HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

GridJM — A way for client job management in ARC

`http://www.tcs.hut.fi/~aehyvari/gridjm/`

Antti E. J. Hyvärinen

antti.hyvarinen@tkk.fi

Helsinki University of Technology

Laboratory for Theoretical Computer Science

Finland

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

Overview

- Grids offer high-throughput computing
  - a large pool of resources
  - an efficient method for discovering resources
- In arc, the discovering poses certain challenges to the client
  - maintain list of resources
  - select targets (brokering)
  - optimize the submission rate
  - minimize overhead
- This talk will give ideas on how the challenges can be answered
- Introduces GridJM (Grid Job Manager) for ARC
- Based on previous work “A Job Manager for the NorduGrid ARC” by H. T. Jensen and J. R. Leth

Submiting jobs in ARC

- Arclib has a 5-stage approach to submitting jobs
- The first two receive information from the grid information system (infosys)
  - `GetClusterResources()` returns a list of URLs pointing to clusters
  - `GetQueueInfo()` queries the states of the queues in the clusters
- The last three are related to matching with job description (xrsl), brokering and final submission
  - `ConstructTargets()`
  - `PerformStandardBrokering()` (or similar)
  - `Submit()` (of the submit-object)

Goals for GridJM

- Job brokering and monitoring is done by the user (not by a centralized authority)

By collecting history and infosys information, GridJM addresses the following:

- Fault tolerance
- Fault avoidance
- Minimizing time between sending the job and receiving the results
- Visualization of resource usage
- Automatic collecting of results

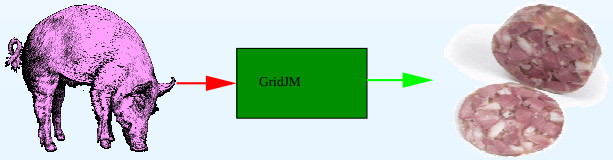
HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

Goals for GridJM

- Job brokering and monitoring is done by the user (not by a centralized authority)

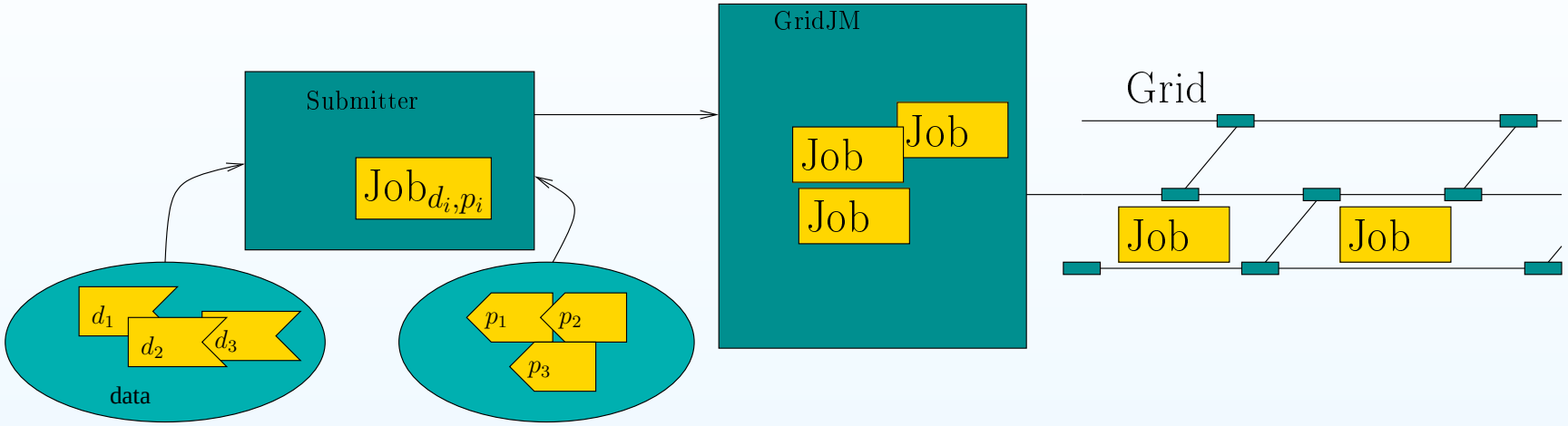
By collecting history and infosys information, GridJM addresses the following:

- Fault tolerance
- Fault avoidance
- Minimizing time between sending the job and receiving the results
- Visualization of resource usage
- Automatic collecting of results



Hide the complexity from the user!

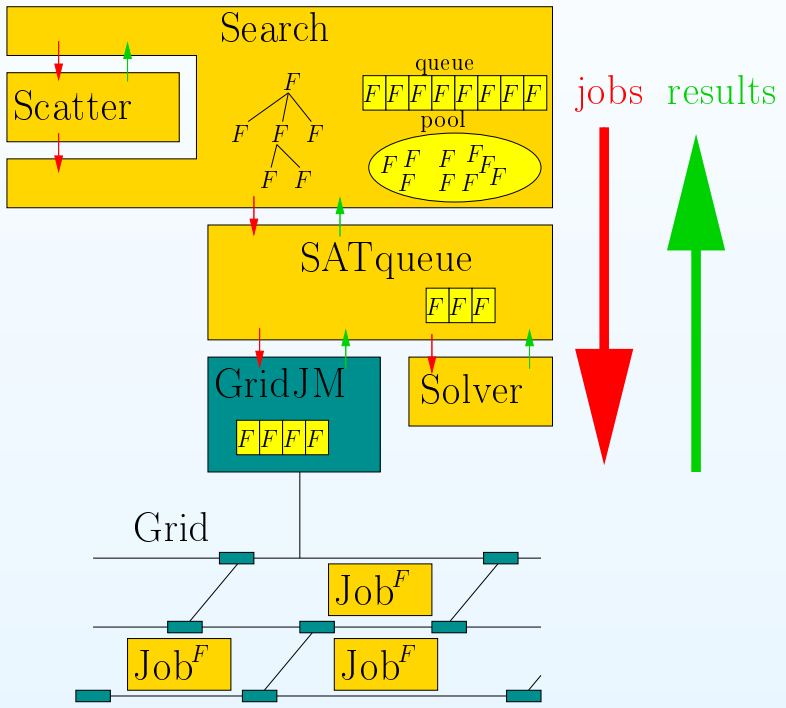
Case Study: Independent jobs with parameters



- A job manager can help here by
  - Submitting a set of previously constructed jobs
  - Ensuring that the jobs are run
  - Collecting the results automatically
  - Enhancing throughput by using history information

Case Study: Constraint Model Solving in Grid

- Constraint Models: Declarative logical formulation of a problem as a set of constraints to the possible solutions
- New subproblems are constructed based on previous results
- Dynamic distribution strategy in solving
- Brokering must be done during the search

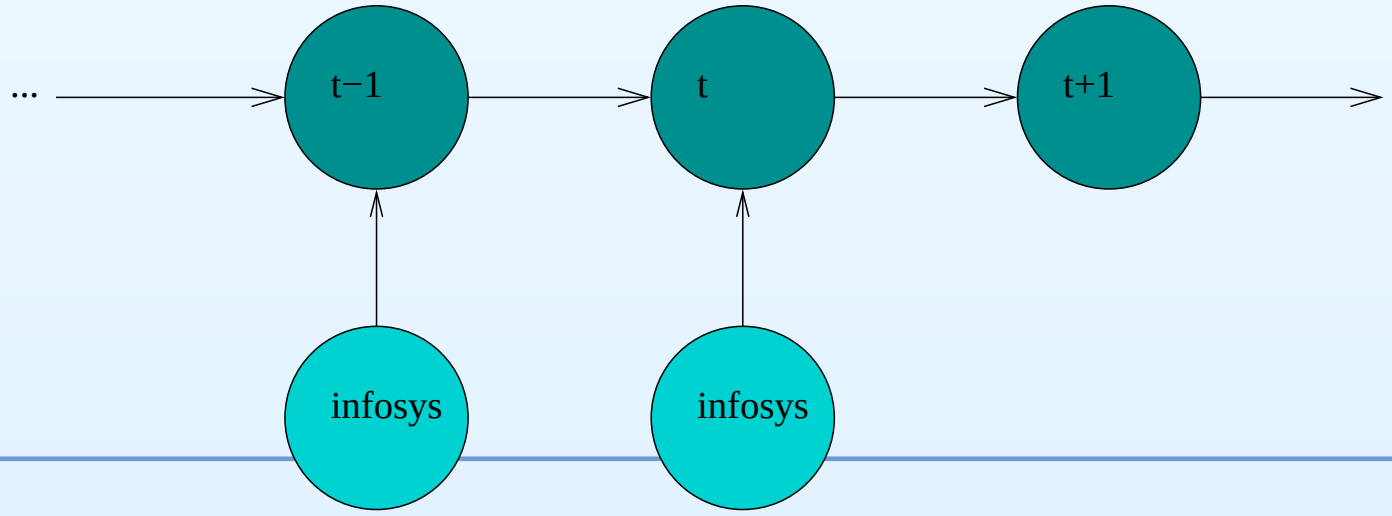


Fault Tolerance and Avoidance

- Users need a reliable execution environment
- Misconfigured clusters and random faults result in failed jobs
- Monitor jobs (constantly) while they are running
- Resubmit failed jobs automatically (limited times)
- Avoid badly working clusters by constructing a dynamic blacklist
  - If certain cluster fails your job once, it will probably do it again soon
  - Try clusters again occasionally, since the problem might disappear

Optimize Total Time to Delivery

- The information about the grid comes from two sources
  - Grid infosystem
  - User experience
- A learning broker
  - Resubmit jobs stuck in queue
  - Avoid loaded clusters where queue time is long
  - Update lists by retrying occasionally loaded clusters
- Maintain a (probabilistic) model of the grid



Efficiency in Job Submission

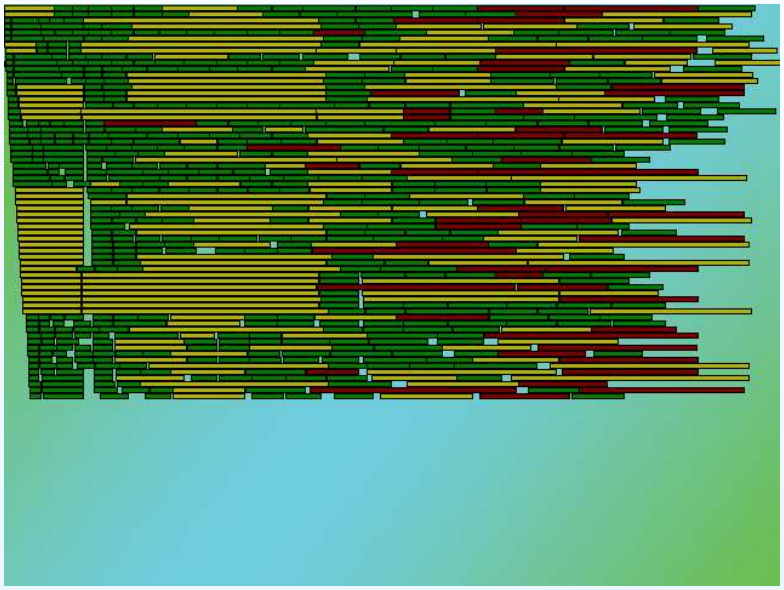
- Information about Clusters, queues and queue statuses are needed to make brokering decisions
- Especially queue status is time-consuming to gather and always out-of-date
- Cache the queue info locally
  - update periodically with queries
  - update local cache when jobs are submitted
- This is available in `ngsub`

Job migration

- What if no resources are available at the time of submission?
  - The job must be submitted to a queue
  - After some time, another queue might become shorter
  - The previously submitted job should now be moved to the new, shorter queue
  - The process is called *job migration*
- The process is complicated, for example due to queue priorities
- Job migration can be approximated and generalized with a simple scheme
  - If a job remains long in a non-running state, Remove the job from the cluster and re-submit it

Visualization

- Long grid runs produce large amounts of log data
- No time information: Difficult to detect performance problems in job creation
- Not easy to detect suspicious failures, such as downloads, resubmission rates
- Solution: Visualize the distributed execution



Automatic result retrieval

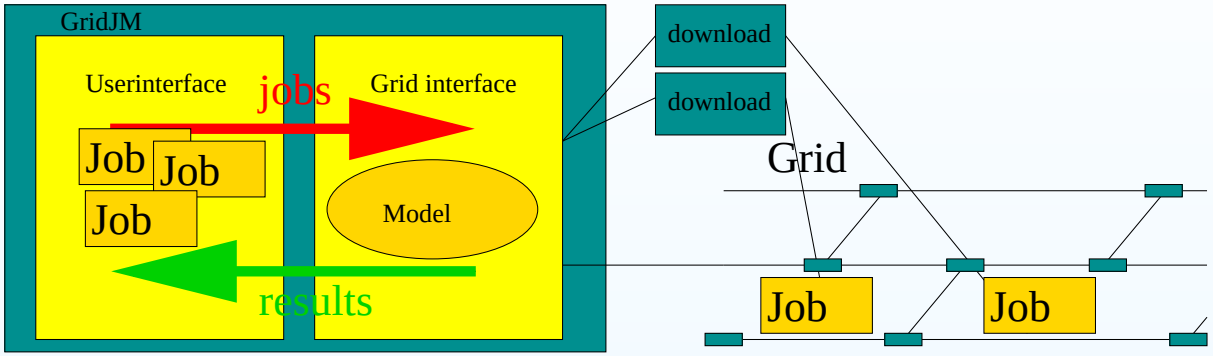
- Simple abstraction: Run a job in the grid, get the result to your self, ASAP.
- Not always this simple
  - Complex workflows
  - Huge result files
- User wishes to have some notification concering finished jobs
- For efficiency reasons, transfers are done in parallel

GridJM: A Set of Scripts or a Process?

- Script approach: Use a set of shell scripts to launch ngsub, ngstat, ngget, ngkill. . .
  - Fast to write (?)
  - Single process failure is not catastrophic
- Process approach: A (single) process handles all communication (by arclib)
  - Efficient communication via low-level primitives
  - Easy to gather history (blacklists. . .)

We selected the process -approach

GridJM: Implementation



- Simple interface to user
- userinterface
  - Listen user socket
  - Listen results from grid interface
  - Queue incoming jobs
- grid interface
  - Maintain / update model
  - Start downloads (separate process)
  - listen to ending downloads (sig_chld)

GridJM: Examples

```
diagram% grid-proxy-init
Your identity: /O=Grid/O=NorduGrid/OU=hut.fi/CN=Antti Hyvarinen
Enter GRID pass phrase for this identity:
Creating proxy ........................................... Done
Your proxy is valid until: Sat Sep 22 04:52:18 2007
diagram% ./gridjm -x 5
Using retry 5
Using net for communication
Listener thread started
proxy subject: /O=Grid/O=NorduGrid/OU=hut.fi/CN=Antti Hyvarinen/CN=2144211867
proxy valid to: 2007-09-22 04:52:18
Accepted Connection
Number of jobs in grid: 0
.User sends data
Received EOF
New job was put into queue
```

GridJM: Examples

```
diagram% nc localhost 12345
# free 64
# free 64
# free 64
# free 64
# free 64
# free 64
```

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

GridJM: Examples

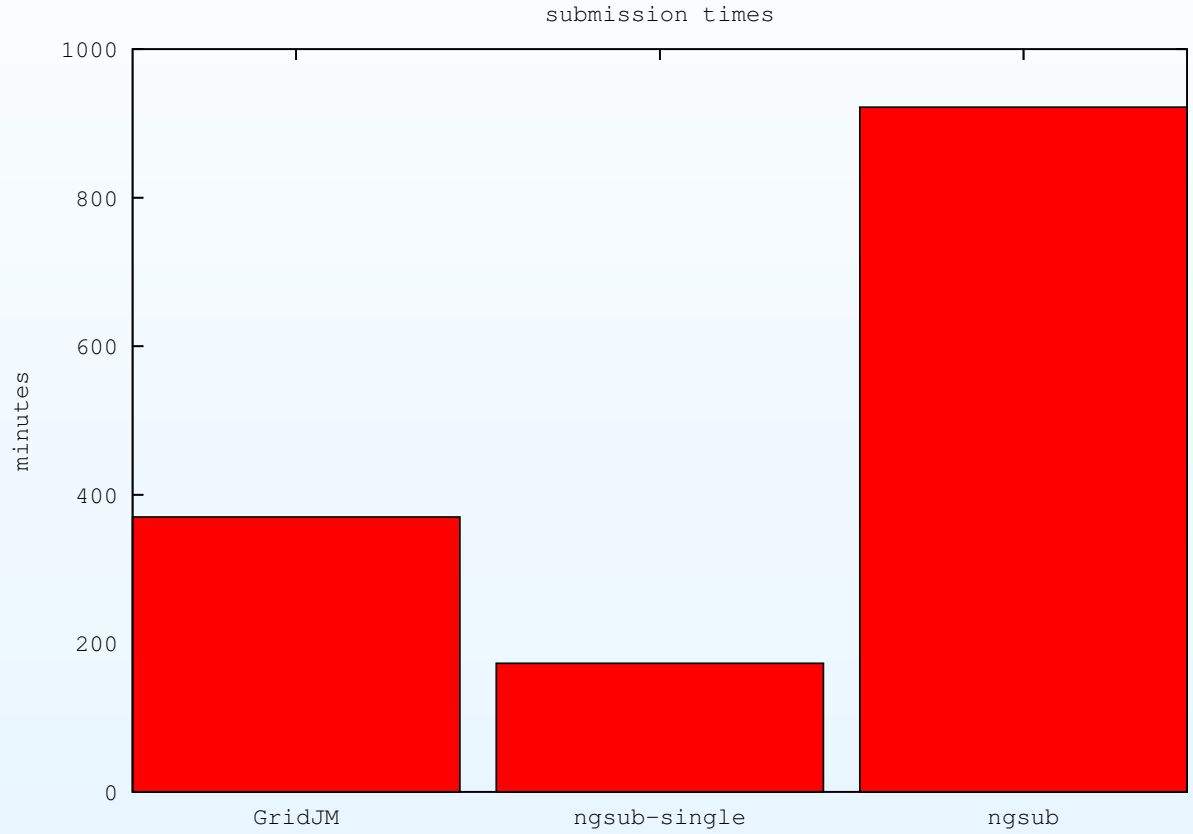
```
diagram% ./gridjm --help
Gridjm version 0.5.1
(c) Antti Hyvärinen and others

Gridjm usage:
  --automatic                Try to automatically guess the available
                              resources in the grid (experimental)
  --cluster-update int,      Cluster update interval (seconds)
    -c int                    (3600)
  --dir str, -d str          Use the download directory given as argument
                              (default is /tmp/ngdownload/)
  --errdir dir, -e dir       Receive failed jobs to directory (default no)
  --help, -h                 This help text
  --instrumentation str,     Instrumentation file (default /tmp/gridjm.ins)
    -i str
  --jobstall int, -j int     Job state stall timeout (seconds)
                              (600)
  --maxjobs int, -m int      Maximum number of simultaneous jobs requested
  --ngget str, -n str        Use ngget program given as argument
                              (default is ngget)
  --port int, -p int         Use tcp port number given as argument (12345)
  --queue-update int, -q int Queue update period (seconds) (300)
  --pidfile str, -r str      Pid file name (default /tmp/gridjm.pid)
  --interval int, -t int     Update interval for jobs in grid (seconds)
  --useuds str, -u str       Use uds for communication
                              (default is not to use)
  --retry int, -x int        Number of submission retries (1)
  --version, -v              Show the version number
Exiting
diagram%
```

Some results

- Benchmarks
  - sleep 300 seconds
  - 3*10 Mb random input files
  - 1000 jobs
- Experiments
  - GridJM using a single resubmission
  - ngsub with a single xrsl
  - ngsub with 1000 xrsl's

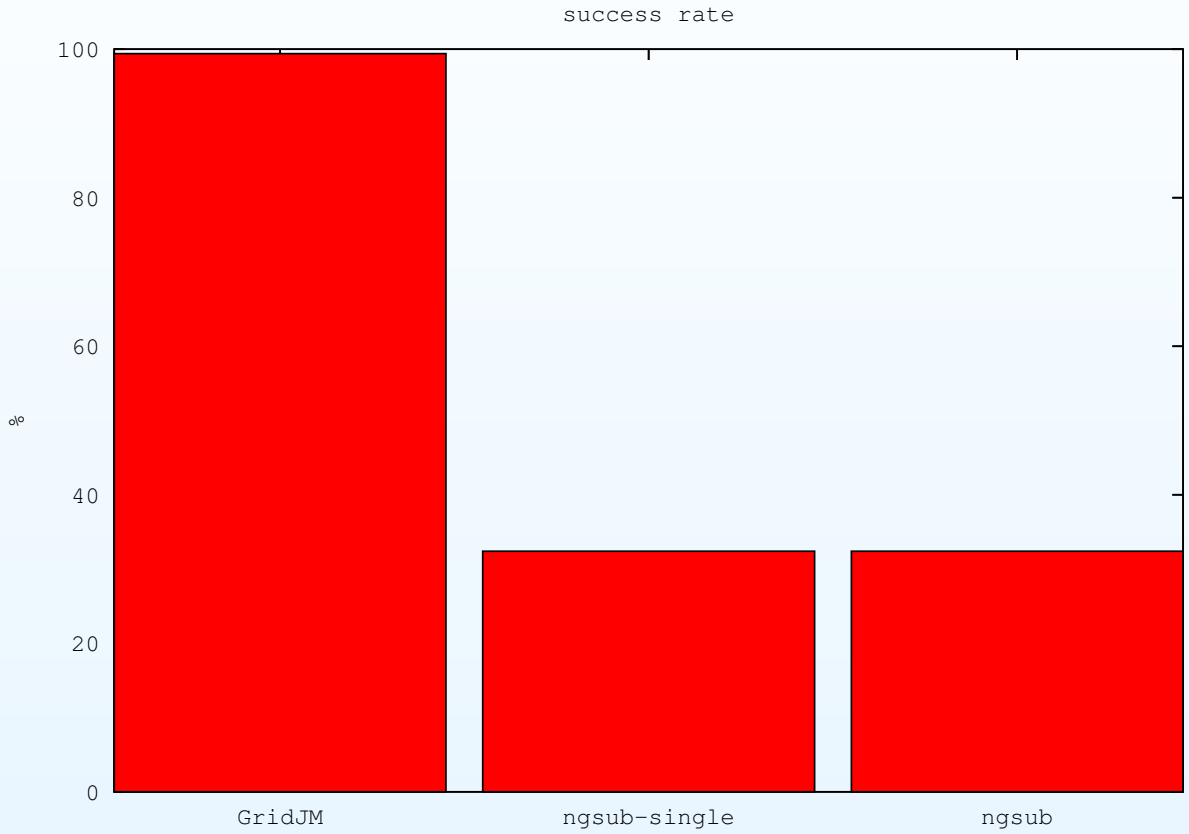
Submit times



GridJM is slower than submitting everything in single xrsl

- However, not everything can be done in single xrsl
  - e.g. the constraint problem

Success rate



GridJM can be considerably more reliable

- The success rates are equally bad for single and multiple submissions!
- Only 6 resubmissions required for GridJM

Conclusions

- `http://www.tcs.hut.fi/~aehyvari/gridjm/`
- Greatly simplifies and streamlines ARC usage

Things to be improved

- Better local grid model
- Time to delivery from sending to end of download
- More realistic visualization (w.r.t. processor time)
- Nicer userinterface