Grid in High Energy Physics

Grid in High Energy Physics

Jakob Nielsen
NorduGrid

Outline

- Introducing High Energy Physics
- Experiments
- Why High Energy Physics needs Grid
- Data-Challenges
- Automatic Production Systems
- Job-managers

High Energy Physics

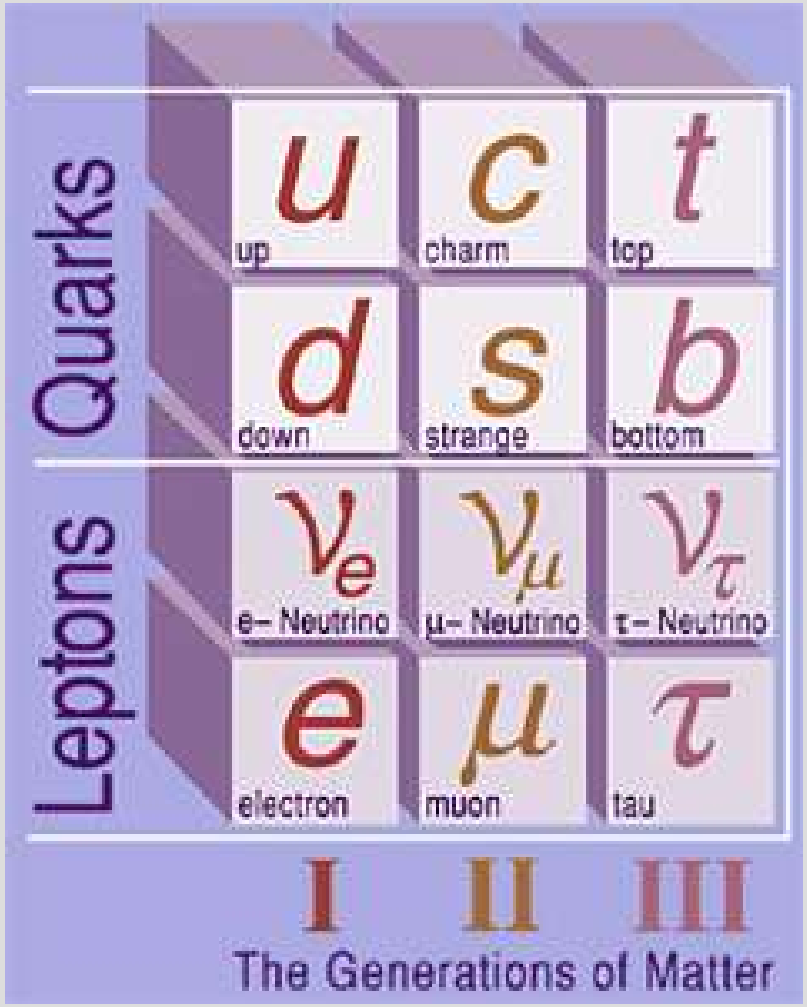
- High Energy Physics is the study of the smallest scales in nature (Paradoxically, maybe!).
- Consequence of the Heisenberg uncertainty relation

$$\Delta \; p \; \Delta \quad \sim$$

- Therefore to be able to study smaller and smaller scales, one needs more and more energy.

The Standard Model

- High Energy Physics is described by the so-called **Standard Model**.
- It consists basically of a set of matter particles interacting with force carriers after some mathematical equations.
- Tested with amazing precision in many, many experiments!
-
- Nevertheless, there are deep theoretical reasons why this is not the complete theory.

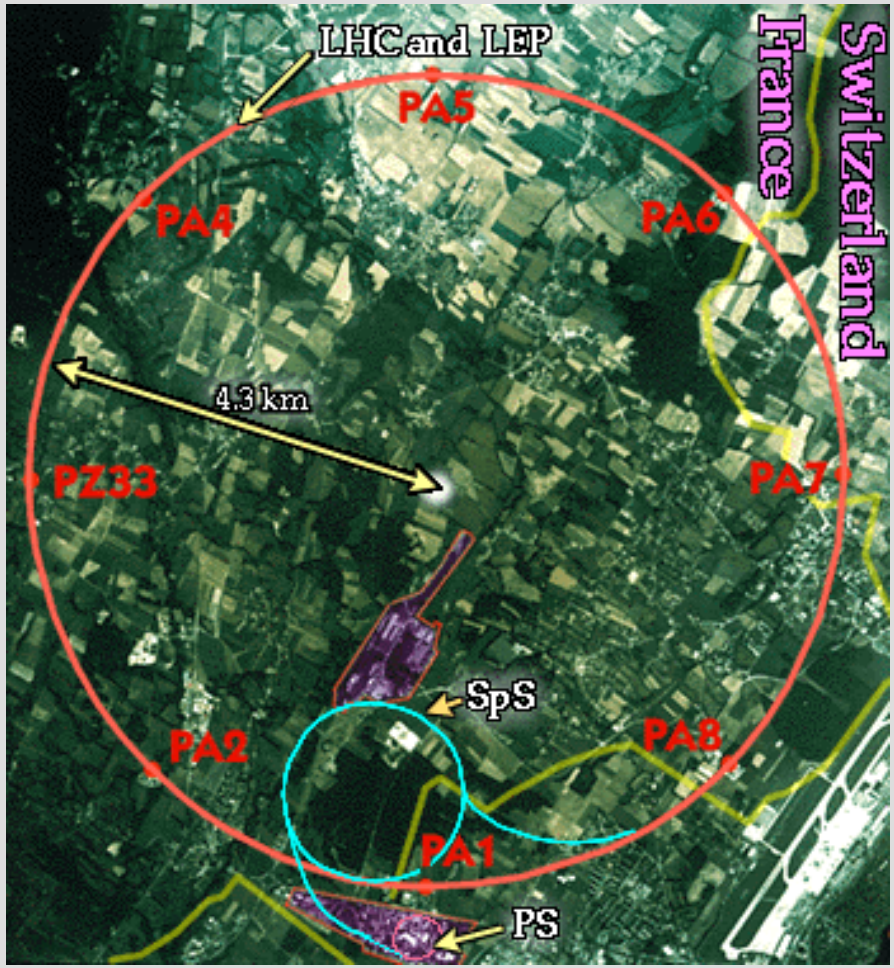


Experiments

- Experiments are performed at large-scale accelerators in several places in the world.
- Usually huge experiments with up to several thousand people involved.
- Examples: CERN, DESY, SLAC, RHIC ...
-
- Particles are accelerated to huge energies and collided.
- The collision is then studied by large detectors hoping to detect new physics/new particles.
- More energy -> smaller scales -> higher probability to discover something new.

CERN

- CERN is the European High Energy Physics Laboratory situated in Geneva, Schwitzerland.
- Huge accelerator – 28 km in circumference!
- At the moment, it is being upgraded to higher energies.
-
- **LHC** = Large Hadron Collider will start in 2007.
- With 4 different experiments studying the collisions.

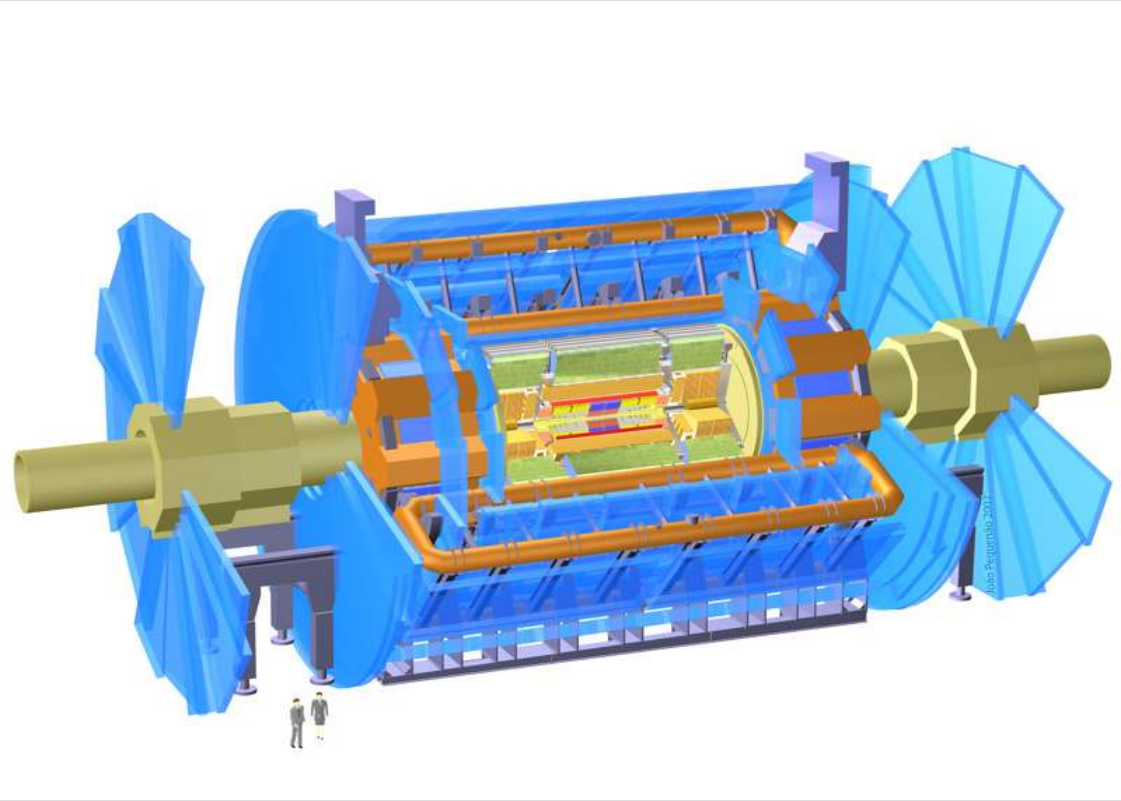


4 Experiments

- 4 different experiments placed at 4 different places in the ring.
  - **LHCb** – detector specially targeted at stduying b-physics.
  - **Alice** – detector to study collisions of nuclei and the generation of quark-gluon plasma.
  - **ATLAS** – general purpose detector looking for the elusive Higgs particle and supersymmetric particles.
  - **CMS** – another general purpose detector with the same goals as **ATLAS**.

ATLAS

- **ATLAS** (A Toriodal Lhc ApparatuS) is one of these experiments.
- Huge detector – 20x40 meters, 8000 tons!



ATLAS

- Major goals include:
  - Detection of the elusive Higgs particle.
  - Observation of the proposed supersymmetric particles.
- Collisions will happen once every 25 ns.
- Signals from collisions from the whole detector needs to be collected, processed and stored.
- Huge synchronization problem.
  - How do ensure that signals from different sub-detectors are from the same event?
- Complicated since speed of light is very small!

ATLAS Event-processing

- During runs,
  - Efficient online data-algorithms will run on partially collected events.
  - Throw away most events as uninteresting.
  - Collect the very few interesting looking events for further processing.
- Event-rate is reduced from 40 MHz to only about 100 Hz.
- But 2 MB pr. event gives
  - 200 MB data pr. second
  - Or 17 TB pr. day.
  - Or in total 6 PB pr. year.

ATLAS Data-management

- Adding simulated data, we end at about 10 TB data pr. year.
- This data should be
  - stored safely at storage servers at CERN and around the world.
  - registered so that it can be found at all times.
  - processed to produce simple analyzable data.
  - analyzed …
-
- Huge computational and logistic problem!
- For this, **ATLAS** needs Grid!

LCG

- The **LCG** (**LHC** Computing Grid) project is to deliver a working Grid solution to the **LHC**-experiments.
- Goal is to solve the enormous logistic and computational problems faced by physicists.
-
- Current middleware, **LCG-2**, is based on the middleware developed by the **EDG**-project.
- To be replaced by gLite during 2005.
- gLite is a lightweight middleware based on a re-engineered version of **Alien**.

Other players

- Other Grid-players participating in LHC grid-computing includes:
  - **US Grid** -- grid solution and infrastructure developed in the US.
  - **NorduGrid** – grid-middleware and infrastructure developed and deployed in the Nordic countries.
-
- Both contribute substantially to the **LHC** Data-challenges.
- Could be considered equal players with **LCG**.
- But politics!

Summary

- A new round of High Energy Physics experiments will start at the LHC in 2007.
- The data-taking will collect an unprecedented amount of data.
- The physicists at the LHC needs Grid for their analysis!
- Grid-solutions like LCG, US-Grid and NorduGrid are being developed, deployed and tested.
- Only the future will show ....

Data-Challenges

- To prepare for the LHC, all experiments have planned a series of **Data-Challenges**.
- The goal is to test the detector software, the Grid-infrastructure and the interplay.
- Each Data-Challenge bigger and more complex than the next.
-
- The Data-Challenges run as a series of production runs generating simulated data.
- Interesting for the physicists to look at.

LHC Data-Challenges

- The 4 experiments at the **LHC** has initiated a set of so-called Data-Challenges.
- The goals are to
  - Prepare for the data-taking and analysis at the scale of the **LHC**.
  - Introduce the use of Grid-middleware as fast as possible.
  - Validate the experiment's software.
-
- In the following, we will describe one of these, the **ATLAS** Data-Challenge 2.

ATLAS Data-Challenge 2

- **ATLAS** Data-Challenge 2 is a large-scale ATLAS-physics-simulation challenge.
- It has run in several stages since June 2004.
-
- Stage 1: Detector-Simulation (Jun-Oct)
- Stage 2: Digitization (Oct-Dec)
- Stage 3: Reconstruction -- just starting ...
-
- In stage 1, 1M events had to be simulated. With about 10 minutes CPU time pr. event, roughly 10M minutes CPU time in total.

Data-Challenge 2

- 3 Grids has participated in the production
  - LCG 4000 CPU's available
  - US-Grid 800 CPU's available
  - NorduGrid 700 CPU's available
- The simulations are divided into different datasets.
- Assigned to the different Grids according to their capacity.
- Reassignment if one Grid turned out to have too many jobs.
- Continuous production for several months!

Data-Challenge jobs

- A very schematic DC jobflow looks like this:
  - The job is submitted to some cluster with the ATLAS software preinstalled.
  - Inputfiles are either located on the cluster already or downloaded before the job is run.
  - The job is run -- producing some output.
  - The output files are stored on some random Storage Element somewhere.
  - The physical locations of the output files are registered in some database.
- Simple -- but there are than more 200.000 of these!

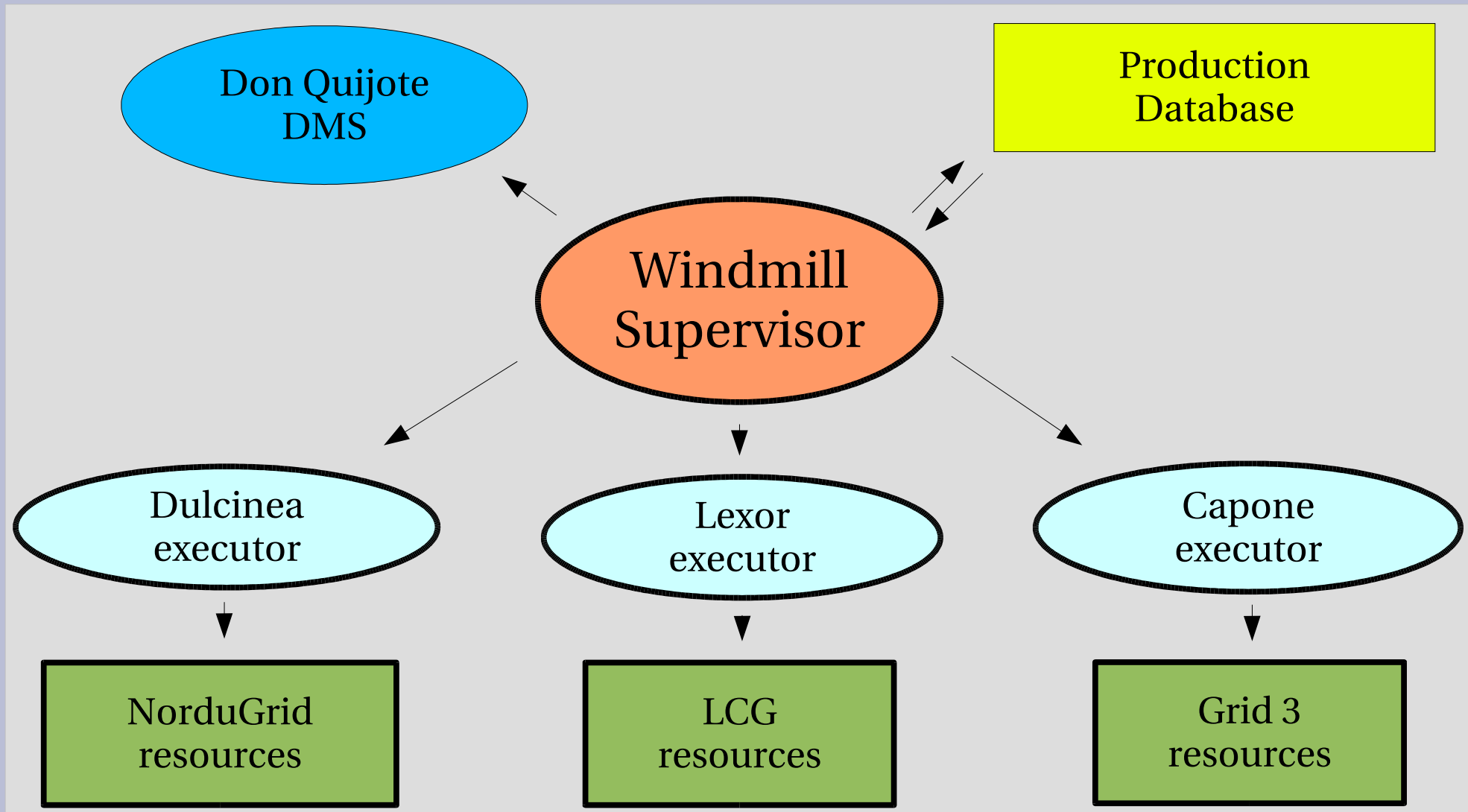
200000+ jobs

- Running 200.000 jobs in shortest possible time is quite a challenge!
- It takes time ...
  - Submitting jobs.
  - Tracking the running jobs and their output.
  - Cleaning after failed jobs.
  - Rerunning failed jobs.
  - Making sure all jobs have been run.
  - Basically babysitting the jobs.
  - All such babysitting takes time!
- And much, much more than expected ...

Production System

- Therefore a combined inter-grid production system has been written for Data-Challenge-2.
- The production system has interfaces to the 3 grid-flavors through grid-specific “executors”.
- All jobs defined in a database in a grid-neutral xml-language.
- Jobs are picked up by a grid-independent supervisor and presented to the “executors”.
- The executor translate the grid-neutral xml-language to the job-description language used in the particular grid-middleware (xRSL for NorduGrid).

Production System



Executor/Supervisor

- Executor tasks consists of
  - Submitting jobs.
  - Answer status queries for submitted jobs.
  - Finalize finished jobs.
- Supervisor tasks consists of
  - Pulling available jobs from the database and presenting them to an executor.
  - Ask status queries for submitted jobs.
  - Make sure finished jobs gets registered as finished.
  - Make sure failed jobs gets resubmitted.

Implementation

- Executor/Supervisor exchange xml-messages using the Jabber-protocol.
- A Jabber-server is setup and the executors and supervisors connect to it.
- Executor/Supervisor prototypes were written in python for easy development.
- Executor-implementations written in python or as a C++-python module (NorduGrid).

Messages

- The supervisor can send the executor 4 different types of messages:
  - How many jobs with given requirements do you want?
  - Submit the following jobs.
  - Did the jobs get submitted?
  - What is the status of all submitted jobs?
- In each case, the executor has to
  - Process the xml-message
  - Get the status of the Grid and jobs on the grid.
  - Construct an xml-answer
  - Send it back
- Format of messages given from a xml-schema.

Dulcinea

- Dulcinea is the NorduGrid executor.
- It is written as a C++-python module on top of the NorduGrid UI API.
- Each message translates into a specific UI-call to the NorduGrid information system:
  - Submit jobs --> ngsub
  - Get status of jobs --> ngstat etc.
- Can be viewed as a fairly simple wrapper around the NorduGrid UI.
- Most work is done by parsing/constructing xml-messages.

Production System

- It actually works!
- Can in principle run unattended for several days with several thousand jobs.
- Cuts the needed manpower for production drastically.
- 2 persons able to do NorduGrid production (70.000 jobs).
- However, the executors/supervisors needs baby-sitting themselves!

Limitations

- Jabber-framework not very suitable for this kind of job!
- Current implementation does not scale to more than a few thousand jobs pr. executor.
  - Huge xml-messages exchanged.
- Huge startup times (read in submitted jobs from database).
- No proxy-forwarding
  - Can run jobs using other people's credentials.
  - Screws up accounting.
- Memory leaks necessitates frequent restarts.

Extensions

- Several ideas:
- Replace Jabber-framework with httpg-daemons exchanging xml-messages over SOAP.
- Would immediately allow proxy-forwarding.
- More advanced job-manager with better schema.
- Flat-file support with job-definitions.

Summary

- **LHC** is running a series of Data-Challenges.
- **ATLAS** Data-Challenge 2 running this year.
- Used an automatic production system on 3 grid-flavors for the production.
- Production system built with
  - grid-specific executors
  - grid-independent supervisors
  - grid-independent job-description language
- It works – and saves (lots of) man-power!
- But also has limitations ...

Job-managers

- A job-manager is an automatic process that when started
  - Reads user job-descriptions.
  - Submit those jobs to the Grid.
  - Track the jobs as they are run.
  - Mark finished jobs.
  - Resubmit failed jobs.
  - Register output-files.
- All without intervention from the user.

Why a job-manager?

- Baby-sitting jobs is much work – usually grossly underestimated.
  - A few hundred jobs sufficient for having advantage from a job-manager.
- Takes care of the boring aspects of the production.
  - Lets the scientist concentrate on the data-analysis instead.
- Quicker! Doesn't sleep or take days off ....
-
- Everybody needs a job-manager!

Components

- A job-manager should contain a plugin-structure with replaceable components:
  - A Job-definition language
  - Job-submission component
  - Job-status component
  - Job-failure component
  - Cluster-efficiency component
  - User-defined components ...
- Pluggable components to avoid monolithic structure.
- Each component is run periodically.

Job-definition-language

- A job-definition-language
  - Should be able to describe a wide array of different tasks to be performed.
  - Independent of the actual simulations performed.
- Standard use-case:
- A job is repeated N times
  - with a varying set of parameters
  - varying set of input-files and output-files
- The job-definition-language should at least support this.
- And it should be simple to describe for a user.

Submitter-component

- The submitter-component should contain:
- A task-reader
  - Reading tasks from a flat file / database
  - Flat file usually for smaller amount of jobs
  - Database for huge amount of jobs
- A job-language parser
  - A parser that translates abstract task-descriptions to the job-description-language of the Grid.
  - Could be fairly general and not experiment-specific
- A job-submitter
  - For submitting jobs.
  - Should spread out job-submission.

Status/Failure-component

- The status component should
  - periodically query the Grid for information about submitted jobs.
  - act upon the information gathered.
  - register finished and failed jobs.
-
- The job-failure component should
  - Investigate job-failures from status-messages
    - What kind of failure was it?
    - Do certain jobs crash very quickly?
    - Has the same job crashed several times?
  - Deduce information from logfile (hard!)
  - Mark failed jobs to be resubmitable.

User-defined components

- It should be possible for users to write their own components that can be plugged in.
  - Maybe a more advanced job-resubmission component could be useful for some users.
  - Or a save-output component that copies the output of the run to a local machine.
  - Or even a better broker in the job-submission component
- Such components should be easily plugged in. No need for recompilation...

Cluster Efficiency component

- A cluster efficiency component
  - Should measure the efficiency of clusters from the number of finished jobs
  - Could rule out job-submissiont to clusters at runtime
  - Especially black-hole-clusters should be detectable (those with many failed jobs)
- The results should be reported back to the job-submission component.

Comparison with DC2 framework

- Missing flat file readin
  - More complicated to use for a small user
- Somewhat fragile framework with two parts
  - Jabber framework does not scale
- No plugin-structure
  - User cannot easily change the behaviour of the executor
- Needs more general language
  - Job-definition language is very HEP-specific.

ARCLib

- ARCLib is a reimplementation of (parts of) the ARC Userinterface written in C++.
- Equipped with a well-defined API.
- Python-wrappers.
-
- Some functionality still missing.
- More testing needed.
-
- Examples:
- Clusters = GetResources(“atlasgiis.nbi.dk”)
- ClusterInfo = GetJobInfo(Clusters)

Job-manager in ARCLib

- A job-manager already exists inside ARC that can handle resubmission of failed jobs.
  - Works but still needs testing in a production environment.
- Also a new job-manager with the above requirements is being developed at the moment.
- Uses ARCLib's python wrappers.
- Has the potential to completely replace the executor/supervisor framework.

Job-managers in HEP

- As we approach 2007, reliable job-managers will be an essential tool for physicists.
- The quantity of data will be so large that it will be impossible to work without it.
- Maybe the single-most important thing!
-
- Such a job-manager as above would be a serious help to many scientists running on Grid.