# Job Dispatching and Scheduling under Heterogeneous Clusters

Ting-Chou Lin, Ching-Chi Lin, Ting-Wei Chang Pangfeng Liu,
National Taiwan University
Chia-Chun Shih, Chao-Wen Huang, Chun-Hua Telecom
Jan-Jan Wu, Academia Sinica

April 19, 2016

# Private Cloud

- Cloud computing has become a very important issue for computer science and IT industry in recent years.
- Many enterprises or institutes are building private clouds by establishing their own data centers, e.g., CHT Telecom.
- These private data centers provide flexible computing resources to accommodate the changing needs of large enterprises.

# Private Cloud

There are two important characteristics in private clouds.

- The amount of computing resources and the number of physical machines are fixed for most of the time.
- The servers are heterogeneous.

# Static Configuration

- Enterprises purchase new hardware to replace obsolete ones according to a fixed schedule.
- There will be no extra "on-demand" computing resources to use in these environments (unlike the case in public cloud).
- It is critical to assign computing resources to various applications so that we can satisfy various time and financial constraints.

# Heterogeneous Configuration

- Equipments are purchased at different times, newly purchased ones are usually more powerful than the previous ones.
- Job scheduling must take the heterogeneity into consideration.

# Jobs

- Jobs may have different characteristics, resource requirements, time constraints, and priority.
- Scientific computation requires large computing power, while a billing sub-system of CHT Telecom needs to generate the bills for each phone subscriber every month before the deadline.
- It is extremely important to allocate computing resources for jobs in a heterogeneous environment so that different requirements are met.

# Framework

- We proposed a cloud resource management framework to dynamically adjust the number of servers for every job to satisfy various job constraints.
- The framework can work alone as an individual cluster management system, or as an extension of an existing cloud system.
- A joint work with CHT Telecom and we use the billing sub-system of CHT Telecom as an example to demonstrate the merits of our system.

# Framework

- Our framework makes decisions according to *scheduling policy*.
- We consider four policies — *priority-based*, *proportion-based*, *workload-based* and *deadline-based*.
- These policies consider remaining workload, priority or deadline of each job, then generate scheduling plans accordingly.

# Policy

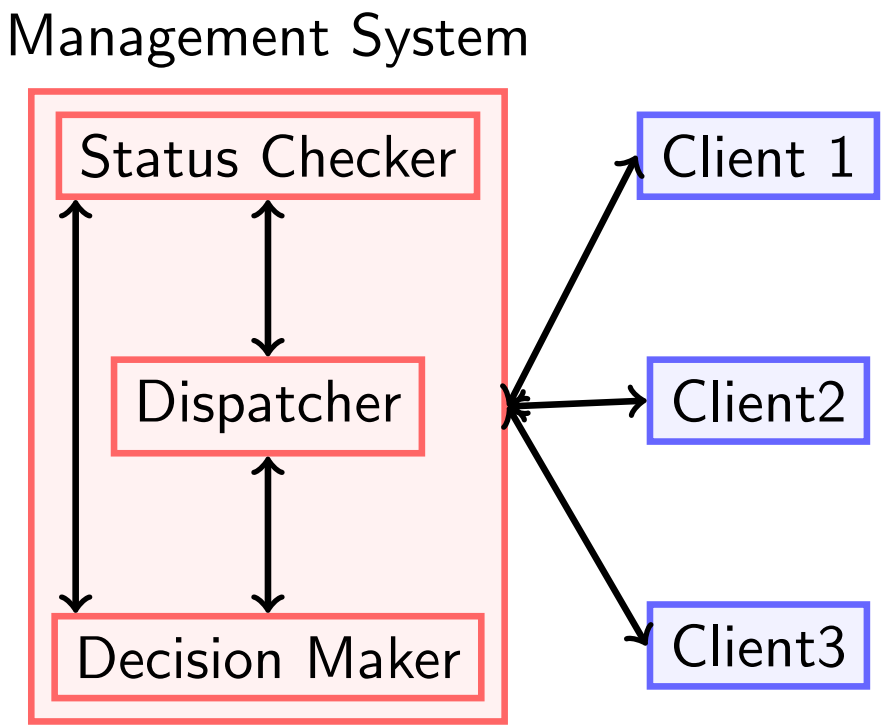
- The system considers both the effects of priority and deadlines, and the three effects of deadlines simultaneously.
- The evaluation criteria are how many jobs do not meet deadlines, by how long they missed the deadline, and the importance of the jobs that miss the deadline.

# Contributions

- A framework designed specifically for private cloud that have heterogeneous and yet static environment, to schedule jobs of different requirements.
- A model to integrate the effects of both priority and deadlines, and algorithms to combine the effects as an objective function.
- Consider three effects simultaneously – how many jobs do not meet deadlines, by how long they missed the deadline, and the importance of the jobs that miss the deadline.
- A pluggable framework architecture in which the components can be hot-swapped without shutting down the service.

# Framework

- Our cloud management framework consists of three parts – a *management system*, many *clients*, and many *workers*.
- The management system is the focus of our framework because users submit jobs from clients to it.
- The workers are the computation servers that process tasks in jobs.


Management System
Status Checker
Dispatcher
Decision Maker
Client 1
Client2
Client3

# Client

- A client provides the programming interface for users to submit jobs.
- Users must specify *job attributes*, such as deadline, priority and execution profile from previous executions.
- These attributes are passed to the decision maker so that it can schedule the jobs to run on suitable servers.

# Worker

- A *worker* is the logical task processing unit in the framework.
- A worker executes at most one task at a time and it is the minimum scheduling unit in the framework.

# Management System

- The management system has three components – a *status checker*, a *decision maker* and a *dispatcher*.
- The status checker collects worker information periodically, and forwards these information to the decision maker, which allocates resources to jobs.
- The dispatcher executes the decisions made by the decision maker and allocates resources to jobs.

# Status Checker

The Status Checker checks if a worker is *available*, *busy*, *occupied*, or *down*.

**Available** Idle and ready to accept new tasks.

**Busy** Executing a task.

**Occupied** Has been assigned jobs, but it is now blocked, e.g., waiting for necessary data.

**Down** Currently non-functional.

# Decision Maker

- The most critical component of the management system because it produces an *allocation plan* according to a specified *policy*, which considers various parameters, e.g., job deadline and priority.
- The decision maker allocates resources passively – the dispatcher only invokes the decision maker when a job is submitted or when a job finishes.

# Dispatcher

- The dispatcher allocates and reallocates physical resources according to the allocation plan from the decision maker.
- The dispatcher receives job requests from the clients, sends the job information to the decision maker, receives the allocation plan from the decision maker, then coordinates the workers and clients to run the jobs according to the allocation plan.

# Policy

Four policies in our framework.

- Priority-based
- Proportion-based
- Workload-based
- Deadline-based

# Priority-based Policy

- Priority-based policy distributes workloads according to job priority.
- Assign workers to the jobs according to their priority. The process repeats until there are no idle workers or all jobs are assigned sufficient workers.
- To make the job with highest priority runs as fast as possible.

# Proportion-based Policy

- The proportion-based policy, (or *fair share scheduling* from the literature), allocates workers to jobs according to the workload proportion of a job to all jobs.
- Does not consider priority.
- First calculates the sum of the workloads from all jobs, then determines the workload ratio of a job to all jobs and the number of workers in proportional to its workload.

# Workload-based Policy

- Similar to Proportion-based policy but also consider worker heterogeneity.
- For each job, workers are sorted by the speed they processing this job.
- Just enough fastest workers are allocated to the job. That is, we try to finish a job as soon as possible by assigning the job to the workers that will finish the job fastest.

# Deadline-based Policy

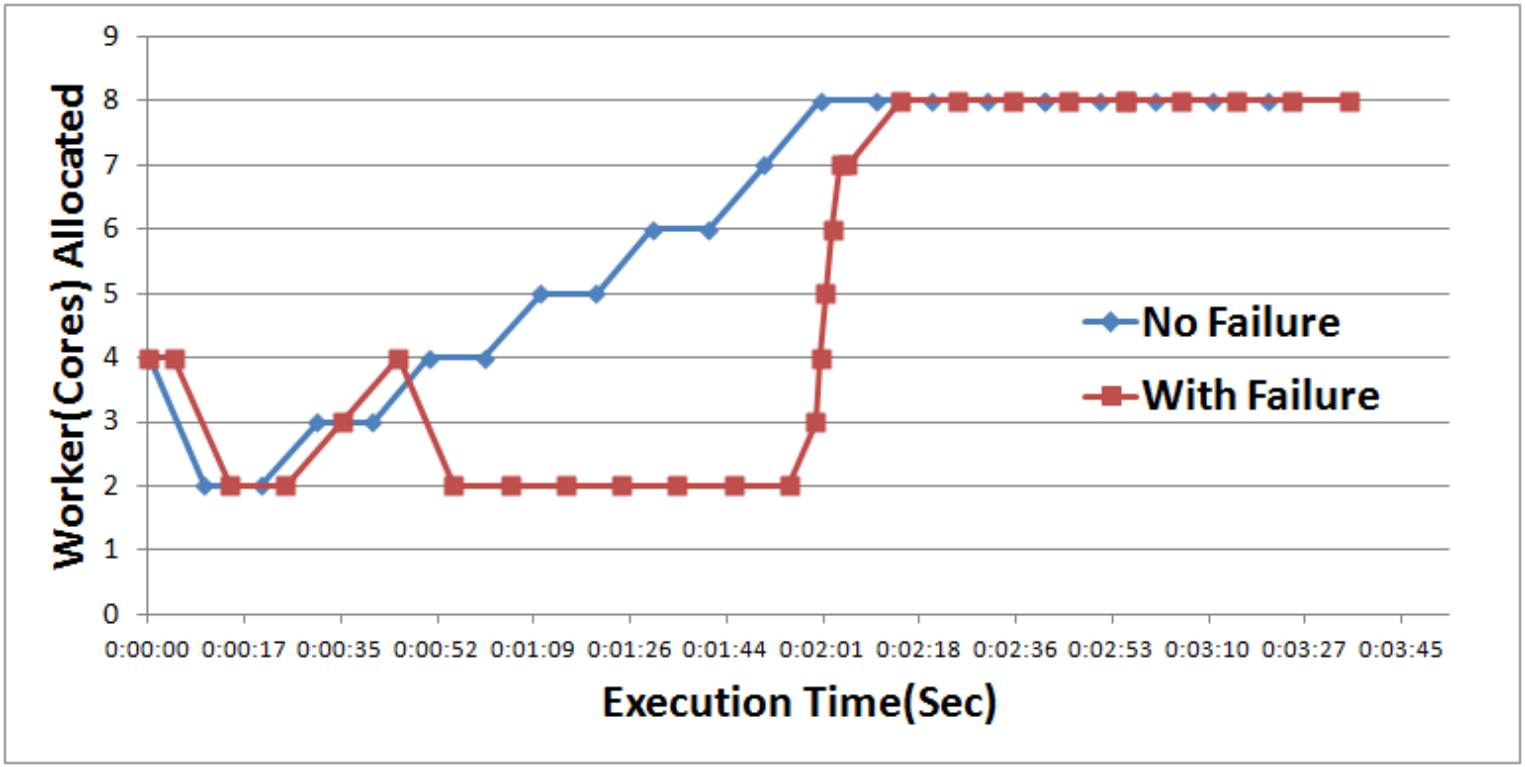
- A greedy approach similar to the workload-based policy.
- Process all jobs in the order of their priority.
- Starting from the job with the highest priority, we sort the remaining workers according to their speed in running this job then assign enough workers to finish this job before its deadline.

# Hardware Setting

- One master and ten worker nodes.
- The master node has two quad-core Intel Xeon CPU X5570 running at 2.93GHz, and has 1.5GB of memory.
- Each worker node is a virtual machine that has a single-core and 768MB of memory.
- Both the management system and the client run on the master node, and two workers run on each working node.

# Failure Tolerance

- First we try a small scale workload trace provided by CHT Telecom Laboratories and intentionally introduce hardware failure.
- The deadline-based scheduling policy schedules jobs to eight workers with deadline in 4 minutes.
- This workload starts with several light-weight tasks and then brings in heavier tasks.


Worker(Cores) Allocated
9
8
7
6
5
4
3
2
1
0
0:00:00 0:00:17 0:00:35 0:00:52 0:01:09 0:01:26 0:01:44 0:02:01 0:02:18 0:02:36 0:02:53 0:03:10 0:03:27 0:03:45
Execution Time(Sec)
No Failure
With Failure

# No Failure

- At the beginning because the scheduler does not have any information about jobs, it just uses a default number of workers.
- The run time statistics collected by the status checker (around 00:00:05) indicates that task execution is going faster than expected, so the system decides to decrease the number of workers.
- The system later (around 00:20:00) realizes that the task execution is slowing down and the current allocation might not be able to meet the deadline, so it schedules more workers to the job.

# With Failure

- Manually shut down six workers at 0:00:55 and resume them at 0:01:55.
- The system realizes that it is way behind schedule, and tries much harder than the no-fault case to catch up with the deadline.
- The slope of the red line (for number of worker) is steeper than the blue line, indicating the ability of the system to adapt to worker failure.

# Effectiveness

- We conducted a trace-based simulation to demonstrate the effectiveness of our system.
- The trace is the SDSC-Par96 log, containing 32135 jobs. This log contains a years worth of accounting records for the 416-node Intel Paragon located at the San Diego Super-computer Center (SDSC).

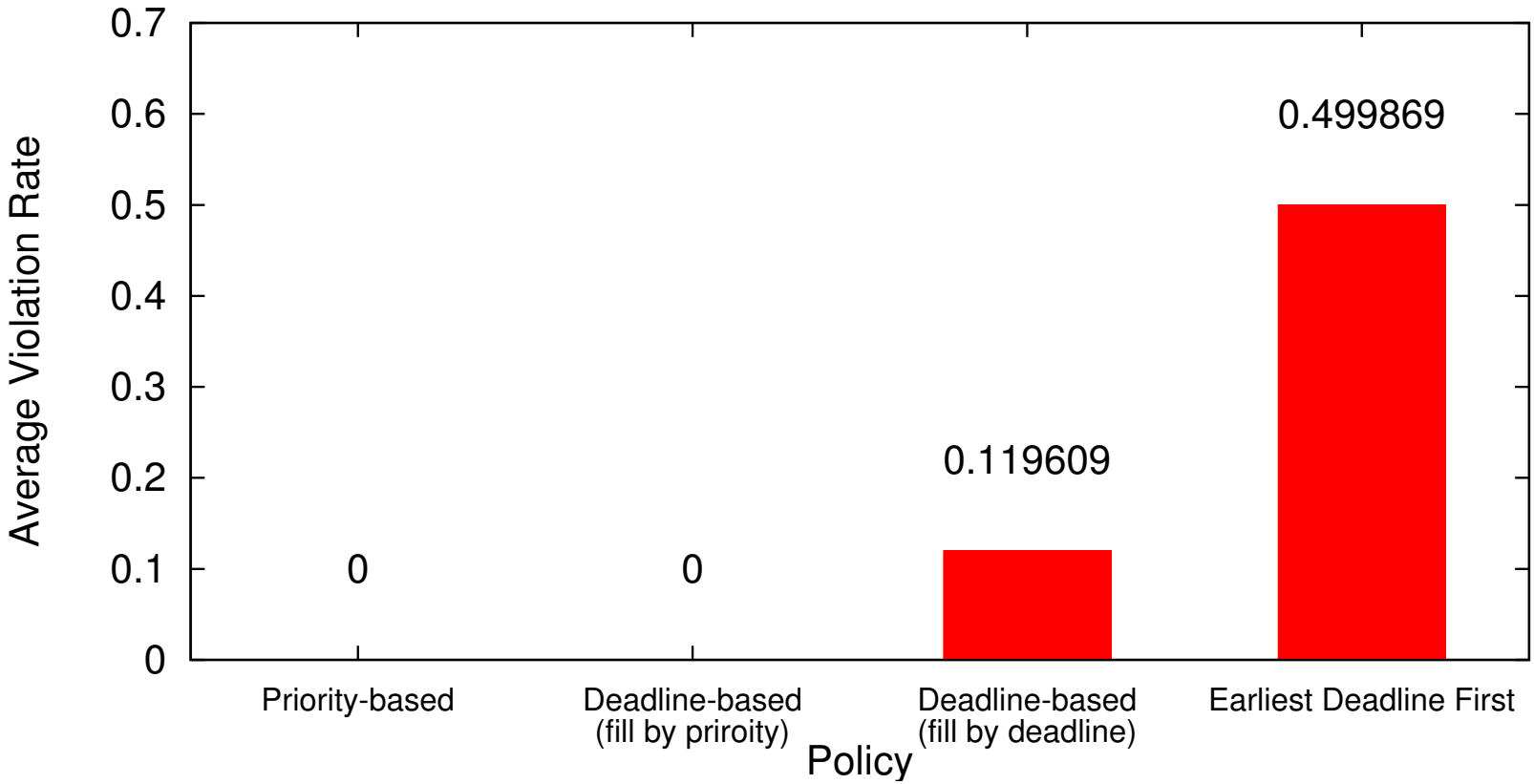
# Evaluation

- We evaluated scheduling policies with *priority inversion rate* and *deadline miss penalty*.
- The baseline for comparison is Earliest Deadline First (EDF) algorithm, an optimal scheduling policy in real-time systems for preemptive jobs running on a processor.

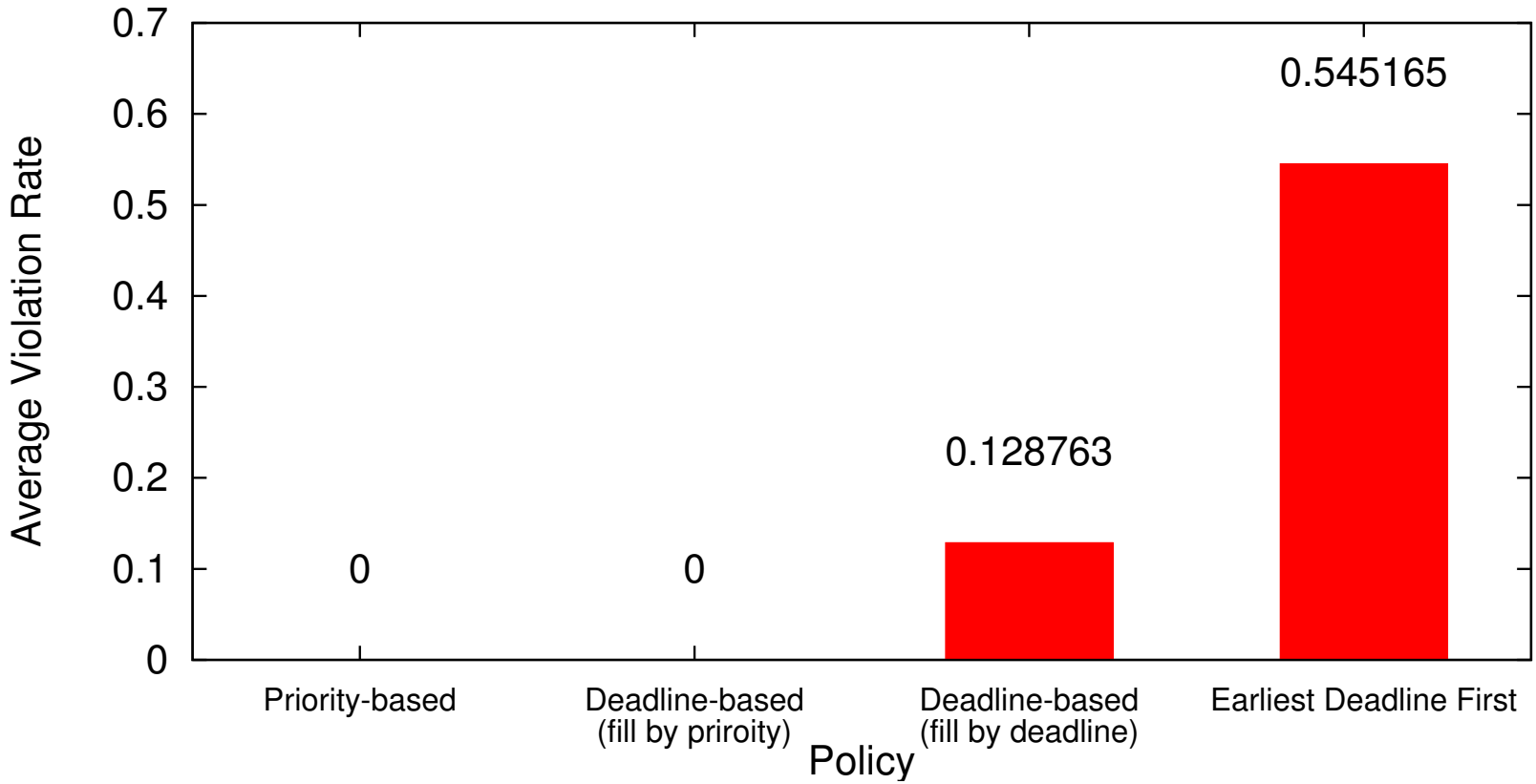
# Priority Inversion Rate

- *Priority inversion rate* measures how often a scheduling policy violates the priority constraint.
- A job is *priority inverted* if it is not allocated any worker, but it has a higher priority than another job that has been allocated workers.
- The priority inversion rate is the percentage of priority inverted jobs in the system.

# Priority Inversion Rate (Homogeneous)



# Priority Violation Rate (Heterogeneous)



# Priority Inversion

- In both homogeneous and heterogeneous environment, the priority inversion rates of priority-based policy and deadline-based policy (with spare resource allocated by priority) are 0.
- High priority jobs are never scheduled later than low priority ones under these two policies.

# Priority Inversion

- Deadline-based (with spare resource allocated by deadline) and Earliest-Deadline-First do not always schedule high priority jobs before low priority ones.
- Since EDF does not take priority into consideration, its average inversion rate is much higher (4 times as much) than deadline-based (with spare resource allocated by deadline).

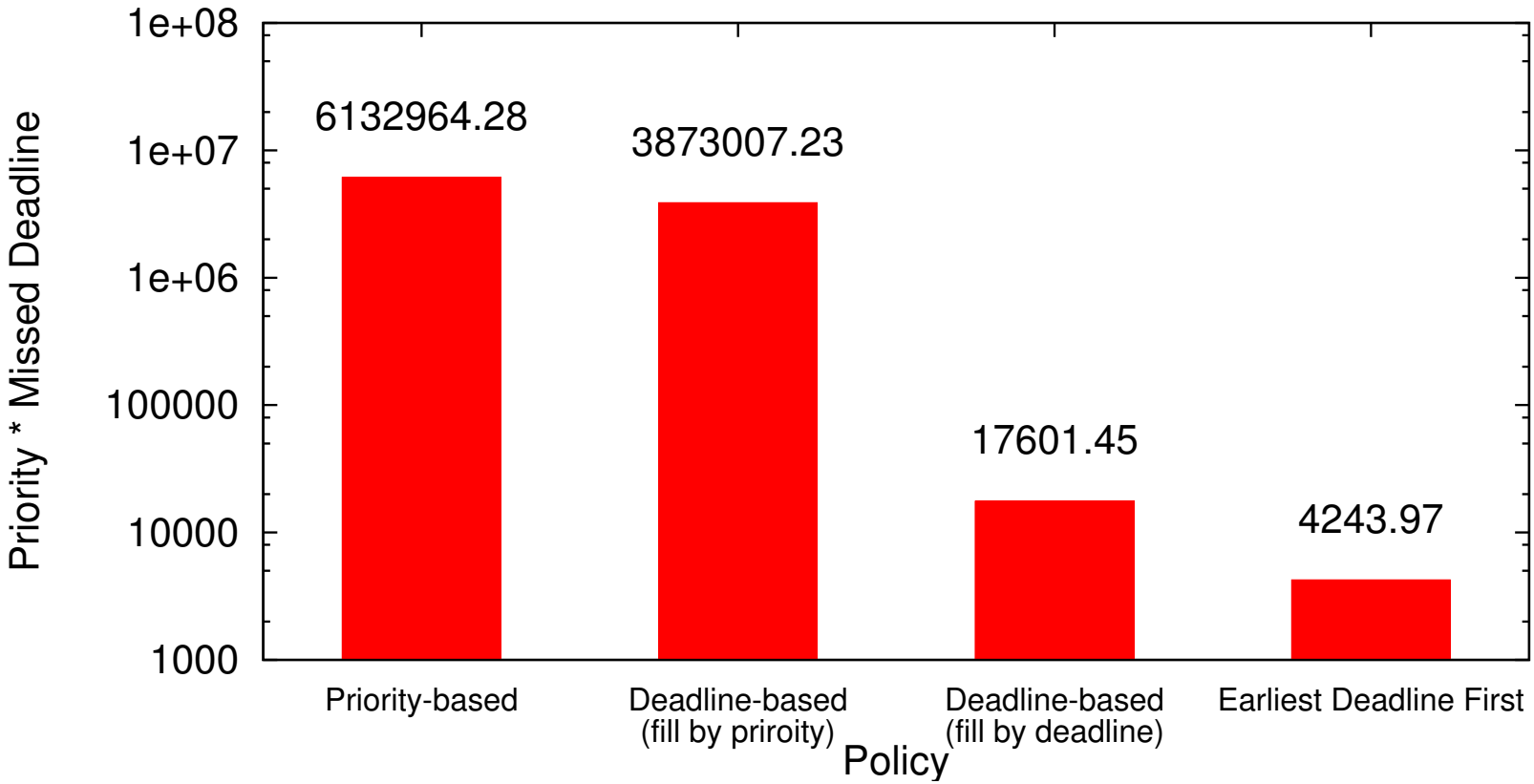
# Deadline Miss Penalty

- *Deadline miss penalty* measures how many jobs do not meet deadlines, by how long they missed the deadline, and the importance of the jobs that miss the deadline.
- We encourage policy to have less jobs missing deadline, less time they miss the deadline, and high priority jobs should miss deadline less often than low priority ones.

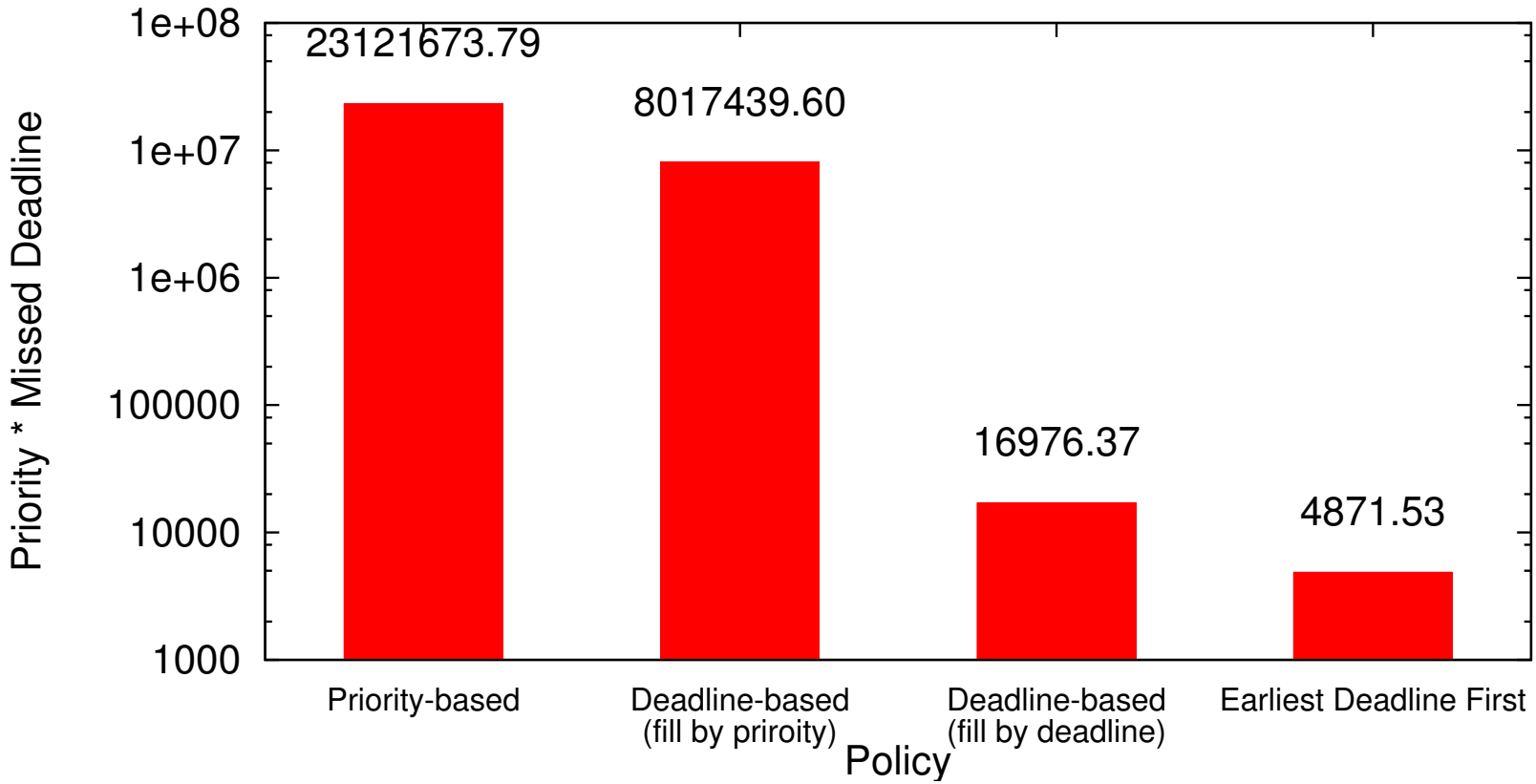
# Combined Metric

- Combine these three aspects by defining panalty as the sum of the product of job priority and the amount of time a job misses the deadline, for all jobs that missed its deadline.
- The goal of scheduling is to minimize the this penalty function.

# Deadline Miss Penalty (Homogeneous)



# Deadline Miss Penalty (Heterogeneous)



# Observations

- EDF preserves deadlines best in both homogeneous and heterogeneous environments.
- Priority-based policy and deadline-based policy (with spare resource allocated by priority) has three times penalty scores as those of EDF.

# Deadline First

- The workload still has jobs that takes 500 to 800 seconds to complete, so the effect of execution time and deadlines will be much higher than the priority, which is limited under 100.
- As a result if deadline-based policy allocate spare resources by deadline instead of priority, the penalty is greatly reduced.

# Conclusion

- A cloud resource management framework that dynamically allocates and reallocates computation resources for jobs that have different requirements, including deadline and priority.
- The framework can work as an individual cloud computing system, or as an extension of an existing cloud system.
- The system is capable of dynamically adjusting the resource allocation plan according to the run-time statistics collected, and also tolerates hardware failures.
- The experiments suggest a trade-off between the priority and deadline.