IMPROVING MAPREDUCE PERFORMANCE IN HETEROGENEOUS ENVIRONMENTS

MATEI ZAHARIA, ANDY KONWINSKI, ANTHONY D. JOSEPH, RANDY KATZ, ION STOICA
UNIVERSITY OF CALIFORNIA, BERKELEY

Presented by: Rehan Abdul Aziz

16th February, 2010

Aalto University School of Science and Technology

Age of data

- We live in the data age.
- Total size of data in the world?
- Data access speeds have not kept up.
- Using multiple disks in parallel is more useful.


Copyright 1996 Randy Glasbergen. www.glasbergen.com
GLASBERGEN
"Sorry about the odor. I have all my passwords tattooed between my toes."

MapReduce and Hadoop

- MapReduce model is very attractive for ad-hoc parallel processing of arbitrary data
- Hadoop, primarily developed by Yahoo, is an open source implementation.
- Runs jobs that produce hundreds of terabytes of data on at least 10,000 cores.
- Used at Facebook, Amazon, Last.fm, various universities for seismic simulation, natural language processing, and mining web data.

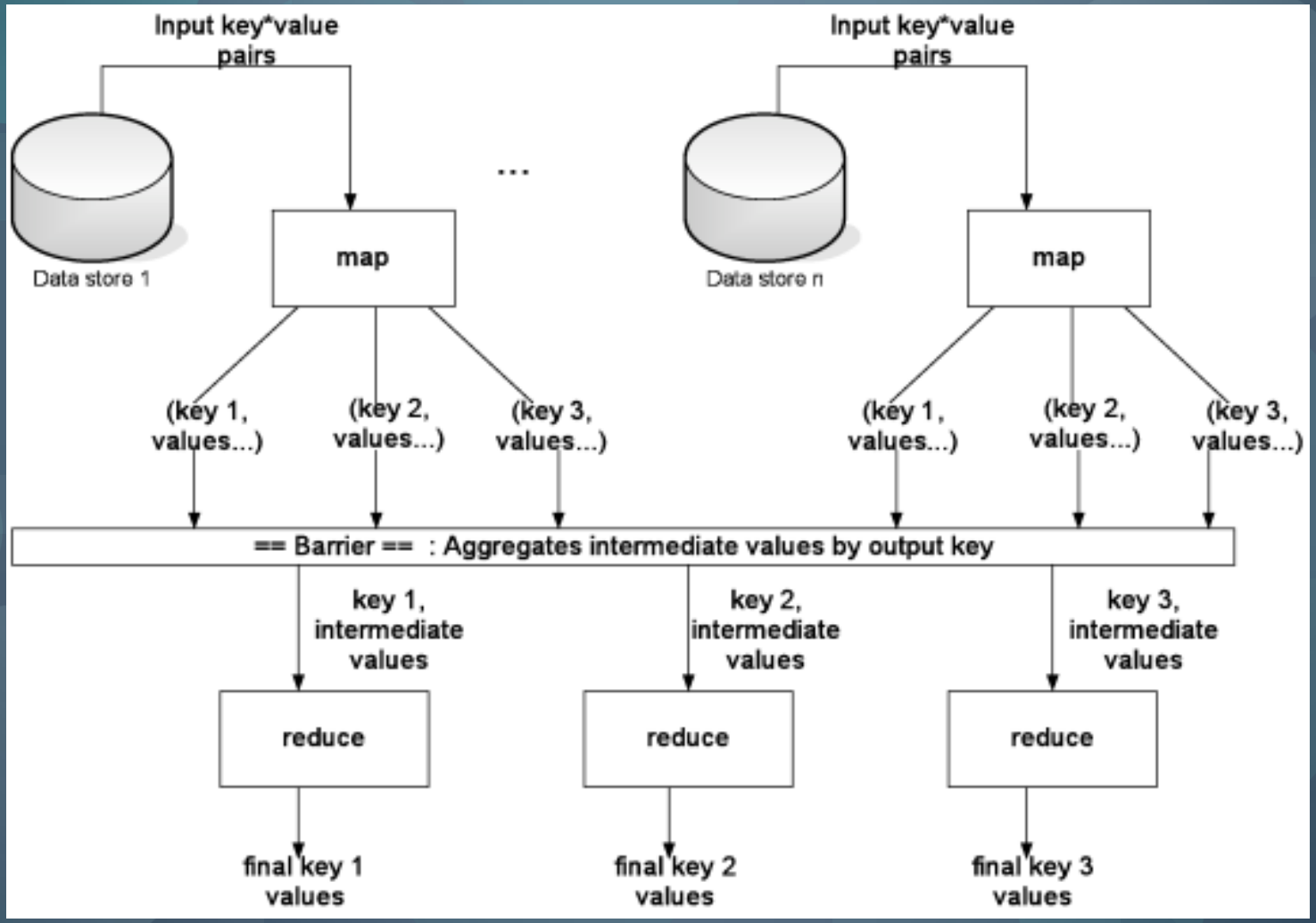
Hadoop Distrtibuted File System (HDFS)

- Quite similar to Google's.
- Single master, managing a number of slaves.
- Input file resides on distributed file system.
- Input file is split into even-sized chunks, replicated for fault tolerance.

MapReduce Computation

- Each job is divided into tasks.
- Each chunk of input is first processed by a *map* task.
- A *map* task outputs a list of key-value pairs.
- Map outputs are split into buckets based on key.
- *Reduce* tasks apply a reduce function to the list of map outputs with each key.

MapReduce Computation



Speculative Execution

- MapReduce automatically handles failures.
- If a node crashes, it reruns the node's task on some other machine.
- If a node is performing poorly (*straggler)*, MapReduce runs a *speculative copy* of its task ("backup task") on another machine.
- This is called *speculative execution*.
- Google has noted that speculative execution can improve job response times by 44%.
- Hadoop's heuristic is to compare each task's progress to the average progress.
- Works fine for homogeneous environments but leads to performance degradation when the underlying assumptions are broken.

Speculative Execution In Hadoop

- For a node with empty task slot, tasks in the order of decreasing priorities are:
  1. Failed tasks (to detect a bug and stop the job)
  2. Non-running tasks
  3. Tasks to execute speculatively
- Selection of speculative tasks is based on *progress score* between 0 and 1.
- For map
  - Progress score is the fraction of input data read
- For reduce
  - Each of the following accounts for 1/3 of the score
    - The copy phase, when the task fetches map outputs.
    - The sort phase, when map outputs are sorted by key.
    - The reduce phase
  - In each phase, the score is fraction of the data processed.

Speculative Execution In Hadoop

- Any task is marked as a *straggler* if:
  - It's progress is less than the average for its category minus 0.2. (*threshold)*
  - It has run for at least one minute.
- All tasks beyond the threshold are considered "equally slow" and ties are broken by data locality.
- The scheduler also ensures that at most one speculative copy of each task is running at a time.

Assumptions In Hadoop's Scheduler

1. Nodes can perform work at roughly the same time.
2. Tasks progress at a constant rate throughout time.
3. Here is no cost to launching a speculative task on a node that would otherwise have an idle slot.
4. A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, sort and reduce phases each take about 1/3 of the total time.
5. Tasks tend to finish in waves, so a task with a low progress score is likely a straggler.
6. Tasks in the same category (map or reduce) require roughly the same amount of work.

Homogeneity Assumptions

- The first two assumptions are about homogeneity which is not always the case.
- Heterogeneity seriously impacts Hadoop's scheduler.
  - Too many speculative tasks may be launched because the scheduler uses a fixed threshold for selecting tasks to speculate. This takes away resources from useful tasks.
  - Preference of data locality may result in the speculation of *wrong* task. Consider the following:

    Average Progress = 70%

    Task 1's Progress = 35% (2x slower than the average)

    Task 2's Progress = 7% (10x slower than the average)

    Task 1 has data on the idle node

    Which task will be speculated first?

Other Assumptions

- Assumptions 3,4, and 5 are broken on both homogeneous and heterogeneous clusters.
- Assumption 3: Speculating tasks on idle nodes costs nothing.
  - Breaks down when resources are shared, e.g. network, I/O etc.
  - When multiple jobs are submitted, needless speculation occupies nodes that could be otherwise running.
- Assumption 4: A task's progress score is approximately equal to its percent completion.
  - May cause incorrect speculation of reducers.
  - The slowest phase in reduce tasks is the *copy* phase since it involves all-pairs communication.
  - However, copy phase only accounts for 1/3 of the progress.
  - What happens when about 30% of the reducers finish?

Other Assumptions (cont.)

- As soon as 30% of reducers finish, the average progress is roughly $0.3(1) + 0.7(1/3) \approx 53\%$. Now all reducers still in the copy phase will be 20% behind average!

- Assumption 5: Progress score is a good proxy for progress rate because tasks begin at roughly the same time.
  - There are potentially tens of mappers per node.
  - Mappers tend to run in waves, and these waves get spread out over time due to variance adding up.
  - As a result, tasks from different generation will run concurrently.
  - New, fast tasks may get speculatively executed instead of old, slow tasks.
- Note that tasks with more than 80% progress can never be speculatively executed.

The LATE Scheduler

- Longest Approximate Time to End.
- Speculatively execute the task that will finish *farthest into the future,* since this task provides the greatest opportunity for a speculative copy to overtake.
- How to estimate the time left?
  - LATE uses a simple heuristic that works well in practice.
  - Define *ProgressRate* = *ProgressScore*/*T* for each task, where *T* is the amount of time the task has been running for.
  - *TimeToComplete* = *(1 – ProgressScore)*/*ProgressRate*

LATE's Other Heuristics

- Speculative tasks should be launched on fast nodes, not stragglers.
- LATE's heuristic: don't launch speculative tasks on nodes that are below *SlowNodeThreshold*.
- *SlowNodeThreshold* = sum of progress scores for all succeeded and in-progress tasks on the node.
- To account for the fact that speculative tasks consume resources:
  - Maximum number of speculative tasks that can be running at once, called *SpeculativeCap*.
  - A *SlowTaskThreshold* that determines whether a task is "slow enough" compared to others.

LATE Algorithm

- If a node asks for a new task and there are fewer than *SpeculativeCap* speculative tasks running:
  - Ignore the request if the node's total progress is below *SlowNodeThreshold*.
  - Rank currently running tasks that are not currently being speculated by estimated time left.
  - Launch a copy of the highest-ranked task with progress rate below *SlowTaskThreshold*.
  - Let the task run for at least a minute before speculating it.
- A good choice for the parameters:
  - *SpeculativeCap* = 10% of available task slots
  - *SlowNodeThreshold* and *SlowTaskThreshold* to 25th percentile of node progress and task progress rates respectively.

Advantages Of Late

- Robust to heterogeneity.
- Prioritizes among the slow tasks.
- Caps the number of speculative tasks to limit contention.
- Takes into account node heterogeneity when deciding *where* to run speculative tasks.
- Executes only tasks that will improve job response time, rather than any slow tasks.
- Consider the scenario:
  - Task A is 5x slower than the average but has 90% progress.
  - Task B is 2x slower than the average and has 10% progress.
  - Task B will be speculated first, because it hurts the *response time* more.

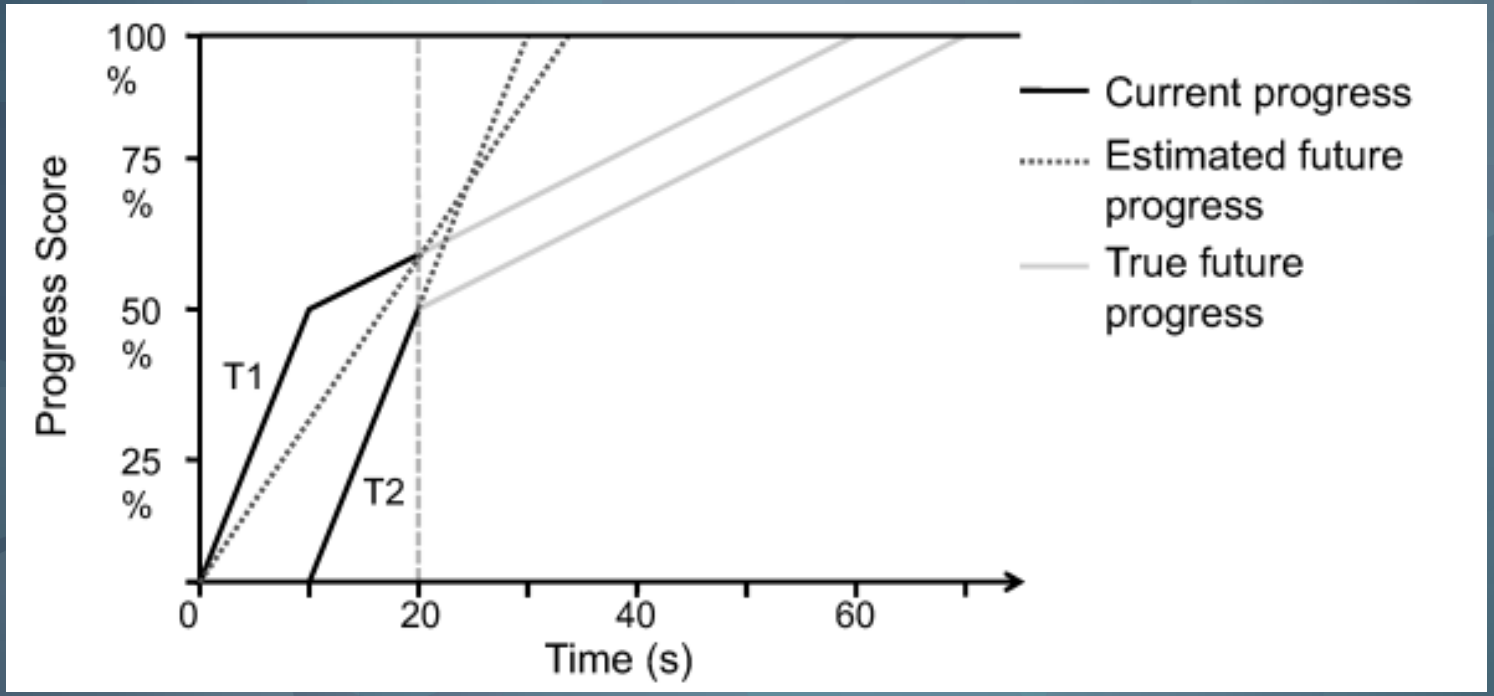
An Exception Where LATE Fails

- Recall the following heuristic:

  TimeToComplete = (1 – ProgressScore)/ProgressRate

- Now consider the following scenario:
  - A task has two phases in which it runs at different rates.
  - It's progress score grows by 5% per second in the 1st upto a total score of 50%, and slows down to 1% per second in the second phase.
  - It spends 10 seconds in the first phase, and 50 seconds in the second.
  - Now suppose we launch two instances of this task, T1 and T2, at times 0 secs, and at time 10 secs, respectively.
  - What happens when we check their estimated remaining times at 20 secs using the heuristic above?

An Exception (cont.)



EVALUATION AND EXPERIMENTS

Measuring Heterogeneity on EC2

- On EC2, Virtual Machines get the full available bandwidth when there is no contention, but are reduced to fair sharing when there is contention.
- A difference of 2.5-2.7x was observed between loaded and unloaded machines.
- Impact of contention of I/O performance:
  - Ran garbage writes on 871 VMs (see next slide)
  - Performance ranged from 62 MB/s for the isolated VMs to 25 MB/s when seven VMs shared a host.
- Impact of contention at the application level:
  - Ran two tests with 100 VMs, one where each VM was on a free separate physical host, and one where all 100 VMs were packed onto 13 physical hosts, with 7 machines per host.
  - In both cases, 100 GB of random data was sorted.
  - Finished in 408 sec and 1094 sec respectively.

EC2 Disk Performance vs. VM co-location

Write performance vs. number of VMs per physical host on EC2. Second column shows how many VMs fell into each load level.

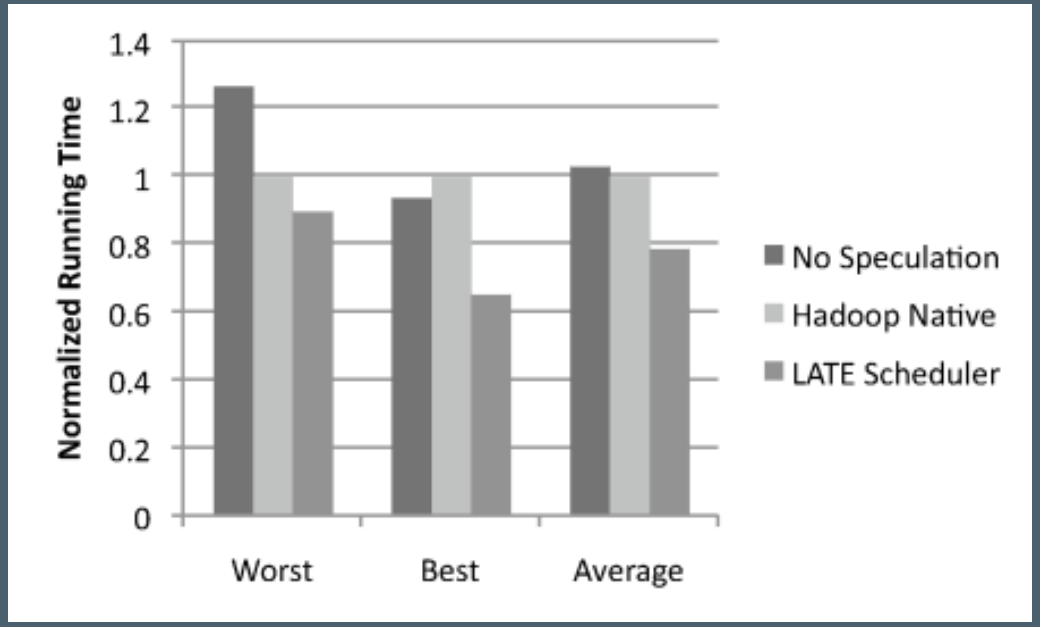
Load Level	VMs	Write Perf (MB/s)	Std Dev
1 VMs/host	202	61.8	4.9
2 VMs/host	264	56.5	10.0
3 VMs/host	201	53.6	11.2
4 VMs/host	140	46.4	11.9
5 VMs/host	45	34.2	7.9
6 VMs/host	12	25.4	2.5
7 VMs/host	7	24.8	0.9

Scheduling Experiments on EC2

- LATE, Hadoop's native scheduler, and no speculation were compared in a variety of experiments on EC2, on clusters of about 200 VMs.
- The three schedulers were compared in two settings: heterogenous but non-faulty nodes, and an environment with stragglers.
- Scheduling in a heterogeneous cluster:
  - 1 to 7 VMs per host, for a total of 243 VMs.
  - Sort job on a data set of 128 MB per VM, or 30 GB of total data.
  - Each job had 486 map tasks and 437 reduce tasks.

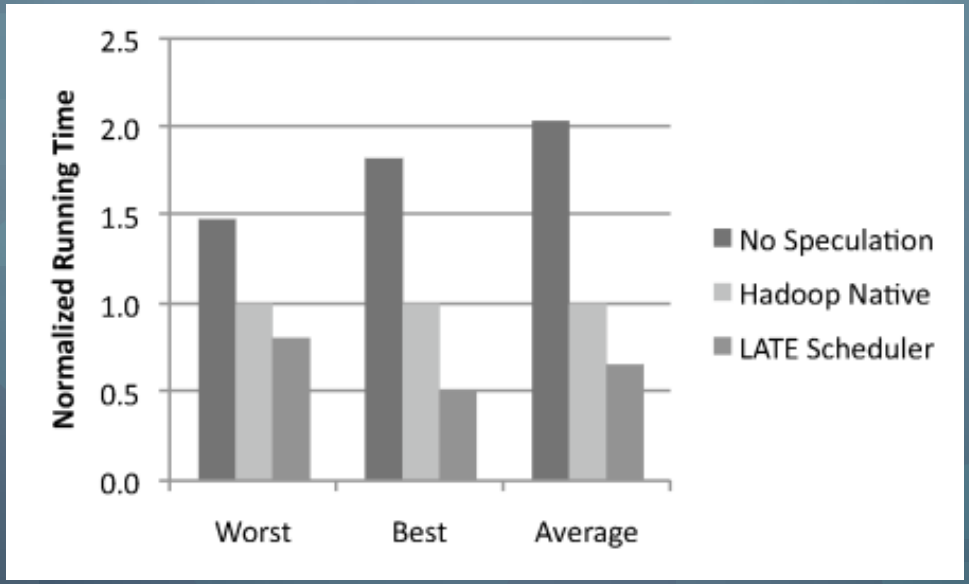
EC2 Sort running times in heterogeneous cluster

Worst, best and average-case performance of LATE against Hadoop's scheduler and no speculation.



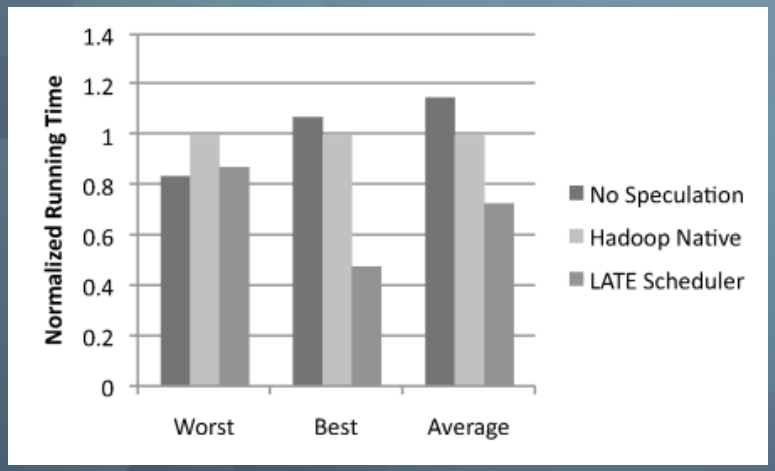
Scheduling Experiments on EC2 (cont.)

- Scheduling with stragglers:
  - Manually slowed down eight VMs in a cluster of 100.
  - The other machines were assigned between 1 and 8 VMs per host. Sorted 256 MB per host.

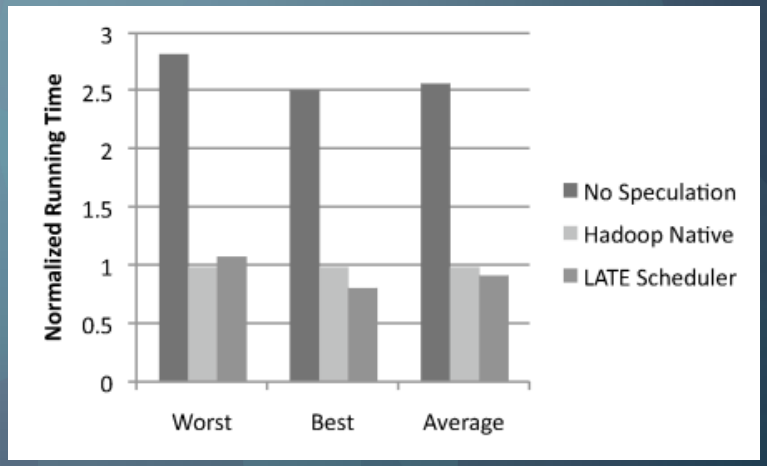


Scheduling Experiments on EC2 (cont.)

- Differences across workloads:
  - Grep and WordCount run on heterogeneous clusters with stragglers.]
  - Applied Grep to 43 GB of text data (about 200 MB per host), and searched for "the".
  - Applied WordCount to 21 GB (100 MB per host).
  - Gain is smaller in WordCount than in Grep and Sort.
  - Element of "luck"?



Grep



WordCount

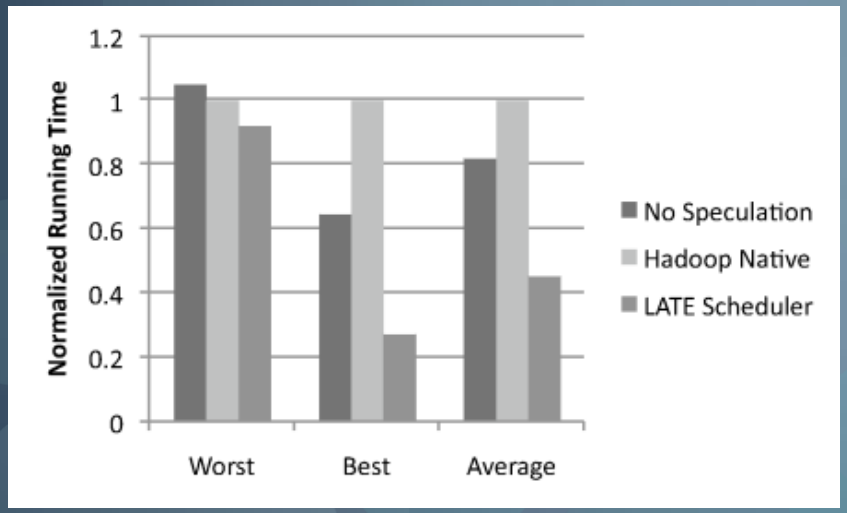
"You are completely free to carry out whatever research you want, so long as you come to these conclusions."

Local Testbed Experiments

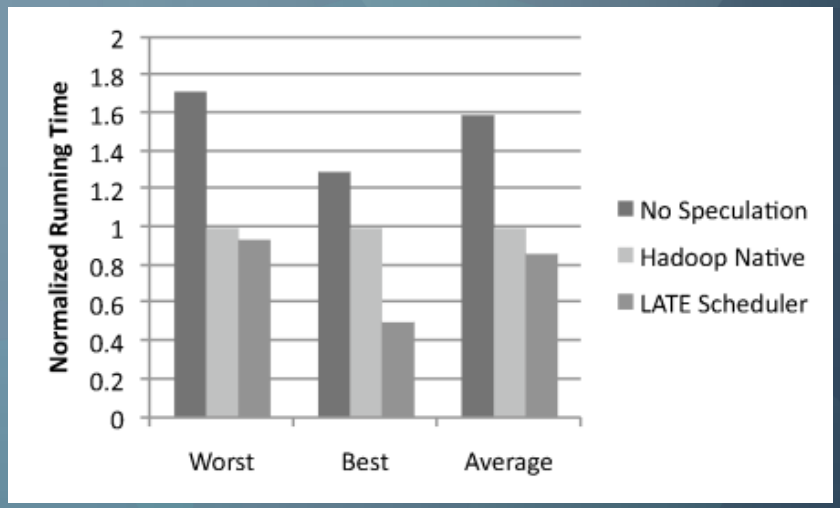
- To validate the results from EC2 in a more tighlty controlled environment.
- Machines were dual processor, dual-core 2.2 GHz Opteron processors, 4 GB memory, and a single 250 GB SATA drive.
- On each machine, one to four virtual machines using Xen virtualization software.
- Local I/O performance heterogeneity.
- Local Scheduling Experiments

Load Level	VMs	Write Perf (MB/s)	Std Dev
1 VMs/host	5	52.1	13.7
2 VMs/host	6	20.9	2.7
4 VMs/host	4	10.1	1.1

Local cluster disk performance



Local sort with heterogeneity



Local sort with stragglers

Thank you for your time.

Questions?