Minimizing Interference and Maximizing Progress for Hadoop Virtual Machines

Wei Zhang[*†], Sundaresan Rajasekaran[*], Shaohua Duan[*], Timothy Wood[*] and Mingfa Zhu[†]

[*]Department of Computer Science, The George Washington University, Washington, D.C., USA

[†]Beihang University, Beijing, China

ABSTRACT

Virtualization promised to dramatically increase server utilization levels, yet many data centers are still only lightly loaded. In some ways, big data applications are an ideal fit for using this residual capacity to perform meaningful work, but the high level of interference between interactive and batch processing workloads currently prevents this from being a practical solution in virtualized environments. Further, the variable nature of spare capacity may make it difficult to meet big data application deadlines.

In this work we propose two schedulers: one in the virtualization layer designed to minimize interference on high priority interactive services, and one in the Hadoop framework that helps batch processing jobs meet their own performance deadlines. Our approach uses performance models to match Hadoop tasks to the servers that will benefit them the most, and deadline-aware scheduling to effectively order incoming jobs. We use admission control to meet deadlines even when resources are overloaded. The combination of these schedulers allows data center administrators to safely mix resource intensive Hadoop jobs with latency sensitive web applications, and still achieve predictable performance for both. We have implemented our system using Xen and Hadoop, and our evaluation shows that our schedulers allow a mixed cluster to reduce web response times by more than ten fold compared to the existing Xen Credit Scheduler, while meeting more Hadoop deadlines and lowering total task execution times by 6.5%.

Keywords

scheduling, virtualization, Map Reduce, interference, deadlines, admission control

1. INTRODUCTION

Virtualization has facilitated the growth of infrastructure cloud services by allowing a single server to be shared by multiple customers. Dividing a server into multiple virtual machines (VMs) provides both a convenient management abstraction and resource boundaries between users. However, the performance isolation provided by virtualization software is not perfect, and interference between guest VMs remains a challenge. If the hypervisor does not enforce proper priorities among guests, it is easy for one virtual machine's performance to suffer due to another guest.

Despite the danger of interference, resource sharing through virtualization has been crucial for lowering the cost of cloud computing services. Multiplexing servers allows for higher average utilization of each machine, giving more profit for a given level of hardware expense. Yet the reality is that many data centers, even those employing virtualization, are still unable to fully utilize each server. This is due in part to fears that if a data center is kept fully utilized there will be no spare capacity if workloads rise, and part due to the risk of VM interference hurting performance even if servers are left underloaded.

In this paper, we first study the causes of interference through virtualization scheduler profiling. We observe that even when set to the lowest possible priority, big data VMs (e.g., Hadoop jobs) interrupt interactive VMs (e.g., web servers), increasing their time spent in the runnable queue, which hurts response times. We control and reduce VM CPU interference by introducing a new scheduling priority for "background" batch processing VMs, allowing them to run only when other VMs are not actively utilizing the CPU.

Our changes in the VM scheduler improve the performance of interactive VMs, but at the cost of unpredictable Hadoop performance. To resolve this challenge, we implement a second scheduler within the Hadoop framework designed for hybrid clusters of dedicated and shared VMs that only use residual resources. We find that when given the same available resources, different tasks will progress at different rates, motivating the need to intelligently match each Hadoop task to the appropriate dedicated or shared server. Our scheduler combines performance models that predict task affinity with knowledge of job deadlines to allow Hadoop to meet SLAs, despite variability in the amount of available resources.

Together, these schedulers form the Minimal Interference Maximal Productivity (MIMP) system, which enhances both the hypervisor scheduler and the Hadoop job scheduler to better manage their performance. Our primary contributions include:

- A new priority level built into Xen's Credit Scheduler that prevents batch processing VMs from hurting interactive VM performance.
- Task affinity models that match each Hadoop task to the dedicated or shared VM that will provide it the most benefit.
- A deadline and progress aware Hadoop job scheduler that allocates resources to jobs in order to meet performance goals and maximize the efficiency of a hybrid cluster.
- An admission control mechanism which ensures high priority jobs meet deadlines, even when a cluster is heavily overloaded.

We have implemented the proposed schedulers by modifying the Xen hypervisor and Hadoop scheduler. Our evaluation shows that

This article is an extended version of [24], which appeared in CCGrid 2014

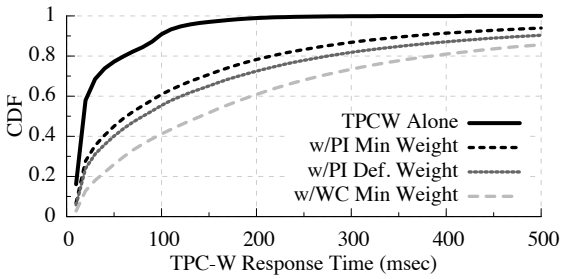


Figure 1: Colocated Hadoop jobs degrade web application performance, despite using the Xen scheduler priority mechanisms.

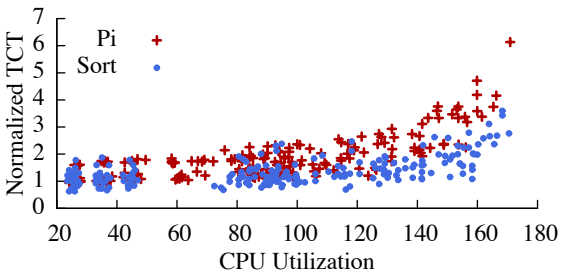


Figure 2: TCT varies by job, and increases non linearly as the web service consumes a larger quantity of CPU (out of 2 cores).

MIMP can prevent nearly all interference on a web application, doubling its maximum throughput and providing nearly identical response times to when it is run alone. For a set of batch jobs, the algorithm can meet more deadlines than EDF(Earliest Deadline First), and reduces the total execution time by over four and a half CPU hours, with minimal impact on interactive VM performance.

Our paper is structured as follows: Section 2 provides the motivation of our paper, and Section 3 provides the problem and system overview for our work. In Section 4 and Section 5, we give a description about interactive VM scheduling in Xen and progress-aware deadline scheduling in Hadoop. Section 6 provides our evaluation using different benchmarks. We discuss related work in Section 7, and conclude in Section 8.

2. MAP REDUCE IN VIRTUALIZED CLUSTERS

Map Reduce is a popular framework for distributing data intensive computation [6]. Hadoop is an open source implementation of Map Reduce developed by Yahoo. Users write a program that divides the work that must be performed into two main phases: Map and Reduce. The Map phase processes each piece of input data and generates some kind of intermediate value, which is in turn aggregated by the Reduce phase.

In this paper we investigate how to run Map Reduce jobs in a hybrid cluster consisting of both dedicated and shared (also known as volunteer) nodes. This problem was first tackled by Clay et al., who described how to pick the appropriate number of shared nodes in order to maximize performance and minimize overall energy costs [5]. Like their work, we focus on scheduling and modeling the Map phase since this is generally the larger portion of the program, and is less prone to performance problems due to slow nodes. Our work extends their ideas both within the virtualization layer to prevent interference, and at the Map Reduce job scheduling level to ensure that *multiple* jobs can make the best use of a hybrid cluster and effectively meet deadlines.

A key issue that has not yet been fully explored is how to prevent batch processing jobs such as Map Reduce from interfering with foreground workloads. Our results suggest that interference can be quite severe if the important performance metric is interactive latency as opposed to coarse grained timing measures (e.g., the time to compile a linux kernel).

As a motivating experiment, we have measured the achieved throughput and response time when running the TPC-W online book store benchmark both alone and alongside a VM running Hadoop jobs. Our results in Figure 1 show that the response time of the web application can be dramatically increased when run with a Pi or WordCount (WC) job. This happens even when the Xen scheduler's parameters are tuned to give Hadoop the lowest possible weight (i.e., the lowest priority). However, the throughput of TPC-W remains similar, as does the amount of CPU that it consumes. Further, we find that if Hadoop is given a separate CPU from TPCW, there is no interference at all. This suggests that the performance interference is due to poor CPU scheduling decisions, not IO interference.

A second major challenge when running in shared environments is that different Hadoop jobs are affected by limitations on available resources in different ways. Figure 2 shows that as the amount of resources consumed by a foreground interactive VM rises, the normalized task completion time (relative to Hadoop running alone) can increase significantly for some jobs. For example, Pi, a very CPU intensive job, suffers more than Sort, which is IO intensive. As a result, the best performance will be achieved by carefully matching a Hadoop job to the servers that will allow it to make the most efficient progress.

3. PROBLEM AND SYSTEM OVERVIEW

This section presents the formal problem MIMP targets, and then gives an overview of the system.

3.1 Problem Statement

The scenario where we believe MIMP will provide the most benefit is in a hybrid cluster containing a mix of dedicated nodes (virtual or physical) and "volunteer" or "shared" nodes that use virtualization to run both interactive applications and Hadoop tasks. We assume that the interactive applications are *higher priority* than the Hadoop tasks, which is generally the case since users are directly impacted by slowdown of interactive services, but may be willing to wait for long running batch processes. While we focus on web applications, the interactive applications could represent any latency-sensitive service such as a streaming video server or remote desktop application. Although we treat Hadoop jobs as *lower priority*, we still take into account their performance by assuming they arrive with a deadline by which time they must be complete.

As discussed previously, we focus on the Map phase of Map Reduce, as this is generally more parallelizable and is less prone to straggler performance problems (i.e., a single slow reduce task can substantially hurt the total completion time). As in [5], we use dedicated servers to run both the shared Hadoop file system and all reduce tasks.

We assume that the interactive applications running in the high priority VMs have relatively low disk workloads, meaning that sharing the IO path with Hadoop tasks does not cause a resource bottleneck. While this is not true for some disk intensive applications such as databases, for others it can be acceptable, particularly due to the increasing use of networked storage (e.g., Amazon's Elastic Block Store) rather than local disks.

Given this type of cluster, a key question is how best to allocate the available capacity in order to maximize Hadoop job per-

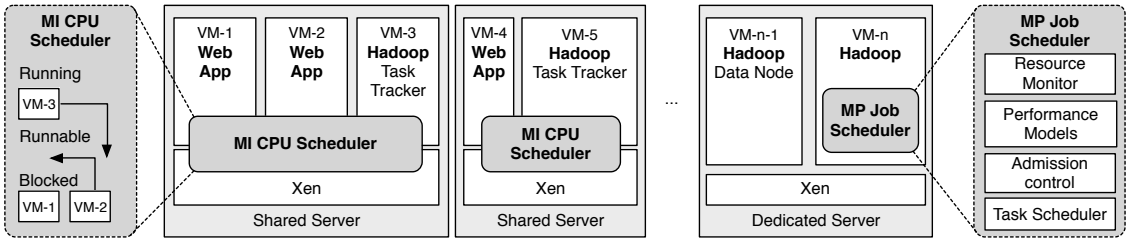


Figure 3: The MI CPU scheduler only runs Hadoop VMs when others are blocked, so VM-2 immediately preempts VM-3 once it becomes ready. The MP Job Scheduler gathers resource availability information from all nodes and schedules jobs based on performance model results.

formance (i.e., minimize the number of deadline misses and the total job completion times) while minimizing the interference on the interactive services (i.e., minimizing the change in response time compared to running the web VMs alone).

3.2 MIMP Overview

We have developed MIMP to tackle this pair of challenges. The system is composed of two scheduling components, as illustrated in Figure 3.

Minimal Interference CPU Scheduler: The MI CPU Scheduler tries to prevent lower priority virtual machines from taking CPU time away from interactive VMs. We do this by modifying the Xen CPU scheduler to define a new priority level that will always be preempted if an interactive VM becomes runnable.

Maximal Productivity Job Scheduler: Next we modify the Hadoop Job scheduler to be aware of how available resources affects task completion time. The MP Scheduling system is composed of a training module that builds performance models, a monitoring system that measures residual capacity throughout the data center, and a scheduling algorithm. Our MP Scheduler combines this information to decide which available resources to assign to each incoming Hadoop Job to ensure it meets its deadline while making the most productive use of all available capacity.

4. VM SCHEDULING IN XEN

This section diagnoses the performance issues in the Xen current Credit scheduler when mixing latency sensitive and computationally intensive virtual machines. We then describe how we have enhanced the Xen scheduler to help minimize this interference.

4.1 Performance with Xen Credit Scheduler

Xen Credit scheduler is a non-preemptive weighted fair-share scheduler. As a VM runs, its VCPUs are dynamically assigned one of three priorities - over, under, or boost, ordered from lowest to highest. Each physical CPU has a local run queue for runnable VCPUs, and VMs are selected by their priority class. Every 30ms, a system-wide accounting thread updates the credits for each VCPU according to its weight share and resorts the queue if needed. If the credits for a VCPU are negative, Xen assigns "over" priority to this VCPU since it has consumed more than its share. If the credits are positive, it is assigned "under" priority. Every 10ms, Xen updates the credits of the currently running VCPU based on its running time. In order to improve a virtual machine's I/O performance, if a VCPU is woken up (e.g., because an IO request completes) and it has credits left, it will be given "boost" priority and immediately scheduled. After the boosted VCPU consumes a non-negligible amount of CPU resources, then Xen resets the priority to "under".

As this is a weight-based scheduler, it primarily focuses on allocating coarse grained shares of CPU to each virtual machine. The Boost mechanism is relied upon to improve performance of interactive applications, but as shown previously, it has limited effect.

TPC-W 600 clients	Alone	+WC (min weight)
Avg. Resp. Time	25 msec	175.5 msec
Avg. CPU Utilization	84.8%	91.1%
Running (sec)	605.9	656.4
Runnable (sec)	1.9	524.4
Blocked (sec)	1092.4	520.2

Table 1: Xen Credit Scheduler statistics when running a web application alone or with a Word Count VM.

Table 1 shows how much time was spent in each scheduler state when a TPCW VM is run either alone or with a VM running the Word Count Hadoop job that has been given the lowest possible scheduler weight. As was shown in Figure 1, this significantly affects TPCW performance, raising average response time by seven times. We find that the Credit Scheduler weight system does do a good job of ensuring that TPCW gets the overall CPU time that it needs—the CPU utilization (out of 200% since it is a 2-core machine) and time spent in the Running state are similar whether TPC-W is run alone or with word count. In fact, TPC-W actually gets more CPU time when run with word count, although the performance is substantially worse. While the overall CPU share is similar, the timeliness with which TPC-W is given the CPU becomes very poor when word count is also running. The time spent in the Runnable state (i.e., TPC-W could be servicing requests) rises substantially, causing the delays that increase response time.

This happens because Credit uses coarse grain time accounting, which means that 1) at times TPC-W may be woken up to handle IO, but it is not able to interrupt Hadoop; and 2) at times Hadoop may obtain boost priority and interrupt TPC-W if it is at the beginning of an accounting interval and has not yet used up its quantum.

4.2 Minimal Interference CPU Scheduler

Our goal is to run processor or data intensive virtual machines in the background, without affecting the more important interactive services. Therefore, we have modified the Xen scheduler to define a new extra low priority class. Virtual machines of this class are always placed at the end of the Runnable queue, after any higher priority VMs. We also adjust the Boost priority mechanism so that "background" VMs can never be boosted, and so that if a regular VM is woken up due to an I/O interrupt, it will always be able to preempt a background VM, regardless of its current priority (i.e., under or over).

This scheduling algorithm minimizes the potential CPU interference between interactive and Hadoop virtual machines, but it can cause starvation for background VMs. To prevent this, we allow a period, p, and execution time, e, to be specified. If over p seconds the VM has not been in the Running state for e milliseconds, then

its priority is raised from background to over. After it is scheduled for the specified time slice, it reverts back to background mode. We use this to ensure that Hadoop VMs do not become completely inaccessible via SSH, and so they can continue to send heartbeat messages to the Hadoop job scheduler. While this mechanism is not necessary when running interactive VMs that typically leave the CPU idle part of the time, it can be important if MIMP is run either with CPU intensive foreground tasks, or with a very large number of interactive VMs.

5. PROGRESS AWARE DEADLINE SCHEDULING IN HADOOP

A Hadoop Job is broken down into multiple tasks, which each perform processing on a small part of the total data set. When run on dedicated servers, the total job completion time can be reliably predicted based on the input data size and previously trained models [25, 17]. The challenge in MIMP is to understand how job completion times will change when map tasks are run on servers with variable amounts of spare capacity. Using this information, MIMP then instructs the Hadoop Job Tracker on how to allocate "slots" (i.e., available shared or dedicated workers) to each job.

Monitoring Cluster Resource Availability: MIMP monitors resource usage information on each node to help guide task placement and prevent overload. MIMP runs a monitoring agent on each dedicated and shared node, and sends periodic resource measurements to the centralized MP Job Scheduler component. MIMP tracks the CPU utilization and disk read and write rates of each virtual machine on each host. These resource measurements are then passed on to the modeling and task scheduling components as described in the following sections.

5.1 Modeling Background Hadoop Jobs

MIMP uses Task Completion Time models to predict the *progress rate* of different job types on a shared node with a given level of resources. As shown previously in Figure 2, each job needs its own task completion time model. The model is trained by running map tasks on shared nodes with different available CPU capacities. This can either be done offline in advance, or the first set of tasks for a new job can be distributed to different nodes for measurement, and then a model can be generated and updated as tasks complete. Our current implementation assumes that all jobs have been trained in advance on nodes with a range of utilization levels. Once a job has been trained for one data input size, it can generally be easily scaled to accurately predict other data sizes [17].

Job Progress: The progress model for a job of type j is a function that predicts the *task completion time* on a shared node with residual capacity r. From Figure 2 we see that this relationship is highly non-linear, so we use a double exponential formula, *exp2*, provided by MATLABs Non-linear Least Squares functionality :

$$TCT_j(r) = a * e^{b*r} + c * e^{d*r} \quad (1)$$

where a, b, c, and d are the coefficients of the regression model trained for each job. The coefficients b and d represent the rate at which $TCT_j(r)$ exponentially grows. In order to compare the progress that will be made by a job on an available slot, we use the normalized TCT:

$$NormTCT_j(r) = \frac{TCT_j(r)}{TCT_j(r_{dedicated})} \quad (2)$$

where the denominator represents the task completion time when running on a dedicated node. This allows MIMP to compare the relative speeds of different jobs.

Checking Deadlines: The task completion time model can then be used to determine if a job will be able to meet its deadline given its current slot allocation. MIMP tracks a resource vector, R, for each active job. The entry R_i represent the amount of resources available on worker slot i that this job has been allocated for use: 100% for an available dedicated slot, 0% for a slot assigned to a different job, or something in between for a shared slot allocated to this job. If there is currently $t_{remaining}$ seconds until the job's deadline, then MIMP can check if it will meet its deadline using:

$$CompletableTasks(j, R) = \sum_{slot\ i=1}^{n} \frac{t_{remaining}}{TCT_j(R_i)} \quad (3)$$

If $CompletableTasks(R)$ is greater than n_{tasks}, the number of remaining tasks for the job, then it is on track to meet its deadline.

Map Phase Completion Time: We can also obtain a direct prediction of the map phase completion time using:

$$CompletionTime(j, R) = n_{tasks} \sum_{slot\ i=1}^{n} \frac{TCT_j(R_i)}{n} \quad (4)$$

which estimates the total map phase completion time based on the average TCT of each slot and the number of remaining tasks.

Data Node I/O: Hybrid clusters like the ones considered in MIMP are particularly prone to disk I/O bottlenecks since there may be a relatively small number of dedicated nodes acting as the data store. If too many I/O intensive tasks are run simultaneously, task completion times may begin to rise [5]. To prevent this, we use a model to predict the I/O load incurred by starting a new map task. During MIMP model training phase, we measure the read request rate sent to the data nodes by a dedicated worker. Since I/O accesses can be erratic during map tasks, we use the 90^{th} percentile of the measured read rates to represent the I/O required by a single worker, per data node available. In order to calculate the read I/O load incurred by a new task on a shared worker, we use the normalized TCT from Equation 2 as a scaling factor:

$$IO_j(r) = \frac{read_j^{90^{th}}}{NormTCT_j(r)} \quad (5)$$

to predict its I/O requirement. This can then be used to determine whether running the task will cause the data nodes to become overloaded, as described in the following section.

5.2 Progress Aware Earliest Deadline First

We now present two standard Hadoop job schedulers, and then discuss how we enhance these in MIMP so that it accounts for both deadlines and the relative benefit of assigning a worker to each job.

FIFO Scheduler: The simplest approach to scheduling Hadoop jobs is to service them in the order they arrive—all tasks for the first job are run until it finishes, then all tasks of the second job, and so on. Not surprisingly, this can lead to many missed deadlines since it has no concept of more or less urgent tasks to perform.

EDF Scheduler: Earliest Deadline First (EDF) is a well known scheduling algorithm that always picks the job with the earliest deadline when a worker becomes available; that job will continue to utilize all workers until it finishes or a new job arrives with a smaller deadline. EDF is known to be optimal in terms of preventing deadline misses as long as the system is preemptive and the cluster is not over-utilized. In practice, Hadoop has somewhat coarse grained preemption—each task runs to completion, but a job can be preempted between tasks. It is also difficult to predict whether a Hadoop cluster is over-utilized since tasks do not arrive

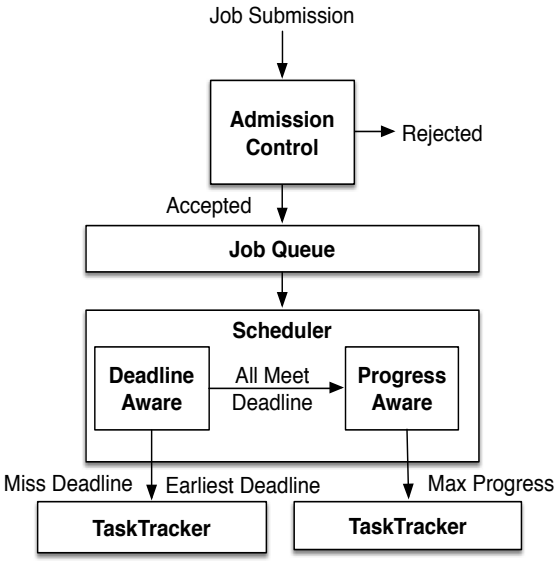


Figure 4: Admission Control Architecture

on strict schedules as is typically assumed in Real Time Operating Systems work. Despite this, we still expect EDF to perform well when scheduling jobs since it will organize them to ensure they will not miss deadlines.

MP Scheduler: Our Maximum Progress (MP) Scheduler uses the models described in the previous section to enhance the EDF scheduler. Whenever a worker slot becomes free, we determine which job to assign to it based on the following criteria.

- First, MP examines each job in the queue to determine whether it can meet its deadline *with its currently allocated set of slots* using Equation 3. If one or more jobs are predicted to miss their deadline, then MP allocates the slot to whichever of those jobs has the closest deadline and returns.
- If all jobs are currently able to meet their deadlines, MP considers each job in the queue and uses Equation 2 to calculate its normalized task completion time if assigned the resources of the free slot. It finds the job with the smallest $normTCT$ value, since that job is best matched for the available resources.
- Before assigning the slot, MP calculates the IO cost of running the selected job using Equation 5. If starting a new task of this type will cause any of the data nodes to become overloaded, then the job with the next highest $normTCT$ is considered, and so on.

This algorithm ensures that the selected job is either currently unable to meet its deadline, or the job that will make the most progress with the slot, without causing the data nodes to become overloaded.

5.3 Admission-Control

High number of incoming jobs can cause the cluster resources to be overloaded. When cluster resources are over-utilized, naturally, due to resource contention, we will have more jobs that will miss their deadlines than usual. EDF scheduler does not have a mechanism that can control resource overload, instead, it tries to schedule as many jobs at it can fit in the incoming job queue. This aggressive approach not only can make a new job miss the deadline but can also cause other jobs in the queue to miss their deadlines as well.

To prevent a cluster from becoming overloaded, we present Admission Control mechanism in MIMP scheduler. The Admission Control maximizes the number of jobs that can meet their deadlines by accepting or rejecting a new incoming job to be executed in the cluster based on whether or not it will overload the resources of that cluster.

We assume that jobs with earlier deadlines always have the highest priority. This means that if a new job arrives with an earlier deadline than some job already in the queue, it may be accepted even though this could cause the job in the queue may now miss its deadline. A job will only be accepted into the queue if MIMP predicts it can meet its deadline with the available resources.We design the admission controller based on the following criteria:

- When a new job J_{new} with deadline $Deadline_{new}$ is submitted, the controller finds the jobs $J_1, J_2, \ldots, J_N$ in the job processing queue that have an earlier deadline than J_{new}. For instance, if J_1 and J_2 are the only jobs that have an earlier deadline than J_{new}, then the controller will find the jobs J_1, J_2, J_{new} to be evaluated. Remember that we assume the jobs J_{new+1}, J_{new+2}, ..., J_N will meet their deadlines.
- In order to accept J_{new}, MIMP must determine if doing so will cause any higher priority job (i.e., one with an earlier deadline) to miss its deadline. The Admission Controller does this by estimating the processing time required by each higher priority job. To make a conservative estimate, we calculate each job's estimated completion time assuming it runs by itself on the cluster, using equation 6. In practice, MIMP will be progress-aware, and will attempt to assign each job to its most efficient slots, so it is likely that this will be an overestimate; this makes the admission controller more conservative, reducing the likelihood of a missed deadline.

$$JCT(j,R) = n_{task\_remaining} \sum_{slot\ i=1}^{n} \frac{TCT_j(R_i)}{n^2} \quad (6)$$

- Once we estimate the processing time, the Admission Controller accepts the new job if and only if all the jobs that has their deadlines lesser than $Deadline_{new}$ can complete successfully. The acceptance condition for a new job is shown in equation 7. If this condition is not met, there is a high probability that some jobs will miss their deadlines and we thus reject J_{new}.

$$Deadline_{new} \geq \sum_{job\ i=1}^{new} JCT_i \quad (7)$$

Figure 4 shows the decision making process of our MIMP scheduler with Admission Control. First, when a new job is submitted, the admission controller, based on the above criteria predicts the new jobs completion time and then makes a decision whether to accept or reject the job. In the case where a job is accepted, the newly accepted job would then be put into a job processing queue where it waits to get scheduled. When the scheduler finds some free task slots that free up, it allocates those free slots to the jobs in the queue based on deadline-aware or progress-aware scheduling. Finally, once the free slots are allocated to the jobs, they are run on their assigned task trackers.

6. EVALUATION

In this section, we present the evaluation results to validate the effectiveness of reducing interference using our Minimizing Interference Scheduler. We then evaluate the accuracy and prediction

error of the TCT models and show the performance improvement as we progressively satturate the data nodes with I/O. We then show how the admission controller improves the performance when the cluster is overloaded. We also describe details of our testbed and three different job scheduling alogorithms in the case study.

6.1 Setup - machines, benchmarks

For our tests we use Xen 4.2.1 with Linux 3.7, running on a heterogeneous cluster of Dell servers with Intel E5-2420 and Xeon X3450 CPUs with each having 16GB of RAM. The E5-2420 has six physical cores at 1.90GHz with 64KB of L1 cache and 256KB of L2 cache per core, and a shared 15MB L3 cache, and X3450 has four physical cores at 2.67GHz with 128KB of L1 cache and 1MB of L2 cache per core, and a shared 8MB L3 cache. The Hadoop version is 1.0.4.

Our virtual cluster contains 13 physical servers with 7 servers running 4VMs per server, two for web server with 1GB of RAM and 2VCPUs each and another two for Hadoop with 4GB of RAM with 2VCPUs each. Four servers run 6 dedicated Hadoop VMs (each with their own disk). Two more servers run web clients.

The web server VM is *always* pinned to shared CPU cores and Hadoop VMs are pinned to either two dedicated or shared CPU cores depending on the server it runs. Xens Domain-0, which hosts drivers used by all VMs, is given the servers remaining cores.

Benchmarks: We use interactive workloads and batch workloads as our workloads. For transactional workloads, we use two applications: TPC-W, which models a three-tier online book store and Micro Web App, a PHP/MySQL application that emulates a multi-tier application and allows the user to adjust the rate and type of requests to control of CPU computation and I/O activities performed on the test system. For batch workloads, we choose the following Hadoop jobs. *PiEstimator:* estimates Pi value using 1 million points; *WordCount:* computes frequencies of words in 15GB data; *Sort:* sorts 18GB data; *Grep:* finds match of randomly chosen regular expression on 6GB data; *TeraSort:* samples the 1GB input data and sort the data into a total order; *Kmeans:* clusters 6GB of numeric data. Both Kmeans and Grep are divided into two types of jobs.

6.2 Minimizing Interference Scheduler

We start our evaluation by studying how our Minimal Interference Scheduler is able to provide greater performance isolation when mixing web and processor intensive tasks. We repeat a variation of our original motivating experiment, and adjust the number of TPC-W clients when running either Pi or Word Count Hadoop jobs on a shared server. As expected, Figure 5(a) shows that the response time when using Xen's default scheduler quickly becomes unmanageable, only supporting about 500 clients before interference causes the response time to rise over 100ms. In contrast, our MI scheduler provides performance almost equivalent to running TPC-W alone, allowing it to support twice the throughput before response time starts to rise. A closer look at the response time CDF in Figure 5(b) illustrates that MIMP incurs only a small overhead when there are 700 clients.

6.3 Task Affinity Models

In this section, we illustrate the accuracy of our task completion time models and how they guide slot allocation.

6.3.1 TCT models

Figure 6 shows the training data and model curves generated by MIMP (green curve is obtained from our model). Each Hadoop VM has one core that is shared with a Micro Web App VM. We run

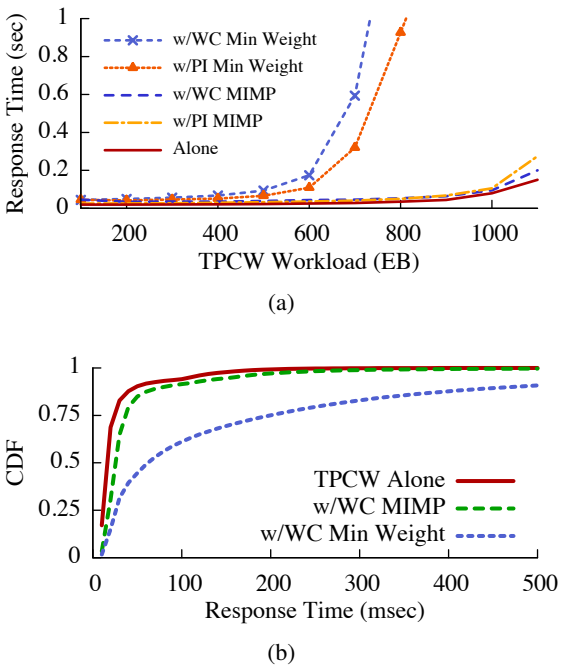


Figure 5: The MIMP scheduler provides response time almost equivalent to running a web application alone.

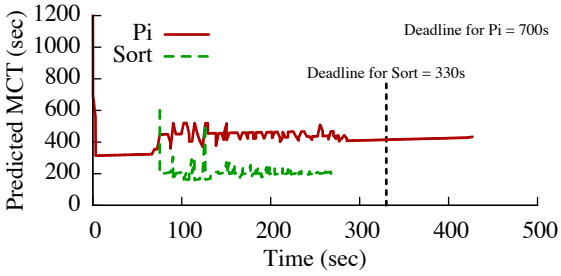


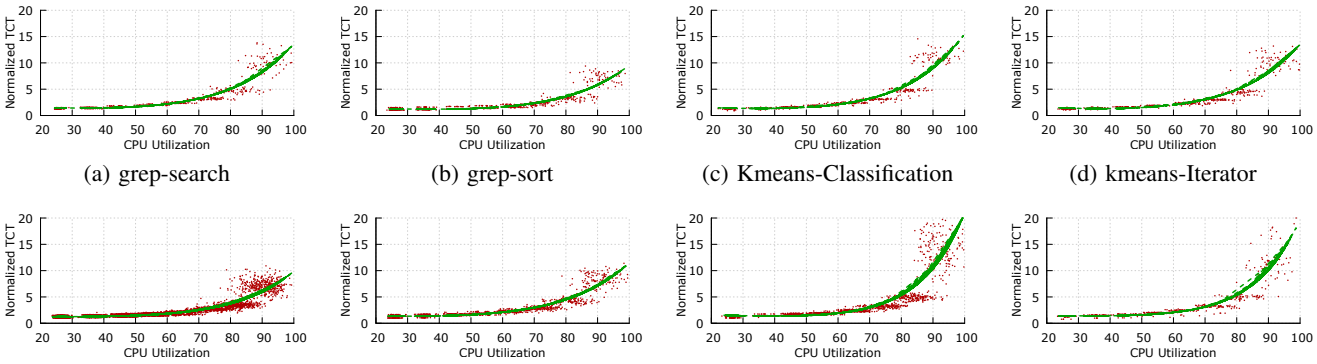
Figure 7: MIMP accurately predicts Map-phase completion time (MCT) and appropriately allocates slots to meet deadlines.

a set of Hadoop jobs across our cluster using a randomly generated web workload ranging from 10 to 100% CPU utilization for each shared node. The x-axis represents the CPU utilization of the web VM before each task is started; we normalize the measured task completion time by the average TCT when running the same type of task on a node with no web workload.

These figures show the wide range of TCTs that are possible even for a fixed level of CPU availability. This variation occurs because the MI scheduler can give an unpredictable amount of CPU to the low priority Hadoop VM depending on fluctuations in the web workload.[1] Thus, it is quite difficult to make accurate predictions, although our models do still capture the overall trends.

When we apply these models to our case study workload described in Section 6.6, we find that 57% of the time our models over predict task completion time, and that the average over prediction is by 35%. The average under prediction is 29%. This is good since we would prefer our model to over predict task completion times, causing it to be more conservative, and thus less likely to miss deadlines.

[1]The variation is *not* simply related to differences in data node I/O levels, since even the PI job (which does not make any storage accesses) sees a particularly high variation in TCT.



(a) grep-search (b) grep-sort (c) Kmeans-Classification (d) kmeans-Iterator

(e) Pi (f) Sort (g) Terasort (h) Word Count

Figure 6: MIMP trains different models for each Hadoop job type.

Jobs	NRMSE	Jobs	NRMSE
Pi	7.74%	KmeansClass	8.61%
Sort	8.02%	KmeansIterator	9.53%
Terasort	8.17%	Grepsearch	8.26%
Wordcount	7.25%	Grepsort	9.64%

Table 2: NRMSE

6.3.2 *Total Map phase time prediction*

The TCT models of MIMP are used to predict whether a job will meet its deadline given its current slot allocation. Figure 7 shows how the predictions change as slots are allocated and removed from a job. We first start a Pi job at time 0 with a deadline of 700 seconds. Within 10 seconds, Pi has been allocated all of the available slots, so its predicted map phase completion time (MCT) quickly drops to about 370 seconds. At time 80 sec, a Sort job is started, causing the MIMP scheduler to divide the available slots between the two jobs. It reduces from Pi, but only enough to ensure that Sort will finish before its deadline. The predicted MCT of each job fluctuates as the number of slots it is given varies, but it remains accurate throughout the run.

6.3.3 *Prediction Error*

To evaluate the accuracy of the TCT models, we compare the error between the predicted task completion times and the actual task completion times. The accuracy is measured by the normalized root mean square error (NRMSE). We use the following formula 8 to calculate the RMSE. Figure 8 shows the RMSE for the different Hadoop jobs. The solid bars represent the difference between the fastest and slowest task completion time, while the error bars represent the RMSE; this shows that the error of our predictor is small compared to the natural variation in completion times on different servers.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{predict,i})^2}{n}} \quad (8)$$

Where X_{obs} is the observed values and $X_{predict}$ is the predicted values at time/place i. We use the formula 9 to calculate the NMRSE. Table 2 shows the NRMSE values for different Hadoop jobs.

$$NRMSE = RMSE/(X_{obs,max} - X_{obs,min}) \quad (9)$$

To illustrate how the prediction error affects the MIMP scheduler, we injected errors into the prediction. We run a trace of Hadoop jobs (pi, terasort) in 4 shared nodes for 1 hour. Figure 9 shows that the total task completion time slightly fluctuates when the insert error was within 10%. The inserted error does not influence the scheduler obviously since from Figure 8 we can clearly see that our TCT model prediction error is around 10%. However, when the error is increased from 15% to 20%, the total time completion time increases significantly from 12100 second to 13790 second.

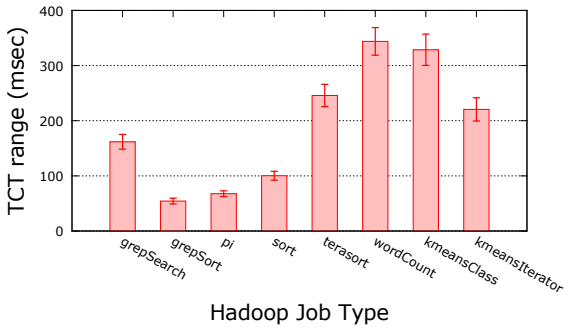


Figure 8: TCT prediction error

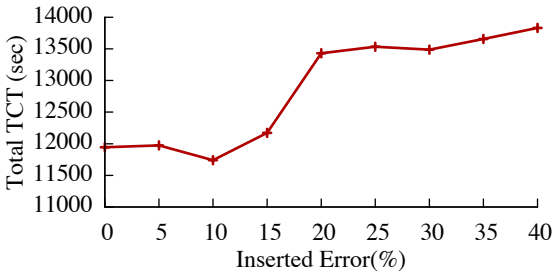


Figure 9: Impact of TCT error

6.4 Data Node I/O Saturation

Allocating more shared worker nodes to a Hadoop job will only increase performance if the data nodes that serve that job are not overloaded. To evaluate this behavior, we measure the map phase completion time (MCT) when the number of available shared nodes is increased from 0 to 10; there are also three dedicated nodes that run map tasks and act as data nodes. Each shared node runs a web server that leaves 164% of the CPU available for the Hadoop VM.

Figure 10 shows the impact of the data node IO bottleneck on the Pi and Grep jobs. We normalize the map phase completion time by the measured time without any shared nodes. Initially, both job types perform similarly, but the Grep job soon sees diminishing returns as the data nodes become saturated. In contrast, Pi is an entirely CPU bound job, so it is able to scale optimally, i.e., ten

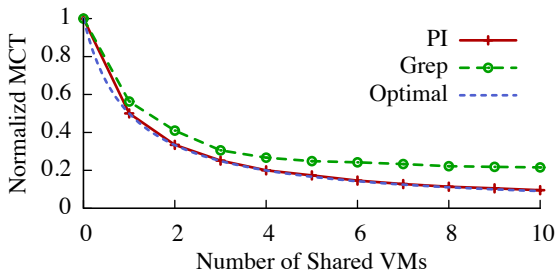


Figure 10: Data intensive Hadoop tasks see diminishing returns once the data nodes become saturated.

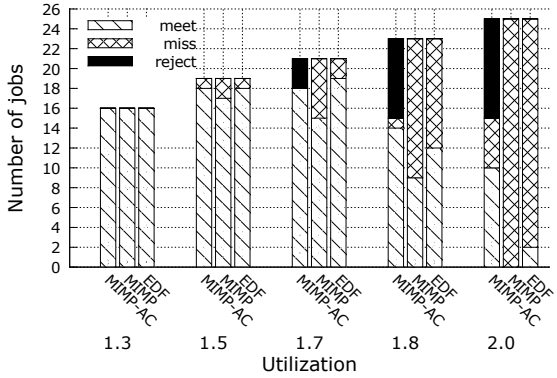


Figure 11: Meet, Miss, Reject jobs vs Utilization

shared servers lowers the completion time by a factor of nearly ten.

6.5 Admission Control

To evaluate how our admission controller affects the performance when the cluster is overloaded, we varied the level of cluster overload and then compare the number of jobs meeting deadlines, missing deadlines, or being rejected among MIMP with admission control(MIMP w/ac), MIMP without admission control and EDF scheduler. With different cluster utilization, we generate 30 minute traces using pi and terasort jobs. Figure 11 shows that when the cluster is slightly overloaded, the number of met deadlines is almost the same for the MIMP w/AC, MIMP and EDF scheduler. However, when the cluster is highly overloaded, most of the jobs will miss deadline for EDF and MIMP. Since MIMP w/AC can detect the overload and reject some jobs, it ensures that the other jobs in the queue will meet their deadlines.

To show how our progress-aware MIMP scheduler affects TCT with varied cluster utilization, in figure 12, we compare the average TCT for MIMP w/AC and EDF scheduler while changing the cluster utilization. We can see that if the cluster is lightly loaded, for example when the utilization is 1, the average TCT of terasort task with MIMP w/AC is less than the EDF scheduler. This is because, while using MIMP w/AC all the jobs can meet their deadline, so our progress-aware rule is used to select which jobs to run. For the same utilization, the average TCT of Pi task in MIMP w/AC is more than the one in EDF scheduler.

One reasonable explanation is that the TCT model for Pi is lower than the TCT model for terasort. At the start of the experiment, almost all slots in the scheduler are running Pi task. However, in order to avoid the terasort job from missing its deadline, the MIMP scheduler also chooses the terasort task to run. In addition, the Hadoop slot which has a low CPU utilization would have shorter schedule time period than the one with high CPU utilization. Therefore, terasort task has a greater chance to gain the slot

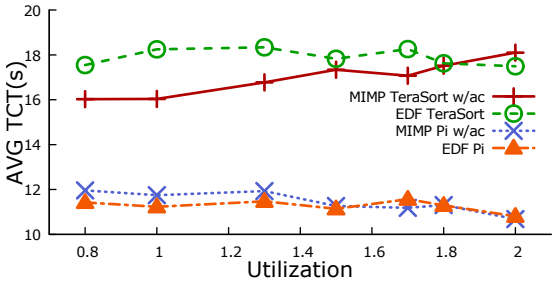


Figure 12: Average TCT vs Utilization

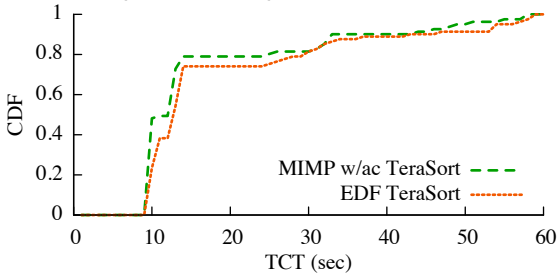


Figure 13: TeraSort TCT CDF

with low CPU utilization than the Pi job. Furthermore, with the utilization increasing, the behavior of MIMP scheduler will be more similar with EDF scheduler. The deviation of TCT between the two types of scheduler will decrease around zero.

Figure 13 and 14 shows CDFs of the TCT for terasort and pi with the 0.8 utilization. One can see that when the utilization is light, the MIMP scheduler tends to be progress-aware, and TCTs of terasort and Pi in MIMP scheduler are shorter than the TCTs in EDF scheduler. In this case, the progress-aware scheduler is only providing a modest benefit since the experimental cluster only has four slots, and only one of those slots provides a significant boost when terasort jobs are scheduled there (as shown by the larger number of jobs finishing within 10 seconds for terasort. We expect that a larger cluster with a more diverse set of servers and jobs would provide even greater benefit.

6.6 Case study

To evaluate our overall system, we perform a large scale experiment where we run a trace of Hadoop jobs on a shared cluster and evaluate the performance of three different job scheduling algorithms. We use a total of 20 Hadoop VMs each with two cores: 6 dedicated hosts, 6 with a light web workload (20-35% CPU utilization), 6 with a medium load (85-95%), and 2 that are highly loaded (130-170%). We generate these workloads based on our observations of the DIT(Division of Information Technology) and Wikipedia data traces, although we use a higher overall utilization level than was found in those traces since this puts more stress on

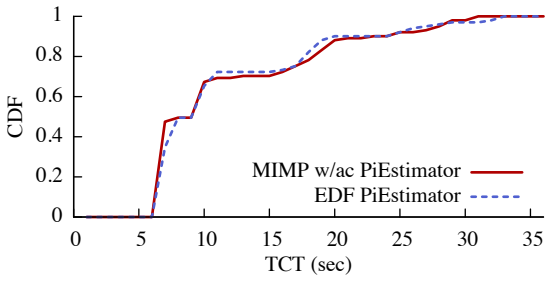
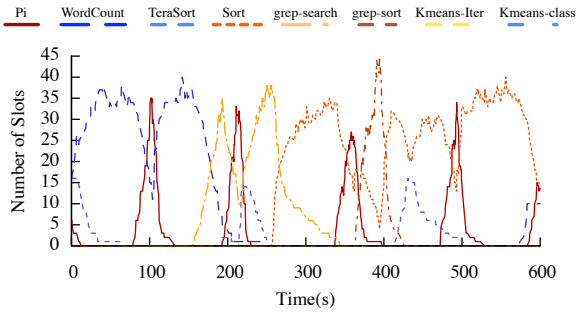
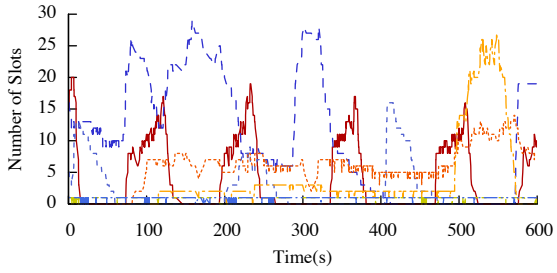


Figure 14: PiEstimator TCT CDF



(a) EDF



(b) MIMP

Figure 15: Slot allocation for the first 10 minutes - MIMP vs. EDF

	FIFO	EDF	MIMP
#Miss Deadline	67	2	1
Total TCT(h)	72.17	72.61	67.95
#Failed Jobs	17	0	0
#Failed Tasks	1080	1	0
Avg Lateness(s)	217.8	6.2	7.9

Table 3: Workload Performance Statistics

making intelligent scheduling decisions. The web workloads are generated using httperf clients connected to our Micro Web App benchmark.

We generate a random Hadoop trace composed of our six representative job types. Jobs of each type arrive following a Poisson distribution; the mean inter-arrival period is used as the deadline for that type of job, with the exception of KMeans jobs which we set to have no deadline. The job trace lasts 2.5 hours and contains 174 jobs in total.

Scheduler Comparison: Table 3 shows the performance statistics of each job scheduler when processing this trace. Unsurprisingly, FIFO performs very poorly, missing deadlines for 67 out of 174 jobs, with an additional 17 jobs failing to complete at all. The EDF scheduler performs much better, but still misses two jobs, with an average lateness of 6.2 seconds. The total task completion time (i.e., the sum of all successful task execution times) is 72.61 hours for EDF; FIFO is slightly lower only because it has failed tasks which do not add to the total.

MIMP provides the best performance, missing only one job deadline by 7.9 seconds. Most importantly, it achieves this while using 4.66 hours less total execution time than EDF. This is possible because MIMP makes smarter decisions about which tasks to run on which nodes, better matching them to the available resources.

Job Distribution: To understand why MIMP improves performance, we now examine how each scheduler assigns workers to jobs. Figure 15(b) shows the number of slots assigned to each job during a 10 minute portion of the trace. This shows that EDF assigns all task slots to whatever job has the earliest deadline, even

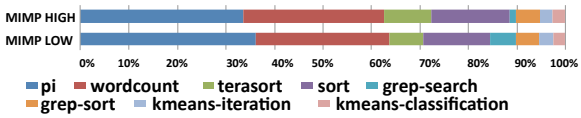


Figure 16: Job Distribution on high vs. low load server

though some slots may be better suited for a different job type. In contrast, MIMP tends to run multiple jobs at the same time, allocating the best fitting slots to each one. While this can result in longer job completion times, MIMP still ensures that all jobs will meet their deadlines and improves overall efficiency due to resource affinity.

Figure 16 breaks down the percent of tasks completed by the highly loaded and lightly loaded servers when using MIMP. Since each job has a different arrival period, some jobs (such as Pi) have more total tasks than infrequent jobs (such as K-means). However, the results still follow our expectations based on the models shown in Figure 6. For example, Pi and grep-search have particularly high normalized TCT when the web server utilization rises, so MIMP runs relatively fewer of those tasks on the highly loaded servers. In contrast, the completion time of sort does not change much, so MIMP runs more of those tasks on the highly loaded servers.

7. RELATED WORK

Virtual Machine Scheduling: Several previous works propose to improve the efficiency of Xen Credit CPU scheduler. For example, [3] proposes a modification to the scheduler that asynchronously assigns each virtual CPU to a physical CPU in order to reduce CPU sleep time. Lin et al. [12] developed *VSched* that schedules batch workloads within interactive VMs without compromising the usability of interactive applications. The drawback of this system is that it was not designed to run on a cluster. Xi et al., use techniques from Real-Time scheduling to give stricter deadline guarantees to each virtual machine [19].

Other work has looked at avoiding interference between these tasks by careful VM placement [7, 21] or dedicating resources [10]. Paragon [7] proposes a heterogeneous and interference-aware data center scheduler. The system prefers to assign the applications on the heterogeneous hardware platform that the application can benefit from and have less interference with the co-scheduled applications. MIMP extends our preliminary study [23] to reduce interference through minor changes to Xen scheduler and then uses the residual resources for big data applications.

To improve I/O performance, Xu et al. [20] propose the use of *vTurbo* cores that have a much smaller time-slice compared to normal cores, reducing the overall IRQ processing latency. Cheng et al. [2] improves I/O performance for Symmetric MultiProcessing VMs by dynamically migrating the interrupts from a preempted VCPU to a running VCPU thereby avoiding interrupt processing delays. Our current focus is on shared environments where disk I/O is not the bottleneck for interactive applications, but view this as important future work.

Hadoop Scheduling & Modeling: Job scheduling in MapReduce environments has focused on topics like fairness [9, 22] and dynamic cluster sharing among users [16]. HybridMR [17] considered running Hadoop across mixed clusters composed of dedicated and virtual servers, but does not consider VM interference. Bu et al. [1] propose a new Hadoop scheduler based on the existing fair scheduler. They present an interference and locality-aware task scheduler for MapReduce in virtual clusters and design a task performance prediction model for an interference-aware policy. Morton et al. [14] provide a time-based progress indicator for a series

of Map Reduce jobs, which can be used to predict job completion times. Polo et al. provide a system to dynamically adjust the number of slots provided for Map Reduce jobs on each host to maximize the resource utilization of a cluster and to meet the deadline of the jobs [15]. [18] decides the appropriate number of slots allocated to Map and Reduce based on the upper and lower bounds of batch workload completion time obtained from the history of job profiling. Our work is distinct from prior work in that we estimate the job completion time of the batch jobs that are running in clusters with unpredictable resource availability due to other foreground applications.

Previous works [5, 8, 11, 13] have shown heterogeneous cluster designs wherein a core set of dedicated nodes running batch jobs are complemented by residual resources from volunteer nodes, or in some cases using "spot instances" from EC2 [4]. The closest work to ours is by Clay et al [5]. They present a system that determines the appropriate cluster-size to harness the residual resources of under utilized interactive nodes to meet user-specified deadlines and minimize cost and energy. Our work extends this by focusing on how groups of jobs should be scheduled across a shared cluster in order to both minimize interference and meet job deadlines.

8. CONCLUSIONS

Virtualization allows servers to be partitioned, but resource multiplexing can still lead to high levels of performance interference. This is especially true when mixing latency sensitive applications with data analytic tasks such as Hadoop jobs. We have designed MIMP, a Minimal Interference, Maximal Progress scheduling system that manages both VM CPU scheduling and Hadoop job scheduling to reduce interference and increase overall efficiency.

MIMP works by exposing more information to both the Hadoop job scheduler and the Xen CPU scheduler. By giving these systems information about the priority of different VMs and the resources available on different servers, MIMP allows cluster utilization to be safely increased. MIMP allows high priority web applications to achieve twice the throughput compared to the default Xen scheduler, and has response times nearly identical to running the web application alone. Despite the increased variability this causes in Hadoop task completion times, MIMP is still able to meet more deadlines than an Earliest Deadline First scheduler and lowers the total execution time by nearly five hours in one of our experiments.

Acknowledgments: We thank the reviewers for their helpful suggestions. This work was supported in part by NSF grant CNS-1253575 and the National Natural Science Foundation of China Grant No. 61370059, 61232009, and Beijing Natural Science Foundation Grant No. 4122042.

9. REFERENCES

[1] X. Bu, J. Rao, and C.-z. Xu. Interference and Locality-aware Task Scheduling for MapReduce Applications in Virtual Clusters. In *Proc. of HPDC*, 2013.

[2] L. Cheng and C.-L. Wang. vBalance: using interrupt load balance to improve I/O performance for SMP virtual machines. In *Proc. of SOCC*, 2012.

[3] T. Chia-Ying and L. Kang-Yuan. A Modified Priority Based CPU Scheduling Scheme for Virtualized Environment. *Int. Journal of Hybrid Information Technology*, 2013.

[4] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See spot run: using spot instances for mapreduce workflows. In *Proc. of HotCloud*, 2010.

[5] R. B. Clay, Z. Shen, and X. Ma. Accelerating Batch Analytics with Residual Resources from Interactive Clouds. In *Proc. of MASCOTS*, 2013.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51, 2008.

[7] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proc. of ASPLOS*, 2013.

[8] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of SOCC*, 2011.

[9] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proc.of SOSP*, 2009.

[10] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. NoHype: virtualized cloud infrastructure without the virtualization. In *Proc. of ISCA*, 2010.

[11] G. Lee, B.-G. Chun, and H. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proc. of HotCloud*, 2011.

[12] B. Lin and P. A. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proc. of Super Computing*, 2005.

[13] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang. MOON: MapReduce On Opportunistic eNvironments. In *Proc. of HPDC*, 2010.

[14] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of MapReduce pipelines. In *Proc. of ICDE*, 2010.

[15] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé. Resource-aware adaptive scheduling for mapreduce clusters. In *Proc. of Middleware*, 2011.

[16] T. Sandholm and K. Lai. Dynamic proportional share scheduling in Hadoop. In *Proc. of JSSPP*, 2010.

[17] B. Sharma, T. Wood, and C. R. Das. HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers. In *Proc. of ICDCS*, 2013.

[18] A. Verma, L. Cherkasova, V. Kumar, and R. Campbell. Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle. In *Proc. of NOMS*, 2012.

[19] S. Xi, J. Wilson, C. Lu, and C. Gill. RT-Xen: towards real-time hypervisor scheduling in xen. In *EMSOFT*, 2011.

[20] C. Xu, S. Gamage, H. Lu, R. Kompella, and D. Xu. vTurbo: accelerating virtual machine I/O processing using designated turbo-sliced core. In *Proc. of Usenix ATC*, 2013.

[21] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: Avoiding Long Tails in the Cloud. In *Proc. of NSDI*, 2013.

[22] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*, 2010.

[23] W. Zhang, S. Rajasekaran, and T. Wood. Big Data in the Background: Maximizing Productivity while Minimizing Virtual Machine Interference. In *Proc. of Workshop on Architectures and Systems for Big Data*, 2013.

[24] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu. Mimp: Deadline and interference aware scheduling of hadoop virtual machines. *CCGrid*, 2014.

[25] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Performance Modeling and Optimization of Deadline-Driven Pig Programs. *ACM TAAS*, 8, 2013.