CMS: Feedback and Requirements

Peter Elmer
Princeton University

Multicore/Virtualization Workshop

21 June, 2010

Overview

- In this talk I will (mostly) avoid any “show and tell” of CMS-specific details. They've been shown before and (mostly) don't interest anybody but people from CMS.
- What I'd really like to discuss with people in this workshop is:
  - The path forward to deployment of multicore applications, including:
    - The “whole node” strategy, file sizes, etc.
    - Site configuration, metrics for memory use, CPU, etc.
    - A schedule/plan for commissioning and initial deployment
  - (Bad) interactions with “virtualization strategies”, at least if they apply to the high throughput part of the system (batch worker nodes)

CMS Multicore Strategy

- Phase 0 – independent jobs on each core (+ hyperthreading?)
- Phase 1 – development and deployment of CMS framework and WM system to fork sub-processes after loading bulk conditions
  - Advantages: reduce memory needs and more flexibility for data/workflow mgmt (with only limited changes to software)
  - First steps with sites, grid providers, etc. to multicore

This year

- Phase 2 – deployment of more fine grained parallelism
  - More difficult, requires greater changes to our software
  - Impacts software development model, may require more sophisticated software development in some cases

Next Year And beyond?

Multicore CPU's – Phase 0

- The current model for HEP applications to exploit multicore CPU's is very simplistic: we exploit the "job" (and event) level parallelism and (typically) simply launch one application process per core
- The local schedulers matched this by configurations to schedule independently (and incoherently) an individual "job" per core
- Hyperthreading and pilot jobs are only minor caveats to the above. The "quantum" of work has become something of order "one batch slot/core".
- This strategy "works" in that we are able to exploit multicore CPU's reasonably efficiently on the kinds of multicore machines deployed today, but there are many consequences (next slide)

Multicore CPU's – "Phase 0" Consequences

- The simple "Phase 0" job-level parallelism has consequences:
  - The memory needs increase with each generation of CPU
  - The number of independent readers and writers (to local disk, to remote storage) increases with each generation of CPU
  - An ever increasing numbers of independent and possibly incoherent jobs running on any given piece of physical hardware.
  - Each of these running "jobs" commands an ever tinier slice of resources and do not explicitly share resources they could share
- And on top of that, there are a number of reasons to believe that this trivial parallelism won't scale efficiently on the CPU's themselves.
- Note that we have been going down the "~one job/core" route simply because we couldn't do anything else, it isn't a "natural" way to use these resources for high throughput.

Beyond “Phase 0”

- Like everyone else, we've been preparing for the deployment of multicore-aware applications: first by “forking after loading constants” then perhaps later some more fine-grained parallelism.
- The exact details and evolution going forward in some sense don't matter. There are two main aspects to note:
  - We want to explicitly manage (and optimize) the use and sharing of resources (Memory, CPU, Disk, Network) with a “quantum” greater than what we have now.
  - How we exploit this larger “quantum” of resources may evolve over time (and in fact may be different for different types of applications)
- Continuing down the road of “~one job/core” scheduling is an unnatural kludge for building future high-throughput systems

"Whole node" scheduling

- (Obviously) the one natural unit in the system is the "whole node", where that is defined as: the physical thing running one (unvirtualized) copy of the OS and sharing a set of resources (CPU, disk, network, etc.) Using "Whole node" scheduling as the resource "quantum" has obvious benefits:
  - The applications *explicitly* take over the management of the sharing of resources within the "whole node" quantum of resources.
  - It is compatible with the current "phase 0" applications as well as more explicitly multicore-aware applications (via forking, threading, etc.) optimized over time to maximize throughput
  - It also makes a number of data/workflow management optimizations possible: I/O caching, local merging, etc.
  - Sites only need to defend the whole node, not individual processes.
  - A move to a proper "whole node" accounting for CPU/memory use, etc. recognizes the role of the OS in optimizing access to resources

CMS Multicore Deployment

- CMS would like proceed with the commissioning and deployment of "multicore-aware" applications in the next six months (i.e. by end of 2010), with a strong preference for commissioning "whole node" scheduling (and accounting) in the process.
- We do not see a use case requiring us to ask for (or limit ourselves to) an arbitrary number of cores. We believe that we can exploit an entire node efficiently, regardless of the number of cores it has. There is no reason to subdivide it.
- One of the things we would like to discuss at this workshop is whether this is coherent with the plans of the other experiments and whether a single statement can be made (e.g. to the GDB, to the sites, to CERN IT, etc.)

Memory Use Accounting

- Already today (with "~one job/core") we have some large problems with memory use accounting. Typically people and tools under Linux use the VSIZE and/or RSS of processes to estimate memory use. Both have problems:
  - VSIZE poorly reflects the memory pages actually used by a process, problem becomes very obvious for 64bit applications
  - RSS poorly reflects any memory sharing (explicit or accidental) which is happening
- Accurate memory accounting is needed to defend the hardware (sysadmins), make future purchases (sysadmins) and estimate what can be done with the software (developers)
- The linux virtual memory system nowadays provides (some) better metrics, such as PSS, which we should adopt. These are useful both for 64bit application deployment and for "whole node" scheduling.
- See notes and PSS proposal: http://elmer.web.cern.ch/elmer/memory.html

How does "whole node" affect Data Mgmt?

- Changes the system working point by an "order of magnitude": the number of schedulable running "entities" within the system will drop by a factor of 4-8 (today), more later, and will become approximately constant in each site going forward
- The resources (local disk, memory, etc.) managed by a (pilot) job start to become significant, providing many opportunities for optimization:
  - Stage-in to and mgmt of local disk, suddenly we are "memory rich"
  - coordinated/coherent I/O access across activity "node"
  - reduced external connections&streams
  - local output merging (or direct write of combined file)
  - "backfill" CPU intensive activities if necessary

File sizes and transfers

- Hand in hand with a change to “whole node” scheduling it is important to make sure we can scale file sizes at the same time
- We removed the 2GB limit some years ago, but in practice the reliability of the system is such that we've not gone very much beyond that.
- The limitations come from the fact that any error during file transfers requires the transfer to restart from scratch. This is seen as the main bottleneck preventing us from increasing significantly the file size.
- CMS Request: we would like the appropriate changes to be made to FTS and/or the storage implementations to avoid restarting transfers from scratch on errors (when possible)

Virtualization

- A side note on virtualization, as there seems to be a disconnect here:
- **CMS (as an experiment) is not asking for the general introduction of virtualization on WN's. In fact we are specifically requesting that if we say "we can run on the actual OS version of the node" that we be allowed to do that, without any virtualization arbitrarily introduced**
- Clearly there are situations (e.g. OS version migration) where we might not be able to do that, and virtualization can help us handle this better than in the past. If we don't *need* it, however, we shouldn't have to pay any cost for it.
- (False) tradeoff between throughput and adminstration
  - Both are relevant for WN's, but not mutually exclusive
  - Virtualization of services is more about administration
- Clearly if we don't own the resources (e.g. commericial cloud computing), the choice is not necessarily ours. But if we do own them, we should engineer it properly to avoid false tradeoffs.

Summary

- CMS would like to proceed with commissioning/deployment of multicore-aware applications in the next 6 months (by end 2010).
- We propose moving to "whole node" scheduling as part of this commissioning and would like to discuss this here, hopefully resulting in some sort of common statement.
- We would like to see improvements to the storage systems and FTS to permit larger files, in particular by resolving the problems with from-scratch restarts after any errors
- We would like to move beyond VSIZE/RSS for memory accounting (PSS proposal, "whole node" accounting needed)
- We see virtualization as appropriate for (most) services, but **not** as a permanent aspect of high-throughput WN's