CMS Multi-core Aware Applications

FNAL Concurrency Workshop

Elizabeth Sexton-Kennedy, David Lange, Chris Jones, Eric Vaandering, and Jose Hernández, and Pete Elmer

Introduction

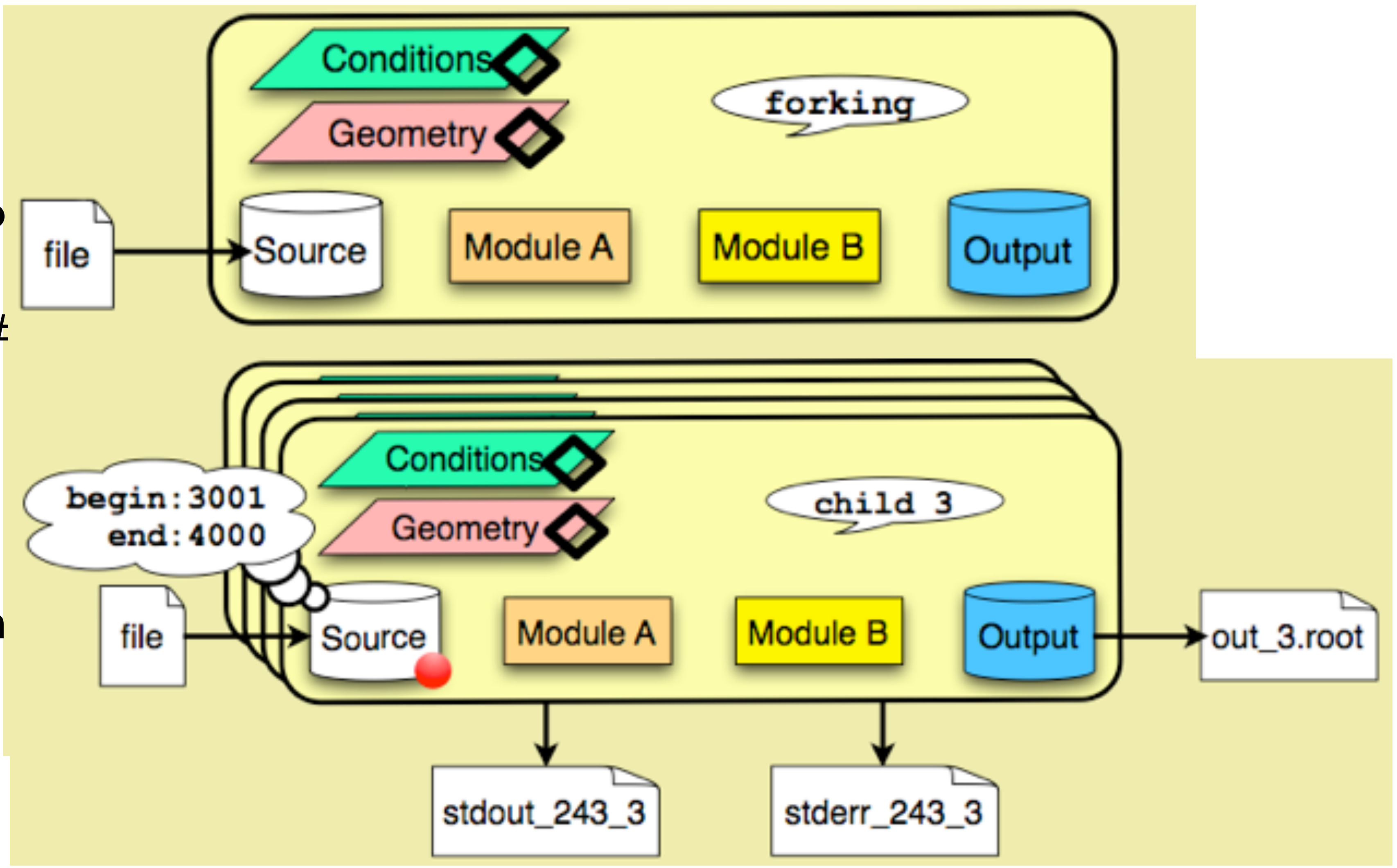
- This is a current status and plans talk meant to cover a broad range of topics without going into great detail about any of them. Topics include:
  - Targets for Concurrency
  - Overview of our Application and it's Performance
  - The Challenge for 2012
  - Whole Node Deployment
  - On Going R&D
  - Longer Term Plans

Targets for Concurrency

- We are not developing/maintaining independent solutions to final ntuple/fitting framework.
  - We will continue to rely on solutions in root/RooFit (etc)
  - User applications are too chaotic. Some analysis is CPU bound, some is Root-IO bound.
  - No easy way to deal with user-produced histograms in forking
  - Analysis jobs can also write arbitrary files
- Thus the CMS focus for R&D on application concurrency is on central/user production activities running on Tier0/Tier1/Tier2 facilities
  - Reconstruction, Simulation
  - Skimming, User jobs running over AOD

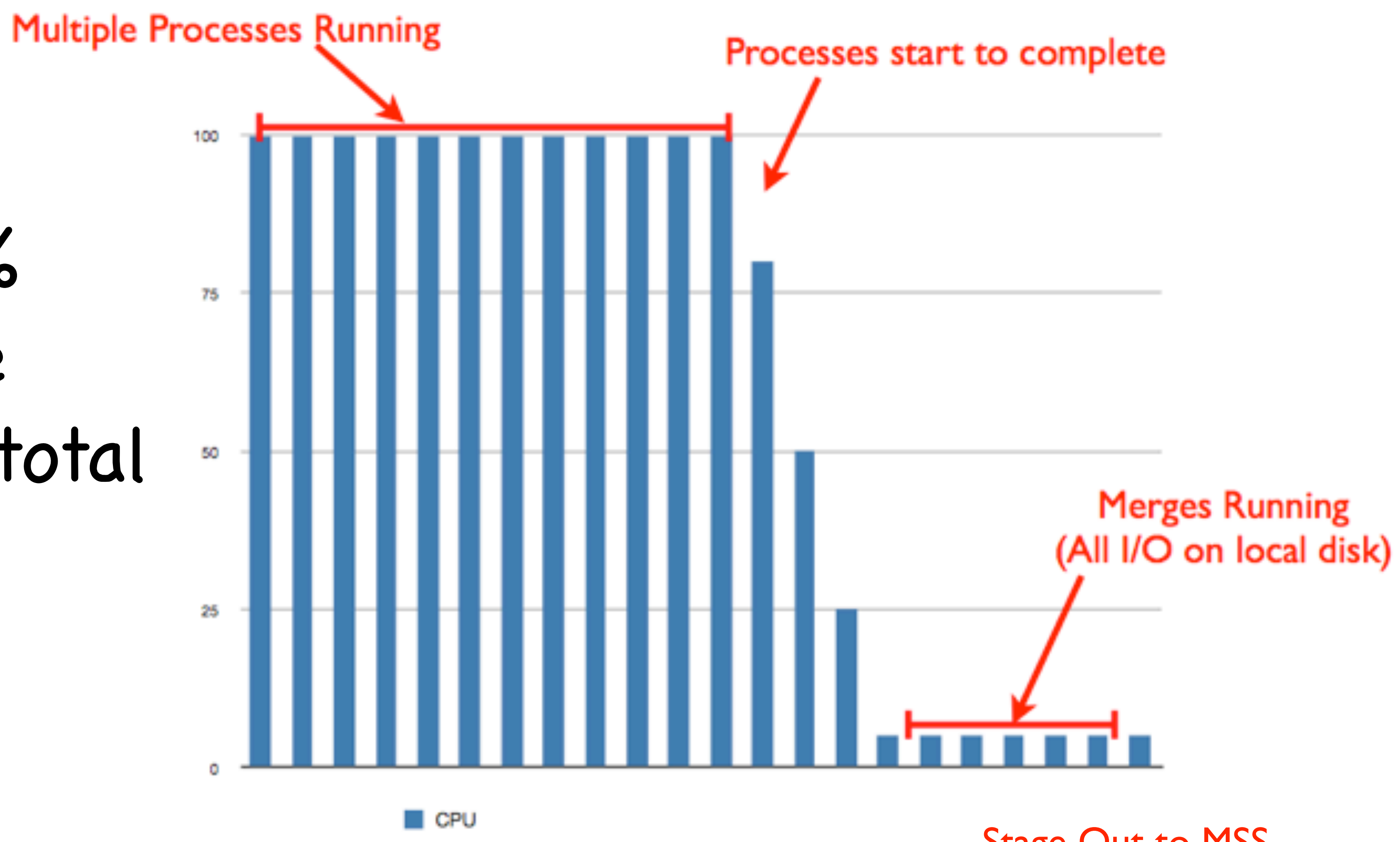
Current Application Overview

- A Fork and Copy On Write application
- The parent process
  - Reads configuration and loads modules. The WMDM system sets configuration of how many children and # events/child to use.
  - 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 then forks
- The child process
  - Redirects stdout and stderr to own files whose names contain parent PID and child #
  - Send messages to modules saying process is child X
  - Sources calculate their event ranges to process and re-open the file
  - Process events in child's start/ end range normally

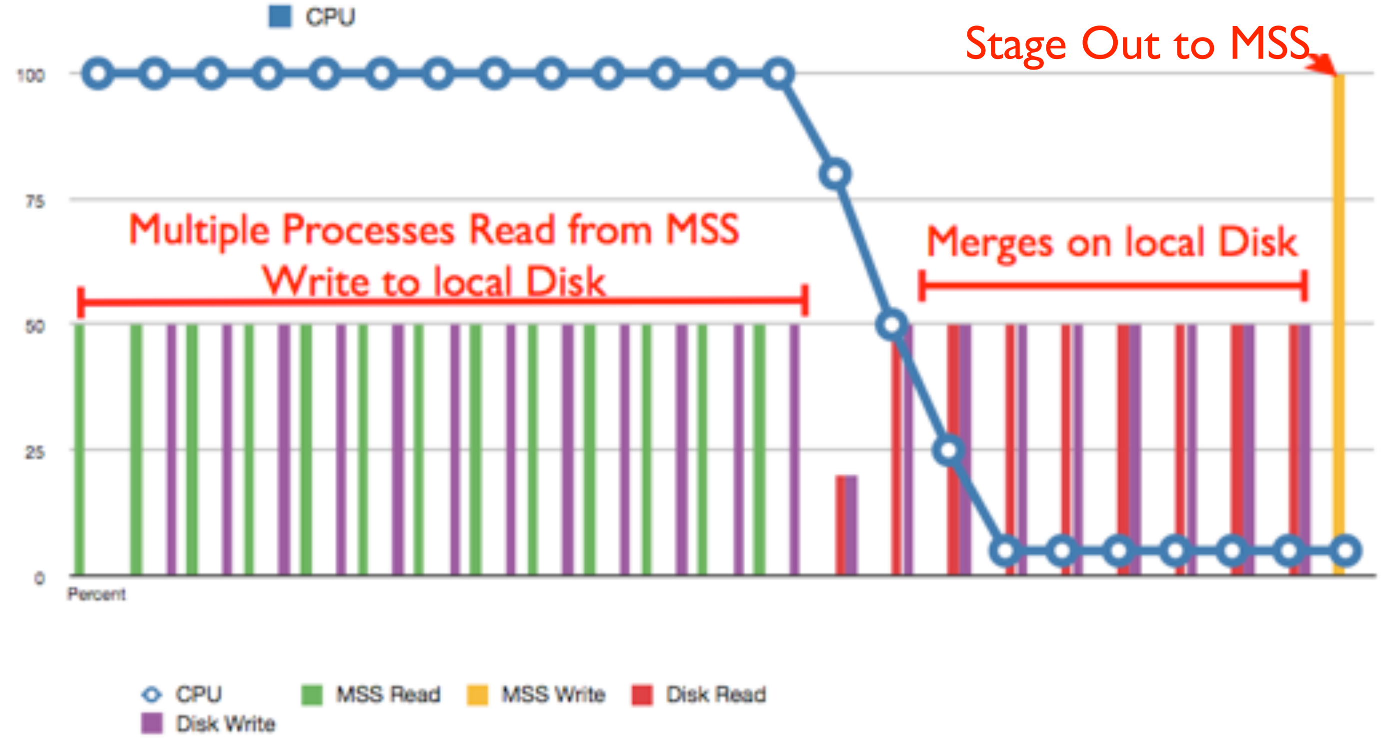


Performance Profiles

- CPU usage in % vs. time (merge time is <1% of total job time)



- I/O and CPU in % vs. time



The Challenge for 2012

- Both memory and CPU of our reconstruction application scale strongly with pileup.
  - Memory is currently the most significant limitation.
  - Recent technical and physics driven changes have already achieved significant improvements (eg 30+% reduction in CPU/memory) over 2011 application
- Memory usage target is 2GB/core*:
  - Easy if LHC bunch spacing is 25ns
  - Hard if 50ns (eg, 30+ pileup events at start of fill).
- For 50ns case, deployment of forking should save 20% to 30% of PSS according to standalone measurements with current software on data from recent high-pileup fill
  - This reduction should be sufficient to bring us below our memory budget

Whole Node Scheduling

- CMS approach for multi-core processing scheduling
- All cores of a node get assigned to a multi-core process application
- We'll be billed for the whole node and all the capability, so we need to make use of everything efficiently
- Need to carefully evaluate the multiprocessing overhead in terms of sufficient use of all cores, merging, etc
- Over the summer we commissioned whole-node queues at all 7 CMS Tier-1s
  - Few WNs behind the whole-node queues
  - FNAL: 25 nodes (8 cores each)
- Whole-node queues accessible via the glideinWMS factory at CERN

Testing Status

- Multi-core workflow (reconstruction) tested with our workflow management system at all Tier-1s
  - So far only functionality testing. Workflow runs fine.
- Now developing the relevant metrics to monitor multi-core workflow performance
  - Monitor memory utilization (RSS, VMEM, PSS)
  - Monitor job CPU inefficiencies (initialization, processing spread, merging)

Testing Status

- For testing on large scale, job reports are aggregated by the WMDM system. They contain:
  - Start/stop times of all steps (startup, processing, merging, stageout)
  - Memory metrics
  - Processing spread inefficiency
- Compare memory, wall/cpu times of N-core job vs N single-core jobs at many sites. Compare workflow turnaround time.
- This strategy maybe most useful at the tier0 where CMS owns whole resources and where memory is most tight. Testing will begin there in the beginning of next year. Will there be buy in from other experiments?

Ongoing R&D

- Constraints of ongoing R&D:
  - Current framework functioning well. “User interface” can not undergo any significant change for the foreseeable future (e.g., 2020) unless changes bring significant improvement that users care about (e.g., not just a technical change such as concurrency support).
  - Adiabatic changes in interface can be made.
    - For example: We do want modules to declare what they require in addition to what they produce.
  - Mostly migrations and algorithm modifications for the sake of technical performance (while leaving physics performance unchanged) are to be done centrally.
  - Luminosity block synchronization is still a requirement.

Ongoing R&D

- Ongoing efforts:
  - Migration to gcc 4.6.x on SLC5 for 2012
  - Prototyping framework multithreaded support [talk by Chris Jones]
  - Prototyping refactored algorithms for fine-grained parallelization
  - Prototyping refactored algorithms to allow gcc to vectorize
  - I/O reduction and persistent object model optimization
  - Continue to reduce the reliance on strings
  - Targeted memory/CPU reduction for high-pileup data taking
  - "Any Data Anywhere" which exploits xrootd technology to remove the constraint that jobs MUST go to the site that stores the data.

Longer term plans (5+ years)

- Goal: Scaling current throughput up to future commodity computing platforms.
  - Forking may be sufficient. Otherwise module level and finegrained parallelization managed by framework.
  - I/O bottleneck? Memory bottleneck?
  - Do not envision exploiting GPUs on this timescale
  - Focus is on scaling production jobs for future hardware and not on enabling rapid job turnaround
- View upcoming shutdown in 2013 as only opportunity to introduce significant changes on 5 year timescale.
  - Changes must be largely behind the scenes. No manpower/interest in adapting algorithms to framework changes
  - Many algorithms essentially frozen. Thus the burden for algorithm changes is on core team. (this is both good and bad)
  - Do not expect major changes to coding model for "physicists" beyond potential requirements for thread safety.