Dynamic Fractional Resource Scheduling vs. Batch Scheduling

Mark Stillwell, Frédéric Vivien, Henri Casanova

INRIA, Université de Lyon, LIP

4th Scheduling in Aussois Workshop
May 29 – June 1, 2011

HPC Scheduling Problem Overview

- We assume a cluster of homogeneous nodes dedicated to processing user jobs
- Users can submit requests at arbitrary times
- Running jobs are made up of nearly identical tasks
  - The number of tasks is specified by the user
  - Tasks may need to block while communicating
- Jobs are temporary
  - May have to wait until resources are available to start
  - Runtime is not known in advance

Current HPC Scheduling Approaches

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

Our Proposal

- Use virtual machine technology.
  - Multiple tasks on one node
  - Sharing of fractional resources
  - 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 rigid requirements and fluid needs
  - All tasks of a job have the same requirements and needs
- For a task to be placed on a node there must be rigid resources available at least equal to its requirements
- A task can be allocated less of a fluid resource than its need, but performance is degraded
- The yield of a job is a value between 0 and 1; tasks are allocated an amount of each fluid resource equal to the job yield multiplied by the need in that resource

The yield of a job gives its performance relative to if it were run on a dedicated system.

Requirements, Needs, and Yield

- Tasks have rigid requirements and fluid needs
  - All tasks of a job have the same requirements and needs
- For a task to be placed on a node there must be rigid resources available at least equal to its requirements
- A task can be allocated less of a fluid resource than its need, but performance is degraded
- The yield of a job is a value between 0 and 1; tasks are allocated an amount of each fluid resource equal to the job yield multiplied by the need in that resource

The yield of a job gives its performance relative to if it were run on a dedicated system.

Stretch

- 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 [Legrand et al., 2008]

Optimal Lower Bound

- Given a clairvoyant scenario and infinite system memory, can compute a max-stretch lower bound in P-time
- Bound may not be achievable in practice
- Useful for comparing the performance of scheduling algorithms

Our Approach

- Basic idea: Consider an on-line maximum stretch minimization problem instance as a sequence of off-line minimum yield maximization problem instances
- Need heuristics to map tasks to nodes
- Need additional heuristics to allocate resources
- Need to decide when to apply heuristics

Task Placement Heuristics

We apply task placement heuristics studied for the off-line problem [Stillwell et al., 2010]:

- **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
  - Achieves higher minimum yield values than Greedy
  - Can *potentially* 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

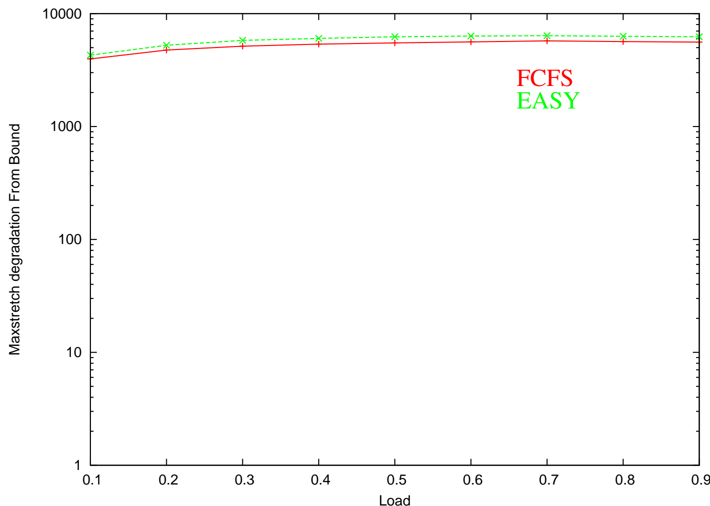
Methodology

- 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

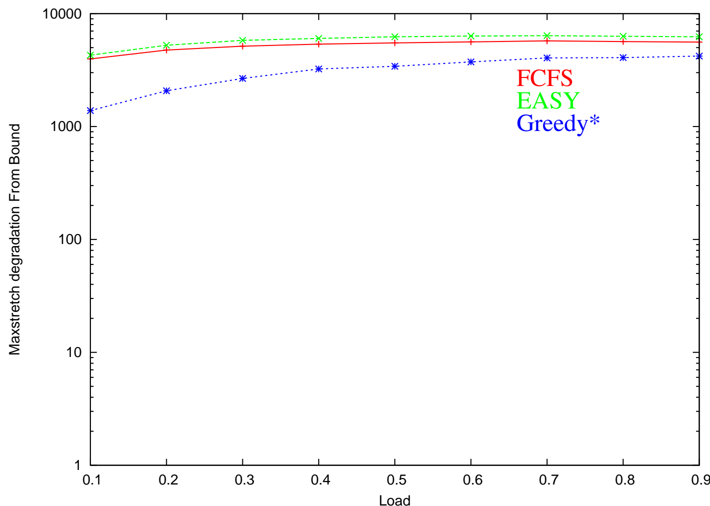
Batch Scheduling Algorithms

- FCFS – Allocates nodes equal to the number of tasks to jobs on a first-come-first-served basis.
- EASY – Only makes a reservation for the first job in the queue. Otherwise allocates nodes to the first job in the queue that can run with the current number of available nodes. Requires (unreliable) user-supplied run-time estimates to make reservations.

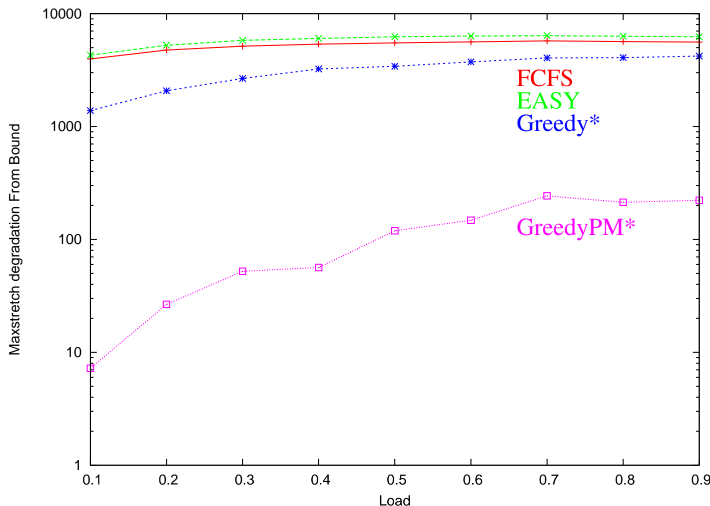
Max Stretch Degradation vs. Load, No Migration Cost



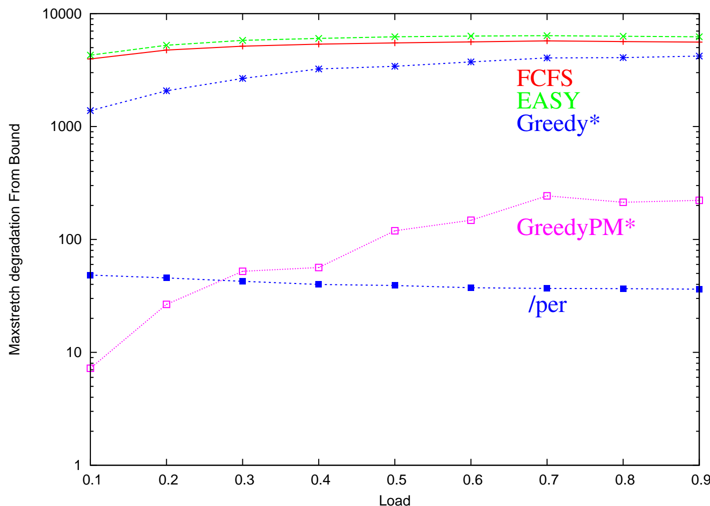
Max Stretch Degradation vs. Load, No Migration Cost



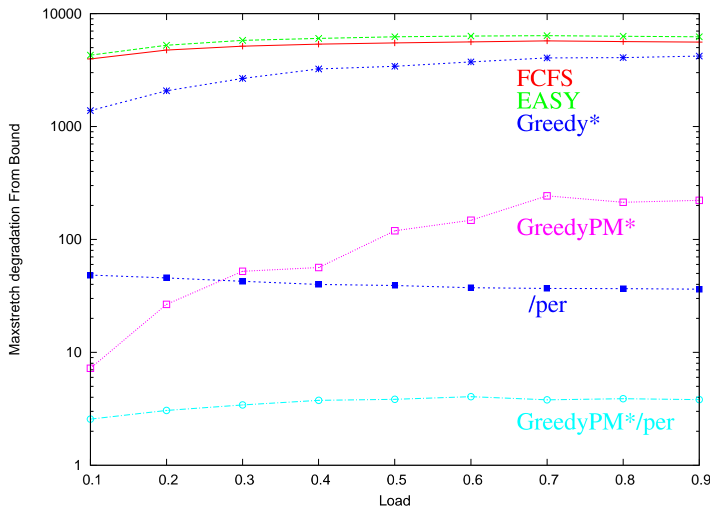
Max Stretch Degradation vs. Load, No Migration Cost



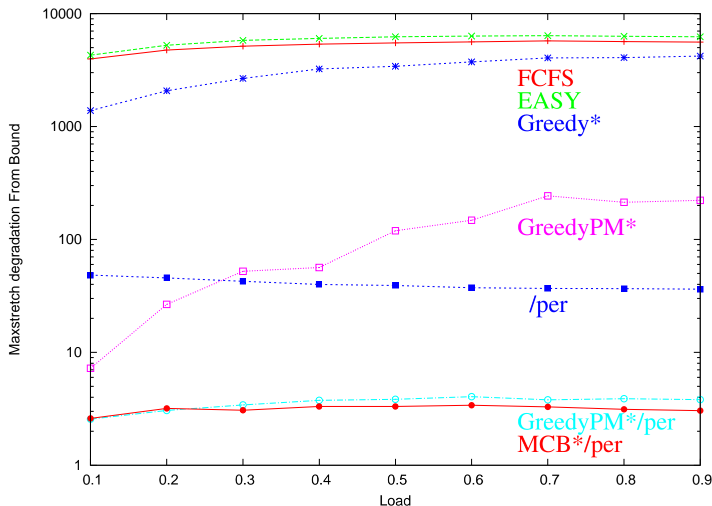
Max Stretch Degradation vs. Load, No Migration Cost



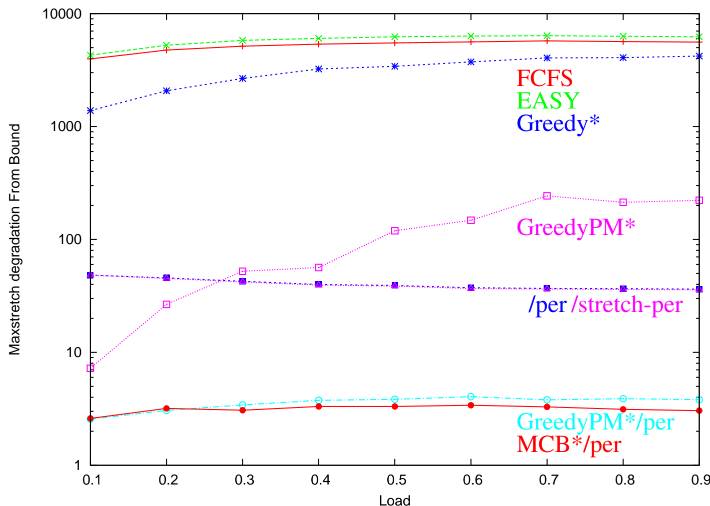
Max Stretch Degradation vs. Load, No Migration Cost



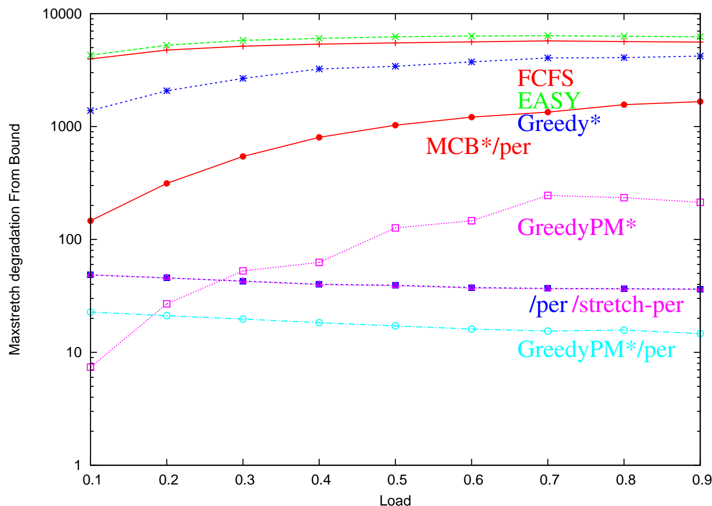
Max Stretch Degradation vs. Load, No Migration Cost



Max Stretch Degradation vs. Load, No Migration Cost



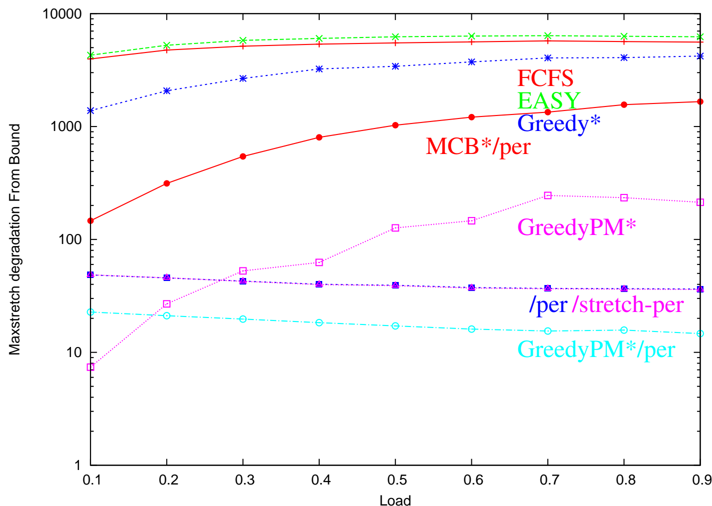
Max Stretch Degradation vs. Load, 5 minute penalty



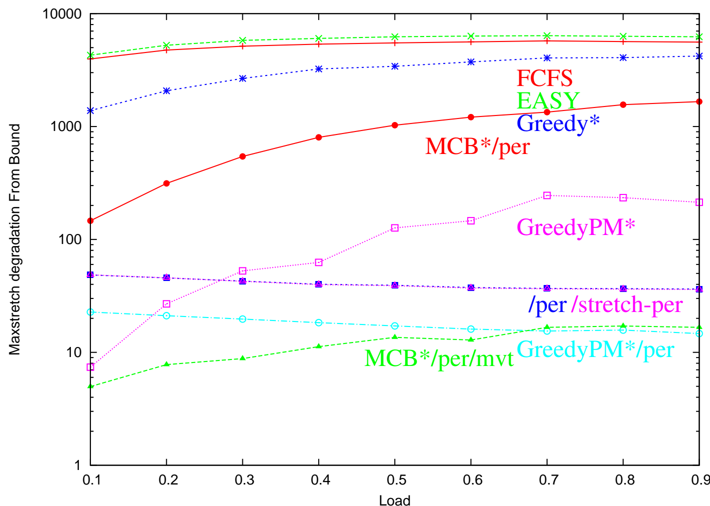
Limiting Migration

- Short-running jobs suffer a greater penalty to stretch from preemption/migration
- No way to tell short from long running jobs a-priori
- We do know the subjective time experienced by a job
- The minvt parameter specifies the minimum virtual time for a job before it can be migrated
- Does not affect preemption due to priority
- We tried 300 seconds and 600 seconds, 600 performed slightly better

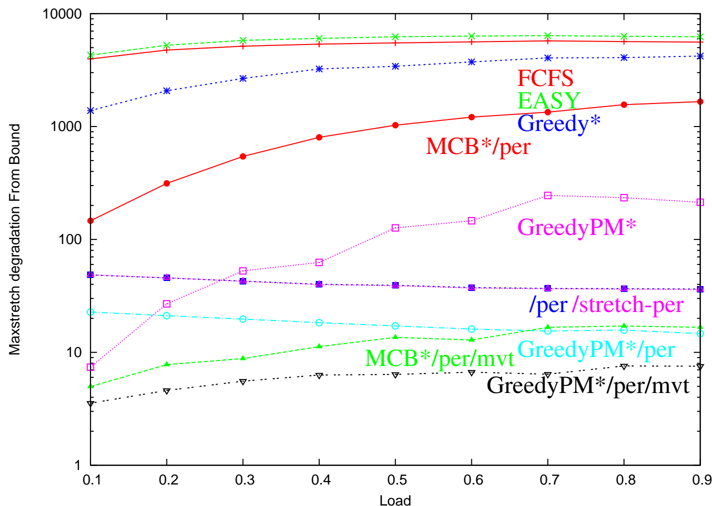
Max Stretch Degradation vs. Load, 5 minute penalty



Max Stretch Degradation vs. Load, 5 minute penalty



Max Stretch Degradation vs. Load, 5 minute penalty



Bandwidth Utilization

Preemption and migration bandwidth costs for selected algorithms. Average and maximum values over scaled synthetic traces with load ≥ 0.7

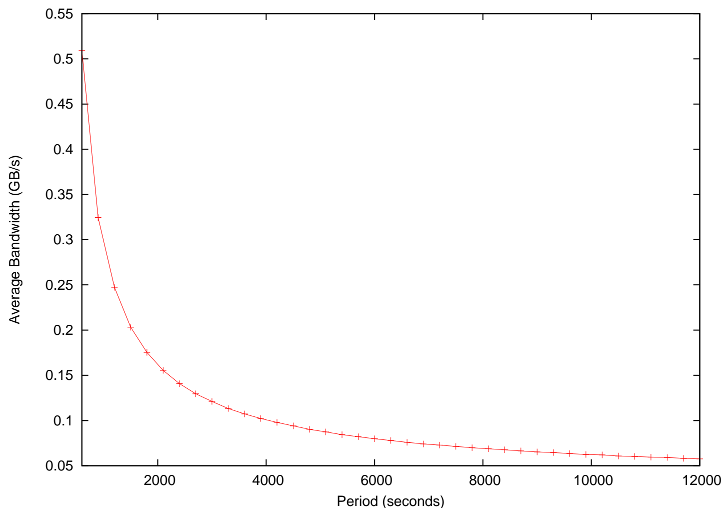
Algorithm	(GB / sec)			
	pmtn		mig	
	avg.	max	avg.	max
GreedyPM *	0.03	0.07	0.02	0.05
GreedyPM */per	0.56	1.37	0.29	0.66
GreedyPM */per/MVT	0.54	1.34	0.26	0.62
MCB */per/MVT	0.54	1.11	0.56	1.53
/per/MVT	0.49	1.08	0.19	0.58
/stretch-per/MVT	0.28	0.64	0.37	0.78

Bandwidth Utilization

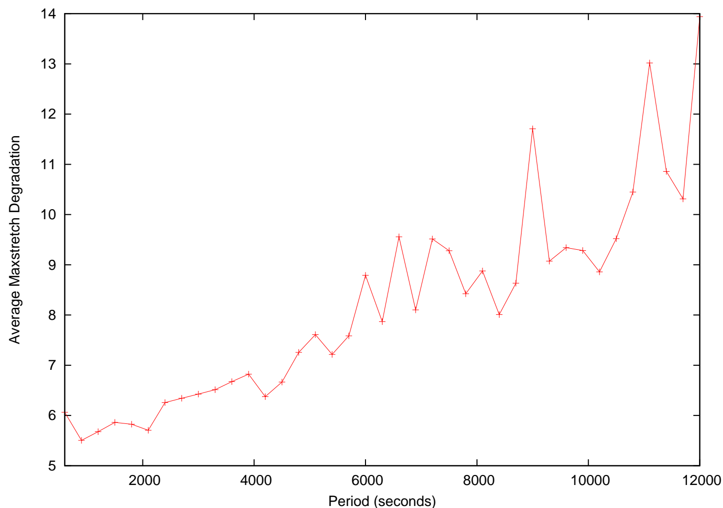
Preemption and migration bandwidth costs for selected algorithms. Average and maximum values over scaled synthetic traces with load ≥ 0.7

Algorithm	(GB / sec)			
	pmtn		mig	
	avg.	max	avg.	max
GreedyPM *	0.03	0.07	0.02	0.05
GreedyPM */per	0.56	1.37	0.29	0.66
GreedyPM */per/MVT	0.54	1.34	0.26	0.62
MCB */per/MVT	0.54	1.11	0.56	1.53
/per/MVT	0.49	1.08	0.19	0.58
/stretch-per/MVT	0.28	0.64	0.37	0.78

Bandwidth vs. Period



Max Stretch Degradation vs. Period



Conclusions

- DFRS algorithms are capable of widely outperforming traditional approaches, even assuming a heavy penalty for migration
- A variety of approaches need to be combined in order to achieve the best results
- Bandwidth costs are reasonable, and can be further reduced without a significant performance penalty by choosing an appropriate rescheduling period from a broad range

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

Future Work

- Complete development of practical implementation
- Heterogeneous Clusters
- New or arbitrary optimization targets
  - Energy or other costs
  - More formal definitions of fairness

References I

Bender, M. A., Chakrabarti, S., and Muthukrishnan, S. (1998).
Flow and stretch metrics for scheduling continuous job streams.
In *Proceedings of the 9th Annual ACM-SIAM Symposium On Discrete Algorithms*, pages 270–279.

Legrand, A., Su, A., and Vivien, F. (2008).
Minimizing the stretch when scheduling flows of divisible requests.
Journal of Scheduling, 11(5):381–404.

References II

Stillwell, M., Schanzenbach, D., Vivien, F., and Casanova, H. (2010).
Resource allocation algorithms for virtualized service hosting platforms.
Journal of Parallel and Distributed Computing, 70(9):962–974.