CMSC 714
Lecture 21
Batch Scheduling

Alan Sussman

Notes

- All readings posted, except last day
- Group Project presentations start Nov. 29
  - Need groups to volunteer to go early
- Today, first any questions/comments on BOINC (and Condor)

Backfilling on IBM SP2

- For space sharing on clusters, distributed memory parallel machines
- Default policy is FCFS in queuing system
  - causes fragmentation, so can do better by allowing later jobs in queue to bypass first job that can't run yet because not enough nodes currently available
  - have to ensure not starving large jobs, and helps to be able to predict when a job will run
  - works better if users give runtime estimate for their job, in addition to number of nodes desired
    - used in EASY scheduler algorithm from Argonne
    - EASY only allows jobs to move up in queue if don't delay the *first* queued job
    - paper shows that can lead to unbounded delay if a job has unbounded execution time (doesn't delay first job in queue, but can delay later jobs)

Backfilling

- Conservative backfilling
  - backfill, but don't delay *any* previous job in queue
  - scheduling decision can be made when job is submitted (later arriving jobs can't affect the decision)
  - no starvation of large jobs
  - problem occurs when job terminates early
    - best solution is to then *compress* schedule, instead of doing nothing, or redoing backfilling algorithm, which could cause a job to run *later*
- Evaluation with workload model derived from real system traces shows that conservative and EASY backfilling perform similarly (except at very high workloads), and same on real traces with user runtime estimates (or real execution times)
- Results also show that user runtime estimates are very inaccurate, but conservative algorithm works better than EASY in that case

Symbiotic Space Sharing

- Idea is to go beyond space sharing, to allow multiple programs to share a node in a cluster/parallel machine, but avoid contention for node resources by mixing jobs with different resource requirements
  - goal is to increase overall system throughput
  - even more relevant now with multi-core nodes
- Even on space shared systems, there are shared resources
  - network bandwidth
  - file system (e.g., DataStar's GPFS)
- Hard part is characterizing an application's resource usage, and how it may interfere with other apps running on same node(s)
- Study divides resources into memory and I/O
  - memory includes all cache levels
  - I/O to both local file system and to GPFS

Symbiotic Space Sharing

- Benchmarks used to stress different parts of 1-node system
  - GUPS for random access to memory
  - Stream for memory bandwidth, and cache effects
  - EP for compute bound (baseline)
  - results show slowdowns from sharing – pretty much linear in amount of sharing
  - IOBench used for disk read/write tests, also shows slowdowns mostly linear, but much bigger than for memory benchmarks
- For parallel codes, thorough study of effects of different job mixes sort of shows that sharing across benchmarks that have different bottleneck resources works well
  - compare NAS Parallel Benchmarks running against each other and against EP and IOBench
  - by running on more/fewer nodes to decrease/increase contention (for same total number of processes)
- Last topic is identifying job characteristics so symbiotic scheduler can decide what to do
  - either use past history of a job (with instrumentation), or maybe use hardware counters
  - either way, need a database of past runs (or current run for a really long running job)