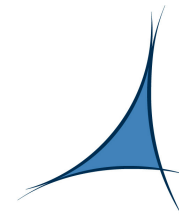


Ciemat

Centro de Investigaciones
Energéticas, Medioambientales
y Tecnológicas

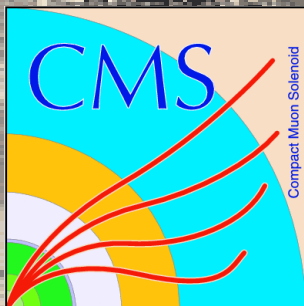


PIC
port d'informació
científica

CMS Multicore Scheduling Strategy

Antonio Pérez-Calero Yzquierdo,
Jose Hernández, Burt Holzman,
Krista Majewski, Alison McCrea

for the **CMS Collaboration**



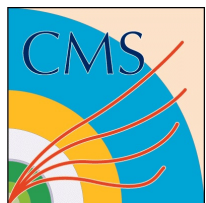


Outline

- Motivation for multicore jobs
- CMS strategy for multicore application and scheduling
- Testing the proposed scheduling strategy
- Outlook for future developments and conclusions



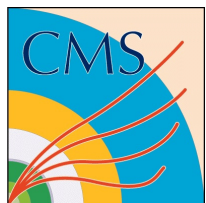
Going multicore



Going multicore

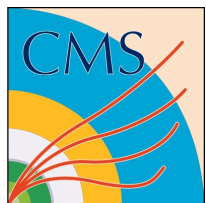
Motivation:

- **Hardware evolution:** over the last decade architecture design goes in the direction of adding processors to the CPU, while individual core performance will probably not increase significantly
- Need to deal with **evolution of LHC conditions:** increased **data volumes** to be processed, with increased **event complexity** due to higher pileup, causing higher processing time per event and memory usage



Going multicore

- Adapt HEP computing **from sequential programming to parallel processing** to efficiently use **multicore capabilities** and avoid running into **memory limitations**
- Modifications at different levels in the **software** we use:
 - Application
 - Grid-wide scheduling
 - Site scheduling
- New era for HEP computing with the integration of elements of Grid Computing and High Performance Computing: ***distributed parallel computing***
- View: LHC upgrades ↔ CMS detector upgrades ↔ CMS software upgrades



Going multicore

Advantages for multicore jobs:

- Fully **exploit future CPU capabilities**, adapting code to new architecture designs
- Reduced **memory consumption** per core, as memory is shared between threads
- Reduced **number of jobs** to be handled by our Workload Management System
- **Output files** of larger size requiring less managing and merging operations

Multicore jobs will be intensively used in the near future: need scheduling strategies to handle them

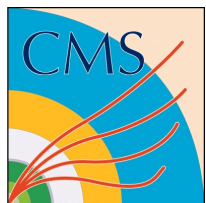


CMS strategy for multicore jobs application scheduling



CMS multicore application

- CMS is exploring approaches to parallel processing at **different levels**:
 - a) Run events in parallel processes
 - b) Process data modules inside an event in parallel
 - c) Both combined: processing in parallel modules not necessarily from the same event
- Parallel threads **share common data in memory**, such as detector geometry, conditions data, etc.
 - Promising preliminary results with parallelization **at event level** implemented as forked subprocesses: remarkable **gain in RAM** (25-40%), for a small CPU inefficiency, due to merging of the output file (CHEP12)
- Parallelization is **limited by sequential parts** of the program.
 - good CPU efficiency running on multicores only when the major part of the code can be run threaded
- **Development status**: first multithreaded application ready for production-size tests by the end of 2013 (see contribution 158, “Transitioning CMS to a hierarchical threaded framework“)



Multicore scheduling

Objectives:

- Integrate scheduling of both **multicore and single-core jobs**, that will still be used for CMS analysis jobs
- Single core jobs will also remain in use by other VOs in shared sites: **avoid splitting resources**, such as dedicated whole node slots and separated queues, which introduce inefficiency and additional complexity in site resources configuration and management
- High efficiency CPU usage, **minimizing inefficiencies** deriving from scheduling
- Intensively automated scheduling system to **reduce manpower** needed to run the whole experiment workflow
- Allow proper **accounting** of resources usage and implementation of **priority policies** for types of jobs, users, etc
- Provide tools for WMS **status monitoring** to be used by system operators

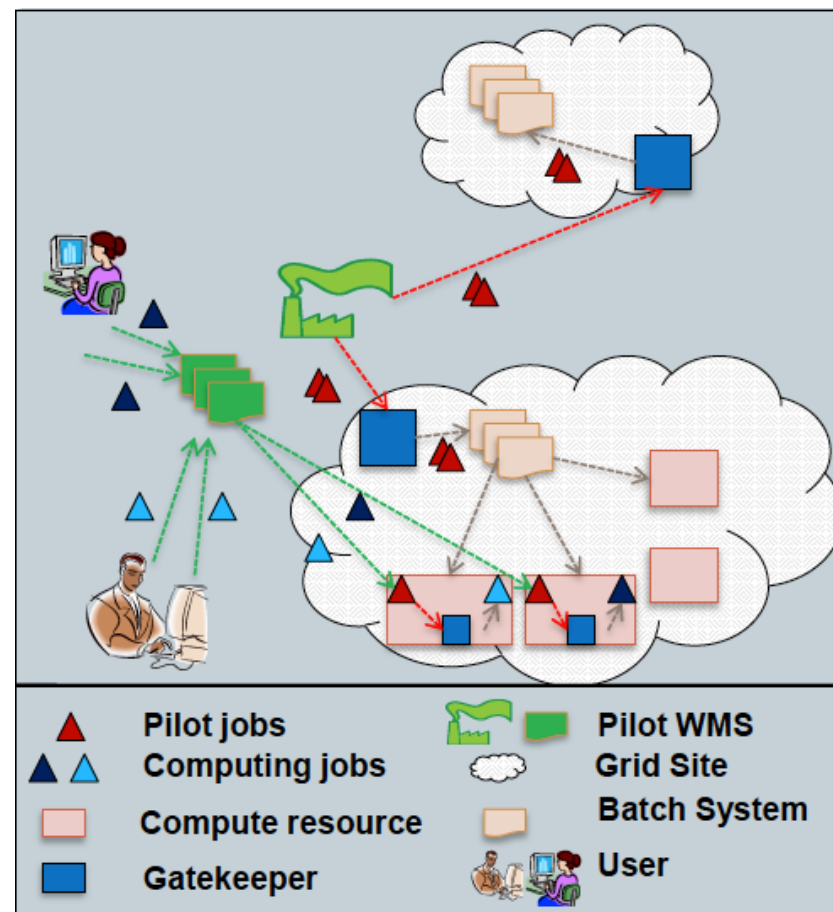


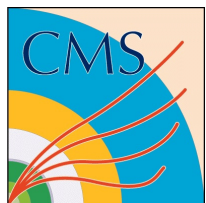
CMS workflow management

- CMS WMS infrastructure is currently built on **glideinWMS**, a grid-wide batch system, derived from **HTCondor** WMS.



Key concept: **pilot jobs pulling user jobs**

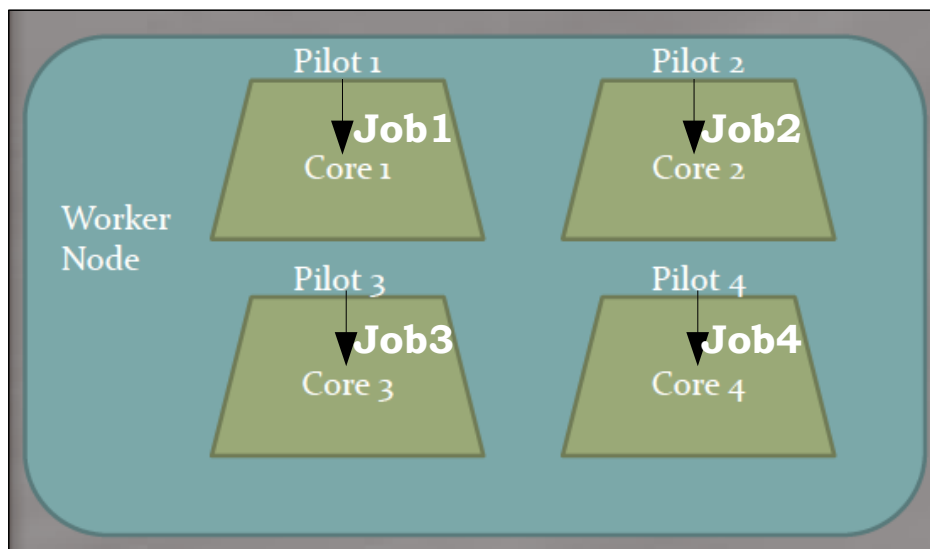
- pilots are sent to all different grid sites matching job resources request
- pilots enter local batch systems queues
- if resources are allocated, running pilots at one or several sites define a virtual pool of computing resources to be used by the grid-wide WMS
- User job is assigned to the first pilot that makes it run

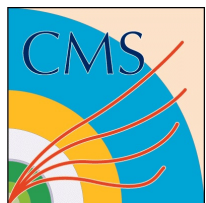




Single-slot pilots

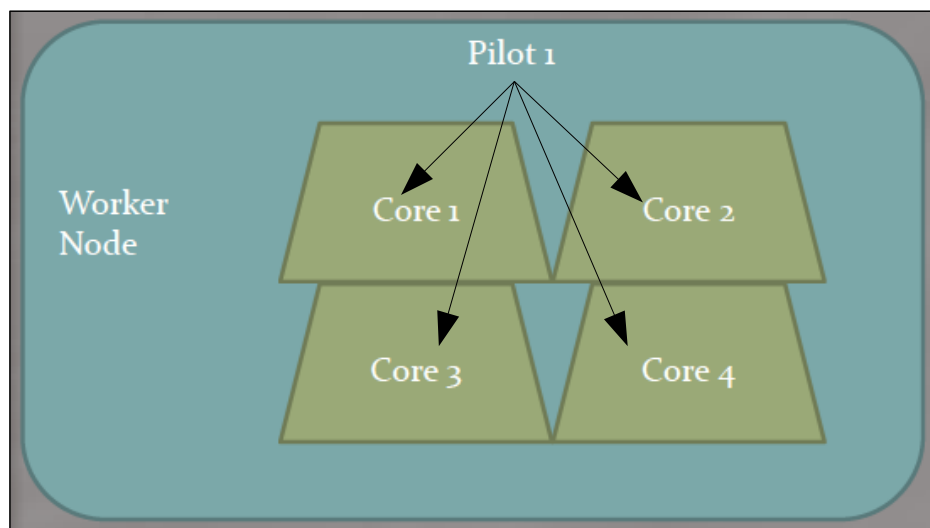
- The model currently in use: one pilot per batch slot and running one single core application
 - No WN draining inefficiencies 
 - No use of multicore capabilities 

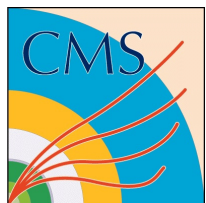




Multicore pilots

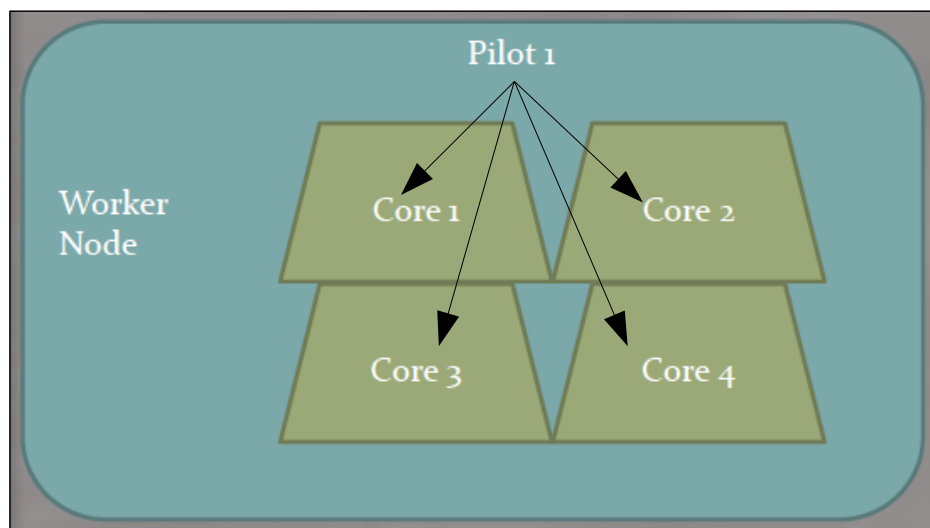
- **Multicore pilots** handle **several batch slots**
- **Past tests** for multicore application jobs used multicore pilots taking **dedicated whole nodes**: it worked but is not the preferred way by sites
- **New schema: multislot pilot** with **dynamic partitioning** of allocated resources:
 - pilots **take N slots** from the local batch system
 - pilots **arrange M internal slots** according to user job's requirements
- **Multicore pilots required** to run **multicore applications**, but **beneficial** even if used with **single core jobs**: reduced number of pilots required to manage whole workflow





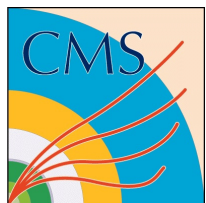
Multicore pilots

- Multicore pilots require **site configuration**:
 - Configure **1 batch slot** ↔ **1 WN core**: no separated dedicated queues for single core and multicore jobs → simplified management
 - Configure **local scheduler allocation policy**: job allocation to **fill machines**, instead of load balance → increased chance for N free slots being found on a WN





Testing the proposed scheduling strategy



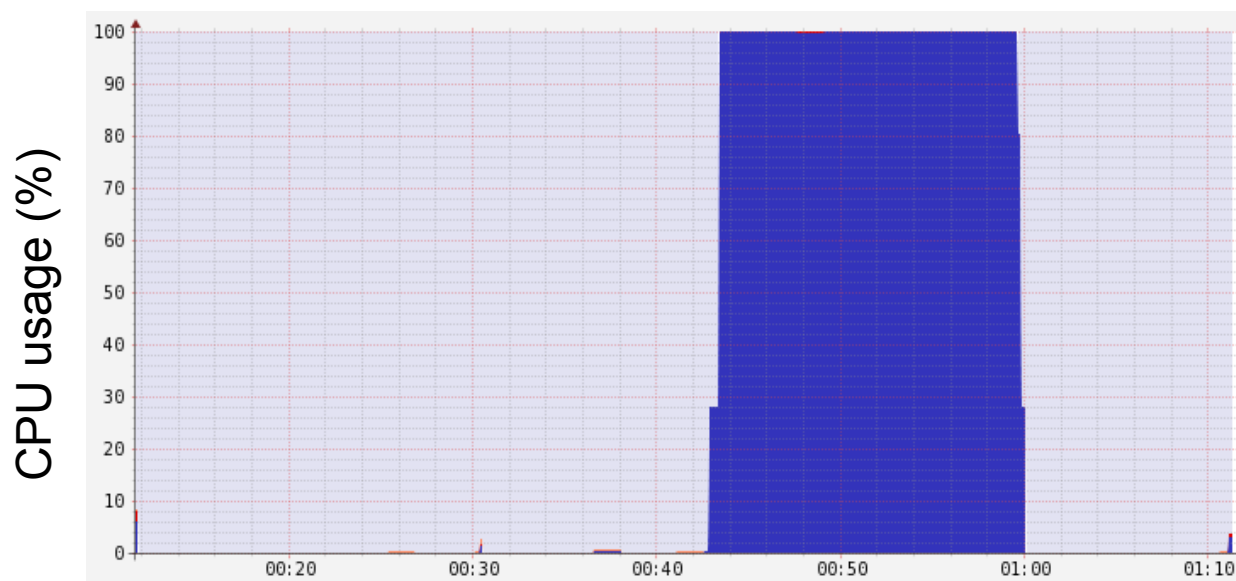
Scheduling tests

- **Infrastructure:**
 - glideinWMS testbed setup at CERN, equivalent to the one used for production
 - test queues in several sites (FNAL, PIC)
- **Objectives:**
 - **Check validity** of the proposal to handle single and multicore jobs simultaneously
 - **Study inefficiencies** in CPU usage deriving from **scheduling strategy** (not the application itself)
- Extra **tool**: jobs 100% efficient in CPU usage: **stress**

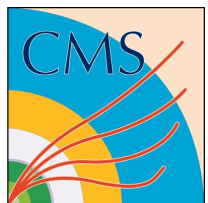


Scheduling tests

- Measure inefficiencies in CPU usage caused by job allocation: execute stress to load CPU
- Tests run in **8-core WNs** at PIC
 - **Example:** `stress --cpu 8 --timeout 1000s`



- Test jobs will execute **1, 2 and 4 stress threads** to load the corresponding number of cores to 100%



Local batch system

- Multislot pilots have been tested in **2, 4 and 8 slot** configurations
- Test queue at PIC: 2 8-core WNS, **1 core = 1 queue slot**, 16 slots in total
- Pilots arrive at local batch system and request resources
- From the site point of view, pilots are just jobs asking for N slots
- Example for 4 slot pilots:

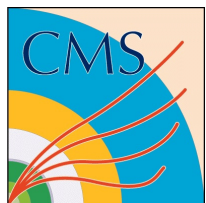
Job ID	Queue	NDS	TSK	Memory	Time	S	Time
23744131.pbs02.p	cms_mco	1	4	--	100:0	R	63:38 td457+td457+td457+td457
23744132.pbs02.p	cms_mco	1	4	--	100:0	R	32:21 td458+td458+td458+td458
23744133.pbs02.p	cms_mco	1	4	--	100:0	R	31:45 td457+td457+td457+td457
23744134.pbs02.p	cms_mco	1	4	--	100:0	R	31:12 td458+td458+td458+td458
23744135.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744136.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744137.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744138.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744139.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744140.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744141.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744142.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744143.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --
23744144.pbs02.p	cms_mco	1	4	--	100:0	Q	-- --



Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

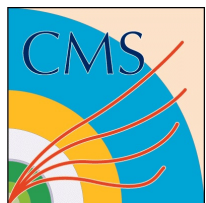
```
├─ /home/cmprd007/home_cream_876670542/CREAM876670542/glic
│   └─ condor_startd -f
│       ├── condor_starter -f -a slot1_4 vocms224.cern.ch
│       │   ├── /bin/bash /home/cmprd007/home_cream_876670542/
│       │   │   ├── stress --cpu 1 --timeout 200
│       │   │   └─ stress --cpu 1 --timeout 200
│       │   └─ condor_starter -f -a slot1_3 vocms224.cern.ch
│       │       ├── /bin/bash /home/cmprd007/home_cream_876670542/
│       │       │   ├── stress --cpu 1 --timeout 200
│       │       │   └─ stress --cpu 1 --timeout 200
│       │       └─ condor_starter -f -a slot1_2 vocms224.cern.ch
│       │           ├── /bin/bash /home/cmprd007/home_cream_876670542/
│       │           │   ├── stress --cpu 1 --timeout 200
│       │           │   └─ stress --cpu 1 --timeout 200
│       │           └─ condor_starter -f -a slot1_1 vocms224.cern.ch
│       │               ├── /bin/bash /home/cmprd007/home_cream_876670542/
│       │               │   ├── stress --cpu 1 --timeout 200
│       │               │   └─ stress --cpu 1 --timeout 200
│       └─ condor_procd -A /home/cmprd007/home_cream_876670542/
└─ /home/cmprd007/home_cream_876670542/CREAM876670542/glic
    └─ condor_startd -f
```



Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

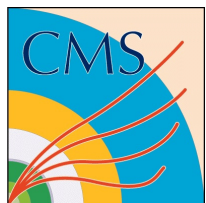
```
bin/bash ./glidein_startup.sh -v std -cluster 26592 -name
- /bin/bash /home/cmprd007/home_cream_412750610/CREAM41275
├─ /home/cmprd007/home_cream_412750610/CREAM412750610/gl
├─┬─ condor_startd -f
│   └─┬─ condor_starter -f -a slot1_1 vocms224.cern.ch
│       └─┬─ /bin/bash /home/cmprd007/home_cream_41275061
│           └─┬─ stress --cpu 4 --timeout 200
│               ├── stress --cpu 4 --timeout 200
│               ├── stress --cpu 4 --timeout 200
│               ├── stress --cpu 4 --timeout 200
│               └─ stress --cpu 4 --timeout 200
│   └─┬─ condor_procd -A /home/cmprd007/home_cream_41275
└─┬─ /home/cmprd007/home_cream_412750610/CREAM412750610/gl
    └─┬─ condor_startd -f
```



Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

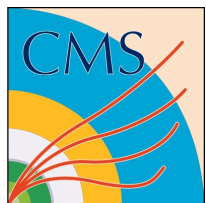
```
- /bin/bash ./glidein_startup.sh -v std -cluster 26592 -name v2_3 -entr
├─ /bin/bash /home/cmprd007/home_cream_114180671/CREAM114180671/glide
├─ /home/cmprd007/home_cream_114180671/CREAM114180671/glide_i1tE5Y
│   └─ condor_startd -f
│       ├── condor_starter -f -a slot1_3 vocms224.cern.ch
│       │   └─ /bin/bash /home/cmprd007/home_cream_114180671/CREAM114
│       │       └─ stress --cpu 2 --timeout 200
│       │           ├── stress --cpu 2 --timeout 200
│       │           └─ stress --cpu 2 --timeout 200
│       ├── condor_starter -f -a slot1_2 vocms224.cern.ch
│       │   └─ /bin/bash /home/cmprd007/home_cream_114180671/CREAM114
│       │       └─ stress --cpu 1 --timeout 200
│       │           └─ stress --cpu 1 --timeout 200
│       ├── condor_starter -f -a slot1_1 vocms224.cern.ch
│       │   └─ /bin/bash /home/cmprd007/home_cream_114180671/CREAM114
│       │       └─ stress --cpu 1 --timeout 200
│       │           └─ stress --cpu 1 --timeout 200
│       └─ condor_procd -A /home/cmprd007/home_cream_114180671/CREAM
└─ /home/cmprd007/home_cream_114180671/CREAM114180671/glide_i1tE5Y
    └─ condor_startd -f
```



Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

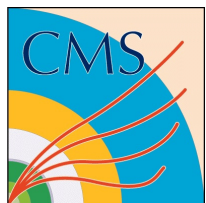
```
└─ /bin/bash ./glidein_startup.sh -v std -cluster 27583 -name
└─ /bin/bash /home/cmprd007/home_cream_626441369/CREAM62644
└─ /home/cmprd007/home_cream_626441369/CREAM626441369/gl
└─ condor_startd -f
└─ condor_starter -f -a slot1_4 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_62644136
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ condor_starter -f -a slot1_3 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_62644136
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ condor_starter -f -a slot1_2 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_62644136
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ condor_starter -f -a slot1_1 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_62644136
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ condor_procd -A /home/cmprd007/home_cream_62644
└─ /home/cmprd007/home_cream_626441369/CREAM626441369/gl
└─ condor_startd -f
```



Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

```
— /bin/bash ./glidein_startup.sh -v std -cluster 27583 -r
└─ /bin/bash /home/cmprd007/home_cream_050643276/CREAM050643276
└─ /home/cmprd007/home_cream_050643276/CREAM050643276
└─ condor_startd -f
└─ condor_procd -A /home/cmprd007/home_cream_050643276/CREAM050643276
└─ condor_starter -f -a slot1_1 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_050643276/CREAM050643276
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ condor_starter -f -a slot1_3 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_050643276/CREAM050643276
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ /home/cmprd007/home_cream_050643276/CREAM050643276
└─ condor_startd -f
```



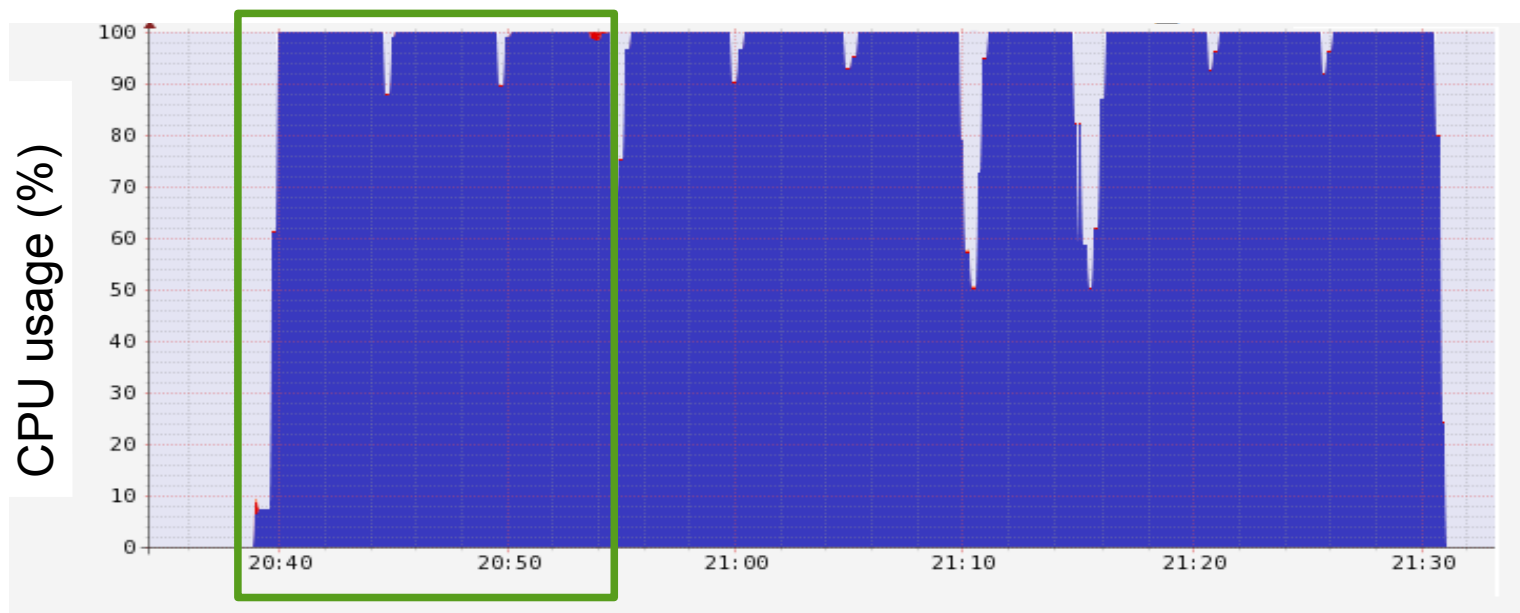
Pilot resources allocation

- Pilots can pull different types of stress test jobs and run them simultaneously

```
└─ /bin/bash ./glidein_startup.sh -v std -cluster 27583 -name
└─ /bin/bash /home/cmprd007/home_cream_050643276/CREAM05064
└─ /home/cmprd007/home_cream_050643276/CREAM050643276/gl
└─ condor_startd -f
└─ condor_procd -A /home/cmprd007/home_cream_05064
└─ condor_starter -f -a slot1_3 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_05064327
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ stress --cpu 4 --timeout 300
└─ condor_starter -f -a slot1_2 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_05064327
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ condor_starter -f -a slot1_1 vocms224.cern.ch
└─ /bin/bash /home/cmprd007/home_cream_05064327
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ stress --cpu 2 --timeout 300
└─ /home/cmprd007/home_cream_050643276/CREAM050643276/gl
└─ condor_startd -f
```

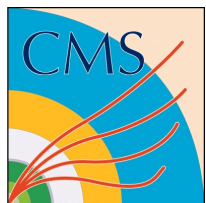


8-core pilot tests

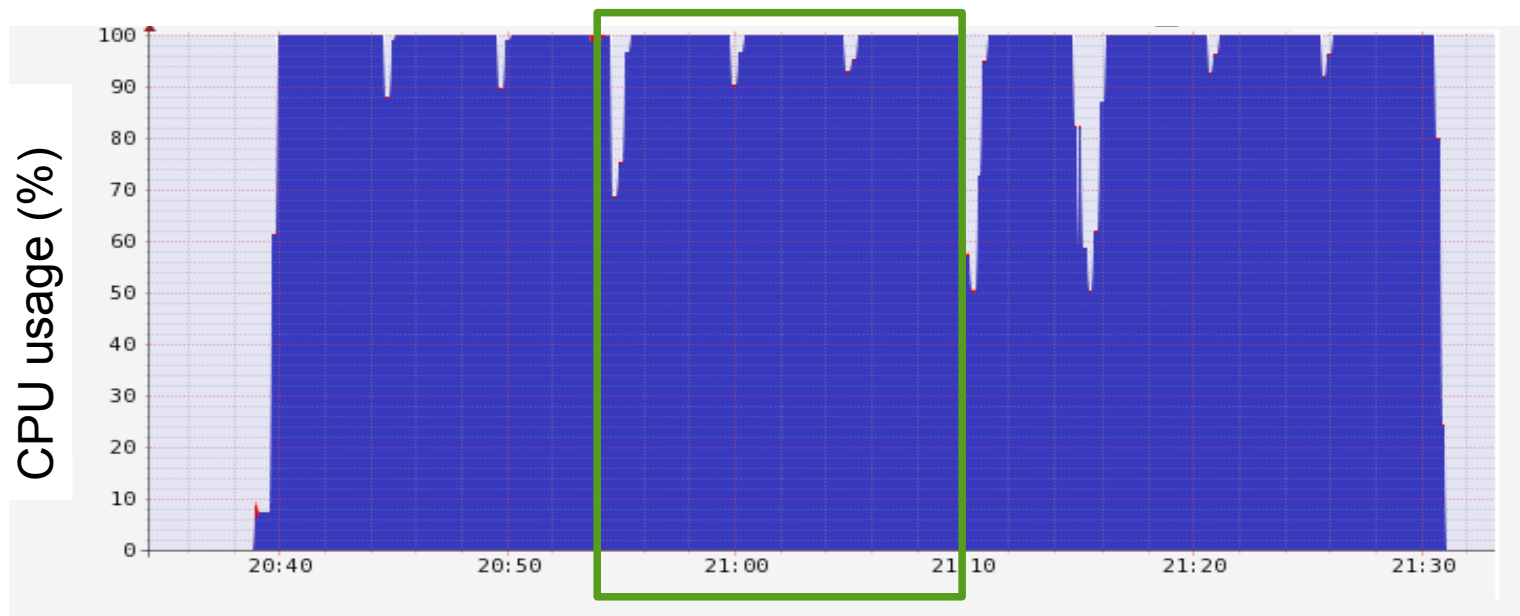


Test efficiency in terms of CPU usage for 8 core pilots:

- Queue is filled with a **mixture of 1, 2 and 4 core jobs** (alternating them)
- Pilots configure themselves internally at the startup according to queue content
 - Each pilot is arranged into **8 slots x 1 core** to start pulling single core jobs
- Pilots run single core jobs efficiently until none remains
 - Inefficiencies at startup and at job completion



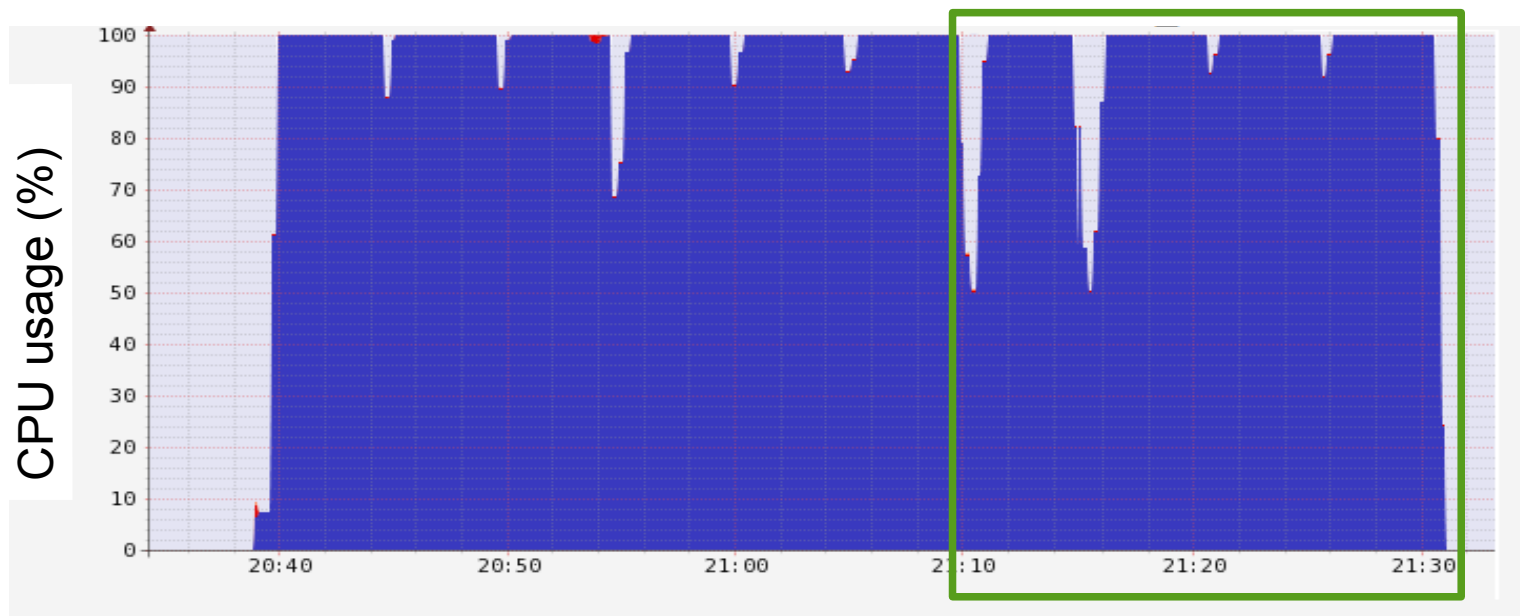
8-core pilot tests



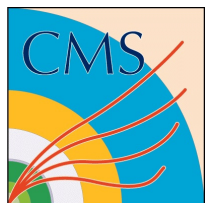
- Once single core jobs are exhausted, **2 core jobs** are next to be run
- Pilot internal slots may take idle resources: as jobs now require more cores in order to run, slots are rearranged: 8 slots x 1 core \rightarrow **4 slots x 2 cores**
- Inefficiency during core reallocation to internal slots and at job completion



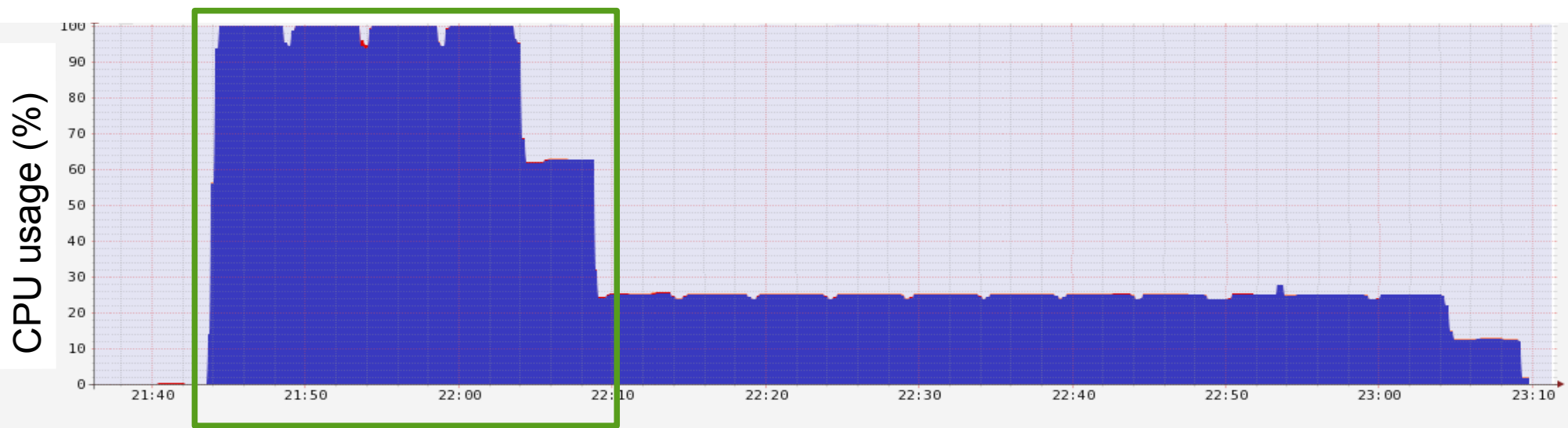
8-core pilot tests



- Once 2 core jobs are finishing, 4 core jobs are next to be run
- Once more, slots are again rearranged: 4 slots x 2 core \rightarrow **2 slots x 4 cores**
- Once slots are rearranged, 4 core jobs start running and continue until queue is empty
- Pilot remains in the system waiting for more possible jobs to enter the queue
- Inefficiencies during core reallocation to internal slots (once per slot), at job completion and when queue is empty until pilot exits

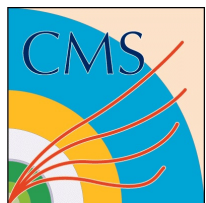


8-core pilot tests

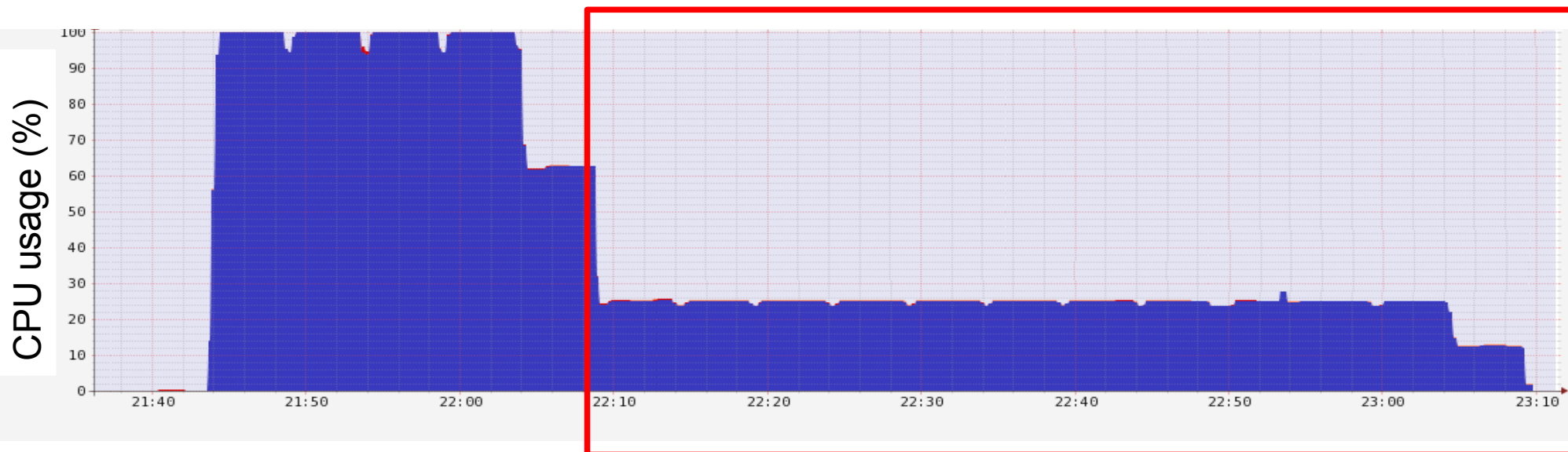


For the next test, fill the queue initially only with **4-core jobs**

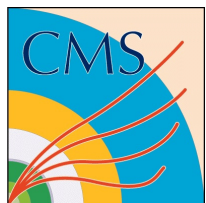
- The queue content causes each pilot to configure as **2 slots x 4 cores**
- 4 core jobs are executed, running efficiently until none remains in the queue
- Some draining inefficiency, as the last 4 core jobs are run



8-core pilot tests



- Once pilots are running, **single core jobs** were added to the queue
- Each pilot is arranged into **2 slots x 4 cores**, as it needed to pull 4 core jobs
- Single core jobs are pulled after 4 core jobs are exhausted
- In **this stage of partitionable slot development**, slots can absorb idle resources but cannot subdivide themselves at running time: **4 core slot will pull a single core job**
- Inefficiency results from this, plus some additional draining inefficiency at the end



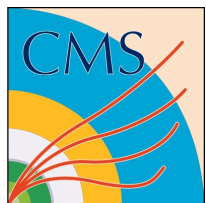
Sources of scheduling inefficiencies

Not specific to multicore pilots:

- At the **beginning**, when pilots start running until they start pulling jobs
- At **job completion**, as another job is pulled and starts to run
- At the **end of pilot life**: when jobs are all finished, pilots remain in the batch system waiting for more potential jobs to appear

Exclusive to multicore pilots

- During **internal slot reconfiguration** to capture idle cores, once initially suitable jobs are exhausted from the queue
- At this stage, as pilot **internal slots can't be subdivided** into smaller pieces at running time, jobs may use more resources than required
- **Draining inefficiency** while finishing long jobs using only a fraction of the cores



Minimize scheduling inefficiencies

Pilot behavior must be controlled to minimize inefficiencies:

- **Dynamic allocation** of internal slots **according to current queue status**, not only at start-up time: rearrange themselves or forced to exit, so that fresh pilots start with an optimal configuration
- Tune parameters controlling **pilot lifetime**: optimize relation between job duration and pilot lifetime to minimize inefficiencies at job completion, draining, etc

Further developments and ideas:

- Pilots **request a range of slots**, instead of a fixed number
- **Improved communication** between job queue, pilots, local batch systems via exchange of parameters:
 - Job features: allocated cores, walltime, memory and disk limits, etc.
 - Machine features: HS06, job slots, etc.
- **Return slots** to local batch system when unused and expected not to be used
- **Job ordering** according to expected run time: schedule shorter ones at the end to increase probability of job completion.



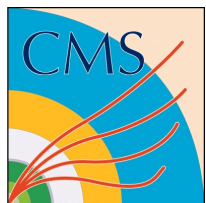
Outlook and conclusions



Outlook

Deployment schedule for CMS multicore strategy:

- Test, then implement scheduling of production single core jobs with multicore pilots.
- Multithreaded application ready by the end of the year
- Test production parallelized jobs with the new scheduling schema during 2014
- Ready for LHC restart by 2015!

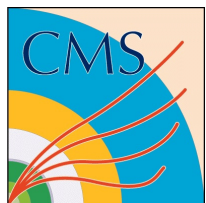


Summary

- Multicore is the future way to go
- Multicore application(s) being developed for CMS software
- Multicore principle to be used at application but also scheduling
- CMS multicore **scheduling strategy** is under development, based on the idea of **multicore pilot with dynamic allocation** of internal slots
- Principle has been tested and **it works!**
- Several sources of inefficiencies in scheduling identified
- Room for efficiency improvement in terms of both new features and fine tuning
- Test during 2014 first with single core jobs, then multicore application
- **Objective: multicore application and scheduling strategy ready for LHC restart by 2015!**

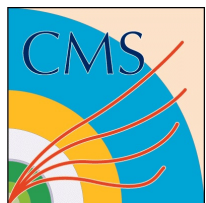


Extra slides



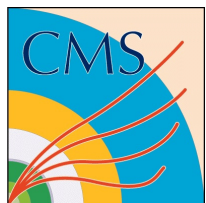
Related contributions at CHEP'13

- Evolution of the pilot infrastructure of CMS: towards a single glideinWMS pool.
<http://indico.cern.ch/contributionDisplay.py?contribId=112&sessionId=9&confId=214784>
- CMS Computing Model Evolution:
<https://indico.cern.ch/contributionDisplay.py?contribId=102&sessionId=5&confId=214784>
- CMS experience of running glideinWMS in High Availability mode:
<https://indico.cern.ch/contributionDisplay.py?contribId=114&sessionId=9&confId=214784>
- Stitched Together: Transitioning CMS to a Hierarchical Threaded Framework:
<http://indico.cern.ch/contributionDisplay.py?contribId=158&sessionId=3&confId=214784>



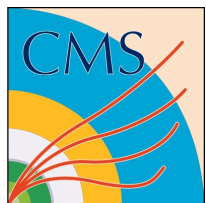
Other references

- The Need for an R&D and Upgrade Program for CMS Software and Computing, P. Elmer et al, <http://arxiv.org/abs/1308.1247>
- GlideinWMS Homepage:
<http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/doc.prd/index.html>
- HTCondor Homepage: <http://research.cs.wisc.edu/htcondor/>
- The pilot way to Grid resources using glideinWMS, I. Sfiligoi et al.
<http://dx.doi.org/10.1109/CSIE.2009.950>
- Stress command: <http://linux.die.net/man/1/stress>
- WLCG Machine/Job Features task force:
<https://twiki.cern.ch/twiki/bin/view/LCG/MachineJobFeatures>



Abstract

In the next years, processor architectures based on much larger numbers of cores will be most likely the model to continue "Moore's Law" style throughput gains. This not only results in many more jobs in parallel running the LHC Run 1 era monolithic applications. Also the memory requirements of these processes push the workernode architectures to the limit. One solution is parallelizing the application itself, through forking and memory sharing or through threaded frameworks. CMS is following all of these approaches and has a comprehensive strategy to schedule multi-core jobs on the GRID based on the glideIn WMS submission infrastructure. We will present the individual components of the strategy, from special site specific queues used during provisioning of resources and implications to scheduling; to dynamic partitioning within a single pilot to allow to transition to multi-core or whole-node scheduling on site level without disallowing single-core jobs. In this presentation, we will present the experiences made with the multi-core scheduling modes and give an outlook of further developments working towards the restart of the LHC in 2015.



Stress command

- **Stress** command can be used to test machine capabilities by causing various types of loads on a machine:
 - **CPU:** *sqrt()* of *rand()* numbers
 - **Memory:** *malloc()*, allocate memory
 - **I/O:** *sync()* flush bytes from memory to disk
 - **Disk:** *write()* files of arbitrary size to disk