WMS overview and Proposal for Job Status

Author: V.Garonne, I.Stokes-Rees, A. Tsaregorodtsev.
Centre de physiques des Particules de Marseille
Date: 15/12/2003

Abstract

In this paper, we describe briefly the workload Management System (WMS) of DIRAC (**D**istributed **I**nfrastructure with **R**emote **A**gent **C**ontrol). Dirac will be the infrastructure used for this year's DC'04 production. This document also discusses the different states of a Job during its lifecycle and the job parameters required by the system.

WMS Architecture Overview

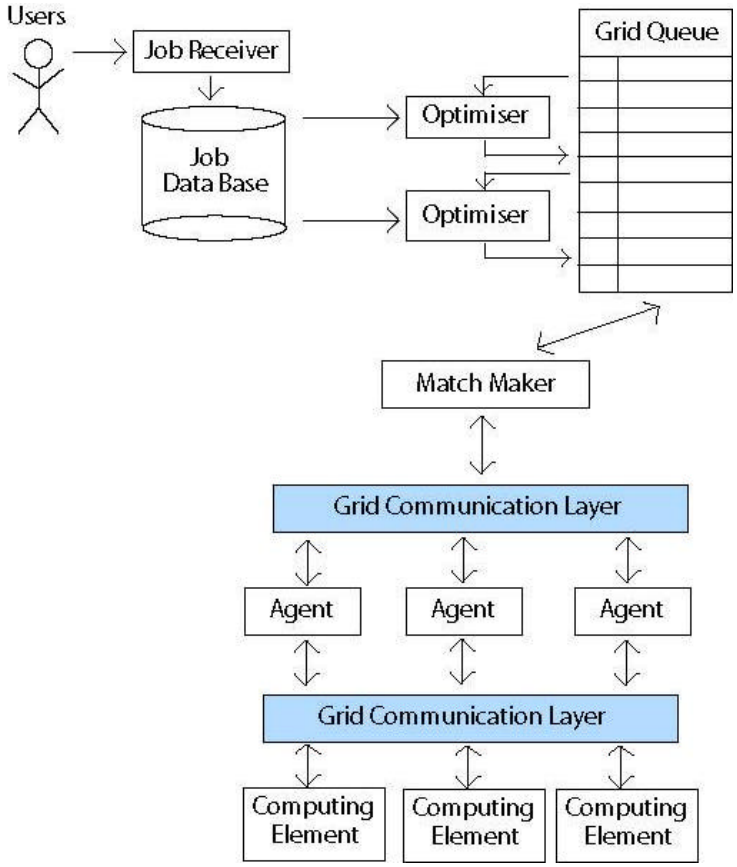


Figure 2. WMS Architecture

The WMS consists of the following key components:

User where jobs originally come from and where results are returned. The User submits jobs in JDL. JDL is used to maintain interoperability with others grid projects such EDG,

	Alien and LCG.
Job receiver	accepts job submissions from the User and registers them into the Job DB.
Job DB	stores information pertaining to each job.
Optimizer	sorts jobs into Grid queues with a priority computed by optimizer.
Match Maker	decides what jobs to give to a particular computing resource
Agent	Monitors a Computing Element (CE) and fetches jobs from the Match Maker
Computing Element	provides computational resources to the user (see also Computing Element chapter)

Computing Element

The Computing Element (CE) is an abstraction which provides an interface to local computing resources. Although single node and "grid" Computing Elements are possible, in general we model each Computing Element as a single Gatekeeper which manages/accesses a cluster of computing Worker Nodes. We assume such a system consists of its own local scheduler and queues (as illustrated in Figure 2). At present, DIRAC provides interfaces to LSF, PBS, BQS and Globus.

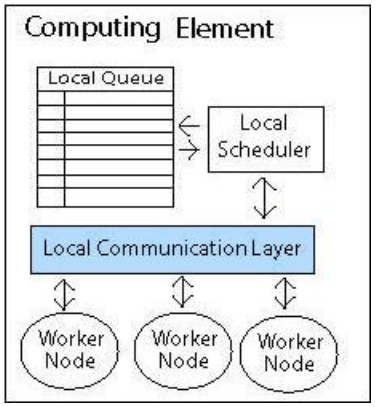


Figure 1: Computing Element

A CE is classified as "Available" if it is immediately (un-utilised CPUs) or imminently (CPUs with no queued jobs) able to execute a job. This criterion of availability depends on the nature of the CE, so we define different criteria of availability:

- For Batch System type

Total Queuing Jobs=0 or Total Queuing Jobs/Total CPUs < 0.3

- For a local PC

Total Running Jobs == 0

Job status

During its lifetime a job may go through a number of states. These states and there meaning are listed below.

Ready	The user has submitted the Job to the User Interface
Waiting	The Job is queued in Grid queues
Queued	The Job is queued in a Cluster on a local queue (for example PBS) of a Computing Element
Running	The Job is running on a worker node
Done	The execution of the Job has completed
Outputready	The Output of the Job is ready
Rescheduled	To be defined
Failed	To be defined

Job Information & Parameters: Example Values

- **Job Information**

JobID	209
Owner	Nobody
Site	ccali.in2p3.fr/bqs-A
Type	Test
SubmissionDate	2003-12-19
SubmissionTime	09:41:07
Status	outputready

- **Job Parameters**

JobID	Name	Value
209	LocalBatchId	I3191743
209	Worker Node	ccwali64
209	Memory	027904kB
209	Model	PentiumIII(Coppermine)

209	CPU	996.894
209	CPU comsumed	36.97
209	Execution Time	37.3256549835

Job Life in the WMS

When the job is done it is not necessary to keep all its information in the system. We plan to use an agent called the "Job Monitor" in order to clear out information from old jobs. This agent will transfer information concerning a job from the Job DB to a Provenance DB when it discovers jobs to be cleared. The criteria for clearing a job is still to be fixed.

The Provenance DB will also be used as an accounting service. For the moment we propose to use the Bookkeeping DB as Provenance DB.

Implementation details

The WMS is implemented as a set of classes written in Python. This provides a clean object-oriented design together with a rapid development environment. It uses the XML-RPC protocol and Jabber to communicate with different services. The Job DB uses MySQL. DIRAC jobs are described with the Condor ClassAds Job Description Language (JDL).

Job splitting and Merging proposal

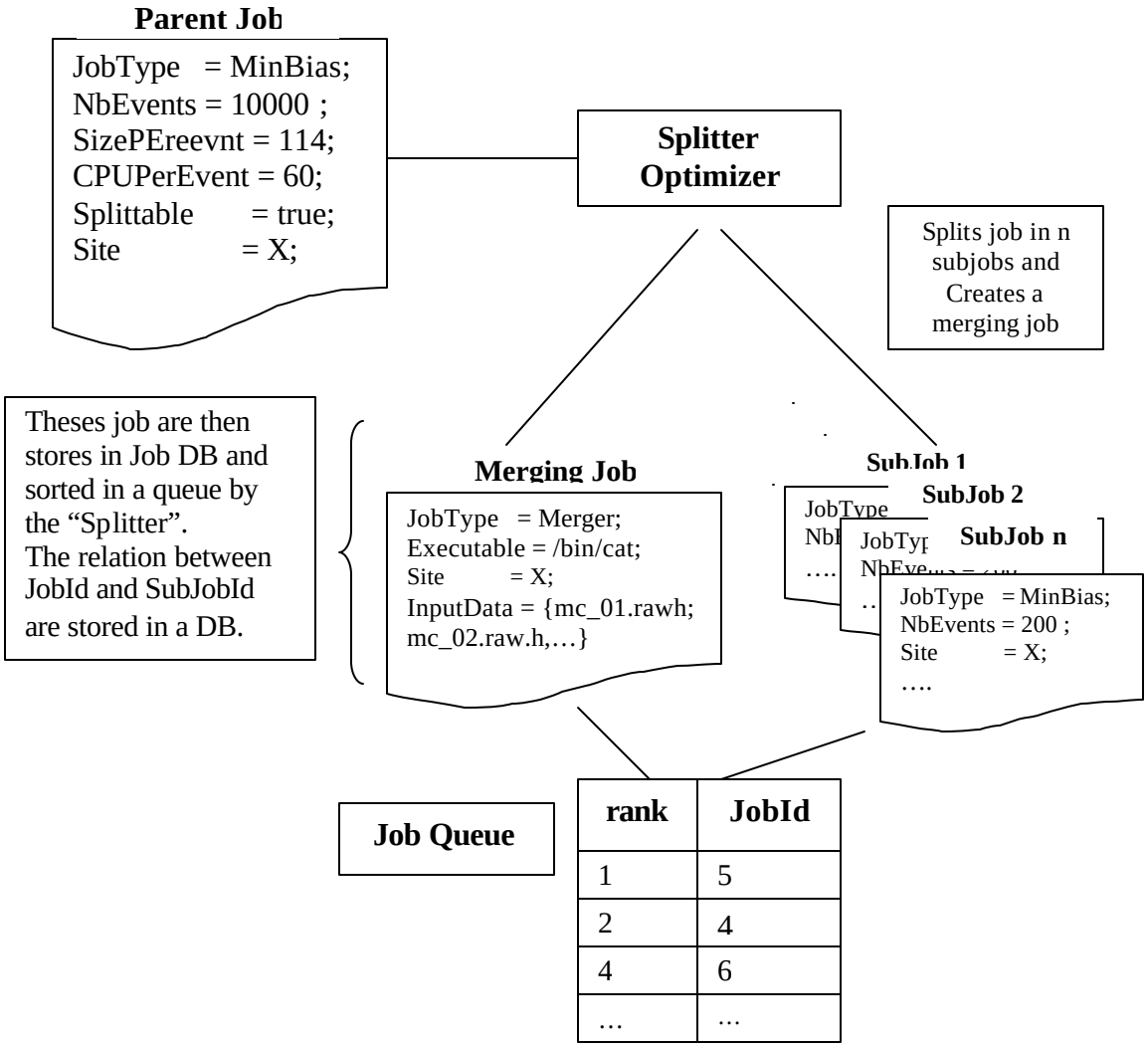
With the WMS architecture, we propose different approaches to do the job splitting and merging. We firstly considered jobs with no input data.

Splitting and Merging approach for jobs with no input data

The job is submitted to DIRAC and then divided into a set of sub-jobs, all of which are targeted for the same destination CE. A "Merge" job is also submitted to the same CE, which allows all sub-job outputs to be locally available, thus facilitating the merge, whose only purpose is to combine subjob outputs. The JDL of the parent job must contain these options in order for splitting to be possible:

- JobType
- NbEvents
- SizePerEvent
- CPUPerEvent
- Splittable (true or false)
- CEName

The Job Receiver stores the parent job in the DB and returns a JobId. Then it notifies a "Splitter Service" which is a specialized Optimiser. This splits the job into the **n subjobs** and **one merging job**. The jobs are queued and will eventually match to the specified CE when its Agent requests a job.
The "Splitter" will also keep track of the relation between the parent's JobId and the SubJob's JobId in a DB. This information will be available via the Job Monitoring Service. The merging job will be executed only if all inputs are ready.



Implication:

There are various implications of this scheme:

- A new optimiser is required which is able to split jobs and manage the sub job dependencies.

- A way to report the status of the parent job from the Job Monitoring Service. For example: 23 % in done status, 2% in running status. The access to this information could be done by a special process which monitors subjobs and computes some statistics.

- A "Dependency Checker Agent" will also check if data is available for a job. This builds on the idea of "virtual data" where a job can be submitted *before* the data is available. Specifically, this will be required to hold the "merge" job until all the outputs are ready. This agent will be interfaced with the file catalogue.

Splitting and Merging approach for jobs with input data

This is the same as the scenario without input data, except that fragmentation of the input data set must be considered. Here the "Dependency Checker" Agent will be make sure that a job is split into groups such that all the files in a sub job are accessible from a single CE. To facilitate this the Dependency Checker can act as or in conjunction with a Replica Manager and initiate data replication to insure that groups of input files for sub jobs are accessible from a single CE which has available computing resources.

Terminology

Glossary

class-ad	Classified advertisement
CE	Computing Element
job-ad	Class-ad describing a job
JDL	Job Description Language
LRMS	Local Resource Management System
PID	Process Identifier
SE	Storage Element
UI	User Interface
VO	Virtual Organisation
WMS	Workload Management System