The STAR Scheduler Project

A job submission tool
for distributed resource environments

Gabriele Carcassi, Jerome Lauret, Claude Pruneau

STAR Collaboration

CHEP03, La Jolla, California, USA

Abstract

The STAR scheduler project was developed to allow users to submit their analysis jobs taking as input a logical collection of files or a specific list of files. The files were initially foreseen as residing on centralized storage or on the various distributed disks of our analysis farm, a pool of 154 Linux Pentium-based processing nodes with a total power equivalent to 24,000 SI95 and 30 TB of storage. As input, the UI uses an XML approach with simple rules. The scheduler analyzes the request taking advantage of the information returned by a meta data catalog and splits the job request into different processes which are then dispatched on our CPU resource pool.

Designed to be modular with a class-based component layout and written in Java, this tool is being further extended to allow submission of jobs on the GRID in a completely transparent way to the users.

This poster presents the software architecture, the status of this tool and an overview of its future development and its migration toward our main resource broker for job submission.

Keywords: Job scheduling, resource broker,

request.xml

```
<job maxFilesPerProcess="100"
  <command> root4star -q -b my
  <stdout URL="file://star/user/ca
  <input URL="catalog:star.bnl.g
  <output fromScratch="*.root" to
</job>
```

JOB DESCRIPTION

The user describes his **job request** by writing an **XML file**.
The request is a **description of what the user wants** to achieve, and not what a particular infrastructure has to do. This assumes a **user model** that we have built **from** a series of **use cases**.
Within this model, a **program** receives a **list of input files** and **produces** one or more **output files**.
Input files can be specified by filename, wildcards or catalog queries. Output files will be produced in a scratch directory on the target machine and later retrieved by the system.

User level

JOB INITIALIZER

Parses the XML and **creates** an **internal representation** of the job **request**. It also **checks** for **consistency**, by checking whether the specified **input files exist**, or if the user selected two options that are mutually exclusive.

STAR Scheduler Resource Broker

FILE CATALOG

catalog:star.bnl.gov?collision=AuAu200

The integration with the file catalog allows the user to **specify** the **input as** a query on the **physics attributes** (metadata). It will also allow the scheduler to **choose** the files that are **more accessible**, or that are on less-used machines.
The **query is opaque** to the scheduler and is passed to the file catalog through an **interface** (abstract class), thus **allowing** the use **different catalog implementations**. We use an implementation based on the STAR file catalog.

POLICY

The policy is the **core of resource brokering**: it decides how to use resources to satisfy requests. It uses GRID middleware to gather the information to make a good decision.
It is an **abstract** class and the **implementation is chosen at runtime**, allowing us to experiment with different schemes.
Our two current main policies are:
PassivePolicy - solves all the queries and forms one **list** with all the **files requested**. This list **is divided** according to user requests and to where the files are located. **Each sub-list** is assigned to a **different job**.
It is called passive because it does not trigger data replication.
SiteForwardPolicy - decides randomly on **which site** the request should be dispatched, and prepares a local scheduler execution to be dispatched.
It is part of our GRID implementation.

MONITORING

services

In STAR we are also working on setting up a **system to monitor** the **load and efficiency** of our clusters. We are investigating the use of **GANGLIA** and its integration with globus **MDS**.
We plan to use this information within the policy of our scheduler to make **better decisions** at a GRID level (aggregate site information) and at a site level (machine by machine information).

JOB SPLITTING

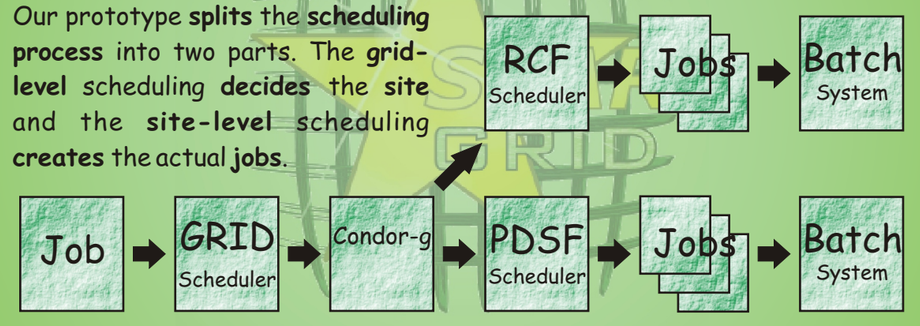
one request for 100s of jobs

Given a **single user request**, the scheduler divides the input file list into **multiple** lists, one for every **job submitted**. In the request the user can set the limit (min/max) of files assigned to a single job.
To allow job splitting, the user has to follow **two simple rules**:

- **take** the list of **input files from the scheduler**, to allow the scheduler to decide on which input files to run on
- give **different names to the output files**, to avoid clash between two different jobs.

GRID Implementation

Our prototype **splits** the **scheduling process** into two parts. The **grid-level** scheduling **decides** the **site** and the **site-level** scheduling **creates** the actual **jobs**.



DISPATCHER

The dispatcher is responsible for **creating scripts**, **filelists**, **class-ads** and everything else the batch system requires and then **submitting the job**. It is defined through an **interface** (abstract class), allowing **different implementations**. We currently provide two implementations:
LSFDispatcher - for each job creates a **script** and a file containing the input **file list**, assigns a **scratch directory** on the target machine for the output files and runs the appropriate **bsub command**.
CondorGDispatcher - for each job creates a **classAd** and **submits** the job to the **gatekeeper** assigned by the policy. This is part of our GRID implementation

PPDG

STAR GRID

BROOKHAVEN NATIONAL LABORATORY

LSF/CONDOR-G

Once the scheduler dispatches the jobs, the **batch system** will monitor the cluster and **start the scripts**. The final part of the script is usually responsible for retrieving the job output files in case the batch system does not support this functionality.
We currently **support LSF** and **Condor-G**, since these are the systems we have currently deployed. The **architecture allows** to integrate **other systems** as well.
Another big advantage of the scheduler is that the **knowledge** of **how to use** the underlying system correctly can be **embedded** directly in the **scheduler**. The administrator can incorporate this knowledge into the policy and into the dispatcher, **instead of** having to **rely** on an **informed and correct usage**. For **example**, we have integrated the use of **LSF resources** to **avoid** multiple jobs **accessing the same NFS server** at the same time thus decreasing performance.

Batch system

Distributed Disk Model

Bringing the job to the data

One of the main objective of the scheduler was to enable the distributed disk model. The data (MuDST) is spread over the local disks of the different nodes of the cluster. The Scheduler dispatches the job on the machine where the data is stored.

This model will enable us to use those computer resources that do not possess a network file-system.

At present, the population is static: it is not triggered by user requests, but decided and fixed by the administrator.