUE) ||
  ( SlotID == 3 ) && ( ( slot2_IsLongJob != TRUE) || ( slot1_IsLongJob != TRUE) ) ||
  ( SlotID == 5 ) && ( slot4_IsLongJob != TRUE) ||
  ( SlotID == 6 ) && ( ( slot5_IsLongJob != TRUE) || ( slot4_IsLongJob != TRUE) ) )
```

Therefore both rules must be combined with an 'OR'.

Finally, jobs not marked as long should be removed from the system, as it's assumed they are not running well or that the current progress should be sufficient to produce results. So after the time elapsed the job should be preempted to stop running. Still, preempted jobs stay on the queue and may restart on another machine. Therefore additional rules must be implemented directly on the queue configuration to remove the job from the system. To address this, either the job contains a rule to remove itself when running over the time limit or a global rule is defined in the scheduler daemon to test all the jobs and remove the non-long ones which bypassed the time limitation. The second way is preferred since this avoid user interference. The expression may be defined as:

```
SYSTEM_PERIODIC_REMOVE = ( ( IsLongJob != TRUE) && ( TotalTimeClaimedBusy > $(
  CERN_MAX_TIME_NON_LONG_JOB) )
```

4.4 Tests and analysis of results

Testing and debugging configuration of Condor is still a little tricky, since all the configurations are on configuration files with a particular syntax, which makes it difficult to check for its correctness or to apply a good highlighting scheme. This may not be a problem for simple expressions, but it surely is when more complex expressions are written.

In this case a general source editor was used to apply a basic highlighting and bracket checking; and afterwards the configuration was directly tested on a working environment using realistic jobs sensible to the behavior in charge. The results were obtained referring to the log files, and using `condor_status` and `condor_q` executables to check cluster and queue status respectively.

4.4.1 Priority behavior and distribution tests

In this section is presented a complete execution example which demonstrates and explains the behavior related to jobs priorities. Two configurations are tested in parallel to evidence performance improvements introduced by preempting jobs.

The test-bed was formed by two Execute machines (PC1 and PC2) and a Central Manager/Submit, from which PC1 advertises 1 node (3 slots, one slot per priority) and PC2, as a dual core, advertises 2 nodes (6 slots).

Submitting Jobs Initially 2 Low priority and 1 Normal priority jobs will be submitted.

```
/run2 > condor_submit submit_test
000 (230.000.000) 03/31 14:14:27 Job submitted from host: <137.138.213.89:60745>
/run2 > condor_submit submit_test
000 (231.000.000) 03/31 14:14:30 Job submitted from host: <137.138.213.89:60745>
001 (230.000.000) 03/31 14:14:46 Job executing on host: <137.138.55.250:37466>
001 (231.000.000) 03/31 14:14:46 Job executing on host: <137.138.55.250:37466>
/run1 > condor_submit submit_test
000 (232.000.000) 03/31 14:15:51 Job submitted from host: <137.138.213.89:60745>
```

```
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
 230.0   fleite       3/31 14:14     0+00:00:00 R  0   0.0  rfluka
 231.0   fleite       3/31 14:14     0+00:00:00 R  0   0.0  rfluka
 232.0   fleite       3/31 14:15     0+00:00:00 R  0   0.0  rfluka
3 jobs; 0 idle, 3 running, 0 held
```

```
Name                               OpSys    Arch    State    Activity LoadAv Mem   ActvtyTime
slot1@pctisrpgs3. LINUX      INTEL   Unclaimed Idle     0.000   167  0+00:13:15
slot2@pctisrpgs3. LINUX      INTEL   Claimed  Busy     0.180   167  0+00:00:20
slot3@pctisrpgs3. LINUX      INTEL   Owner    Idle     0.000   167  0+00:13:17
slot1@pctisrpjv1.c LINUX      INTEL   Unclaimed Idle     0.000   83   0+00:13:46
slot2@pctisrpjv1.c LINUX      INTEL   Owner    Idle     0.020   83   0+00:13:56
slot3@pctisrpjv1.c LINUX      INTEL   Claimed  Busy     0.000   83   0+00:00:04
slot4@pctisrpjv1.c LINUX      INTEL   Unclaimed Idle     0.000   83   0+00:13:49
slot5@pctisrpjv1.c LINUX      INTEL   Owner    Idle     0.000   83   0+00:13:59
slot6@pctisrpjv1.c LINUX      INTEL   Claimed  Busy     0.000   83   0+00:00:09
```

Priority	Node1	Node2	Node3		Node1	Node2	Node3
H							
M	164.R				232.R		
L		162.R	163.R			230.R	231.R

Since there are three nodes available it seems obvious each node to get a job. But actually job 232 could have chosen a slot from the fastest machine since the normal priority slot in both cases is free and it can Suspend a lower priority job. This behavior is not taken because the resources are firstly ranked by its availability, as described in 4.3.5.2.

Both Node2 and Node3 have a ranking of $1 - 0.25 (= 0.75)$ while Node 1 has a ranking of 1.

Suspending and preempting jobs For testing suspension, another normal priority job will be submitted to the system, but let's mark this one as Long.

```
/run1 > condor_submit submit_test
000 (233.000.000) 03/31 14:15:55 Job submitted from host: <137.138.213.89:60745>

001 (233.000.000) 03/31 14:16:07 Job executing on host: <137.138.55.250:37466>

010 (230.000.000) 03/31 14:16:28 Job was suspended.
004 (230.000.000) 03/31 14:16:28 Job was evicted
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:07
slot2@pctisrjv1.c	LINUX	INTEL	Claimed	Busy	0.000	83	0+00:00:02
slot3@pctisrjv1.c	LINUX	INTEL	Claimed	Suspende	0.900	83	0+00:00:06

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:28
slot2@pctisrjv1.c	LINUX	INTEL	Claimed	Busy	0.160	83	0+00:00:26
slot3@pctisrjv1.c	LINUX	INTEL	Owner	Idle	1.320	83	0+00:00:04

Priority	Node1	Node2	Node3		Node1	Node2	Node3
H							
M	164.R	165.R			232.R	233.R	
L		162.S	163.R			230.S	231.R

No surprises so far, the cluster is full and so job 233 will take the resource of a Low priority job, which successfully tests priority rules (4.3.4). But, the new job is a Long one, therefore the job 230 is Preempted (in the log: “14:16:28 Job was evicted”). In the next refresh of the condor_status, the job has already quit the execution machine.

Priority	Node1	Node2	Node3		Node1	Node2	Node3
H							
M	164.R	165.R			232.R	233.R	
L		162.S					

230.I

This behavior confirms that the rule for preemption (in 4.3.7 which implements transition 3 in Figure 4.6 on page 31) is working correctly.

In the meanwhile, job 231 finishes.

Preempted job continues Since job 230 is waiting in the queue and Node3 gets free, the resources will be matched in the next negotiation cycle.

```
005 (231.000.000) 03/31 14:17:44 Job terminated.
001 (230.000.000) 03/31 14:17:44 Job executing on host: <137.138.55.250:37466>
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:01:52	R	0	244.1	rfluka

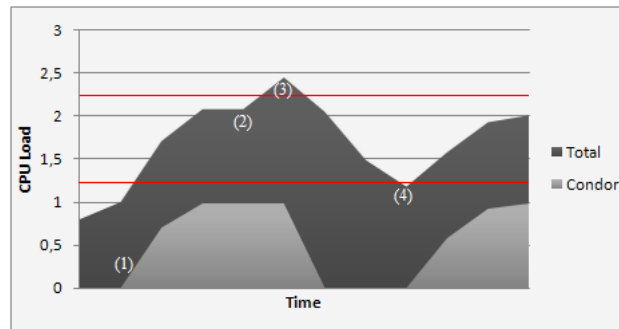


Figure 4.10: Load control - machine fills up with local jobs

```
232.0   fleite           3/31 14:15   0+00:01:48 R  0   0.0   rfluka
233.0   fleite           3/31 14:15   0+00:01:48 R  0   0.0   rfluka
3 jobs; 0 idle, 3 running, 0 held
```

Priority	Node1	Node2	Node3	Node1	Node2	Node3
H						
M	164.R	165.R		232.R	233.R	
L		162.S				230.R

The three jobs are now running in parallel, while in the previous scheme one of them would remain suspended. This simple comparison demonstrates how job preemption can be useful to better distribute jobs across the cluster.

4.4.2 Load control and distribution tests

Now it will be presented a test case demonstrating the load control mechanism.

The test-bed is formed by one dual-core Execute machine, and a Central Manager/Submit server. It will address the cases of controlling Condor jobs as more jobs are submitted into Condor or the local running jobs change. The main rule: local jobs have always priority. The complete output from this tests can e found in Section B.2.

4.4.2.1 Running 1 local and 1 Condor jobs

A dual-core machine advertises two Nodes and therefore, without load control, it would always allow Condor to run two jobs.

In this test Condor will have to stop its job since the machine will be filled up with two simulations running locally.

Figure 4.10 on page 42 shows the temporal evolution of the system, whose numbers are the main events:

1. There was already one local job running when a Condor job is started. The Condor load increases and so the total machine load, which stabilizes around 208%.

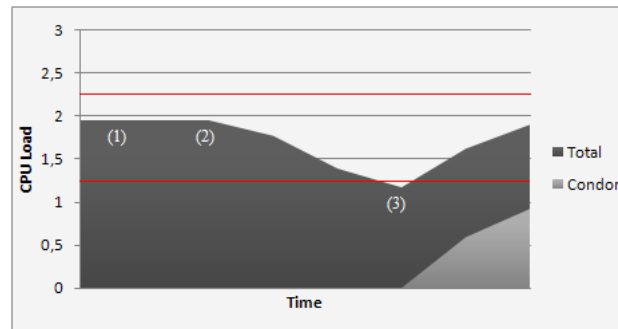


Figure 4.11: Load control - machine waits until gets free

2. A second local job is started so the total machine load starts to increase. Condor's load stays the same.
3. Total machine load surpasses the allowed maximum limit ($\text{CPUS} + \text{BackgroundLoad} = 2.25$) and Condor suspends the job.
4. One local job is stopped and the load decreases until it reaches the CPU-free barrier. The Condor job is allowed to resume.

4.4.2.2 Running 2 local jobs

Without load control, condor would have started the job right away in the machine. Load control checks the machine cannot start more jobs and therefore Condor job waits in the queue until some resources get freed.

The main events, numbered from 1 to 3 in Figure 4.11 on page 43, are:

1. Machine is running 2 local jobs, holding a constant load near 200%. In the meanwhile a job is submitted into Condor but, even the machine Condor slots are free, it doesn't start the job as the load is above the CPU-free line. So the job is kept idle in the queue:

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
750.0	fleite	6/20 09:56	0+00:00:00	I	0	0.0	cfluka

1 jobs; 1 idle, 0 running, 0 held

2. One local job finishes. The total CPU load starts to decrease.
3. CPU load reaches CPU-free barrier and, in the next negotiation cycle the job is assigned to the machine. Condor load increases and so the total machine load.

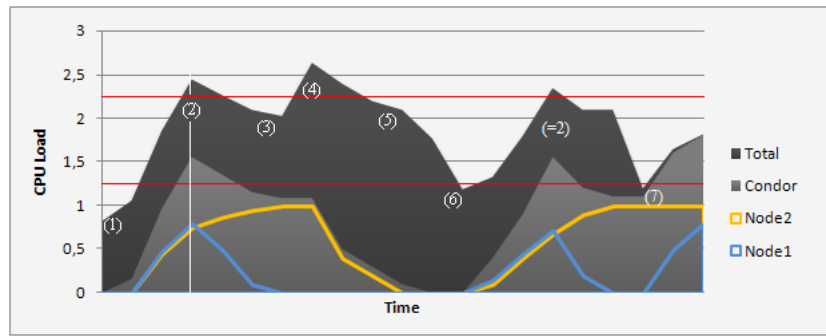


Figure 4.12: Load control - extensive test

4.4.2.3 Extensive test

This test is intended to check most situations related to load management and how the system behaves to complex chains of events.

With a running local job, two more jobs will be submitted through Condor and additional variations to local jobs will be performed.

Figure 4.12 on page 44 shows how the system responds to changes that affect the load, always trying to stabilize it on the optimal load, 200% in this case. The events, which are numbered from 1 to 7, are the following:

1. One local job was running (no Condor load), when two jobs are submitted through Condor. Both jobs start running immediately because the machine load is below the CPU-free line.
2. Total CPU-load bypasses the allowed maximum load and so one Condor job is suspended. As both jobs just started, the situation is detected and only the first Node suspends the job (section 4.3.5.1), making the load to stabilize around 200%.
3. A new local job is started and the load quickly increases.
4. Again, too much load is detected in the machine and Condor suspends the remaining job on Node2.
5. Condor has two suspended jobs so the loading is approaching 200%. Now one of the user local jobs is stopped, which manages to quickly drop system load.
6. When the total CPU load goes below the available mark, both Condor jobs which were suspended may now resume and, as in event 2, one of them will have to be suspended.
7. The other local job finishes, making room for the Condor job which was recently suspended.

Chapter 5

Development of supplementary software tools

So far the cluster is installed and enabled for efficient job distribution. It's then the time to introduce the development interfaces to improve cluster usability.

People are not machines, people are different and they should not need to know about Condor to do their job. And in this case this could be really tricky since there are new job attributes (4.3.2). Therefore, the process would become even more complicated which would narrow its usage.

This is the reason for the implementation of the COFLU (COndor-FLUka), a set of applications to simplify and make more powerful the submission of simulations to the cluster.

5.1 High-level architecture and specification

The objective of COFLU is clear: to work as a layer between the user and Condor, specially designed to handle FLUKA simulation jobs. This layer must be (1) powerful, so the advanced user can take the maximum advantage of the system, and (2) simple, so users can easily submit jobs within two or three steps and without having to care about the details.

These two objectives somehow push the development into opposite directions: either the system gets fully-featured but complex or simple but providing limited functionalities. The key to achieve both in a consistent manner is structuring the tools and differentiate usage profiles.

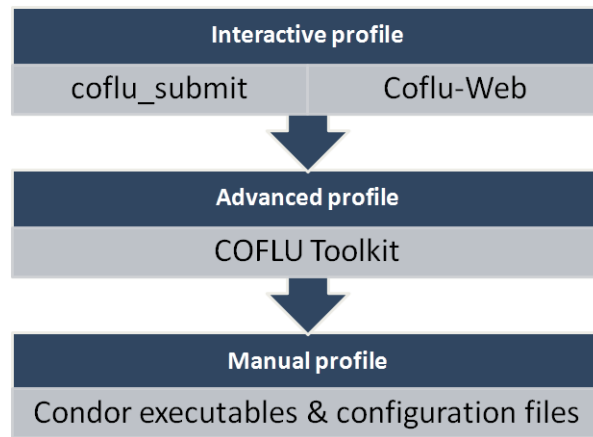


Figure 5.1: System's usage profiles and application dependencies

5.1.1 Logical structure

Better than differentiation users, one can differentiate usage profiles to model the structure of the application. This logic works towards the user since he can decide how to use the system depending on his skills and the specific situation. The user can also combine profiles in order to obtain both speed and control. This approach is not only beneficial for user but also provides guidelines for the development.

For this system three execution profiles can be identified: the manual profile, the advanced profile and the interactive profile (Figure 5.1 on page 46). Each profile implements a subset of the profile in the previous level and therefore can rely on it for execution:

1. Until now the only submission mode was preparing all necessary files for the simulation, create a submission file for condor and submit it. This Manual Profile is, and should be, always available as it grants users the maximum control over the system and it is assured to work on the worst scenarios.
2. The Advanced Profile should be formed by a set of tools which automates the processes of prepare and submit simulation jobs to the cluster. The maximum possibilities will have to be available in this layer to allow high level of control and wide possibilities. Besides being used by users, this layer should provide the upper layer interface to the system, working as an API, which is guaranteed since the commands are not interactive. This layer will be formed by several interrelated shell commands - the COFLU Toolkit.
3. The Interactive Profile tries to provide all the common options a job may have through a simple and user-friendly interface. Two versions are available:
 - (a) `coflu_submit` - an interactive shell interface which is intended for shell users,

so they can do the necessary steps without changing the working environment, and to allow easy remote execution via a remote shell.

- (b) Coflu-Web - A website which provides the user a simple, organized and platform independent graphical interface for managing simulations over the cluster. Because of its simplicity and organization, coflu-web includes many additional high-level functionalities, making it the ideal solution for most FLUKA simulations.

5.1.2 Coflu-Toolkit requirements

COFLU Toolkit is a set of non-interactive programs for independent tasks, and may be used in conjunction to meet the requirements regarding simulations characteristics.

One of the objectives which received more attention was the possibility for one simulation to split up its cycles over the cluster as sub-simulations. Even it was possible in the manual profile it was rather complex to set up and the users easily trapped in some misconfigurations which could lead to the loss of data.

When creating sub-simulation the user had to care about generating a distinct seed for each one and to place all the necessary files in a different directory, as completely independent simulations. To address this in an automated fashion and summarizing all the needed features for the advanced profile, one can specify the toolkit to meet the following requirements:

- [REQ_CT1] It should be possible to define as many seeds as needed and allow for seed re-generation - by given the same initial seed we should obtain the same set of seeds.
- [REQ_CT2] The submission to the condor pool should be transparent for the user, so the jobs submit file must be generated automatically after the user provided the simulation's parameters.
- [REQ_CT3] The user is able to easily create a groups of sub-simulation based on an Input File and a group of seeds to be used. An organized directory structure must be specified to start the whole package at once and help results analysis.
- [REQ_CT4] The submission process should not become more complicated.

5.1.3 Coflu_submit requirements

The COFLU shell interface can be a very valuable resource for any simulation which doesn't need any specific advanced parameter, since it dramatically decreases the time

needed of setting up a simulation. By asking the user and providing him with default values, it turns to be more user-oriented and error-avoiding mechanism.

Besides the interactive mode, which must be limited to the basic details, this program supports more advanced usage through arguments from the command line. These arguments influence the interactive mode in such way there's no redundancy of information. For instance the user may specify a configuration file to be loaded and therefore all configuration related questions are suppressed.

Summarizing, the requirements for `coflu_submit` are:

- [REQ_CS1] Support importing shared configuration files, which provides the defaults values for system dependent expressions;
- [REQ_CS2] Support advanced parameters through command line arguments;
- [REQ_CS3] Question the user about missing details for the current simulation, providing defaults as available and validating user input.
- [REQ_CS4] Sequentially invoke COFLU tools and handle results, providing user information about progress and errors.

5.1.4 Coflu-Web requirements

Even the shell interactive interface eases the submission of simulation jobs to the cluster, it has some intrinsic limitations:

- Users must log in on a Linux machine that is running the submission agent
- Most users working O.S. is Windows, as so they needed to copy the files to the submission machine
- It's still not much user friendly.
- Develop a consistent shell module for results analysis is impracticable.

To surpass these limitations, a web interface was found to be the best solution which can yield many more features. Since it is completely disconnected from user's working computer, such interfaces offer an incredible transparency and flexibility to the user.

But on the other hand, it must provide authentication mechanisms and allow COFLU commands to execute with current user privileges on specific machines so that: (1) users are responsible for their own jobs, (2) fair share policies can be applied, (3) the job have the correct permissions to access user directories and (4) user is free to use both interfaces since underlying commands execute on the same machine. This can turn to be a problem

during development but, nevertheless, the web environment is so extensible and powerful that the implementation of additional functionalities, other than the job submission, was considered for this application. Putting all together, Coflu-Web should:

- [REQ_CW1] Allow a user to authenticate with his NIS credentials;
- [REQ_CW2] Execute commands remotely on the machine defined as his home;
 - [REQ_CW2.1] Generate commands from simulation specification
 - [REQ_CW2.2] Inform user about execution progress, handling eventual errors;
- [REQ_CW3] Allow for server directory browser and file listing;
- [REQ_CW4] Import default values from a shared configuration file;
- [REQ_CW5] Support combined usage with lower level profiles:
 - [REQ_CW5.1] Specify advanced parameters through the use/load of configuration files, as seeds and submit file for Condor;
 - [REQ_CW5.2] Allow for simulation set up without actually submitting it, so user may personalize it;
- [REQ_CW6] Import configurations to the current edition view, parsing the file;
- [REQ_CW7] Thoroughly validate user inputs;
- [REQ_CW8] Inform about cluster status and user jobs;
- [REQ_CW9] Allow users for own jobs' removal;
- [REQ_CW10] Provide a simple administration for defining users' home computers.

5.2 COFLU-Toolkit architecture

Regarding the previous requirements, the different tasks are carried by four programs forming the toolkit:

- `coflu_genprime`: a program which generates seeds to be used by each FLUKA simulation cycle. It allows the user to re-generate seeds by entering the first element of the sequence.

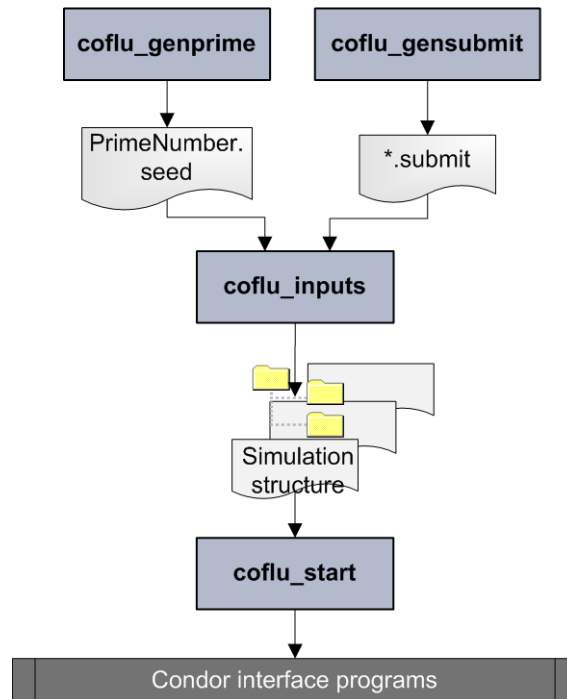


Figure 5.2: COFLU-Toolkit

- **coflu_gensubmit**: a program which generates a condor submission file for a specific FLUKA simulation. This generated file will contain information about the job executable itself but also its arguments and special flags for Condor, like “+IsLongJob” or “+IsNormalPrioJob”.
- **coflu_inputs**: a program gathers necessary files and creates the directory structure for the simulation, from where the actual FLUKA simulation will run. When creating the structure, this program may split the simulation cycles over sub-simulations, introducing the powerful possibility of run the simulation in parallel.
- **coflu_start**: a program which submits a whole simulation into Condor, including all sub-simulations.

Figure 5.2 on page 50 shows how programs may be combined to set up and submit one simulation. The advantage of this sparse architecture is that the user can interfere in every stage and personalize properties. For instance a user may need to add additional details to the submit file or use a specific seed.

5.2.1 Simulation structure (coflu_inputs)

Each condor job has compulsorily a submit file and each individual simulation has an input file and some optional auxiliary files. For a simple simulation this is it and all the

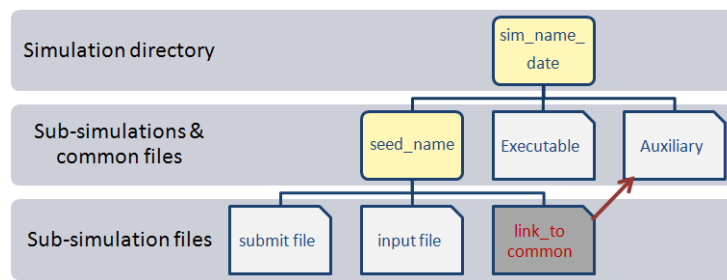


Figure 5.3: Simulation file structure

files are located in the same directory, but when it comes to sub-simulations, created from the split of a simulation, it gets complicated to handle every file in the same directory:

1. It's difficult for the user to distinguish between original, generated and output files.
2. The user would have to manually copy files to different directories to start other simulations with the same sources.
3. Two simulations with the same name in the same directory would overwrite each other outputs.

So it is a good idea to create a file structure to organize all this data files from where sub-simulations may run independently. The structure is composed by a main directory for each prepared simulation (to where all the necessary files are copied) and a subdirectory for each sub-simulation containing its specific files and links to simulation common files, as illustrated in Figure 5.3 on page 51. Therefore file consistency is guaranteed among sub-simulations and file overhead is reduced.

Moreover, the simulation main directory name includes a date stamp, in order to avoid overwriting previous generated simulation structures. This detail is also useful for user orientation and file archiving.

5.2.2 coflu_submit interface architecture

The `coflu_submit` shell interface is responsible to handle the COFLU-Toolkit programs so they execute the desired actions based on information provided by the user. As a single program it was included in the COFLU-Toolkit package.

To support such characteristics `coflu_submit` is based on a parser for command line arguments, a configuration file parser, an user interaction/validation module and a command building/execution module (Figure 5.4 on page 52).

The command line parser, based on the `optParse` library from python, checks the information passed as argument through the command line and stores as properties of an internal object.

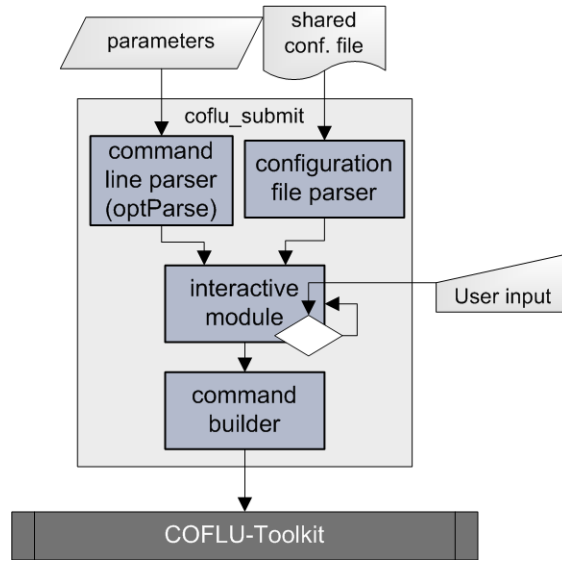


Figure 5.4: coflu_submit architecture

The configuration file parser performs a similar task to the shared configuration file, parsing its macro-like entries. In order to keep configuration methods consistent, the latter was implemented as an independent class module. It loads and parses configuration files and stores definitions into an internal data structure formed by a python dictionary.

The interactive module checks the existing information collected by the previous modules and request additional details to the user, validating the answer.

The command builder module generates the commands to be executed, checking execution outputs to inform user about status or an eventual errors.

5.3 Coflu-Toolkit implementation

As a first note, the COFLU tools are developed in Python [28]. The main reason behind this selection is that Python was already used in some basic script used at the RP group and is claimed to (1) increase programmer productivity (from 5 to 10 times [29]), well suited for large or complex projects with changing requirements and easy to learn. Python comes with extensive standard libraries and is available for all major operating systems.

Besides not being complex programs, the tools were implemented with two main concerns: maintainability and robustness in error handling.

5.3.1 Error handling

COFLU tools are not interactive programs, so they can be invoked from other programs. Therefore they all must parse information from the command line and if this is some parameter doesn't comply the program should abort. To implement this, all programs

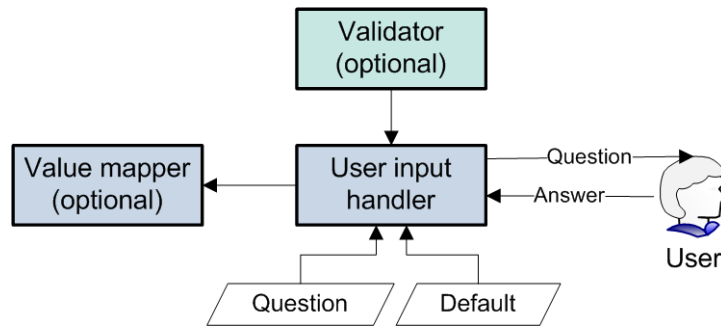


Figure 5.5: coflu_submit interactive mode

include a python function which checks, parses and stores parameters into a list.

When a parameter does not comply, an explaining message is printed to `STDERR` and depending on the severity of the error the application may continue or exit immediately.

Error handling within coflu_submit The interactive interface for COFLU implements a more sophisticated parameter parsing mechanism. As in Figure 5.4 on page 52, the Python `optParse` library was used to execute such task, which was configured through validation rules.

Although very complete, `optParse` doesn't come with support for multi-values per option (e.g. “-aux file1 file2”). Fortunately, `optParse` is flexible to the extent of supporting user defined parsing functions for each option. For this `optParser` has to define option parsing as:

```
parser.add_option("-a", "--aux", action="callback", callback=vararg_callback, (...))
```

Since `coflu_submit` relies on the COFLU tools, it parses the return code of the executed command and aborts with a specific message and return code.

5.3.2 coflu_submit interactive mode

Depending on the given parameters, `coflu_submit` will ask the user about additional simulation details. For each required detail the program must provide (1) a description of the necessary detail, (2) provide a default value (if it exists) and (3) validate user input.

To implement this requirements in an organized fashion, so additional user input is easy to implement, all the independent tasks were split into: validators, functions to handle input (one for numeric and other for all other content), and value mappers.

As illustrated in Figure 5.5 on page 53, the “input handle functions” need to be provided with the question to be asked and an optional validation callback. The aim of these functions is to show user the question with an eventual default value and ask the user input until it's accepted by the validation function. When a default value is specified, the

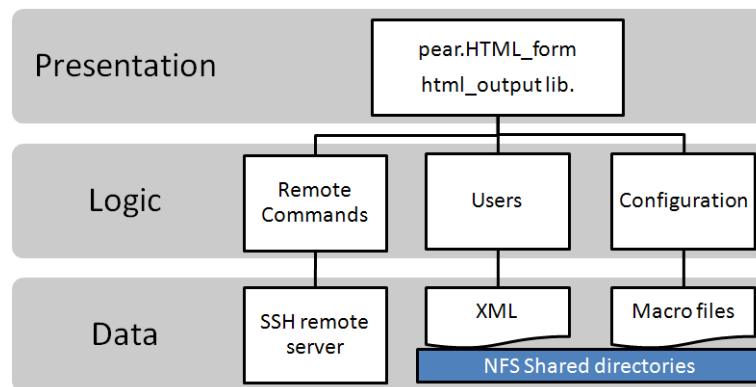


Figure 5.6: Coflu-Web architecture: Horizontal decomposition

validator returns true for empty strings as well, so it the default is assumed.

For reading numeric values there is a specific function because there is always the intrinsic validation to test whether the user input is actually a number or not.

At the end the developer may want to convert types or map values (e.g. string Yes/No to boolean True/False), so he must provide the mapping function with the outputs from user handling function.

The result is that all three steps are executed in a single line, as in the following example where user is asked about the job priority:

```
args.lowprio = mapYesNo_bool( read_valid( "[INPUT] Is this a Low priority job? (
    Enter for default: No)", yesno_validator, "No" ) )
```

For reference, these functions are included in Chapter C.2.

5.4 Coflu-Web architecture

Coflu-web was designed as following a well defined structure in order to support for expandability and maintainability. Furthermore it concerns in providing well-specified interfaces so addition of new features can be easily implemented in a modular style.

5.4.1 Horizontal decomposition

The application can be analyzed from a horizontal perspective by distinguishing higher level from lower level functions, creating a three layer model composed by presentation, application logic and data access (Figure 5.6 on page 54).

Presentation layer As a web-application, the presentation layer is defined by the mechanisms which are used to produce html to be rendered by the client's browser. In order to increase productivity and assure consistency between interface components, two libraries

were used to generate the html instead of output it directly: PEAR/HTML_form and html_output.

HTML_form is a package from PEAR PHP framework for generating html forms depending on specified fields and option. In turn, html_output is a set of functions from a modified version of the GNU osCommerce tep_html_output library, which generates html output for specific form elements. Therefore HTML_form is used whenever possible for complex forms, while html_output demonstrated a big flexibility and high ease-of-use for complex elements in short forms.

Logic layer Coflu-Web logic layer is composed by three main abstractions which handle (1) Remote commands, (2) Users and (3) Configurations.

In order to run Condor operations on a specific machine from a web interface, mechanisms for authentication and remote execution are required. Since web-applications intrinsically follow a client-server architecture, these challenging requirements will be handled at the Remote Commands. This module will handle SSH authentication, command requests and outputs, being responsible for REQ_CW1 and REQ_CW2.

User abstraction will be handling information regarding the user, principally his home machine, to allow for REQ_CW2. Write operations are only allowed to administrator.

Configuration abstraction has the purpose of load macro-like configuration files and make them available to be used as variables, thus is responsible for REQ_CW4 and REQ_CW5.1.

Data layer Data layer will provide to the upper layers the data they need, being responsible for the acquisition of information from the different sources:

1. From a SSH remote server: Authentication and remote commands and will be sent through a SSH client running at the web-server.
2. From a local XML file: Because of such little amount of information, data was stored as XML files and is handled at data layer through PEAR/XML_serializer.
3. From shared macro-style configuration files: Since Coflu-web only needs read-access mode to configuration files, the files can be directly accessed from the web-server process.

5.4.2 Vertical decomposition

Coflu-Web provides a set of features grouped into modules. Each module can be added to or removed from the system without impact on the other modules. Modules can access

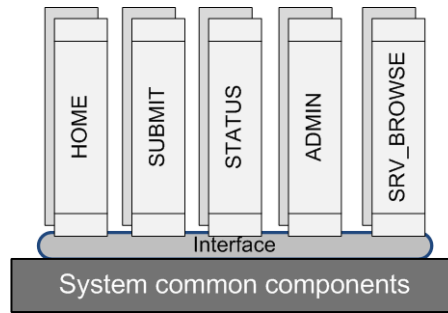


Figure 5.7: Coflu-Web: Vertical decomposition

predefined existing resources, like those specified in 5.4.1 or include specific resources no matter the layer they belong, since it is the developer concern to define each module architecture.

Five modules were implemented, as in Figure 5.7 on page 56:

1. [MOD_HOME] A default home module, containing application overview and instructions. It's the only screen available when the user isn't authenticated, from where he can authenticate (REQ_CW1). As so this module cannot be removed unless substituted.
2. [MOD_SUBMIT] A submission module, from where the user can set up a whole simulation and submit it. This is the essential module of this web-application which requires the most resources and depends on the secondary module MOD_SRV_BROWSE. Through this module the following requirements are met: REQ_CW2.1, REQ_CW2.2, REQ_CW5.1, REQ_CW5.2 and REQ_CW6.
3. [MOD_STATUS] This module provides the user information about the cluster status as well as the status of its jobs (REQ_CW8). It also includes the job removal feature, which allows users to stop their jobs from executing (REQ_CW9).
4. [MOD_ADMIN] This module allows the administrator to define some details about the users, like their home computer (REQ_CW10). Although removable, without this module administrators would have to edit an XML file manually when to make changes to users profile.
5. [MOD_SRV_BROWSE] This module allows for server file system browsing (REQ_CW3). Using Ajax to communicate with the server, it provides user a highly-responsive and intuitive interface for directory selection, which can be easily integrated with other modules.

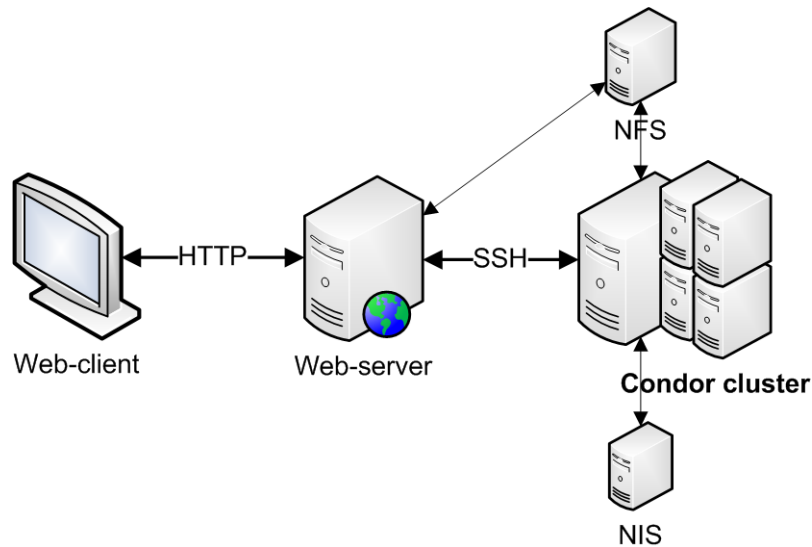


Figure 5.8: Physical architecture

5.4.3 Physical architecture

Coflu-Web execution will require interaction among the physical elements present at Figure 5.8 on page 57, whose communication is supported by the existing network.

The user interacts with the application via the web-interface displayed at the client browser, which communicates with the web-server via the HTTP protocol. The web-server handles client requests, gathers the necessary data (whether it is local or remote) and formats it into html pages. For remote data it translates the request into SSH commands and communicates with a specific server via the SSH protocol. Authentication to SSH requires services from the NIS and access to shared files requires NFS.

5.5 Coflu-Web Implementation

One first step to start implementation is selecting the technologies. Two factors were determinant for selecting PHP as the implementation language for Coflu-Web:

1. The machines are running Linux SLC4, and therefore will run Apache HTTP Server, which avoids Microsoft-based solutions, like .NET.
2. PHP has an immense set of extensions and libraries, developed and maintained all around the world, including the PEAR and PECL extension managers [30], which some very useful packages for this project like HTML_Form, HTML_AJAX, XML_serializer and PECL/ssh2 [31].

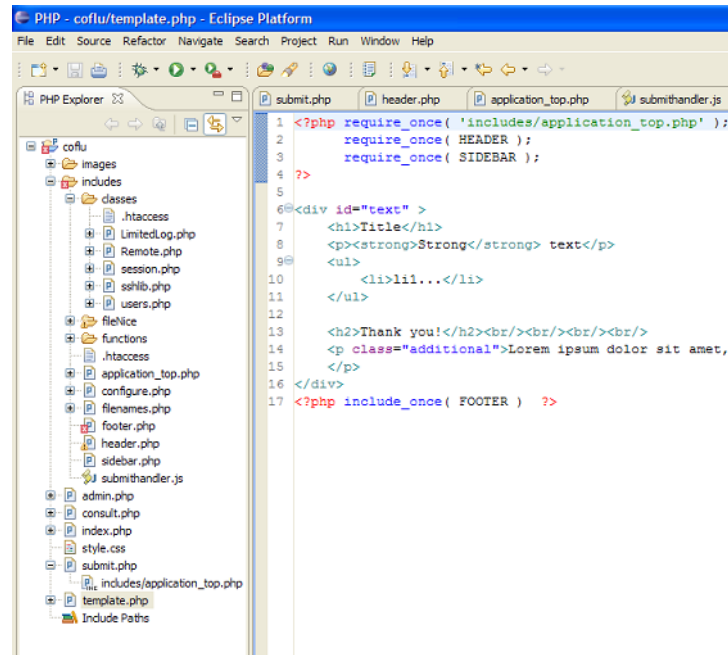


Figure 5.9: File structure and module template

5.5.1 Project structure

To provide easy addition of features and modules, two strategies were applied: main COFLU-Web sources were placed in dedicated directories depending on its function, and code was structured as much as possible and imported to where necessary.

So, the main directory contains only php files which generates html pages for the user interface. These files work as modules, since they only implement their specific look and functionalities; all other page elements are just included, as seen in the template in Figure 5.9 on page 58. These page elements are placed in “includes” directory, which besides the header, the sidebar and the footer, contains two important configuration files: “application_top.php” which sets the site initializations, and “configure.php” which defines specific configurations including available SSH servers. Usually the site administrator won’t have to change any file but the later.

5.5.2 Authentication and remote execution

One of the biggest challenges in this application was to define how the web server could execute commands making the current user the owner of the process. And it is not only a matter of permissions (as explained in the specification - 5.1.4). Nevertheless, allow Apache to execute programs as a high privilege user (like root) or scripts which to gain administrative permissions leads to severe security issues [32, 33]. Additionally, the authentication mechanism should rely on the NIS server, so the accounts data is kept

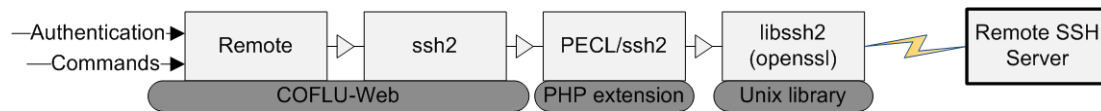


Figure 5.10: Remote execution dependencies

centralized. As a third point, the commands should execute on specific machines, so the load is distributed and users can freely switch between web and shell interfaces.

These three issues led to a single and comprehensive solution: to create a remote SSH connection from the web-server to the user home computer. This method is plausible since (1) all computers have SSH server installed, (2) command to be executed remotely (mainly from COFLU-Toolkit) are non-interactive and (3) SSH client does not need to run as root and is available as PHP extension.

So in order to execute remote commands from a higher level, without invoking `ssh2` library directly, class `ssh2` (in `sshlib.php`) was created to handle a SSH session. So, from the moment a user open a SSH connection, providing username and password, he is allowed to submit commands whose output information from standard output and standard error is collected to internal variables. For the specific case of this web-application, the `ssh2` class was extended to implement some general functions like listing remote files, changing directory, logging previous commands and errors, etc. Figure 5.10 on page 59 shows the library dependencies for this mechanism.

5.5.3 Three-tier data validation

Validation is an extremely important point in this project for many reasons. Besides having direct consequences on the submission machine and on the user work, actions on the web-interface may introduce a significant bandwidth and web-server load.

Additionally, to increase user experience any non-complying user input should be detected and asked for correction as soon as possible. Regarding Figure 5.6 on page 54, each layer should implement validation of the data it handles.

Tier 1 - Client side validation

Presentation layer validation is mainly used for performance purposes. Since it is executed on the web-client machine (see Figure 5.8 on page 57), it avoid communication through the network and therefore provides fast response to the user. This validation was implemented in separate JavaScript files which are integrated into the corresponding interface module. For instance, the submission module “`submit.php`” imports its JavaScript validation file “`submit_handler.js`”, which checks the entered data and when a rule is not met it shows alert messages and avoids form submission.

Figure 5.11: Coflu-Web: simulation configurations

Tier 2 - Web-server module validation

With a working client side validation one could omit server-side validation. But due to security and fail-safe reasons - like clients which don't support browser scripting, communication errors, etc - data is always checked in server-side. This validation is executed at the beginning of each corresponding module and, in case some issue is found, it generates messages which are added to the user's interface.

Tier 3 - Remote execution validation

The last validation layer is held directly by each Coflu tool, and therefore the web application must handle their behavior regarding exception situations.

After the commands' generation at the web server, their execution may produce errors and warnings because of the most reasons, including operating system restrictions. Since all these messages are written to standard error stream which is available to the web-server through the ssh2 library, Coflu-Web can use this information to avoid further execution of the commands and to include error description in the user's generated interface.

5.5.4 Using AJAX to import configuration files

The Condor submission file (*.submit) contains many relevant details about the simulation (Figure 5.11 on page 60). Coflu-Web can either generate this file from user's specified settings or may use an existing submission file to create a similar simulation. A third option was to allow users to import submission files and change certain details in the web interface, so they could adapt it for related executions. Additionally users were given the option to edit the file manually, which makes the bridge to advanced users. However this feature must be explicitly activated in order to prevent accidental changes.

To bypass reloading of the whole page, which would take a considerable amount of time and make the user to lose changes, file import mechanism was implemented using AJAX. The HTML_AJAX library from PEAR showed up to be a good option because of the

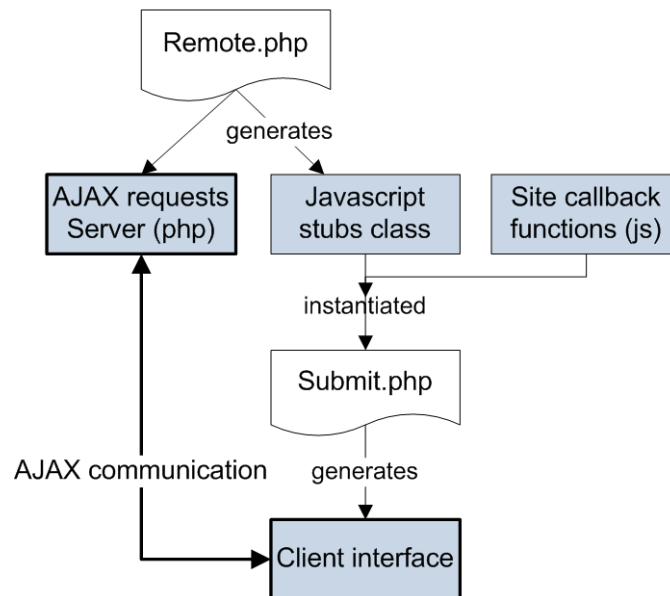


Figure 5.12: PEAR/HTML_AJAX in Coflu-Web

well structured implementation architecture it enforces. HTML_AJAX on Coflu-Web was configured to generate a JavaScript class from a PHP class which can automatically invoke callback handlers (Figure 5.12 on page 61). The AJAX Requests Server was enabled for session persistence and therefore keeps loaded configurations available during application usage. On the client-side, the results are parsed and inserted in the specific form elements through callback functions. The definition of these functions is implemented as in the following snippet:

```

//AJAX Callback functions
var configLoaderCallback = {
    parsefile: function( result ) {
        (...)
    },
    (...)
}

//Ajax remoteObject
var remoteLoad = new confloader( configLoaderCallback );

```

5.6 Tests and result analysis

On a first step a program must be tested for syntax correction. Python, PHP, JavaScript are all interpreted languages, which do not need explicit compilation. This avoids the need of compiling a program each time it's changed, but as so, makes it difficult to detect programming errors. Fortunately there are many good development environments and tools which automatically check syntax for these popular languages. For PHP it was used Eclipse with PDT (PHP Development Tools), which revealed to be an excellent and comprehensive platform, supporting HTML, PHP and JavaScript syntax highlight-

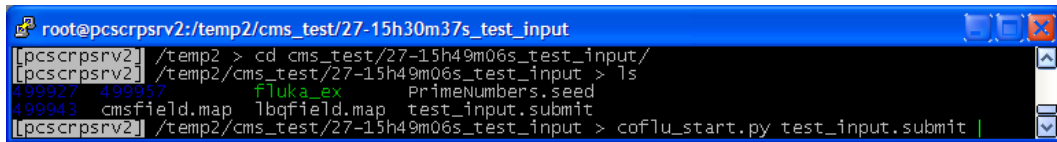


Figure 5.13: Main simulation directory

ing, checking and auto-complete. Python was developed in a general editor with syntax highlighting and checked for correctness with PyChecker¹.

For testing their behavior, both applications include a debug mode, which outputs long and verbose details about the operations and data structures.

5.6.1 coflu_submit execution example

A good example to show results is to do comprehensive tests directly in the interfaces, which use the toolkit and so also their messages are printed in the current view. In this test the user want to submit a simulation with 3 cycles, each cycle in one computer, two auxiliary files and a different executable. These options are all supported in coflu_submit.

```
[pcscrpsrv2] /temp2/cms_test > coflu_submit.py test_input.inp -a cmsfield.map lbqfield.map -e fluka_ex
[INPUT] Generate new seed file? (Enter for default: No)Y
[INPUT] Initial seed (Enter for random)?
[INPUT] How many runs?3
552469
552481
552491
[INPUT] Fluka -N? (Enter for default: 0)
[INPUT] Fluka -M?3
[INPUT] Is this a long job? (Enter for default: No)
[INPUT] Is this a Low priority job? (Enter for default: No)
# Condor submit file for FLUKA generated by coflu_gensubmit.py
(...)
[INFO] Current marked as auxiliary files: ['cmsfield.map', 'lbqfield.map']
[INPUT] Add auxiliary file: (Enter for no (more) files)
[INPUT] Should the job start immediately (Enter for default: Yes) y
coflu_inputs.py test_input.inp test_input.submit -run -aux:cmsfield.map -aux:lbqfield.map -aux:fluka_ex
added new parameter to input file:552469
added new parameter to input file:552481
added new parameter to input file:552491
(cluster status)
Detected input file: test_input_0.inp
Submitting job(s).
1 job(s) submitted to cluster 801.
Detected input file: test_input_2.inp
Submitting job(s).
1 job(s) submitted to cluster 802.
Detected input file: test_input_1.inp
Submitting job(s).
1 job(s) submitted to cluster 803.
[FINISH STATUS] 3 jobs submitted!
```

The bold lines are the output of the executed commands from COFLU tools. The first bold block shows the three generated seeds by coflu_genprime. After, coflu_gensubmit is invoked to generate the submission file for Condor. Since coflu_inputs is the most complex

¹<http://pychecker.sourceforge.net>

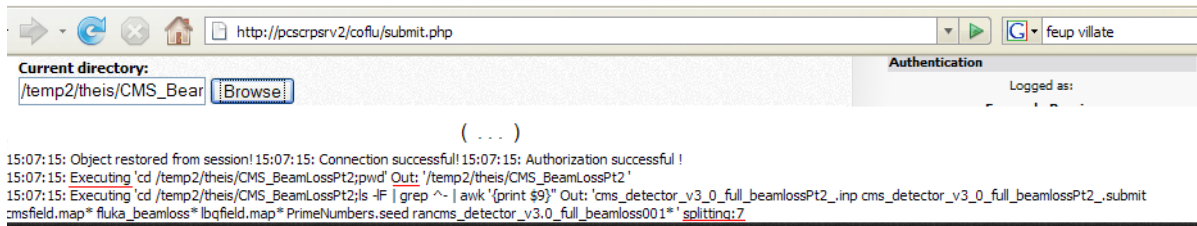


Figure 5.14: Submission page debugging data

tool, its generated command is printed along some output while creating the simulation structure. In this case the main simulation directory will contain: three directories for the sub-simulations, a different executable, the auxiliary files, the file with the seeds and the global submit file (Figure 5.13 on page 62). This structure and the directory name itself follow the model defined in Figure 5.3 on page 51.

As a last step, `coflu_start` is invoked. It analyses directory-by-directory and submits each sub-simulation to the cluster. As in the log, submission information is printed and includes the finish status with the number of jobs submitted or eventual errors.

5.6.2 Coflu-Web execution example

In this example a similar simulation will be submitted through Coflu-Web. To analyze the progress, debugging mode was activated to print extended details. Some illustrative screens showing the Coflu-Web usage are available in Appendix D.1 on page 91.

The user should first choose the directory where the simulation source files are located. At any reload or directory change, the web-server communicates with the user's remote server to list files in the current directory. These files are then filtered by its properties or extension regarding their final place in the page.

Checking debugging information at the bottom of Figure 5.14 on page 63, the SSH connection is reestablished at the page load and a few commands are sent via the channel. The last command is a complex `ls` which outputs only the file names (not directories) to be parsed in PHP.

To set up the simulation, the user should fill the fields accordingly. As in the shell interface example, the simulation is defined with two auxiliary files, a different executable and three sub-simulations having new random seeds.

When the form is submitted, the web-server parses the job information and generates commands which invoke the Coflu tools.

As in Figure 5.16 on page 64, the results page shows user the outputs from the Coflu tools. In the lower part of the picture, the debugging information shows the executed commands (with red underline) and their output. As it's possible to see, `coflu_genprime`

Base input file (Fluka .inp)

☒ Existing file ☐ Upload new file

Load from:

Aux file: (remove)

Aux file: (remove)

[Add aux file](#)

Job Parameters

☒ New configuration ☐ Existing condor submit *(new)*

Fluka Cycles per group:

Number of groups:

Total cycles:

Executable: ☐ Use default

Fluka Dir./\$FLUPRO: ☒ Use default

Fluka Script: ☒ Use default

Options: ☐ Long Job ☐ Low Priority

Seed Source

☒ Create new seeds ☐ Existing seed file

Initial seed:

Figure 5.15: Coflu-Web - Submitting simulation

JOB SUBMISSION

Commands completed successfully.

```

74177
92203
111697

#
# Condor submit file for FLUKA.
# Generated automatically by gensubmit.
(...)

Detected input file: test_input_1.inp
Detected input file: test_input_2.inp
Detected input file: test_input_0.inp

[FINISH STATUS] 3 jobs submitted!

[lbqfield.map', 'cmsfield.map', 'fluka_ex']
added new parameter to input file:74177
added new parameter to input file:92203
added new parameter to input file:111697

```

13:07:00: Object restored from session! 13:07:00: Connection successful! 13:07:00: Authorization successful !
13:07:00: Executing 'cd /temp2/cms_test;pwd' Out: '/temp2/cms_test'
13:07:00: Executing 'cd /temp2/cms_test;coflu_genprime.py 3' Out: '74177 92203 111697'
13:07:01: Executing 'cd /temp2/cms_test;coflu_gensubmit.py test_input.inp -N0 -M3 -e fluka_ex' Out: '# # Condor submit file for FLUKA. # Generated automatically by gensubmit. # Executable = /temp2/Software/FLUKA/2006.3b.10/fluti/cfluka Arguments = "-e fluka_ex -N0 -M3 test_input" Universe = vanilla Error = error.log Output = output.log Log = log.log Notification = Never Environment = "FLUPRO=/temp2/Software/FLUKA/2006.3b.10" Getenv = True Requirements = Memory > 50 +IsLongJob = False +IsNormalPriorityJob = True Queue'
13:07:01: Executing 'cd /temp2/cms_test;echo '#GROUPS=3' >> test_input.submit; echo 'Group count saved to submit file' Out: 'Group count saved to submit file'
13:07:01: Executing 'cd /temp2/cms_test;coflu_inputs.py test_input.inp test_input.submit PrimeNumbers.seed -n3 -aux:cmsfield.map -aux:lbqfield.map -run' Out: '[lbqfield.map', 'cmsfield.map', 'fluka_ex'] added new parameter to input file:74177 added new parameter to input file:92203 added new parameter to input file:111697'
13:07:07: Executing 'cd /temp2/cms_test;ls -lF | grep '^-' | awk '{print \$9}' Out: 'cmsfield.map fluka_ex* lbqfield.map PrimeNumbers.seed test_input.inp test_input.submit' splitting:6

Authentication

Logged as:
Fernando Pereira
(on pctsrpgs3)

[Logout](#)



Powered by condor HTC platform.



Fluka - the Monte Carlo simulation software.

CERN.CH

Figure 5.16: Job submission results in debugging mode

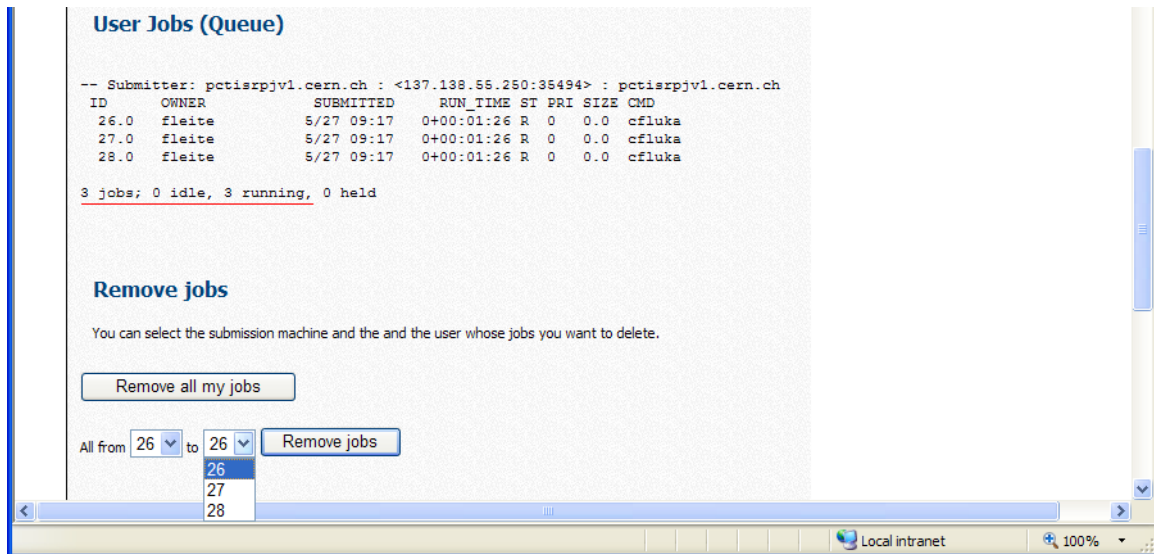


Figure 5.17: job-status

is the first invoked tool which outputs the generated seeds, followed by `coflu_gensubmit` and its output. The third command adds the “number of groups” parameter to the submit file, so this setting is defined when importing configuration (5.5.4). As a last command, `coflu_inputs` is executed to create the whole simulation structure and is given the “-run” option to start the simulation immediately.

Checking status and removing jobs

Module Status allows the user to check and remove their jobs from the cluster. Continuing the previous example, the user should be able to see the three jobs he submitted.

In Figure 5.17 on page 65 it is possible to see a first group of information showing jobs’ details from the current user, including their status, run time, memory size, etc. In the lower group, besides the “Remove all my jobs” button, the system provides the user the possibility to select a range of jobs owned by him for removal. This operation only removes jobs from being executed within Condor, which means no files are deleted.

Chapter 6

Summary and conclusions

The design and installation of a cluster for a very specific environment is always a challenge. Even though the constant advancements in technology have permitted the development of new architectures and the evolution of cluster management software, it can turn out to be a rather complex task to tune a cluster to obtain the maximum profit from a set of resources. Since the performance of a cluster is highly dependent on its usage, an optimized configuration has to take into account a number of factors including the software which will be using the resources, how people use it and how they expect to use it within the new cluster framework. Additionally, the hardware platform introduces some restrictions, principally when it is composed by a group of existing and diverse resources.

The analysis of the problem, which addresses the previous questions, is performed in chapter 2. The objective is to distribute individual simulation runs to a number of cluster nodes so that the resources are used more efficiently. The simulations are performed using the Monte-Carlo code FLUKA and may run up to several weeks. Because of its complexity and copyright license, FLUKA offers no possibility to be modified for parallel execution. From the point of view of the execution profile, each simulation runs through a script file which may run FLUKA for several times and plays an important role for gathering results. Furthermore, simulations are usually submitted in batches whose frequency and size depends on the project.

In order to select the appropriate software to manage the cluster jobs, an analysis of the current state-of-the-art is carried at chapter 3. Since a number of jobs are to be run concurrently, the system should be design towards High-Throughput-Computing (HTC). Based on the project requirements, a comparison between three of the most popular Job Management Systems was performed and analyzed: the Portable Batch System (PBS), the Sun Grid Engine (SGE) and Condor. As result, Condor was found to be the most suitable option, mainly because it is extremely customizable and specially designed for HTC and CPU harvesting.

The installation of the cluster comprised several milestones. Firstly, the architecture of the cluster was defined to be constituted by two main servers and a number of cluster nodes. The two servers implement some fault-tolerance mechanisms and run the vital cluster services: the Condor Central Manager for jobs and resource matching, a NFS server for shared file system, and a NIS server for centralized authentication. After the architecture had been set, a policy to optimize the system's global efficiency was defined taking into account the machine and the jobs characteristics. Most importantly, it defines that the jobs may have one of three priorities, the resources should be ranked by its availability and CPU performance, and local user processes should always have priority over cluster jobs. To implement this policy into Condor, a set of algorithms were developed and translated into Condor's own configuration syntax.

To automate and simplify the submission and the control of the simulations a group of high-level interfaces, which extend the cluster management tools, were developed. On one hand these interfaces were designed to support both the submission of simulations through a user-friendly interface which provides the most common options, and on the other hand, the control of non-standard and personalized simulations. Therefore, they were developed in terms of two layers. The first one, composed by the COFLU-Toolkit, directly handles the simulation sources, creates the necessary files and directory structure for the simulation, and eventually submits it to the cluster. This layer supports the division of a simulation job into several smaller ones, which allowed for a considerable improvement of the performance. The second layer is dependent on this package and provides an interactive and higher-level interface for the user. This layer comprises two front ends: on one hand a shell based program which is specially convenient for shell users and may be executed from a remote shell; on the other hand an intuitive and full-featured website from which the user can set up, submit and consult his simulations, working transparently on his machine via a SSH connection established from the web-server.

Project contribution and achievements

The development of the cluster for the RP group was certainly challenging but also a very rewarding project. The in-depth analysis and appropriate definition of policies was definitely the key factor for the good performance of the system, while the implementation of the high-level user interfaces highly contributed to its fast adoption within the group.

Taking advantage of the new system's interfaces, specially the web front-end, a user may now set up and submit a simulation in less than one minute, which can include several personalized parameters and the possibility of splitting the simulation into sub-groups running in parallel. Actually, there is not much time difference between setting up different

simulations and users are guaranteed that their jobs will execute as long as any resource is free. Previously, users had to set up their simulations manually and in comparison to the new cluster framework this could take a long time, depending on the complexity of the jobs plus the time overhead to find out which computers were free. Because of the complexity of using other resources, the users often choose to execute their simulations merely on their own computers. Considering a recent real case, a user submitted three simulations divided into 5 groups, which occupied immediately 15 resources. Previously, the effort to manually set up a simulation in such a way would simply not have been worth the effort, and the user would have had to wait 5 times longer for each simulation to end before starting the next one. Additionally, thanks to Condor's fair-share policy and the custom implemented job priorities, other jobs may suspend or postpone current job execution to provide equal and fair access to resources for all users.

The main cluster is currently running with 17 nodes from 10 machines, while the second cluster at the Meyrin site, which is still in its initial state, counts 6 resource nodes. Both clusters are expected to grow gradually, depending on computational requirements, until about 30 nodes.

Beside the application for the current scenario in CERN's RP group, the investigation and results found in the framework of this thesis can be used as valuable foundations for the development of clusters where similar constraints are present. Actually, the configuration of Condor is as flexible as working with all non-interactive UNIX software, but it is especially intended for situations where there is no possibility to parallelize neither save the state (checkpoint) of the program.

Looking at the present state of the system and comparing it to the initial requirements, one can affirm that the project has completely accomplished the objectives and surpassed the expectations regarding performance and especially usability.

Future work

“Nearly every man who develops an idea works it up to the point where it looks impossible, and then he gets discouraged. That's not the place to become discouraged.”

Thomas A. Edison

This project yielded significant advancements in the execution of the Monte-Carlo simulations performed at the RP group. Nevertheless, the type of project itself offers such a large range of possibilities for future work that it seems a mere drop in the ocean.

The better acquaintance with the problem and its related issues, and the acquired knowledge from the project opened a wide set of new possibilities.

As a first outlook, a future version of project could provide a set of new functionalities.

Particularly, it would be very useful to enhance the web interface with additional modules, for instance to provide online administration of the cluster parameters or an advanced status module with continuous monitoring and statistics.

On another line of sight, regarding performance improvements, an investigation could be started to try to mitigate checkpointing restrictions for some FLUKA related points, like the usage of shell scripts. By checkpointing simulations, there would be no progress loss when the jobs move from one machine to another. Therefore, jobs would be able to move freely on the cluster each time they could get better conditions on a different resource or simply because its resource was claimed by a job with higher priority.

Even with the support of jobs to save their state and move freely, the efficiency of the cluster is always limited in the amount of jobs to execute. If the cluster had 20 nodes and only one job would be submitted without explicit manual splitting into sub-groups, the job would occupy only one node and therefore, the cluster efficiency would be as low as 5%. This fact leads us to the root of the problem: FLUKA does not currently support inherent parallelism and therefore the granularity unit is the simulation job. Because of simulation characteristics (see [Fluka restrictions]) changing this scenario would be an extremely complex task and therefore it is still far from reality, even though it's certainly an option for the future.

Final words

The fields of parallel and distributed computing are a state-of-the-art subject within computer architecture. Everyday research leads to constants evolvments and higher and higher objectives which were never thought of before. During the writing of this thesis a new milestone was achieved by IBM whose new supercomputer - Roadrunner - broke the petaflop barrier [34]. Something like 1.024 quadrillion floating point operations per second, which is four times the performance of the last year's most powerful computer.

High performance computing is utilized in all kind of todays' fields, especially in education and research. CERN is one good example, where those technologies support a noble mission - to learn and better understand the laws of the Universe. Even being a drop in the ocean, the fact of the current project to contribute for such fascinating objectives has been a powerful motivation source for the author. And that's about what history has shown: high objectives and passionate work has always lead to great results. Let them be used for human knowledge and progress.

Bibliography

- [1] Cern about-us webpage. <http://public.web.cern.ch/public/en/About/About-en.html>.
- [2] Cern's mission. <http://public.web.cern.ch/public/en/About/Mission-en.html>.
- [3] History highlights. <http://public.web.cern.ch/public/en/About/History-en.html>.
- [4] *CERN LHC: the guide. faq. frequently asked questions*. CERN, Geneva, 2006.
- [5] Atlas webpage. www.atlas.ch.
- [6] C Lefevre. Grid brochure (english version). Mar 2008.
- [7] S Dasu, V Puttabuddhi, S Rader, D Bradley, M Livny, and W Smith. Use of condor and glow for cms simulation production. 2005.
- [8] B.Mellado Sau Lan Wu M.Chen, A. Leung and N.Xu. Uw-atlas experiences with condor. In *Paradyn / Condor Week*. University of Wisconsin-Madison, 2008.
- [9] L. Gardner, R; Perini. *ATLAS eNews*, 200109, 2001.
- [10] Safety commission website. safety-commission.web.cern.ch.
- [11] CERN. *The Safety Commission (SC)*, number A-12, May 2005.
- [12] Radiation protection group webpage. www.cern.ch/radiation/.
- [13] CERN. *SAFETY CODE: Protection against Ionizing Radiation, Radiation Safety Manual*, March 1996.
- [14] Los alamos national laboratory mcnp home page. <http://mcnp-green.lanl.gov/>.
- [15] Fluka website. www.fluka.org.
- [16] A Fasso, A Ferrari, P R Sala, and J Ranft. *FLUKA: a multi-particle transport code*. CERN Yellow Report, INFN/TC_05/11, SLAC-R-773, October 2005.

-
- [17] A Fasso, A Ferrari, S Roesler, P R Sala, F Ballarini, A Ottolenghi, G Battistoni, F Cerutti, E Gadioli, M V Garzelli, A Empl, and J Ranft. The physics models of fluka: status and recent development. Technical Report hep-ph/0306267, SLAC, Stanford, CA, Jun 2003.
 - [18] Joseph D Sloan. *High Performance LINUX Clusters: With OSCAR, Rocks, Open-Mosix & MPI*. O'Reilly, Sebastopol, CA, 2005.
 - [19] Nug30 press release, 2000. <http://www-unix.mcs.anl.gov/metaneos/nug30/pr.html>.
 - [20] Aske Plaat Jonathan Schaeffer. Kasparov versus deep blue: The re-match. *ICCA Journal*, 20(2):95–102, 1997.
 - [21] Michael Feldman. Ibm roadrunner takes the gold in the petaflop race. *HPCwire*, June 2008.
 - [22] Nikitas Alexandridis Frederic Vroman Nguyen Nguyen Jacek R. Radzikowski Preeyapong Samipagdi Suboh A. Suboh Tarek El-Ghazawi, Kris Gaj. A performance study of job management systems. *Concurrency and Computation: Practice and Experience*, 16(Issue 13):1229–1246, October 2004.
 - [23] Branimir RadicEmir Imamagic Dobrisa Dobrenic Branimir Radic Emir Imamagic, Dobrisa Dobrenic. Job management systems analysis. In *6th Carnet Users Conference*. University Computing Centre - Srce, Croatia, 2004.
 - [24] Maui user manual. <http://www.clusterresources.com/products/maui/docs/mauiusers.shtml>.
 - [25] Condor & nfs. http://www.cs.wisc.edu/condor/manual/v7.0/2_5Submitting_Job.html.
 - [26] Using condor with afs. http://www.cs.wisc.edu/condor/manual/v7.0/3_12Setting_Up.html.
 - [27] The network information system. <http://tldp.org/LDP/nag2/x-087-2-nis.html>.
 - [28] Python programming language – official website. <http://www.python.org/>.
 - [29] Stephen Ferg. Python & java - a side-by-side comparison. 2007. http://www.ferg.org/projects/python_java_side-by-side.html.
 - [30] Pear - php extension and application repository. <http://pear.php.net/>.
 - [31] Pecl ssh2 package information. <http://pecl.php.net/package/ssh2>.
 - [32] Apache security tips. http://httpd.apache.org/docs/2.0/misc/security_tips.html.
 - [33] Thomas Akin. Dangers of suid shell scripts. *Sys Admin Magazine*, June 2001. <http://www.samag.com/documents/s=1149/sam0106a/0106a.htm>.

-
- [34] Erica Ogg. Ibm's roadrunner breaks petaflop barrier, tops supercomputer list. *CNET news*, June 2008. http://news.cnet.com/8301-10784_3-9971006-7.html.

Glossary

Condor pool - A collection of computers used by Condor which may be dedicated or interactive. The primary definition for “cluster” assumes computers are not for interactive use.

Job Scheduling - The task of assigning jobs to cluster resources by matching their characteristics. In most Job Management Systems, administrators are able to define a customized policy to improve matching and obtain higher efficiency on resources usage.

Advance reservation - Mostly used for grid clusters, advanced reservation stands for the possibility of resources to be used by a pre-determined set of jobs, including those from the same user or assigned by a specific Job scheduler.

Fair share - A scheduling algorithm that tracks users’ resource consumption history and limits his job execution in order to enforce fair cluster usage among users.

Backfilling - A scheduling algorithm that distributes short jobs on resources that are reserved for usage in future. Backfilling is therefore very important for Schedulers that enable advanced reservation.

Checkpointing - The procedure of storing the state of the active process on hard drive. Stored state of process is used to restart the process from that point.

Process migration - The movement of jobs or processes from one node to another. Jobs are then restarted from the last available checkpoint or from the beginning.

Dynamic load balancing - The process of evenly distribute the load between available cluster nodes, by moving or changing jobs’ behavior.

File stage in/out - File stage in/out is process of copying defined set of files on the node before the job is executed, and copying back another set of files after the job is finished. This capability is important for JMSs having no support for remote system calls neither access to a shared file system.

CPU harvesting - The capability of a cluster to take advantage of non-dedicated computers without interfering with its local processes.

Preemption - The process of a resource to evict the job it's running, usually to allow a higher priority job to execute. The job gets back to the queue and waits until being assigned a new resource.

Suspension - The process of a resource to pause its executing job. This measure can be taken both for enable load balancing and CPU harvesting as well.

Appendix A

Relevant Condor configuration

```
1 #####
2 ## Part 1: Settings you must customize:
3 #####
4
5 ## What machine is your central manager?
6 CONDOR_HOST = pcscrpsrv2.cern.ch
7
8 ## Where is the local condor directory for each host?
9 LOCAL_DIR = $(RELEASE_DIR)/local
10 ## Where is the machine-specific local config file for each host?
11 LOCAL_CONFIG_FILE = $(RELEASE_DIR)/etc/localconf.sh|
12
13
14 #####
15 ## Part 2: Settings you may want to customize:
16 #####
17
18 ## The user/group ID <uid>.<gid> of the "Condor" user.
19 CONDOR_IDS=0.0
20
21 ## What machines have administrative rights for your pool?
22 HOSTALLOW_ADMINISTRATOR = $(CONDOR_HOST)
23 ## What machines should have "owner" access to your machines
24 HOSTALLOW_OWNER = $(FULL_HOSTNAME), $(HOSTALLOW_ADMINISTRATOR)
25
26 # No checkpoint server. Useless in Vanilla.
27 #CKPT_SERVER_HOST = checkpoint-server-hostname.your.domain
28
29
30 #####
31 ## Part 3: Settings control the policy for running, stopping, and
32 #####
33
34 ## This section contains macros are here to help write legible
35 ## expressions:
36 MINUTE = 60
37 HOUR = (60 * $(MINUTE))
38 DAY = (24 * $(HOUR))
39 StateTimer = (CurrentTime - EnteredCurrentState)
40 ActivityTimer = (CurrentTime - EnteredCurrentActivity)
41 ActivationTimer = (CurrentTime - JobStart)
42 LastCkpt = (CurrentTime - LastPeriodicCheckpoint)
43
44 RealCpus = (TotalSlots / 3)
45 NonCondorLoadAvg = (TotalLoadAvg - TotalCondorLoadAvg)
46 AvailableCPU = ( $(RealCpus) - TotalLoadAvg )
47 BackgroundLoad = 0.2
48 HighestLoad = ( $(RealCpus) + $(BackgroundLoad) )
49
50 StartIdleTime = 15 * $(MINUTE)
51 ContinueIdleTime = 5 * $(MINUTE)
52 MaxSuspendTime = 10 * $(MINUTE)
```

```

53 MaxVacateTime          = 10 * $(MINUTE)
54
55 KeyboardBusy           = ( KeyboardIdle < $(MINUTE) )
56 ConsoleBusy           = ( ConsoleIdle < $(MINUTE) )
57 CPUIdle               = ( TotalLoadAvg <= $(BackgroundLoad) )
58 CPUNotBusy            = ( $(AvailableCPU) > ( 1 - $(BackgroundLoad) ) )
59 CPUBusy               = ( TotalLoadAvg > $(HighestLoad) )
60
61 KeyboardNotBusy        = ( $(KeyboardBusy) == False )
62
63 BigJob                 = ( TARGET.ImageSize >= ( 50 * 1024 ) )
64 MediumJob              = ( TARGET.ImageSize >= ( 15 * 1024 ) && TARGET.ImageSize < ( 50 *
    1024 ) )
65 SmallJob               = ( TARGET.ImageSize < ( 15 * 1024 ) )
66
67 JustCPU                = ( $(CPUBusy) && ( $(KeyboardBusy) == False ) )
68 MachineBusy            = ( $(CPUBusy) || $(KeyboardBusy) )
69
70
71 ## RANK is 0, meaning that all jobs have an equal preference.
72 #RANK                  = 0
73
74 ##### POLICY DEFINITION #####
75
76 # When should we only consider SUSPEND instead of PREEMPT?
77 WANT_SUSPEND           = $(CERN_WANT_SUSPEND)
78
79 # When should we preempt gracefully instead of hard-killing?
80 WANT_VACATE            = $(CERN_WANT_VACATE)
81
82 ## When is this machine willing to start a job?
83 START                  = $(CERN_START)
84
85 ## When to suspend a job?
86 SUSPEND                = $(CERN_SUSPEND)
87
88 ## When to resume a suspended job?
89 CONTINUE               = $(CERN_CONTINUE)
90
91 ## When to nicely stop a job?
92 ## (as opposed to killing it instantaneously)
93 PREEMPT                = $(CERN_PREEMPT)
94
95 ## When to instantaneously kill a preempting job
96 ## (e.g. if a job is in the pre-empting stage for too long)
97 KILL                   = $(CERN_KILL)
98
99 PERIODIC_CHECKPOINT    = $(UWCS_PERIODIC_CHECKPOINT)
100 PREEMPTION_REQUIREMENTS = $(CERN_PREEMPTION_REQUIREMENTS)
101 PREEMPTION_RANK        = $(UWCS_PREEMPTION_RANK)
102
103 NEGOTIATOR_PRE_JOB_RANK = $(CERN_NEGOTIATOR_PRE_JOB_RANK)
104
105 NEGOTIATOR_POST_JOB_RANK = $(CERN_NEGOTIATOR_POST_JOB_RANK)
106 MaxJobRetirementTime  = $(UWCS_MaxJobRetirementTime)
107 CLAIM_WORKLIFE          = $(UWCS_CLAIM_WORKLIFE)
108
109 # Faster Adjust of user priorities
110 PRIORITY_HALFLIFE       = $(CERN_PRIO_HALFLIFE)
111
112 #####
113 ## CERN SC/VP Configuration. ##
114 #####
115
116 ##### Configuration Values (Adjust as needed) #####
117 CERN_MAX_TIME_NON_LONG_JOB = 7 * 24 * $(HOUR)
118 CERN_MAX_SUSPENDED_TIME = 10 * $(MINUTE)
119 CERN_PRIO_HALFLIFE = 1 * $(HOUR)
120 NEGOTIATOR_INTERVAL = 10
121
122 ##### RULES #####

```

```

123
124 CERN_START = $(CPUNotBusy) && $(START_PRIO)
125
126 CERN_SUSPEND = ( ( $(CPUBusy) && ( ( SlotID < 4 ) || ( CondorLoadAvg > 0.40 ) )
127   ) \
128     || $(SUSPEND_PRIO))
129 CERN_CONTINUE = $(CPUNotBusy) && $(CONTINUE_PRIO)
130
131
132 START_PRIO = ( ( SlotID == 1) && $(SLOTH1_START) || \
133   ( SlotID == 2) && $(SLOTN1_START) || \
134   ( SlotID == 3) && $(SLOTL1_START) || \
135   ( SlotID == 4) && $(SLOTH2_START) || \
136   ( SlotID == 5) && $(SLOTN2_START) || \
137   ( SlotID == 6) && $(SLOTL2_START) )
138
139 SLOTH1_START = (Owner == "condor")
140 SLOTH2_START = (Owner == "condor")
141
142 SLOTN1_START = (slot1_Activity != "Busy") && (TARGET.IsNormalPrioJob == TRUE)
143 SLOTN2_START = (slot4_Activity != "Busy") && (TARGET.IsNormalPrioJob == TRUE)
144
145 SLOTL1_START = (slot1_Activity != "Busy") && (slot2_Activity != "Busy") && (TARGET.
146   IsNormalPrioJob != TRUE)
147 SLOTL2_START = (slot4_Activity != "Busy") && (slot5_Activity != "Busy") && (TARGET.
148   IsNormalPrioJob != TRUE)
149
150 SUSPEND_PRIO = ( ( ( SlotID == 2) && (slot1_Activity == "Busy" ) ) || \
151   ( ( SlotID == 3) && ( (slot1_Activity == "Busy" ) || (
152     slot2_Activity == "Busy" ) ) ) || \
153   ( ( SlotID == 5) && (slot4_Activity == "Busy" ) ) || \
154   ( ( SlotID == 6) && ( (slot4_Activity == "Busy" ) || (
155     slot5_Activity == "Busy" ) ) ) )
156
157 CONTINUE_PRIO = ( ( ( SlotID == 2) && (slot1_Activity != "Busy" ) ) || \
158   ( ( SlotID == 3) && ( (slot1_Activity != "Busy" ) && (
159     slot2_Activity != "Busy" ) ) ) || \
160   ( ( SlotID == 5) && (slot4_Activity != "Busy" ) ) || \
161   ( ( SlotID == 6) && ( (slot4_Activity != "Busy" ) && (
162     slot5_Activity != "Busy" ) ) ) )
163
164 # Preempts Jobs suspended for more than 10H / Jobs running for a short time / When
165 # a Long job is taking its resources (higher priority).
166 CERN_PREEMPT = ( ( (TotalJobSuspendTime != UNDEFINED) && (TotalJobSuspendTime > $(
167   CERN_MAX_SUSPENDED_TIME)) ) || \
168   ( (Target.isLongJob != True) && (TotalJobRunTime != UNDEFINED) &&
169     (TotalJobRunTime > $(CERN_MAX_TIME_NON_LONG_JOB)) ) ) || \
170   ( ( SlotID == 2 ) && (slot1_IsLongJob == TRUE) ) || \
171   ( ( SlotID == 3 ) && ( (slot2_IsLongJob == TRUE) || (
172     slot1_IsLongJob == TRUE) ) ) || \
173   ( ( SlotID == 5 ) && (slot4_IsLongJob == TRUE) ) || \
174   ( ( SlotID == 6 ) && ( (slot5_IsLongJob == TRUE) || (
175     slot4_IsLongJob == TRUE) ) ) ) )
176
177 ##### RANKING FLAGS #####
178
179 #Define Job Load per "CORE"
180 CORE1_JOBLOAD = ( 0.5 * (MY.slot2_RemoteOwner != UNDEFINED) + 0.25 * (MY.
181   slot3_RemoteOwner != UNDEFINED) )
182 CORE2_JOBLOAD = ( 0.5 * (MY.slot5_RemoteOwner != UNDEFINED) + 0.25 * (MY.
183   slot6_RemoteOwner != UNDEFINED) )
184
185 #Machine Availability (0 - Used / <1 - Lower priority slots are being used)
186 SLOT_AVAILABILITY = ( 1 - ( ( MY.SlotID < 4 ) * $(CORE1_JOBLOAD) + ( MY.SlotID > 3
187   ) * $(CORE2_JOBLOAD) ) )
188
189 #Ranks machines considering "neighbor" slots availability

```

```

179 CERN_NEGOTIATOR_PRE_JOB_RANK = ( RemoteOwner == UNDEFINED ) * $(SLOT_AVAILABILITY)
180
181 #Rank Machines as they're faster
182 CERN_NEGOTIATOR_POST_JOB_RANK = KFlops
183 #CERN_NEGOTIATOR_POST_JOB_RANK = ( ( MY.SlotID < 4 ) * ( slot1_KFlops +
    slot2_KFlops + slot3_KFlops ) + ( MY.SlotID > 3 ) * ( slot4_KFlops +
    slot5_KFlops + slot6_KFlops ) )
184
185
186 ##### DECISION FLAGS #####
187
188 #Prefer suspension - Preempt whenever a non-long job is running for more time than
    allowed
189 CERN_WANT_SUSPEND = True
190
191 #CERN_WANT_SUSPEND = True
192
193 #Kill iff too much time to vacate
194 CERN_KILL = $(UWCS_KILL)
195
196
197 ##### USER PRIORITIES #####
198
199 ## Will only preempt running jobs (suspended jobs cannot be replaced)
200 CERN_PREEMPTION_REQUIREMENTS = ( ( Activity == "Busy" ) && ( RemoteUserPrio >
    SubmitterUserPrio * 1.2 ) )
201
202 ### Suspended resources should not be considered for user priority
203 NEGOTIATOR_DISCOUNT_SUSPENDED_RESOURCES = True
204
205
206 #####
207 ## Part 4: Advanced daemon settings
208 #####
209
210 ##### condor_startd #####
211
212 ## Information Shared across machine Slots
213 STARTD_SLOT_ATTRS = State, Activity, RemoteOwner, IsLongJob
214
215 ## Send updates often because we need to know the state of the slots
216 UPDATE_INTERVAL = 10
217
218 ## When a machine unclaimed, when should it run benchmarks?
219 BenchmarkTimer = (CurrentTime - LastBenchmark)
220 RunBenchmarks : (LastBenchmark == 0 ) || ($(BenchmarkTimer) >= (4 * $(HOUR)))
221
222 STARTD_ATTRS = COLLECTOR_HOST_STRING, IsDesktop
223
224 ## Advertised attributes from the ClassAd of the job its working on.
225 STARTD_JOB_EXPRS = ImageSize, ExecutableSize, JobUniverse, NiceUser, IsLongJob
226
227 ## How many CPUs your machine has. Specify default to 3 Slots
228 NUM_CPUS = 3
229
230
231 ##### condor_schedd #####
232
233 ##Auto Remove NonLong jobs which ran for too long
234 SYSTEM_PERIODIC_REMOVE = ( (IsLongJob != TRUE) && (TotalTimeClaimedBusy > $(
    CERN_MAX_TIME_NON_LONG_JOB) ) )
235
236 ## What users do you want to grant super user access to this job
237 ## queue? (These users will be able to remove other user's jobs).
238 QUEUE_SUPER_USERS = root, condor
239
240 ##### condor_starter #####
241
242 #Runner mode defaults to Server
243 IsDesktop = False
244 JOB_RENICE_INCREMENT = ( 0 + 15 * $(IsDesktop) )

```

Appendix B

Condor policy test outputs

B.1 Priority behavior and resources ranking

B.1.1 Without preemption

```
Log1
1 job(s) submitted to cluster 162.

— Submitter: pccrpsrv2.cern.ch : <137.138.213.89:60885> : pccrpsrv2.cern.ch
  ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
  162.0    fleite      3/27 14:32      0+00:00:07 R  0   0.0  rfluka
1 jobs; 0 idle, 1 running, 0 held

1 job(s) submitted to cluster 163.

— Submitter: pccrpsrv2.cern.ch : <137.138.213.89:60885> : pccrpsrv2.cern.ch
  ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
  162.0    fleite      3/27 14:32      0+00:00:24 R  0   0.0  rfluka
  163.0    fleite      3/27 14:33      0+00:00:04 R  0   0.0  rfluka
2 jobs; 0 idle, 2 running, 0 held

Name                               OpSys      Arch      State      Activity  LoadAv  Mem  ActvtyTime
slot1@pctisrpgrs3. LINUX      INTEL     Unclaimed  Idle      0.000    167  0+00:07:55
slot2@pctisrpgrs3. LINUX      INTEL     Owner      Idle      0.060    167  0+00:03:49
slot3@pctisrpgrs3. LINUX      INTEL     Unclaimed  Idle      0.000    167  0+00:03:47
slot1@pctisrpjv1.c LINUX      INTEL     Unclaimed  Idle      0.000    83   0+00:17:49
slot2@pctisrpjv1.c LINUX      INTEL     Owner      Idle      0.000    83   0+00:03:05
slot3@pctisrpjv1.c LINUX      INTEL     Claimed    Busy      0.410    83   0+00:00:27
slot4@pctisrpjv1.c LINUX      INTEL     Unclaimed  Idle      0.000    83   0+00:17:52
slot5@pctisrpjv1.c LINUX      INTEL     Owner      Idle      0.360    83   0+00:17:58
slot6@pctisrpjv1.c LINUX      INTEL     Claimed    Busy      0.000    83   0+00:00:09

#####

cd ../run1 (Normal priority jobs)
1 job(s) submitted to cluster 164.
1 job(s) submitted to cluster 165.

— Submitter: pccrpsrv2.cern.ch : <137.138.213.89:60885> : pccrpsrv2.cern.ch
  ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
  162.0    fleite      3/27 14:32      0+00:01:06 R  0   0.0  rfluka
  163.0    fleite      3/27 14:33      0+00:00:46 R  0   0.0  rfluka
  164.0    fleite      3/27 14:33      0+00:00:21 R  0   0.0  rfluka
  165.0    fleite      3/27 14:33      0+00:00:01 R  0   0.0  rfluka
4 jobs; 0 idle, 4 running, 0 held

Name                               OpSys      Arch      State      Activity  LoadAv  Mem  ActvtyTime
slot1@pctisrpgrs3. LINUX      INTEL     Unclaimed  Idle      0.000    167  0+00:08:55
slot2@pctisrpgrs3. LINUX      INTEL     Claimed    Busy      0.370    167  0+00:00:32
slot3@pctisrpgrs3. LINUX      INTEL     Owner      Idle      0.000    167  0+00:00:28
slot1@pctisrpjv1.c LINUX      INTEL     Unclaimed  Idle      0.000    83   0+00:18:50
slot2@pctisrpjv1.c LINUX      INTEL     Claimed    Busy      0.030    83   0+00:00:11
```

```

slot3@pctisrnpjv1.c LINUX      INTEL  Claimed  Suspende  0.860    83  0+00:00:08
slot4@pctisrnpjv1.c LINUX      INTEL  Unclaimed Idle       0.000    83  0+00:18:43
slot5@pctisrnpjv1.c LINUX      INTEL  Owner    Idle       0.430    83  0+00:18:49
slot6@pctisrnpjv1.c LINUX      INTEL  Claimed  Busy       0.880    83  0+00:01:00

```

```
#####
```

```
cd ../run2 (Again low priority)
```

```
1 job(s) submitted to cluster 166.
```

```
— Submitter: pcscrpsrv2.cern.ch : <137.138.213.89:60885> : pcscrpsrv2.cern.ch
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
162.0	fleite	3/27 14:32	0+00:02:55	R	0	0.0	rfluka
163.0	fleite	3/27 14:33	0+00:02:35	R	0	0.0	rfluka
164.0	fleite	3/27 14:33	0+00:02:10	R	0	0.0	rfluka
165.0	fleite	3/27 14:33	0+00:01:50	R	0	0.0	rfluka
166.0	fleite	3/27 14:34	0+00:00:00	I	0	0.0	rfluka

```
5 jobs; 1 idle, 4 running, 0 held
```

```
— Submitter: pcscrpsrv2.cern.ch : <137.138.213.89:60885> : pcscrpsrv2.cern.ch
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
162.0	fleite	3/27 14:32	0+00:03:09	R	0	0.0	rfluka
165.0	fleite	3/27 14:33	0+00:02:04	R	0	0.0	rfluka
166.0	fleite	3/27 14:34	0+00:00:07	R	0	0.0	rfluka

```
3 jobs; 0 idle, 3 running, 0 held
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:10:35
slot2@pctisrpgrs3.	LINUX	INTEL	Owner	Idle	0.900	167	0+00:00:05
slot3@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:00:06
slot1@pctisrnpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:20:31
slot2@pctisrnpjv1.c	LINUX	INTEL	Claimed	Busy	1.020	83	0+00:02:02
slot3@pctisrnpjv1.c	LINUX	INTEL	Claimed	Suspende	0.000	83	0+00:01:59
slot4@pctisrnpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:20:34
slot5@pctisrnpjv1.c	LINUX	INTEL	Owner	Idle	0.270	83	0+00:20:40
slot6@pctisrnpjv1.c	LINUX	INTEL	Claimed	Busy	0.820	83	0+00:00:07

```
— Submitter: pcscrpsrv2.cern.ch : <137.138.213.89:60885> : pcscrpsrv2.cern.ch
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
162.0	fleite	3/27 14:32	0+00:03:50	R	0	0.0	rfluka
166.0	fleite	3/27 14:34	0+00:00:48	R	0	0.0	rfluka

```
2 jobs; 0 idle, 2 running, 0 held
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:11:25
slot2@pctisrpgrs3.	LINUX	INTEL	Owner	Idle	0.390	167	0+00:00:56
slot3@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:00:56
slot1@pctisrnpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:21:21
slot2@pctisrnpjv1.c	LINUX	INTEL	Owner	Idle	0.810	83	0+00:00:11
slot3@pctisrnpjv1.c	LINUX	INTEL	Claimed	Busy	0.060	83	0+00:00:06
slot4@pctisrnpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:21:24
slot5@pctisrnpjv1.c	LINUX	INTEL	Owner	Idle	0.290	83	0+00:21:30
slot6@pctisrnpjv1.c	LINUX	INTEL	Claimed	Busy	0.770	83	0+00:00:48

B.1.2 With preemption

```
/run2 > condor_submit submit_test
```

```
000 (230.000.000) 03/31 14:14:27 Job submitted from host: <137.138.213.89:60745>
```

```
/run2 > condor_submit submit_test
```

```
000 (231.000.000) 03/31 14:14:30 Job submitted from host: <137.138.213.89:60745>
```

```
001 (231.000.000) 03/31 14:14:46 Job executing on host: <137.138.55.250:37466>
```

```
001 (230.000.000) 03/31 14:14:46 Job executing on host: <137.138.55.250:37466>
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:00:00	R	0	0.0	rfluka
231.0	fleite	3/31 14:14	0+00:00:00	R	0	0.0	rfluka

```
2 jobs; 0 idle, 2 running, 0 held
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:13:15
slot2@pctisrpgrs3.	LINUX	INTEL	Owner	Idle	0.000	167	0+00:13:23

```

slot3@pctisrpgrs3. LINUX      INTEL  Unclaimed Idle    0.000  167  0+00:13:17
slot1@pctisrpjv1.c LINUX      INTEL  Unclaimed Idle    0.000   83  0+00:13:46
slot2@pctisrpjv1.c LINUX      INTEL  Owner    Idle    0.020   83  0+00:13:56
slot3@pctisrpjv1.c LINUX      INTEL  Claimed  Busy    0.000   83  0+00:00:04
slot4@pctisrpjv1.c LINUX      INTEL  Unclaimed Idle    0.000   83  0+00:13:49
slot5@pctisrpjv1.c LINUX      INTEL  Owner    Idle    0.000   83  0+00:13:59
slot6@pctisrpjv1.c LINUX      INTEL  Claimed  Busy    0.000   83  0+00:00:09

```

```

/run1 > condor_submit submit_test
000 (232.000.000) 03/31 14:15:51 Job submitted from host: <137.138.213.89:60745>
/run1 > condor_submit submit_test
000 (233.000.000) 03/31 14:15:55 Job submitted from host: <137.138.213.89:60745>

001 (232.000.000) 03/31 14:16:06 Job executing on host: <137.138.55.251:40684>
001 (233.000.000) 03/31 14:16:07 Job executing on host: <137.138.55.250:37466>

010 (230.000.000) 03/31 14:16:28 Job was suspended.
004 (230.000.000) 03/31 14:16:28 Job was evicted

```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:01:42	I	0	244.1	rfluka
231.0	fleite	3/31 14:14	0+00:01:51	R	0	0.0	rfluka
232.0	fleite	3/31 14:15	0+00:00:31	R	0	0.0	rfluka
233.0	fleite	3/31 14:15	0+00:00:31	R	0	0.0	rfluka

4 jobs; 1 idle, 3 running, 0 held

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:14:45
slot2@pctisrpgrs3.	LINUX	INTEL	Claimed	Busy	0.180	167	0+00:00:20
slot3@pctisrpgrs3.	LINUX	INTEL	Owner	Idle	0.020	167	0+00:00:16
slot1@pctisrpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:07
slot2@pctisrpjv1.c	LINUX	INTEL	Claimed	Busy	0.000	83	0+00:00:02
slot3@pctisrpjv1.c	LINUX	INTEL	Claimed	Suspende	0.900	83	0+00:00:06
slot4@pctisrpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:24
slot5@pctisrpjv1.c	LINUX	INTEL	Owner	Idle	0.210	83	0+00:15:33
slot6@pctisrpjv1.c	LINUX	INTEL	Claimed	Busy	0.900	83	0+00:01:42

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@pctisrpgrs3.	LINUX	INTEL	Unclaimed	Idle	0.000	167	0+00:14:45
slot2@pctisrpgrs3.	LINUX	INTEL	Claimed	Busy	0.180	167	0+00:00:20
slot3@pctisrpgrs3.	LINUX	INTEL	Owner	Idle	0.020	167	0+00:00:16
slot1@pctisrpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:28
slot2@pctisrpjv1.c	LINUX	INTEL	Claimed	Busy	0.160	83	0+00:00:26
slot3@pctisrpjv1.c	LINUX	INTEL	Owner	Idle	1.320	83	0+00:00:04
slot4@pctisrpjv1.c	LINUX	INTEL	Unclaimed	Idle	0.000	83	0+00:15:24
slot5@pctisrpjv1.c	LINUX	INTEL	Owner	Idle	0.210	83	0+00:15:33
slot6@pctisrpjv1.c	LINUX	INTEL	Claimed	Busy	0.900	83	0+00:01:42

```

/run2 > condor_submit submit_test
000 (234.000.000) 03/31 14:17:30 Job submitted from host: <137.138.213.89:60745>

```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:01:42	I	0	244.1	rfluka
231.0	fleite	3/31 14:14	0+00:02:48	R	0	0.0	rfluka
232.0	fleite	3/31 14:15	0+00:01:28	R	0	0.0	rfluka
233.0	fleite	3/31 14:15	0+00:01:28	R	0	0.0	rfluka
234.0	fleite	3/31 14:17	0+00:00:00	I	0	0.0	rfluka

5 jobs; 2 idle, 3 running, 0 held

```

005 (231.000.000) 03/31 14:17:44 Job terminated.
001 (230.000.000) 03/31 14:17:44 Job executing on host: <137.138.55.250:37466>

```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:01:52	R	0	244.1	rfluka
232.0	fleite	3/31 14:15	0+00:01:48	R	0	0.0	rfluka
233.0	fleite	3/31 14:15	0+00:01:48	R	0	0.0	rfluka

234.0 fleite 3/31 14:17 0+00:00:00 I 0 0.0 rfluka
4 jobs; 1 idle, 3 running, 0 held

005 (232.000.000) 03/31 14:18:22 Job terminated.
001 (234.000.000) 03/31 14:18:46 Job executing on host: <137.138.55.251:40684

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
230.0	fleite	3/31 14:14	0+00:02:51	R	0	244.1	rfluka
233.0	fleite	3/31 14:15	0+00:02:47	R	0	0.0	rfluka
234.0	fleite	3/31 14:17	0+00:00:04	R	0	0.0	rfluka

4 jobs; 1 idle, 3 running, 0 held

005 (233.000.000) 03/31 14:18:51 Job terminated.
005 (230.000.000) 03/31 14:19:44 Job terminated.
005 (234.000.000) 03/31 14:21:03 Job terminated.

B.2 CPU Load management

```
1 LOCAL RUNNING + SUBMITTED 1 CONDOR + SUBMITTED 1 LOCAL
```

```
[pctisrpbjv1] /temp2/test/run1 > cfluka -N0 -M1 carbon_test1 &
[1] 21305
```

```
### Submitted 1 job to the cluster
1 job(s) submitted to cluster 742.
[FINISH STATUS] 1 jobs submitted!
```

```
[pcscrpsrv2] /temp2/test/run1 > condor_status -f "%s" Activity -f "      \t%s"
      TotalLoadAvg -f "      \t%s" TotalCondorLoadAvg -f "      \t%s" LoadAvg -f "      \t%
s\n" CondorLoadAvg
Busy          1.710000          0.330000          0.330000          0.330000
```

```
[pctisrpbjv1] /temp2/test/run1 > cfluka -N0 -M1 carbon_test1 &
[2] 31293
```

```
Busy          2.080000          0.350000          0.350000          0.350000
Suspended     2.450000          0.360000          0.360000          0.360000
```

```
### STOPPED 2nd local CFULKA
```

```
[pcscrpsrv2] /temp2/test/run1 > condor_status -f "%s" Activity -f "      \t%s"
      TotalLoadAvg -f "      \t%s" TotalCondorLoadAvg -f "      \t%s" LoadAvg -f "      \t%
s\n" CondorLoadAvg
Suspended     2.050000          0.000000          0.000000          0.000000
Suspended     1.490000          0.000000          0.000000          0.000000
Busy          1.180000          0.000000          0.000000          0.000000
Busy          1.580000          0.220000          0.220000          0.220000
Busy          1.930000          0.330000          0.330000          0.330000
```

```
2 Local Job RUNNING + SUBMITTED 1 CONDOR + STOPPED 1 Local
```

```
[pctisrpbjv1] /temp2/test/run1 > cfluka -N0 -M1 carbon_test1 &
[1] 1409
[pctisrpbjv1] /temp2/test/run1 > cfluka -N0 -M1 carbon_test1 &
[2] 1438
```

```
[pcscrpsrv2] /temp2/test/run1 > condor_status -f "%s" Activity -f "      \t%s"
      TotalLoadAvg -f "      \t%s" TotalCondorLoadAvg -f "      \t%s" LoadAvg -f "      \t%
s\n" CondorLoadAvg
Idle          1.950000          0.000000          0.950000          0.000000
```

```
### Submitted 1 job to the cluster
1 job(s) submitted to cluster 750.
[FINISH STATUS] 1 jobs submitted!
```

```
[pcscrpsrv2] /temp2/test/run1 > condor_q
```

```
ID      OWNER      SUBMITTED  RUN_TIME ST PRI SIZE CMD
750.0   fleite      6/20 09:56   0+00:00:00 I  0   0.0  cfluka
```

```
1 jobs; 1 idle, 0 running, 0 held
```

```
### Stopped 1 Local job
```

```

Idle          1.770000          0.000000          0.770000          0.000000

Idle          1.390000          0.000000          0.390000          0.000000

[pcscrpsrv2] /temp2/test/run1 > condor_q

— Submitter: pscrpsrv2.cern.ch : <137.138.213.89:57593> : pscrpsrv2.cern.ch
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
750.0    fleite      6/20 09:56      0+00:00:00 I  0   0.0  cfluka

1 jobs; 1 idle, 0 running, 0 held

Idle          1.170000          0.000000          0.170000          0.000000

[pcscrpsrv2] /temp2/test/run1 > condor_q
— Submitter: pscrpsrv2.cern.ch : <137.138.213.89:57593> : pscrpsrv2.cern.ch
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
750.0    fleite      6/20 09:56      0+00:00:01 R  0   0.0  cfluka

1 jobs; 0 idle, 1 running, 0 held

Busy          1.620000          0.590000          0.590000          0.590000

=====
1 JOB RUNNING + 2 Submitted into Condor + 1 Local Started + 1 Local Stopped + 1
  LocalStopped
=====

[pctisrjv1] /temp2/test/run1 > cfluka -N0 -M1 carbon_test1 &
[1] 3904

[pcscrpsrv2] /temp2/test/run1/20-16h11m17s_carbon_test1 > condor_status -f "%s"
Activity -f "      \t%s" TotalLoadAvg -f "      \t%s" TotalCondorLoadAvg -f "      \t
%s" LoadAvg -f "      \t%s\n" CondorLoadAvg
Idle          0.820000          0.000000          0.000000          0.000000
Idle          0.820000          0.000000          0.820000          0.000000

Busy          0.890000          0.000000          0.000000          0.000000
Busy          1.060000          0.000000          0.000000          0.000000

Suspended    2.450000          0.130000          0.010000          0.010000
Busy          2.450000          0.130000          0.120000          0.120000

Suspended    2.080000          0.340000          0.000000          0.000000
Busy          2.080000          0.340000          0.340000          0.340000

Suspended    2.030000          0.330000          0.000000          0.000000
Busy          2.030000          0.330000          0.330000          0.330000

###Submitted a second local job
cfluka -N0 -M1 carbon_test1 &
[2] 4012

Suspended    2.640000          0.410000          0.000000          0.000000
Busy          2.030000          0.330000          0.330000          0.330000

Suspended    2.640000          0.410000          0.000000          0.000000
Suspended    2.590000          0.360000          0.360000          0.360000

Suspended    2.540000          0.320000          0.000000          0.000000
Suspended    2.500000          0.280000          0.280000          0.280000

Suspended    2.240000          0.000000          0.000000          0.000000
Suspended    2.220000          0.000000          0.000000          0.000000

### STOPPED 1 local jobs

```

Suspended	1.770000	0.000000	0.000000	0.000000
Suspended	1.700000	0.000000	0.000000	0.000000
Suspended	1.330000	0.000000	0.000000	0.000000
Suspended	1.300000	0.000000	0.000000	0.000000
Busy	1.190000	0.010000	0.000000	0.000000
Suspended	1.200000	0.000000	0.000000	0.000000
Busy	1.190000	0.010000	0.000000	0.000000
Busy	1.330000	0.030000	0.020000	0.020000
Busy	1.800000	0.220000	0.080000	0.080000
Busy	1.800000	0.220000	0.140000	0.140000
Suspended	2.210000	0.540000	0.200000	0.200000
Busy	2.190000	0.550000	0.360000	0.360000
Suspended	2.210000	0.540000	0.200000	0.200000
Busy	2.190000	0.550000	0.360000	0.360000
Suspended	2.160000	0.560000	0.180000	0.180000
Busy	2.160000	0.560000	0.380000	0.380000
### STOPPED 2nd Local				
Suspended	1.620000	0.760000	0.000000	0.000000
Busy	1.680000	0.690000	0.690000	0.690000
Busy	1.180000	0.990000	0.000000	0.000000
Busy	1.220000	0.990000	0.990000	0.990000
Busy	1.820000	1.820000	0.990000	0.990000
Busy	1.820000	1.820000	0.980000	0.980000

Appendix C

Coflu-Toolkit implementation

C.1 Shared configuration parser

```
1  #!/usr/bin/python
2  #
3
4  class Configs:
5
6      #Class constructor
7      def __init__(self):
8          self.options = dict()
9
10     #Import configuration entries from a specific file
11     def parseFile( self , path ):
12         try:
13             filed = open(path)
14             for line in filed.readlines():
15                 self.__parseline(line)
16         except: return 1
17
18     #Parses a single line and adds the entry to the structure.
19     def __parseline( self , line ):
20         tline = line.strip()
21         if( len (tline) > 0 and tline[0] != '#' ):
22             [key,val] = [tline[:tline.find('=')],tline[tline.find('=')+1:]]
23             self.options[key.strip()] = val.strip()
24
25     #Returns a value given the key
26     def get( self , key ):
27         try:
28             return self.options[key]
29         except: return False
30
31     #Returns the number of entries in the structure
32     def length( self ):
33         return len( self.options )
34
35     #Adds/sets a configuration entry.
36     def set( self , key , val):
37         self.options[key.strip()] = val.strip()
```

C.2 Interactive mode functions (coflu_submit)

```
##### VALIDATION #####
### Function to read numerical from stdin.
def read_number( prompt, default = None):
    ok = False
    number = 0
    while not ok:
        raw = raw_input( prompt )
        if raw == "" and default != None: return default
        try: number = long( raw ); ok = True
        except: print "Invalid number"
    return number

### Function to read string from stdin until it meets validation
def read_valid( prompt, validation, default = None ):
    while True:
        raw = raw_input( prompt )
        if raw == "" and default != None: return default
        if validation( raw ): return raw
        else: print "Invalid input"

### Validates an input as representing Yes or No
def yesno_validator( inp ):
    return inp[0].lower() == 'n' or inp[0].lower() == 'y'

### Maps Yes/No into True/False
def mapYesNo_bool( inp ):
    return inp[0].lower() == "y"
```

Appendix D

Coflu-Web

D.1 User interface

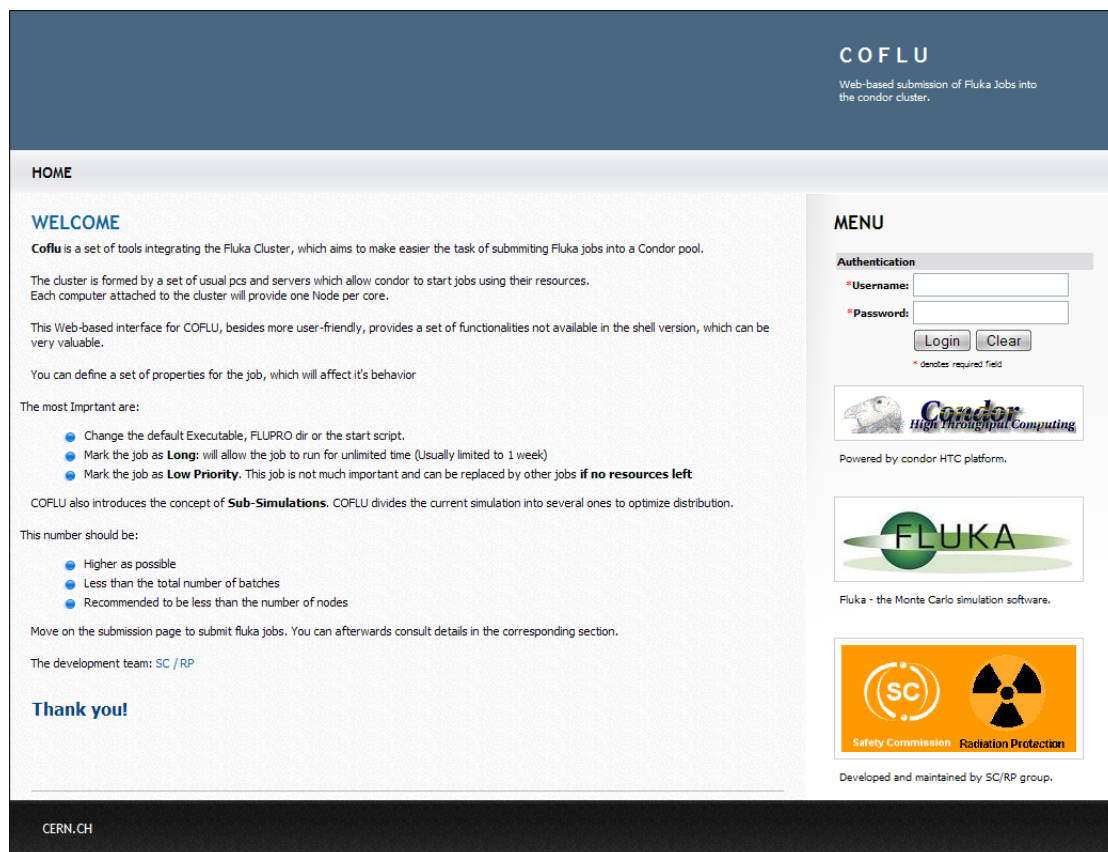


Figure D.1: Coflu-Web Home

The home page shows general information and instructions on how to use the cluster. It's the entry-point of the site and so the user can login using the authentication form.

Figure D.2: Coflu-Web Submission

The submitssion page allows users to submit simulations to the cluster. To change the current working directory, users may click “Browse” button which open a server side explorer. After choosing the location, the page reloads and allows the user to select files from that location. After filling the form with all the options and needed files, the user may “Setup only” or “Submit job”. The difference between them is that besides creating the simulation structure, “Submit job” also invokes `coflu_start` to immediately submit the simulation.

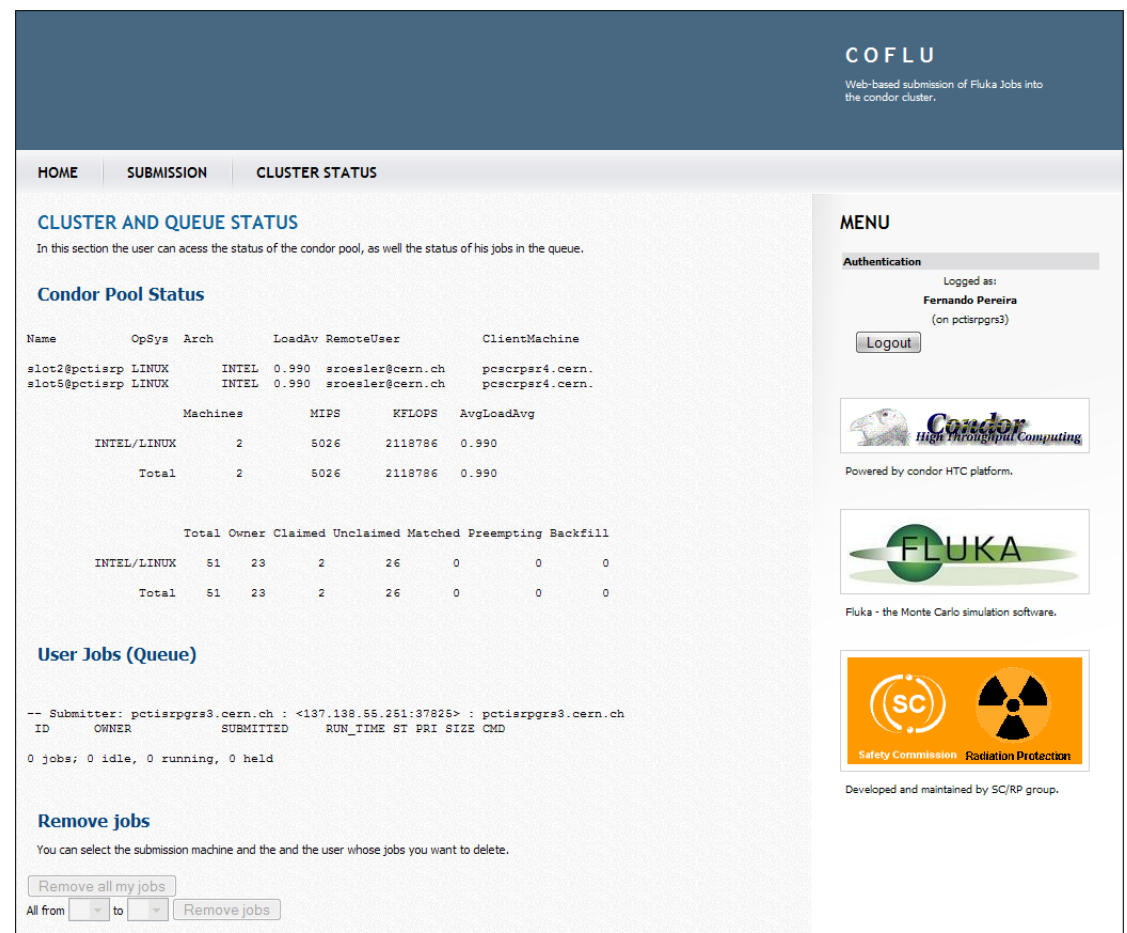


Figure D.3: Coflu-Web Status

At the status page, users may consult the global status of the cluster, the status of their jobs and remove them from execution. For security reasons, this page behavior changes whether the logged user is root or not. As in Figure D.3 on page 93, user fleite can see that two jobs are running on the cluster but cannot see their queue status neither less remove them. On the other hand, root can access all queues in the system and is allowed to remove any running job.

COFLU
Web-based submission of Fluka Jobs into the condor cluster.

[HOME](#) | [CLUSTER STATUS](#) | [ADMINISTRATION](#)

ADMINISTRATION

These are the registered users.
You can change display name and select home computer.

Username	Display name	Home computer		
theis	Chris Theis	pcscrpsrv2	Update	Remove
efeldbau	Eduard Feldbaumer	pcscrpsrv2	Update	Remove
vincke	Helmut Vincke	pcscrphv2	Update	Remove
root	root	pcscrpsrv2	Update	Remove
fleite	Fernando Pereira	pctsrpgrs3	Update	Remove
condor	condor	pctsrpjr1	Update	Remove
sroesler	Stefan Roesler	pcscrpsr4	Update	Remove
sgourgou	sgourgou	pcscrpsrv2	Update	Remove
jsampaio	jsampaio	pcscrpsrv2	Update	Remove
mfuerstn	mfuerstn	pcscrpm2	Update	Remove
hvincke	Heinz Vincke	pcscrphv2	Update	Remove

New user

MENU

Authentication
 Logged as:
root
(on pscrpsrv2)


Powered by condor HTC platform.


Fluka - the Monte Carlo simulation software.


Developed and maintained by SC/RP group.

Figure D.4: Coflu-Web Administration

Coflu-Web administration is only available for root user. In this page the administrator may add, remove or change users' details, including the machine they will be connection to via ssh. Commonly this machine is their own local machine, so they can change between web and shell interface and continue their work. Users which login for the first time - remember, authentication is providade by NIS - have their account automatically created and connecting to a default server, defined at the configuration file (configure.php).

D.2 Relevant implementation

D.2.1 sshlib.php source

```

1 <?php
2
3 // ssh protocols
4 // note: once openShell method is used, cmdExec does not work
5
6 class ssh2 {
7
8     var $host = SSH_HOST;
9     var $port = SSH_PORT;
10    var $user;
11    var $password;
12    var $con;
13    var $shell_type = 'xterm';
14    var $shell = null;
15    var $log = '';
16    var $lasterrors;
17
18    function ssh2() {
19        // $this->connect( );
20    }
21
22    function connect($host='', $port='') {
23
24        if( $host!='' ) $this->host = $host;
25        if( $port!='' ) $this->port = $port;
26
27        $this->con = ssh2_connect($this->host, $this->port);
28        if( !$this->con ) {
29            $this->log .= date('H:m:s') . " :␣Connection␣failed␣!";
30        } else {
31            $this->log .= date('H:m:s') . " :␣Connection␣successful!";
32            return true;
33        }
34        return false;
35    }
36
37    function authenticate( $user = '', $password = '', $host = '' ) {
38
39        if( $user!='' ) $this->user = $user;
40        if( $password!='' ) $this->password = $password;
41        if( $host!='' ) $this->host = $host;
42
43        if( $this->user == '' || $this->password == '' ) return false;
44
45        if( !($this->con > 0) )
46            if( !$this->connect() ) return false;
47
48        if( @!ssh2_auth_password( $this->con, $this->user, $this->password ) ) {
49            $this->log .= date('H:m:s') . " :␣Authorization␣failed␣!";
50        } else {
51            $this->log .= date('H:m:s') . " :␣Authorization␣successful␣!";
52            return true;
53        }
54
55        return false;
56    }
57
58    function openShell( $shell_type = '' ) {
59
60        if( !($this->con > 0) ) $this->connect();
61        if ( $shell_type != '' ) $this->shell_type = $shell_type;
62        $this->shell = ssh2_shell( $this->con, $this->shell_type );
63        if( !$this->shell ) $this->log .= "␣Shell␣connection␣failed␣!";
64
65    }
66

```

```

67  function writeShell( $command = '' ) {
68      fwrite($this->shell, $command."\n");
69  }
70
71
72  function cmdExec( ) {
73
74      $argc = func_num_args();
75      $argv = func_get_args();
76      if( $argc < 1 ) return false;
77
78      if( !( $this->con > 0 ) ) {
79          if( !$this->authenticate() ) return false;
80      }
81
82      $cmd = '';
83      for( $i=0; $i<$argc-1 ; $i++)
84          $cmd .= $argv[ $i ]. " && ";
85      $cmd .= $argv[ $argc-1 ];
86
87
88      $this->log .= "<br>" . date( 'H:m:s' ) . " : Executing '$cmd'";
89
90      $stream = ssh2_exec( $this->con, $cmd );
91      $stderr_stream = ssh2_fetch_stream( $stream, SSH2_STREAM_STDERR );
92      $stdio_stream = ssh2_fetch_stream( $stream, SSH2_STREAM_STDIO );
93      stream_set_blocking( $stdio_stream, true );
94      stream_set_blocking( $stderr_stream, true );
95
96
97      $resp = '';
98      while ( $line=fgets( $stdio_stream ) ) {
99          $resp .= $line;
100     }
101
102     $this->lasterrors = '';
103     while ( $line=fgets( $stderr_stream ) )
104         $this->lasterrors .= $line;
105
106
107     $this->log .= "Out: '$resp'";
108     return substr( $resp, 0, -1 );
109 }
110
111
112
113 function getLog() {
114     return $this->log;
115 }
116
117 function getLastError() {
118     return $this->lasterrors;
119 }
120
121 function copy_file( $origin, $remotedest, $mode = 0644 ) {
122     if( !( $this->con > 0 ) ) {
123         if( !$this->login() ) return false;
124     }
125     if( ssh2_scp_send( $this->con, $origin, $remotedest, $mode ) ) {
126         $this->log .= "<br>" . date( 'H:m:s' ) . " : Uploaded '$remotedest'";
127         return true;
128     }
129     else $this->log .= "<br>" . date( 'H:m:s' ) . " : Error uploading '$remotedest'";
130
131     return false;
132 }
133
134
135 }
136
137 ?>

```

D.2.2 AJAX server to import submission files

```

1 <?php
2 /**
3  * *** PHP AJAX SERVER FOR PARSING SUBMIT FILE *****
4  */
5
6 //PHP Class to parse submit files
7 class confloader {
8
9     var $confs;
10
11     function confloader( ) {
12         $this->confs = array();
13     }
14
15     //Parses a submission file into a local structure
16     function parsefile( $file , $return_all = false ) {
17         $text = '';
18         $fx = fopen( $file , 'r' );
19         if( $fx ) {
20             while( !feof( $fx ) ) {
21                 $tline = trim( fgets( $fx ) );
22                 $text .= $tline . "\n";
23                 if( strlen( $tline ) > 0 ) {
24                     if( $tline[0] != '#' || $tline[0] == '#' && strlen( $tline ) > 1 && $tline[1] != '_' ) {
25                         $spair = split( '=', $tline );
26                         if( sizeof( $spair ) == 2 )
27                             $this->confs[trim( $spair[0] )] = trim( $spair[1] );
28                     }
29                 }
30             }
31             //Save object to session
32             $_SESSION['confloader'] = serialize( $this );
33             return $text;
34         } else return false;
35     }
36
37     function get_value( $key ) {
38         if( isset( $this->confs[ $key ] ) )
39             return $this->confs[ $key ];
40         else return false;
41     }
42
43     function get_all() {
44         return sizeof( $this->confs );
45         $out = '';
46         foreach( $this->confs as $key => $val )
47             $out .= $key . '=' . $val . "\n";
48
49         return $out;
50     }
51 }
52
53 //The Ajax server.
54 require_once ( 'HTML/AJAX/Server.php' );
55
56 class AutoServer extends HTML_AJAX_Server {
57     // this flag must be set for your init methods to be used
58     var $initMethods = true;
59
60     // init method for my confloader class
61     function initconfloader() {
62         //Tries to restore object from session
63         if( session_is_registered( 'confloader' ) )
64             $submitloader = unserialize( $_SESSION['confloader'] );
65         else {
66             //New object. Save to session
67             $submitloader = new confloader();
68             $_SESSION['confloader'] = serialize( $submitloader );

```

```
69         }
70         $this->registerClass($confloader);
71     }
72 }
73
74 session_start();
75
76 //generate javascript stubs for the object methods
77 $server = new AutoServer();
78 $server->handleRequest();
79
80 ?>
```