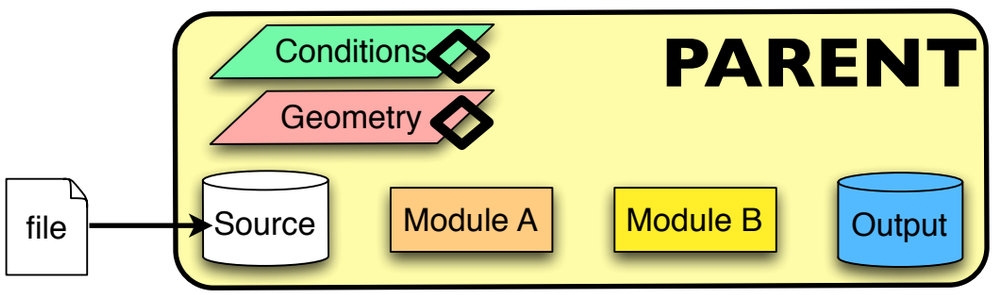
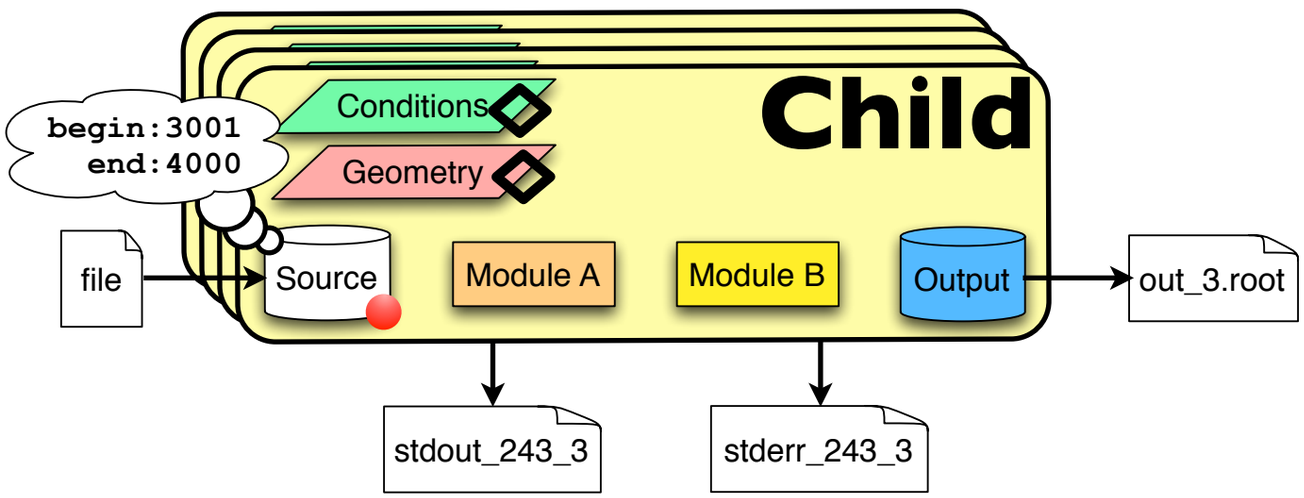
Whole Machine Jobs in CMS

OSG All Hands Meeting
20. March 2012

José Hernández (CIEMAT)
Oliver Gutsche (FNAL)

Multi-Core CMSSW jobs



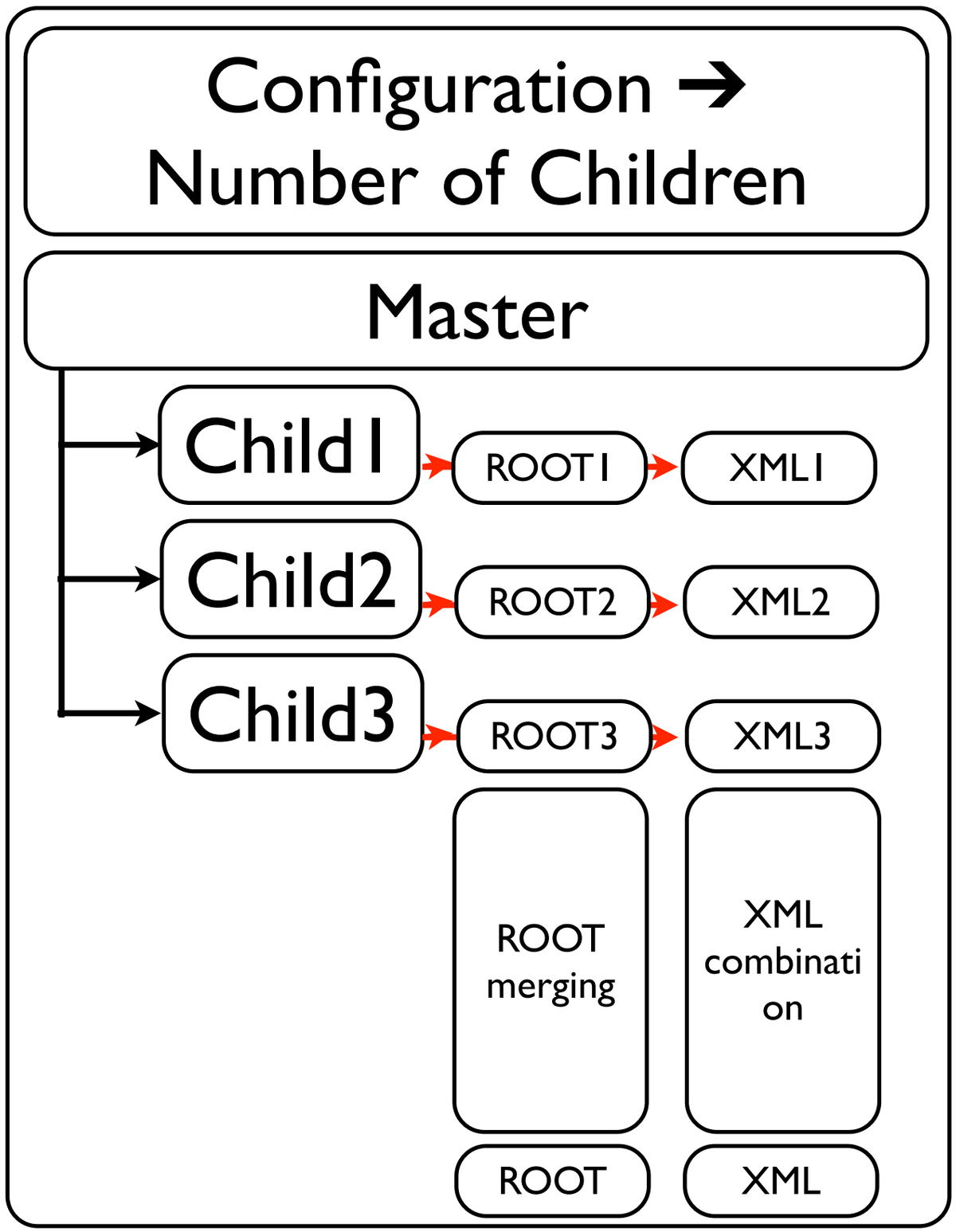


For the Expert:

- one CMSSW configuration file
- Select number of children via parameter
- Write out one file per child
- Provide one FrameworkJobReport.xml per child and one master xml

- **Parent**
  - Reads configuration and loads modules
  - Configuration says how many children and # events/child
  - Opens input file and reads first run
  - modules are not called
  - Pre-fetches conditions, calibrations and geometry
  - Sends message to all modules that forking is going to happen
  - source closes file
  - Forks
- **Children**
  - Redirects stdout and stderr to own files whose names contain parent PID and child #
  - Send messages to modules saying process is child X
  - Output modules append child # to file names
  - Sources calculate their event ranges to process (no IP communication) and re-open the file
  - Process events in child's start/end range normally

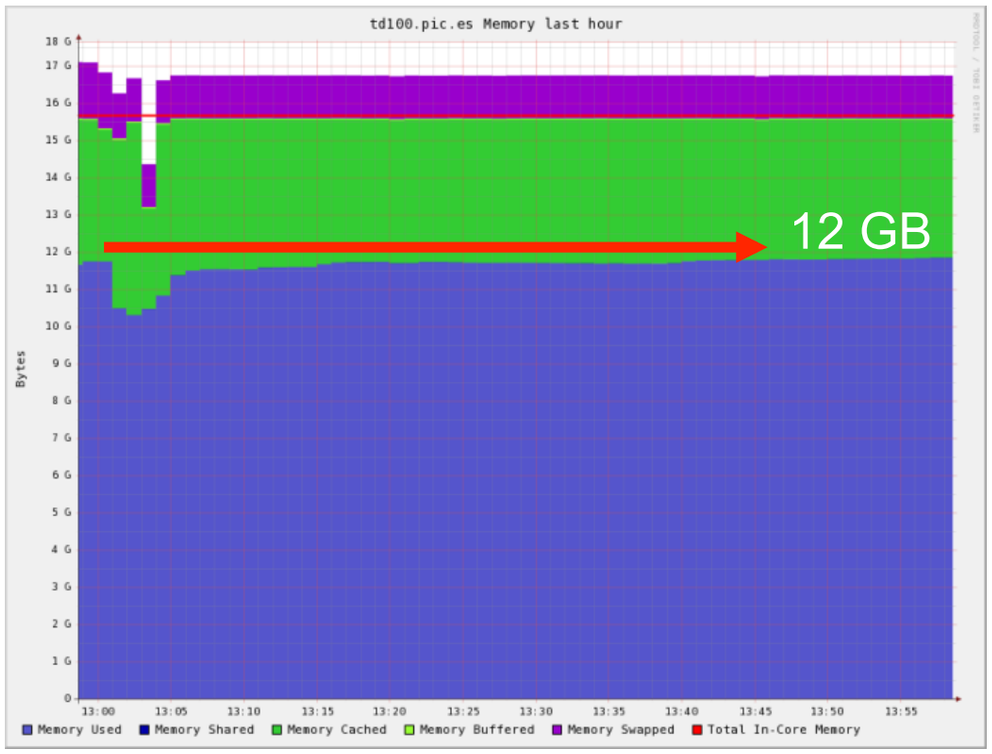
Multi-Core WMAgent job



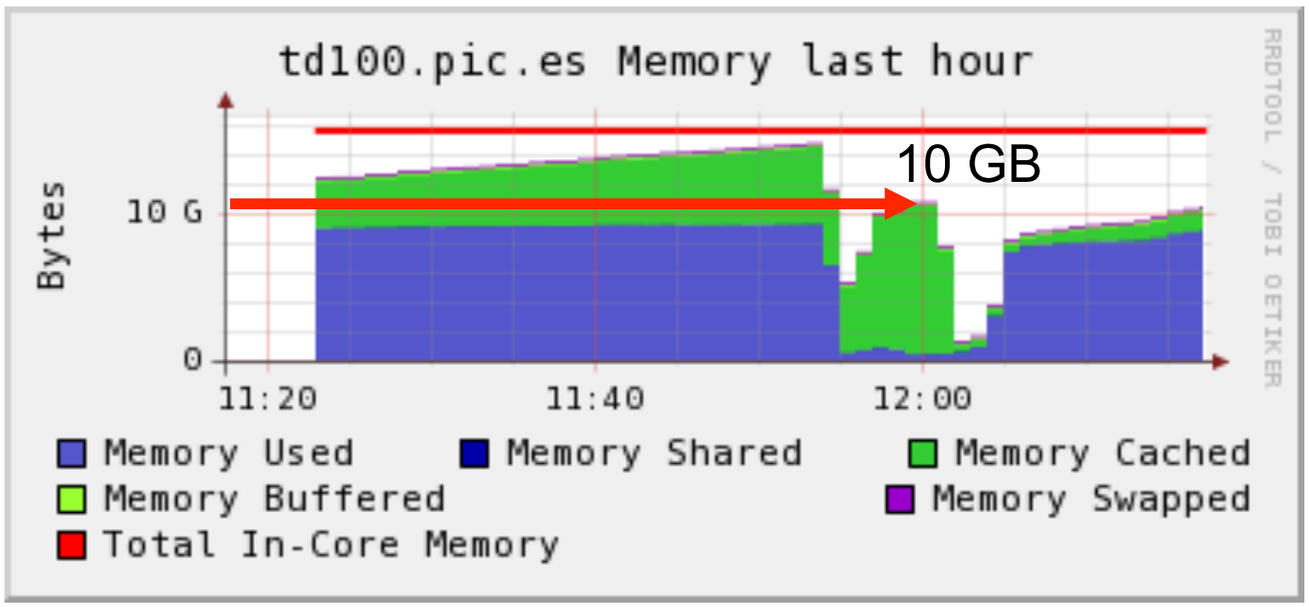
- JobWrapper configures number of children
  - Either via workflow settings
  - Or using /proc/cpuinfo to use the whole node
- JobWrapper executes single CMSSW job producing multiple FrameworkJobReport.xml plus master xml and multiple ROOT output files
- JobWrapper merges all ROOT files and stages it out to MSS and also combines all xml into one

Multi-Core Memory Performance

8 Single-core jobs



One 8-core job

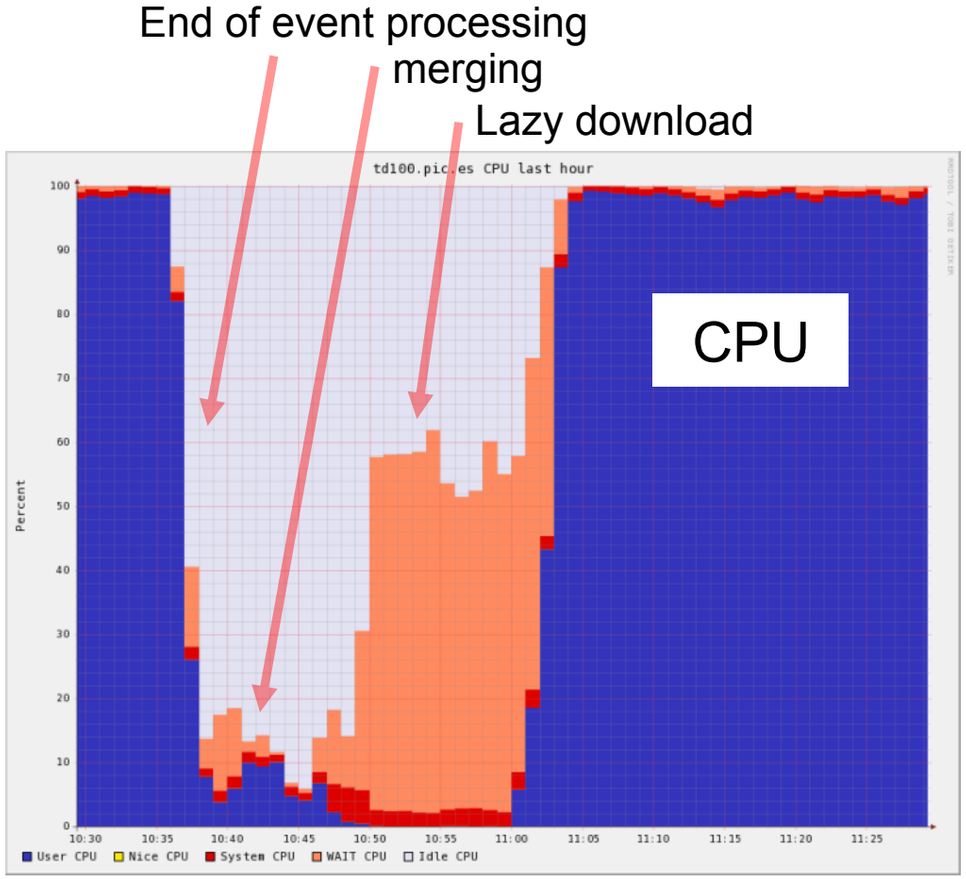


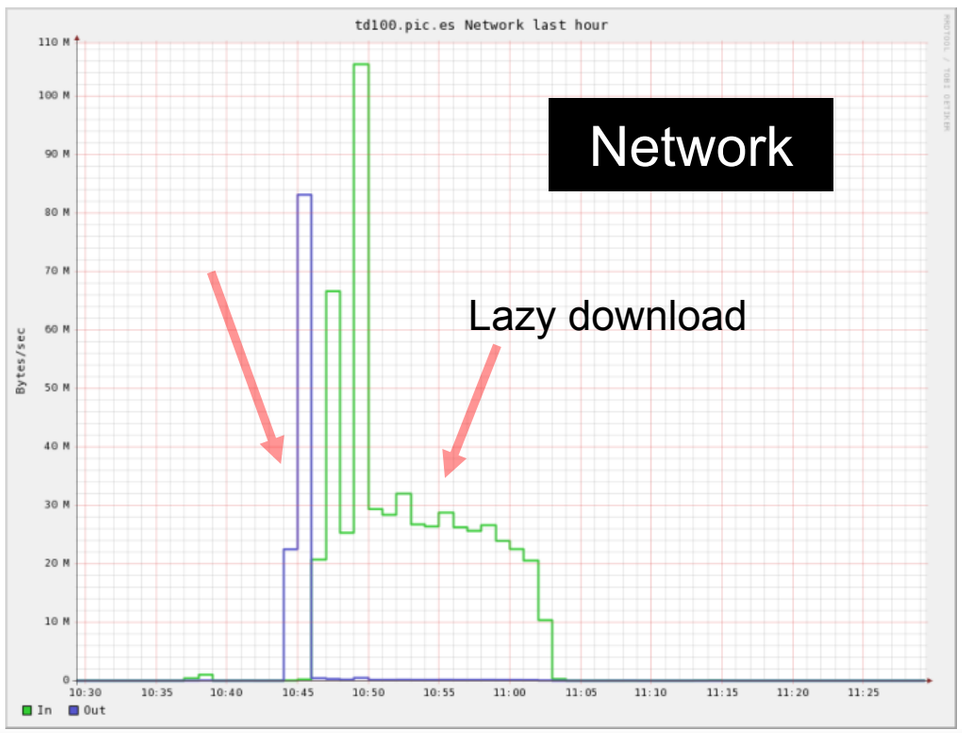
- Clear memory gain (~20%) with multi-core processing
  - 8-core job, ~2 hour long, ~9 GB total memory used by the machine
  - Reported in framework job report for each processing child:
    VSIZE: 2 GB, RSS: 1.5 GB, PSS: 900 MB
  - Parent process also consumes some memory
- To compare with 8 simultaneous single-core jobs, same workflow, ~11 GB

```
top - 09:20:35 up 1 day, 15:37,  3 users,  load average: 7.96, 5.07, 2.20
Tasks: 158 total,   9 running, 148 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.7%us,  0.1%sy,  0.0%ni, 99.1%id,  0.1%wa,  0.0%hi,  0.0%si,
Mem:  16437844k total,  9414440k used,  7023404k free,   239148k buffers
Swap:  4192924k total,        0k used,  4192924k free,  1234428k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+
COMMAND
13411 cmsdc04  418   0 1917m 1.5g  91m R 101.0  9.6   4:49.59 cmsRun
13414 cmsdc04   25   0 1946m 1.5g  91m R 101.0  9.7   4:49.08 cmsRun
13407 cmsdc04   18   0 1935m 1.5g  91m R  99.0  9.6   4:47.88 cmsRun
13408 cmsdc04   20   0 1934m 1.5g  91m R  99.0  9.7   4:48.14 cmsRun
13409 cmsdc04   21   0 1980m 1.6g  91m R  99.0  9.9   4:41.31 cmsRun
13410 cmsdc04   18   0 1946m 1.5g  91m R  99.0  9.7   4:45.93 cmsRun
13412 cmsdc04   18   0 1947m 1.5g  91m R  99.0  9.7   4:49.49 cmsRun
13413 cmsdc04   25   0 1917m 1.5g  91m R  99.0  9.5   4:49.41 cmsRun
13404 cmsdc04   22   0 1184m 964m 168m S   0.0  6.0   0:43.70 cmsRun
```

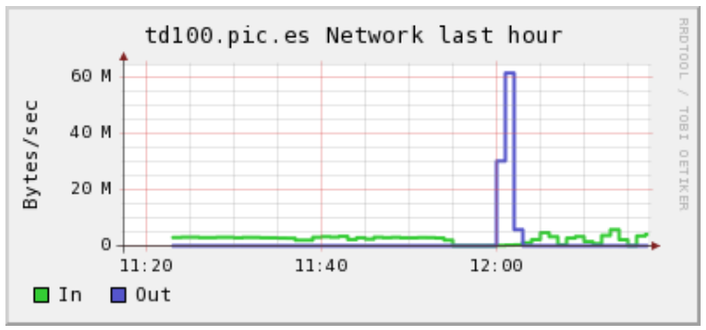
Lazy Download

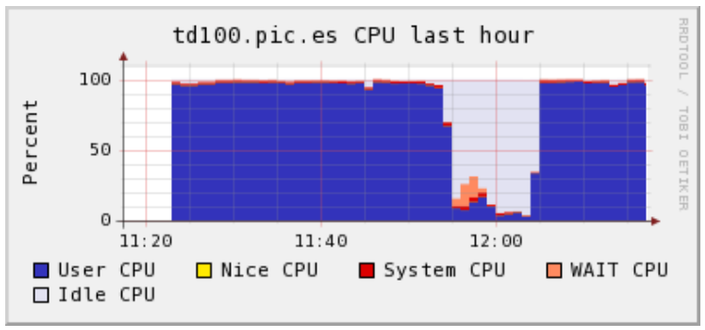




- Lazy download produces a large startup overhead for multi-core jobs
  - Large IO/wait, local disk hammered by processing children downloading input file(s)
- With lazy download off no startup overhead
- Small overhead due to children processing dispersion, file merging and stage-out (~10 minutes all cores ~idle)

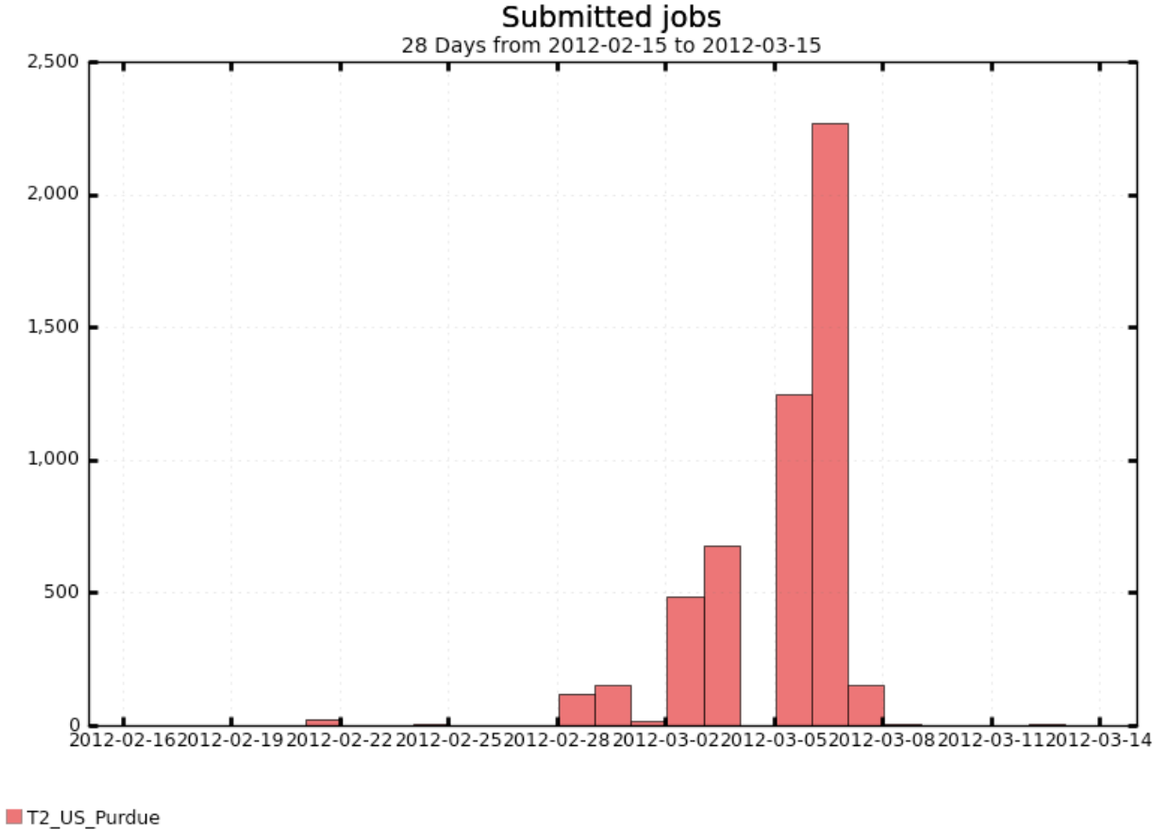
Without Lazy download

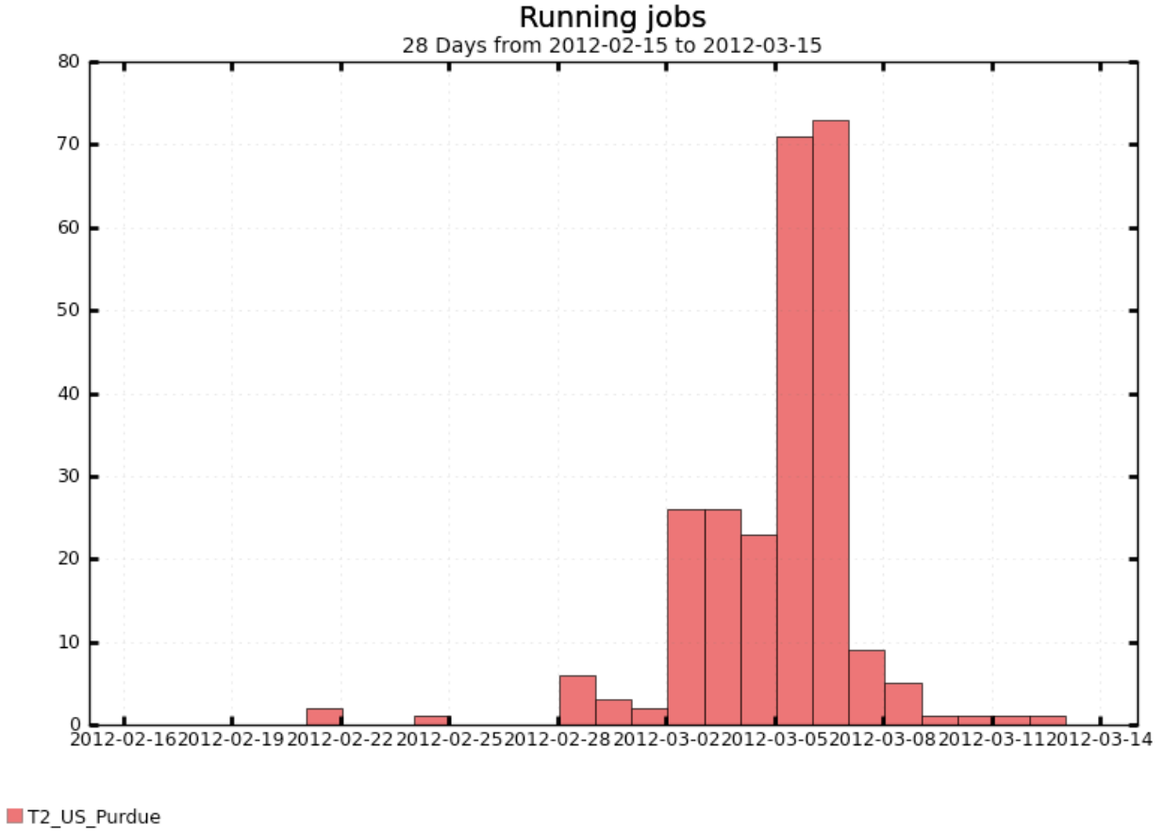




Tests

- Setup
  - Dedicated small whole-node queues at all T1 sites and one T2 site
  - Configured as separate HTPC queues in glideIn factory
  - New: Dynamic 8-core queue at Purdue
    - globus_rsl = (jobtype=single)(queue=cms)(xcount=8)(host_xcount=1)(maxWallTime=2800)





Summary

- 20% memory gain compared to single core jobs
  - Asynchronous merging very much reduced
  - Number of processing jobs very much reduced

- Dedicated queues at Tier-1 sites used for initial tests
  - Tier-1 sites will not like to move parts of their resources to multi-core usage

- Dynamic multi-core slots at Purdue are working and simple to use
  - ~5k jobs run with about 70 jobs in parallel (70x8 cores!)
  - Preferred solution of Tier-1 sites to use multi-core jobs, but still questions about accounting (for example when draining a node to have enough cores for 1 job)
- T1_DE_KIT will be providing similar queues with 4 and 8 cores available per slot very soon

- WLCG TEG recommendation
  - Number of cores configurable during job submission and site provides dynamically access to multi-core slots