Dynamic Fractional Resource Scheduling for HPC Workloads

Mark Stillwell[1] Frédéric Vivien[2] Henri Casanova[1]

[1]Department of Information and Computer Sciences
University of Hawai'i at Mānoa

[2]INRIA, France

The 24th IEEE International Parallel and Distributed Processing Symposium
April 19–23, 2010
Atlanta, USA

High Performance Computing

- Today, HPC usually means using clusters
  - Homogeneous nodes connected via high speed network
  - These are ubiquitous
  - But large ones are expensive
- Users submit requests to run jobs
  - Running jobs are made up of nearly identical tasks
    - The number of tasks is generally specified by the user
    - Tasks in a job are nearly identical
    - Tasks can block while communicating with each other
    - Most systems put each task on a dedicated node
    - Many jobs are serial, a few require all of the system nodes
  - Jobs are temporary
    - The user wants a final result
    - Quick turnaround relative to runtime is desired
  - Jobs may have to wait until resources are available to start
- The assignment of resources to jobs is called scheduling

Current HPC Scheduling Approaches

- Batch Scheduling, which no one likes
  - Usually FCFS with backfilling
  - Backfilling needs (unreliable) compute time estimates
  - Unbounded wait times
  - Inefficient use of nodes/resources
- Gang Scheduling, which no one uses
  - Globally coordinated time sharing
  - Complicated and slow
  - Memory pressure a concern
  - Large granularity limits improvement over batch scheduling

Our Proposal

- Use virtual machine technology.
  - Multiple tasks on one node
  - Sharing of fractional resources
    - Similar to preemption
  - Performance isolation
- Define a run-time computable metric that captures notions of performance and fairness.
- Design heuristics that allocate resources to jobs while explicitly trying to achieve high ratings by our metric.

Requirements, Needs, and Yield

- Tasks have memory requirements and CPU needs
- All tasks of a job have the same requirements and needs
- For a task to be placed on a node there must be memory available at least equal to its requirements
- A task can be allocated less CPU than its need, and the ratio of the allocation to the need is the yield
- All tasks of a job must have the same yield, so we can also speak of the yield of a job
- The yield of a job is the rate at which it progresses toward completion relative to the rate if it were run on a dedicated system

Stretch

- Our goal: minimize maximum stretch (aka slowdown)
  - Stretch: the time a job spends in the system divided by the time that would be spent in a dedicated system [Bender et al., 1998]
  - Popular to quantify schedule quality post-mortem
  - Not generally used to make scheduling decisions
  - Runtime computation requires (unreliable) user estimates.
  - Minimizing average stretch prone to starvation
  - Minimizing maximum stretch captures notions of *both* performance and fairness.

Approach

- Job arrival/completion times are not known in advance
- We avoid the use of runtime estimates
- Instead we focus on maximizing minimum yield
- Similar, but not the same, as minimizing maximum stretch

Task Placement Heuristics

We apply task placement heuristics studied in our previous work [Stillwell et al., 2008, Stillwell et al., 2009]

- **Greedy Task Placement** – Incremental, puts each task on the node with the lowest computational load on which it can fit without violating memory constraints
- **MCB Task Placement** – Global, iteratively applies multi-capacity (vector) bin-packing heuristics during a binary search for the maximized minimum yield
  - Much better placement than greedy
  - Can cause lots of migration
- But what if the system is oversubscribed?
  - Need a priority function to decide which jobs to run

Priority Function?

- Virtual Time: The subjective time experienced by a job
- First Idea: $\frac{1}{\text{VIRTUAL TIME}}$
  - Informed by ideas about fairness
  - Lead to good results
  - But theoretically prone to starvation
- Second Idea: $\frac{\text{FLOW TIME}}{\text{VIRTUAL TIME}}$
  - Addresses starvation problem
  - But lead to poor performance
- Third Idea: $\frac{\text{FLOW TIME}}{(\text{VIRTUAL TIME})^2}$
  - Combines idea #1 and idea #2
  - Addresses starvation
  - Performs about the same as first priority function

Use of Priority

- By Greedy
  - **GreedyP** – Greedily schedule tasks, and suspend lower-priority tasks if necessary to run higher-priority tasks
  - **GreedyPM** – Like **GreedyP**, but can also migrate tasks instead of suspending them
- by MCB
  - If no valid solution can be found for any yield value, remove the lowest priority task and try again

Resource Allocation

- Once tasks are placed on nodes we iteratively maximize the minimum yield
- Based on network resource allocation ideas about fairness
- Easy to compute and slightly better than maximizing average yield

When to apply Heuristics

We consider a number of different options:

- Job Submission – heuristics can use greedy or bin packing approaches
- Job Completion – as above, can help with throughput when there are lots of short running jobs
- Periodically – some heuristics periodically apply vector packing to improve overall job placement

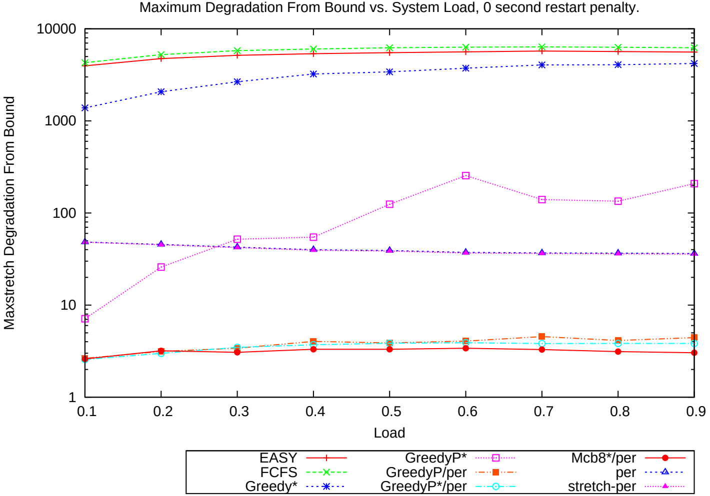
MCB-Stretch Algorithm

- Like MCB, but tries to minimize maximum stretch
- Requires knowledge of time until next rescheduling period, uses current and estimated future stretch
- Second phase focuses on iteratively minimizing the maximum stretch

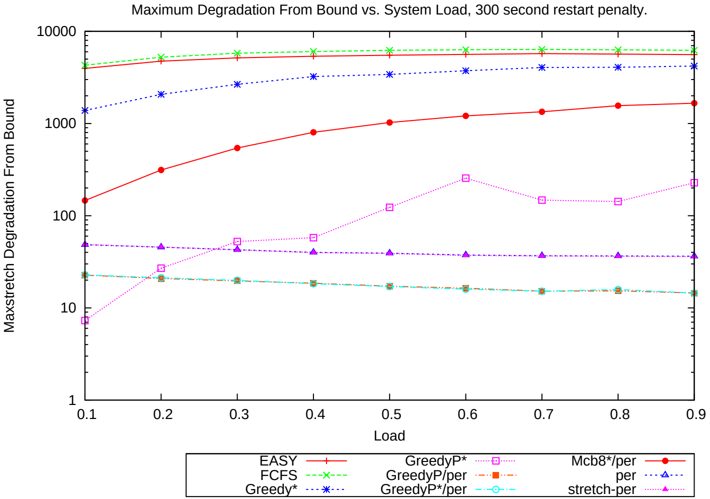
Methodology

- Experiments conducted using discrete event simulator
- Mix of synthetic and real trace data
- Ran experiments with and without migration penalties
- Periodic approaches use a 600 second (10 minute) period
- Absolute bound on max stretch computed for each instance
- Performance comparison based on max stretch degradation from bound

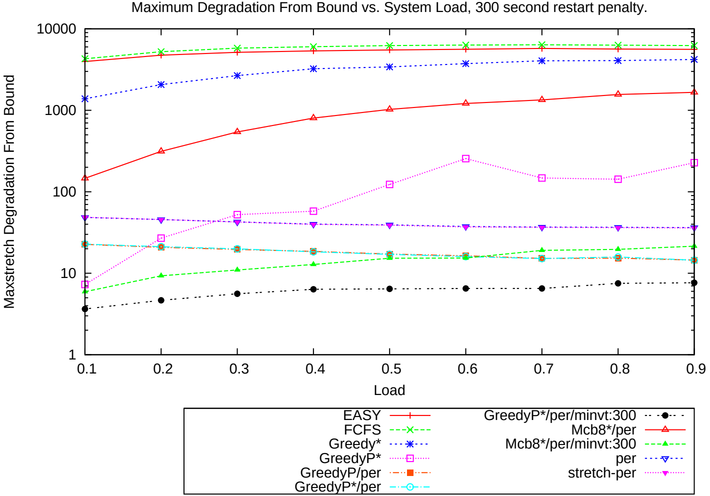
Max Stretch Degradation vs. Load, No Migration Cost



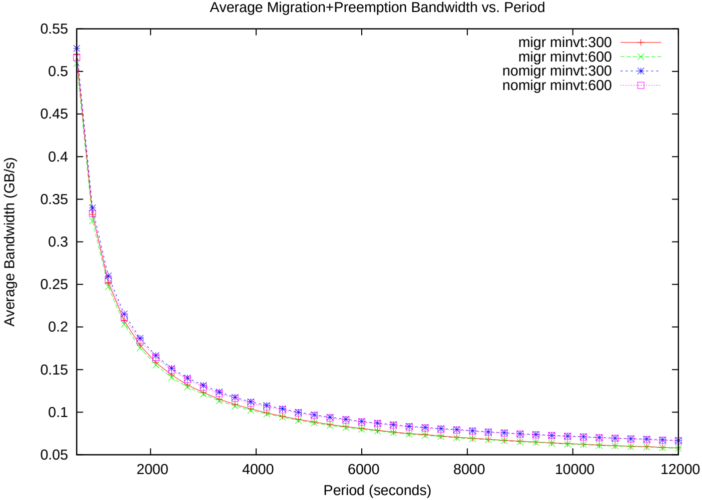
Max Stretch Degradation vs. Load, 5 minute penalty



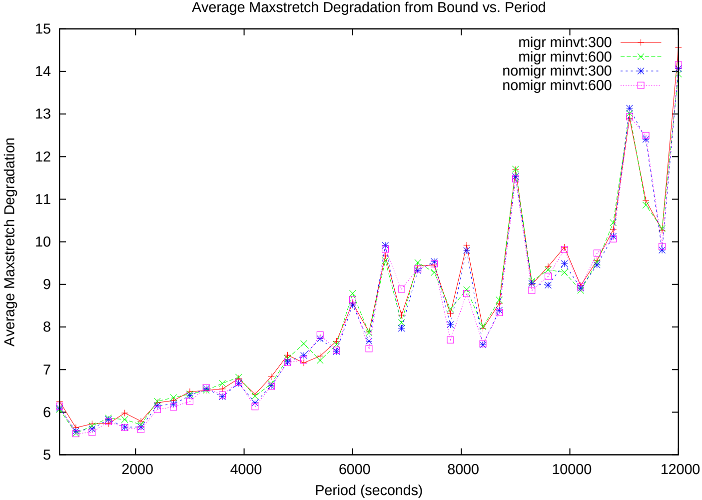
Max Stretch Degradation vs. Load, 5 minute penalty



Bandwidth vs. Period



Max Stretch Degradation vs. Period



Conclusions

- DFRS approaches can significantly outperform traditional approaches
- Aggressive repacking can lead to much better resource allocations
  - But also to heavy migration costs
- A combination of opportunistic greedy scheduling, with limited periodic repacking has the best average case performance across all load levels
- Bandwidth costs can be reduced by extending the period without much loss of performance
- Greedy migration is not that useful
- Attempting to maximize the minimum yield does about the same as trying to minimize the maximum stretch

Summary

- We have proposed a novel approach to job scheduling on clusters, Dynamic Fractional Resource Scheduling, that makes use of modern virtual machine technology and seeks to optimize a runtime-computable, user-centric measure of performance called the minimum yield
- Our approach avoids the use of unreliable runtime estimates
- This approach has the potential to lead to order-of-magnitude improvements in performance over current technology
- Overhead costs from migration are manageable

References I

- Bender, M. A., Chakrabarti, S., and Muthukrishnan, S. (1998).
  Flow and Stretch Metrics for Scheduling Continuous Job Streams.
  In *Proc. of the 9th ACM-SIAM Symp. On Discrete Algorithms*, pages 270–279.

- Stillwell, M., Shanzenbach, D., Vivien, F., and Casanova, H. (2008).
  Resource Allocation using Virtual Clusters.
  Technical Report ICS2008-09-01, University of Hawai'i at Mānoa Department of Information and Computer Sciences.

References II

Stillwell, M., Shanzenbach, D., Vivien, F., and Casanova, H. (2009).
Resource Allocation using Virtual Clusters.
In *CCGrid*, pages 260–267. IEEE.

Stillwell, M., Vivien, F., and Casanova, H. (2010).
Dynamic fractional resource scheduling for HPC workloads.

In *IPDPS*.
to appear.