STAR Scheduler

Gabriele Carcassi
STAR Collaboration

What is the STAR scheduler?

- Resource Broker
  - receives job requests from the user and decides how to assign them to the resources available
- Wrapper on evolving technologies
  - by and by that GRID middleware fit for STAR needs is available is integrated in the scheduler flexible architecture

Scheduler benefits

- Enables the Distributed Disk framework
  - Data files are distributed on the local disk of each node of the farm
  - The job requiring a given files is dispatched where the file can be found
- Interfacing with STAR file catalog
  - User specify job input through a metadata/catalog query (ex. Gold-Gold at 200 GeV, Fullfield, minbias, ...)
  - File catalog implementation is modular

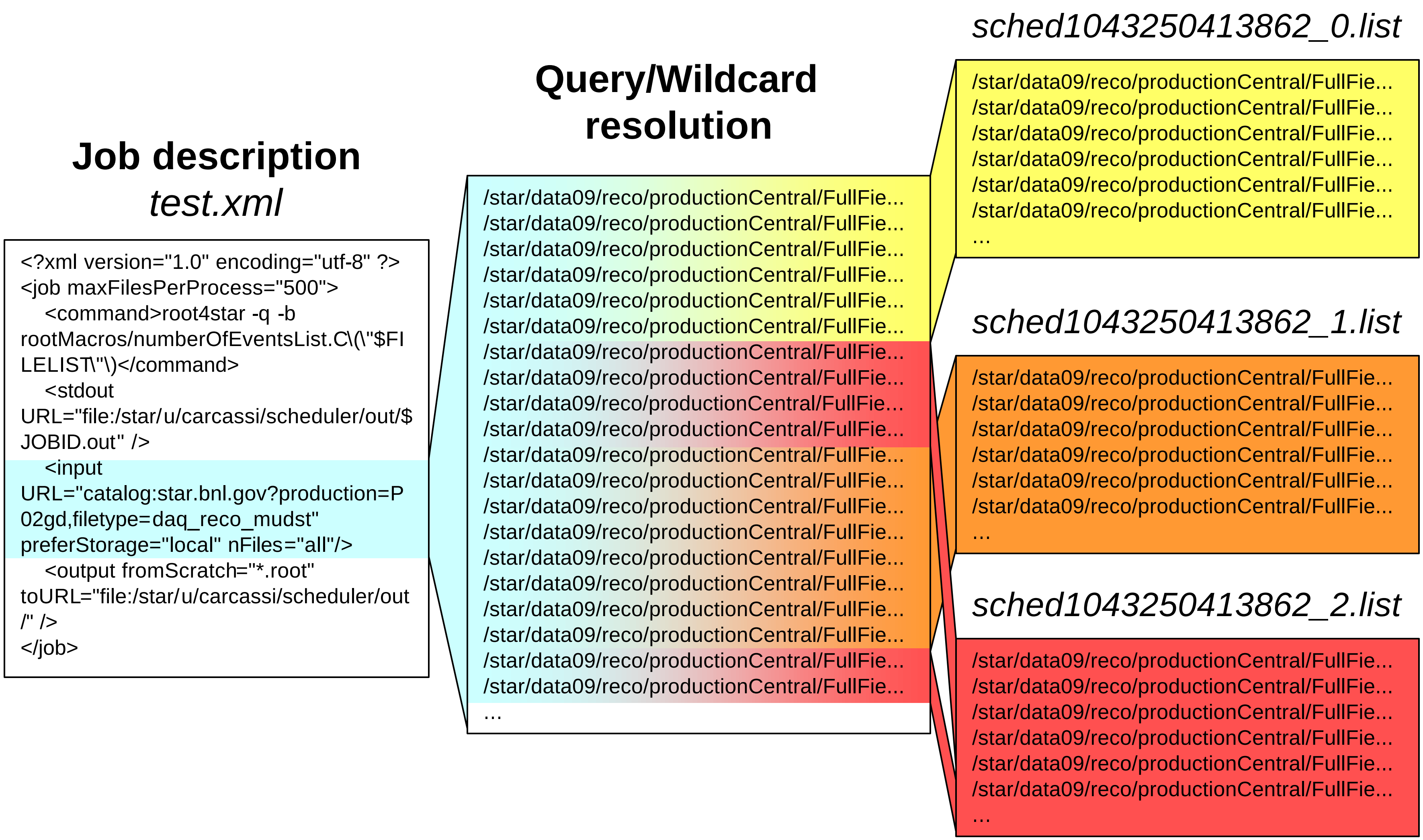
Scheduler benefits

- User interface: description and specification
  - Well defined user interface and job model
  - Abstract description allows us to embed in the scheduler the logic on how to use resources
  - Allows us to experiment and migrate to other tools with minimal impact for the user (for job submission)
  - Makes it clearer for other groups collaborating with us to understand our needs
- Extensible architecture

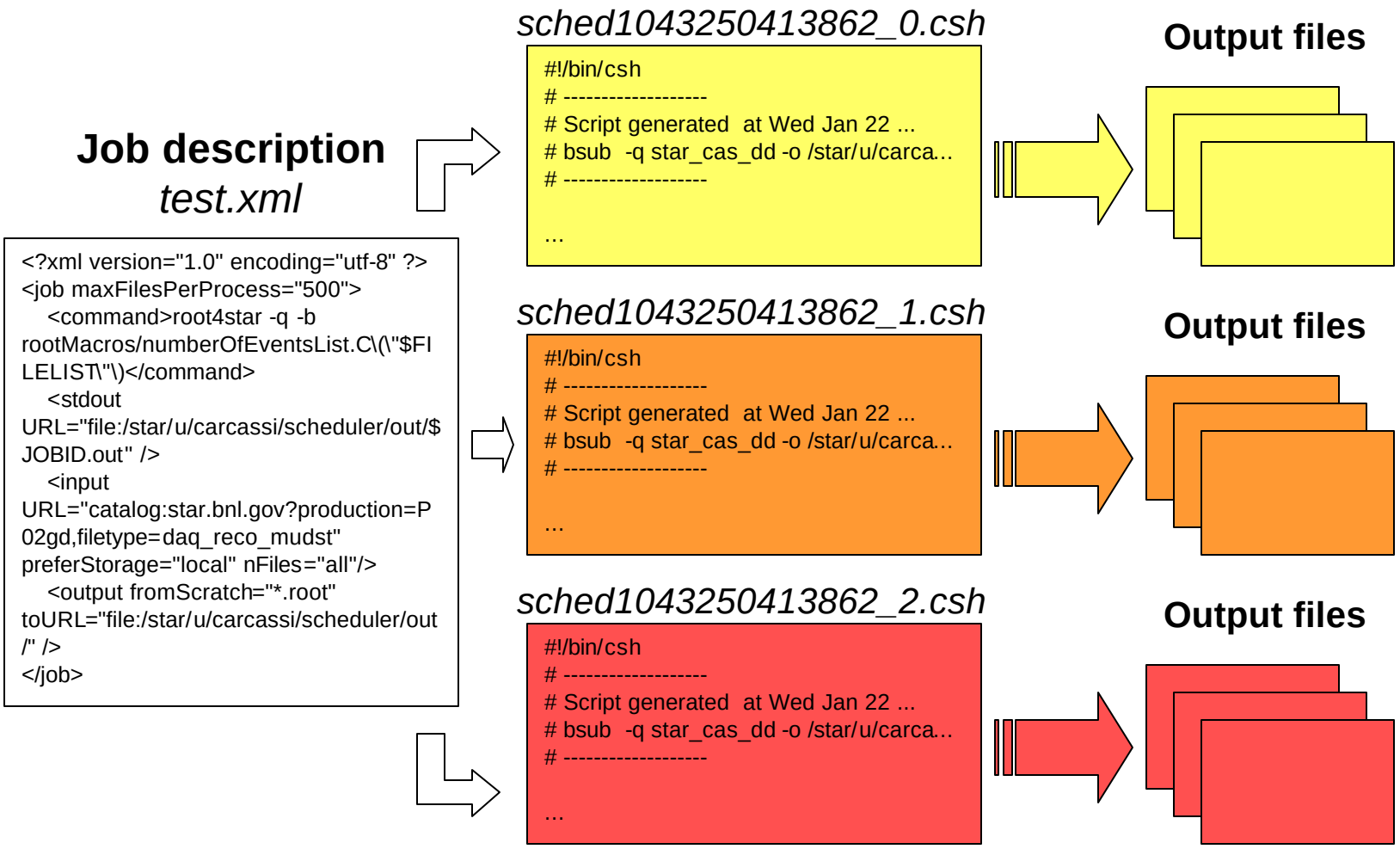
Technologies used

- Scheduler is written in Java
- Job description language is an XML file
- Current implementation uses
  - LSF for job submission
  - STAR catalog as the file catalog
- Experimenting with Condor-g for GRID submission

How does it work?



How does it work?



Distributed disk

- Motives
  - Scalability: NFS requires more work to scale
  - Performance: reading/writing on local disk is faster
  - Availability: every computer has local disk, not every computer has distributed disk
- Current model
  - Files are distributed by hand (Data carousel) according to user needs
  - File catalog is updated during distribution
  - Scheduler queries the file catalog and divides the job according to the distribution
- Future model
  - Dynamic distribution

File catalog integration

- Enables distributed disk
  - If not present, users would have to know where the files are distributed on which machines
- Allows users to specify their input according to the metadata
- On small number of files requests, the scheduler can choose which files are more available

File catalog integration

- Implemented through an interface (pure abstract class
  - The query itself is an opaque string passed directly to the file catalog
  - Other tags tell the scheduler how to extract the desired group
    - single copy or all copies of the same files
    - prefer files on NFS or local disk
    - number of files requires

User Interface

- Job description
  - an XML and it’s tag used to describe to the scheduler which command is to be dispatched and on which input files
- Job specification
  - a set of simple rules that define how the user job is supposed to behave

The Job description

- XML file with the description of our request

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="500">
    <command>root4star -q -b
rootMacros/numberOfEventsList.C\(\"$FILELIST\"\)</command>
    <stdout
URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
    <input URL="catalog:star.bnl.gov?
collision=dAu200,trgsetupname=minbias,filetype=MC_reco_MuDst"
preferStorage="local" nFiles="all"/>
    <output fromScratch="*.root"
toURL="file:/star/u/carcassi/scheduler/out/" />
</job>
```

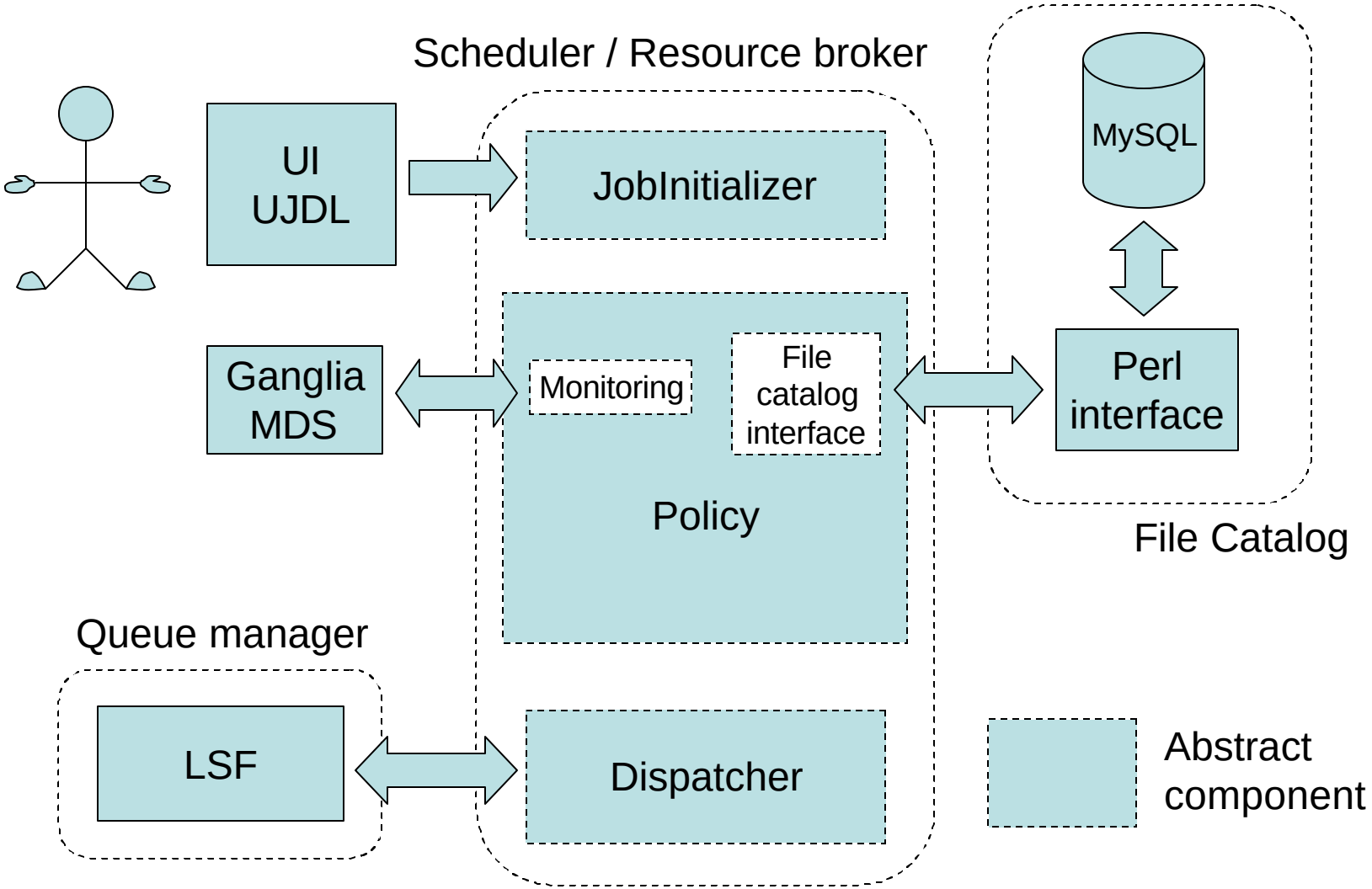
Job specification

- The scheduler prepares some environment variables to communicate the job its decision about job splitting
  - $FILELIST, $INPUTFILECOUNT and $INPUTFILExx contain information about the input files assigned to the job
  - $SCRATCH is a local directory available to the job to put it’s output for later retrieval

Job specification

- The other main requirement is that the output of the different processes won’t clash one another
  - One can use $JOBID to create filenames that are unique for each process

STAR Scheduling architecture



Job Initializer

- Parses the xml job request
- Checks the request to see if it is valid
  - Checks for elements outside specification (typically errors)
  - Checks for consistency (existence of input files on disk, ...)
  - Checks for requirements (require the output file, ...)
- Creates the Java objects representing the request (JobRequest)

Job Initializer

- Current implementation
  - Strict parser: any keyword outside the specification stops the process
  - Checks for the existence of the stdin file and the stdout directory
  - Forces the stdout to prevent side effects (such as LSF would accidentally send the output by mail)

Policy

- The core of resource brokering:
  - From one request, creates a series of processes to fulfill that request
  - Processes are created according to farm administrator’s decisions
  - The policy may query the file catalog, the queues or other middleware to make an optimal decision (ex. MDS, Ganglia, ...)

Policy

- We anticipate a lot of the work in finding an optimal policy
- Policy is easily changeable, to allow the administrator to change the behavior of the system

Policy

- Current policy
  - Resolves the queries and the wildcards to form a single file list
  - Divide the list into several sub-lists, according to where the input files are located and the maximum number of files set per process
  - Creates one process for every file list.

Dispatcher

- From the abstract process description, creates everything that is needed to dispatch the jobs
  - Talks to the underlying queue system
  - Takes care of creating the script that will be executed: csh based (widely supported)
  - Creates environment variables and the file list

Dispatcher

- Current implementation:
  - creates file list and script in the directory where the job was submitted from
  - creates environment variables containing the job id, the list of files and all the files in the list, assigns a scratch directory.
  - creates a command line for LSF
  - submits job to LSF

Conclusion

- The tool is available and working
  - In production since September 2002 and slowly acquiring acceptance (difficult to get people to try, but once they try it they like it)
- Allows the use of local disks
- Architecture is open to allow changes
  - Different policies
  - Catalog implementation (MAGDA, RLS, GDMP, ... ?)
  - Dispatcher implementation (Condor, Condor-g – Globus, ...)
- We are preparing an implementation that uses Condor-g and allows us to dispatch jobs to the GRID