1-1-2010

Cost-Based Dynamic Job Rescheduling: A Case Study of the Intel Distributed Computing Platform

Linh T.X. Phan
University of Pennsylvania

Zhuoyao Zhang
University of Pennsylvania

Saumya Jain
University of Pennsylvania

Godfrey Tan
Intel Corporation

Boon Thau Loo
University of Pennsylvania, boonloo@cis.upenn.edu

See next page for additional authors

Cost-Based Dynamic Job Rescheduling: A Case Study of the Intel Distributed Computing Platform

Abstract
We perform a trace-driven analysis of the *Intel Distributed Computing Platform (IDCP)*, an Internet-scale data center based distributed computing platform developed by Intel Corporation for massively parallel chip simulations within the company. IDCP has been operational for many years, and currently is deployed "live" on tens of thousands of machines that are globally distributed at various data centers. Our analysis is performed on job execution traces obtained over a one year period collected from tens of thousands of IDCP machines from 20 different pools. Our analysis demonstrates that job completion time can be severely impacted due to job suspension when higher priority jobs preempt lower priority jobs. We then develop cost-based dynamic job rescheduling strategies that adaptively restart suspended jobs, which better utilize system resources and improve completion times. Our trace-driven evaluation results show that dynamic rescheduling enables IDCP to significantly reduce job completion times.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-14.

Author(s)
Linh T.X. Phan, Zhuoyao Zhang, Saumya Jain, Godfrey Tan, Boon Thau Loo, and Insup Lee

Cost-based Dynamic Job Rescheduling: A Case Study of the Intel Distributed Computing Platform

Linh T.X. Phan Zhuoyao Zhang Saumya Jain Harrison Duong
University of Pennsylvania
{linhphan,zhuoyao,saumyaj,hduong}@seas.upenn.edu

Godfrey Tan
Intel Corporation
godfrey.tan@intel.com

Boon Thau Loo Insup Lee
University of Pennsylvania
{boonloo,lee}@cis.upenn.edu

ABSTRACT

We perform a trace-driven analysis of the *Intel Distributed Computing Platform (IDCP)*, an Internet-scale data center based distributed computing platform developed by Intel Corporation for massively parallel chip simulations within the company. IDCP has been operational for many years, and currently is deployed "live" on tens of thousands of machines that are globally distributed at various data centers. Our analysis is performed on job execution traces obtained over a one year period collected from tens of thousands of IDCP machines from 20 different pools. Our analysis demonstrates that job completion time can be severely impacted due to job suspension when higher priority jobs preempt lower priority jobs. We then develop cost-based dynamic job rescheduling strategies that adaptively restart suspended jobs, which better utilize system resources and improve completion times. Our trace-driven evaluation results show that dynamic rescheduling enables IDCP to significantly reduce job completion times.

1. INTRODUCTION

We present a trace-driven analysis of a distributed computing platform for performing compute-intensive operations within a large enterprise. Our main driving application is the *Intel Distributed Computing Platform (IDCP)* developed by Intel Corporation for running concurrently tens of thousands of chip simulations. IDCP has been operational for many years but constantly evolving from a pure job execution engine into a distributed grid computing platform [10] and now a service-oriented computing *cloud* for Intel. The IDCP cloud at Intel consists of tens of thousands of machines that are globally distributed at dozens of data centers. IDCP enables Intel engineers to focus more on their primary job of designing chips and validating through simulations and less on how the computing service is delivered. By pooling together machines all over the enterprise network and coordinating the scheduling of jobs over those machines, IDCP strives to meet the demands of jobs with varying priorities while keeping the system utilization high.

Our study is motivated by the practical needs of IDCP within Intel. Our longer-term vision is to develop a holistic understanding of the resource management challenges of similar computational clouds at the global scale. As data center computing environments mature, there will be a need for such infrastructures to gracefully handle multiple classes of jobs with different priorities and SLAs (Service Level Agreements).

Intel's IDCP platform resembles what is described as a *private cloud* [2]. These are internal data centers operated by a business or an enterprise for internal use. The difference between public and private clouds is that public clouds, such as Amazon's EC2 [1], are available to any user with a credit card. Private clouds like IDCP share with many recent cloud computing efforts similar characteristics and challenges, including deployment at Internet-scale, supporting a large group of heterogeneous concurrent applications, and supporting different quality of service guarantees for different classes of users. To put the relevance in context, the IDCP deployment today is estimated to involve hundreds of machine clusters called *pools*, distributed globally at dozens of data centers with varying wide-area network characteristics, utilizing tens of thousands of heterogeneous multi-core compute machines. At any moment, there are thousands of concurrent jobs with varying priorities and requirements being dispatched by engineers through Intel, and in aggregate, hundreds of millions of jobs per year.

One important challenge that IDCP faces today is the need to accommodate jobs with varying priorities and goals while keeping both the system utilization high and latency low. Specifically, there are two primary job classes sharing common resources but having differing goals: i) high-priority jobs and ii) low priority jobs. High priority jobs typically are used for time-sensitive workload and hence, they can pre-empt lower priority jobs by suspending them so that they can complete as quickly as possible. Low priority jobs are typically less time-sensitive but users of low priority jobs also need to get the simulation results from those jobs in a timely fashion. Based on our analysis of a year-long trace data, we find that high priority jobs can significantly impact the completion time of low priority jobs even when the overall system utilization is relatively low. We study this issue in depth and present preliminary work to improve the completion time of lower priority jobs that get suspended, thereby improving the average job completion time of the entire system.

Our contributions are as follows:

Trace-driven analysis: We present an analysis of job execution traces obtained over a year long period collected from tens of thousands of machines deployed in 20 IDCP pools. To our best knowledge, this is one of the first trace-driven efforts at empirically understanding the infrastructure requirements of an Internet-scale distributed computing platform deployed in various data centers for use by a global enterprise.

Performance bottlenecks: The focus of our analysis is on *goodput* (useful work) achievable in the entire system. From our IDCP traces, we observe that despite only utilizing around 40% of computing resources on average during that period, the IDCP is not currently designed to handle flash crowds. There have been severe cases of slow turnaround of jobs during peak times at certain pools, primarily caused by high priority jobs preempting low priority jobs. The introduction of high priority jobs tends to be bursty, isolated to particular pools at a few sites, and can last for several days; i.e., the overall resource utilization at each pool can be highly diverse.

Dynamic cost-based rescheduling: Using a trace-driven simulator developed in-house at Intel, we perform an initial feasibility study on the potential benefits of selectively and dynamically rescheduling preempted (and hence suspended) jobs to available resources elsewhere. We adopt a cost-based approach, where rescheduling decisions are made based on expected job completion time in each candidate pool. The completion time of a job is the time from when the job enters the system until it completes. Our trace-driven evaluation results suggest that dynamic rescheduling enables IDCP to significantly reduce job completion times.

2. ARCHITECTURE AND MOTIVATION

We provide an architectural overview of IDCP and highlight its performance insights gathered from traces.

2.1 IDCP Architecture

As shown in Figure 1, IDCP is designed as a hierarchical system consisting of several dozens of *physical pools*, each of which consists of hundreds or thousands of multi-core machines. Intel has many sites of operation across the globe. Each typical site has many physical pools that are often located in multiple data centers. To simplify computing operations and make the geography of physical pools transparent to the users, IDCP deploys a virtual pool at each site. A virtual pool accepts job submissions from users at that site, and then distributes jobs across one or more physical pools located at the same site or different sites according to resource availability and IDCP configurations. Thus, IDCP inherently supports remote execution of jobs in a manner that is transparent to the users. In practice, chip simulation jobs are also I/O intensive, and they require access to a large amount of data. Data synchronization and large data transfers are typical handled through out-of-band mechanisms and are out of scope for this paper.

Jobs that arrive at a physical pool are immediately queued. The decision on when to dispatch a job and to which machine depends on the job requirements (e.g., OS and memory), resource availability and the priority of the job relative to that of other jobs in the queues.

2.2 Priority-based Preemption

In IDCP, priority-based job preemption is enforced at the host level of each physical pool. When a job is dispatched by the virtual pool manager for execution at one of the physical pools, if the physical pool has no available cores and there is a lower priority job running on a particular host of the pool, the lower priority job can be *suspended* to allow the new job (with higher priority) to execute. When a pool is highly utilized, low priorities jobs may get suspended more than once. Such a priority-based job preemption is necessary for the IDCP environment due to the business need to periodically run a large amount of jobs in a relatively short time.

Preemption may also arise in cases even where the system is not overloaded, due to resource bottlenecks for specific types of resources such as machines with large physical memory. In addition, latency sensitive jobs may be configured to only run in specific sets

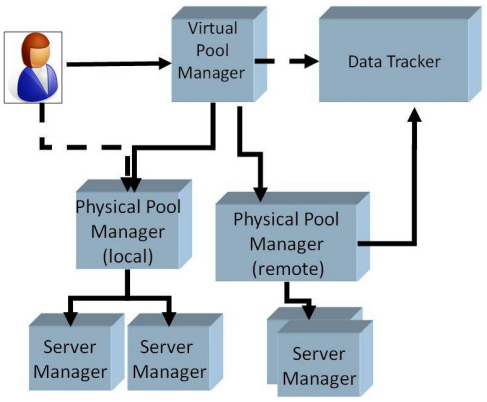

Figure 1: IDCP High-level System Components.

of physical pools so that desired results can be achieved.

To illustrate the impact of higher priority jobs on lower priority ones, Figure 2 shows the CDF of job suspension time (in seconds) collected from a large site with 20 physical pools, for a time period of roughly 10 weeks or 100,000 minutes. We focus on this particular period where a large number of job suspensions occurred due to the presence of high-priority jobs. We note that 20% of all jobs are suspended for more than 5000 seconds (83 minutes), and the median suspension time is 275 seconds (4.6 minutes). In addition, we observe a long-tailed distribution of jobs that require more than 100k seconds to complete.

The impact in some cases goes beyond the higher completion times experienced by suspended jobs and can impact the engineering productivity. For example, some classes of chip simulation work has logical notions of tasks, each of which represents a set of jobs completing a specific function. Typically, 100% or a high percentage of jobs associated with a particular task needs to complete before the task result (combined from the results of those jobs) can be useful. Often when one or more of those low priority jobs cannot complete in a timely fashion, engineers lose productivity and/or system resources are wasted since the same task execution needs to be manually repeated at a different time.

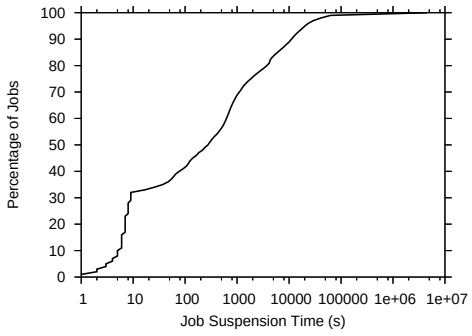

Figure 2: CDF of job suspension time.

We further note that high suspensions of jobs can occur even when there are resources available to execute the jobs. Figure 3 shows the IDCP utilization (darker line) in terms of cores that are in use for job execution, and the number of suspended jobs over a period of 500,000 minutes (roughly a year) of IDCP traces. We

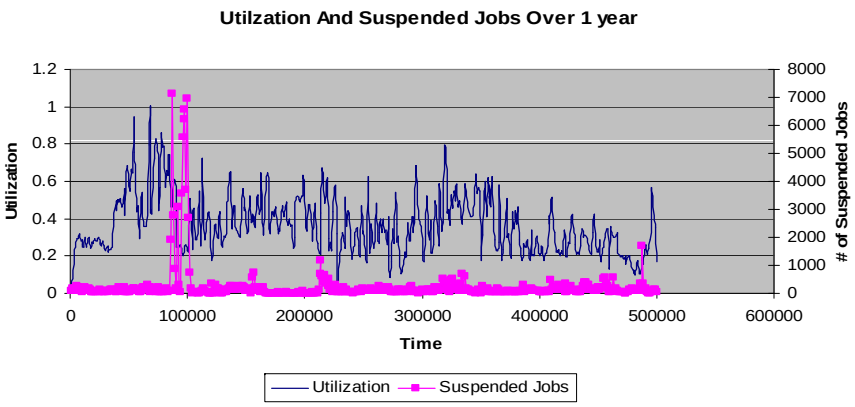

Figure 3: Utilization and Suspension over a one year period.

observe that overall system utilization averages around 40%, and is typically in the range of 20%-60%. However, during the same period, higher priority jobs can preempt lower priority jobs, leading to a large number of suspended jobs and hence a poor utilization of resources.

Given that overall system utilization is only 40%, better techniques need to be developed to ensure faster turnaround of jobs. We observe from our traces that higher priority jobs tend to be bursty and hence, job suspension happens in burst and lasts for several hours up-to a week. Furthermore, when global resources are spread out across multiple data centers, it is not practical for any scheduler to make perfect scheduling decisions. As a result, utilization is uneven across different pools. By adapting our job scheduling to such situations, we are able to improve the utility of low priority jobs with little or no adverse impact to high priority jobs.

3. DYNAMIC RESCHEDULING

Our approach in this section is based on two major observations. First, as explained in detail in the previous section, even when there were many suspended jobs in some physical pools, there were still available computing resources at other physical pools. This leads us to our approach of dynamically *restarting* (i.e., rescheduling) of suspended jobs at different pools, including remote pools, which may have more availability of the type of resources required by the suspended jobs.

Second, the job arrival patterns in the IDCP environment cannot be easily predicted in advance. As shown in Figure 3, the utilization of the system over the two-week period varied widely as a result of highly-random job arrivals. In fact, we were involved in an effort lasting several months to develop an analytical model of the arrival process of jobs using this same trace and was unable to do so. We conclude that it is impractical for virtual or physical pool managers to have apriori knowledge of job arrivals and to schedule jobs in such a way that suspensions of low priority jobs are largely avoided when system still has resources available (at other pools).

Under our restart-based approach, it is clear that there will be wasted work already completed by the suspended and then restarted jobs. One valid question to ask is why process migration (e.g., as used in Condor [3]) or VM migration approaches (e.g., as used in VMWare [7]) are not used. Prior work [3] has shown that process migration does not work well when multiple threads are involved and thus not suitable for the IDCP environment. VM migration approaches are practical but require virtualized environments in which virtual machine monitors are present on the servers. However, our internal experiments have shown that for many chip simulation workloads, the overhead of running those workloads on virtualized hosts often lead to performance overhead between 10% to 20% [8]. This is a primary reason why we explore the restart-based approach. Nonetheless, we discuss in Section 5 how the VM migration methods could be employed within the context of job-driven dynamic scheduling framework that we plan to work in the near future. The rest of this section describes our dynamic cost-based rescheduling method in detail.

3.1 Cost-based Decision

When a job is preempted and suspended by a higher priority job, a decision has to be made whether to keep the suspended job in its current pool (and resume when consecutive higher priority jobs are completed), or restart the job at another pool with the goal of reducing completion time. The choice of an alternate pool is achieved via *cost-based* decisions.

For each job, j running in a given pool, i, we identify the following measures:

- $RT_{j,i}$: The run time of job j in pool i
- $ST_{j,i}$: The suspend time of job j in pool i (when it is suspended by a higher priority job)
- $WT_{j,i}$: The wait time of job j spent in the queue of pool i before it starts running
- $CT_{j,i}$: The completion time of job j computed as the time between when the job enters the system and when it completes

We also use a superscript to denote each partial component of the measure. For example, when a low priority job, lj, gets suspended once before completion, there will be two run time components, $RT^1_{lj,i}$ and $RT^2_{lj,i}$.

Figure 4 illustrates how a decision can be made when a low priority job, lj, is first suspended at a pool i by a high priority job $hj1$ at time $t1$. If the system decides to keep job lj in pool i, it will remain suspended until $hj1$ completes. lj will then continue executing until another high priority job, $hj2$, suspends it again. Assuming that the system decides to keep job lj in i, the job will complete at $t3$, note that:

$$ST_{lj,i} = RT_{hj1} + RT_{hj2} \quad (1)$$

Alternatively, the system may decide to send lj to another pool k, where it may need to wait in the queue for a certain amount of time ($WT_{ij,k}$) before beginning to run. Note that by queuing lj

at another pool k, we preserve the first-in-first-out (FIFO) queuing principle typically used in systems like IDCP. In fact, while running, the job may get suspended by a high priority job, $hj3$, before running to completion at time $t2$. In this example, since $CT_{lj,k} < CT_{lj,i}$ and thus the system should have chosen to restart lj in k.

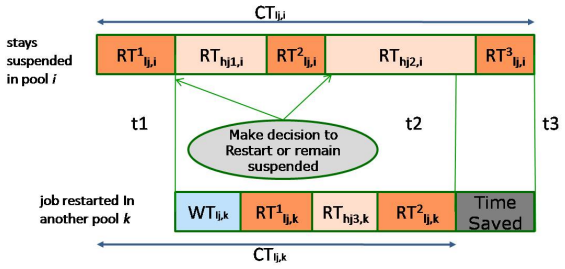

Figure 4: Timeline with and without scheduling a low priority job that has been preempted in pool i under high utilization.

In fact, at any decision point upon suspension, the system chooses a candidate pool that leads to lowest *remaining completion time* (RCT), resulting in lowest completion time for the job (if every decision till completion is correct). It is also important to note that when a job is restarted all work done thus far by the job will be wasted. Thus, we call $RT^1_{lj,i}$ *wasted run time*. For convenience, we call the sum of $RT^2_{lj,i}$ and $RT^3_{lj,i}$ the *remaining run time*, $RRT_{lj,i}$.

3.2 Computing Expected Completion Times

In practice, we do not know the future arrival time of jobs and their run time in each pool to be able to make an ideal decision about remaining completion time. Thus, for each job lj that gets suspended at a pool, i, we compute *expected remaining completion time*, $E[RCT_{lj,i}]$ as follows:

$$E[RCT_{lj,i}] \leftarrow E[ST_{lj,i}] + E[RRT_{lj,i}] \quad (2)$$

Similarly, we compute $E[RCT_{lj,k\neq i}]$ for any pool k other than i as follows:

$$E[RCT_{lj,k\neq i}] \leftarrow E[WT_{lj,k}] + E[ST_{lj,k}] + E[RT_{lj,k}] \quad (3)$$

An ideal algorithm will choose a pool, p, such that $E[RCT_{lj,p}]$ is lowest. If $p = i$ then the job will remain suspended in the pool. Otherwise, the job will get restarted at $p \neq i$.

For simplicity, we assume that a job's priority does not change even when it moves from one pool to another. We estimate $E[ST_{lj,i}]$ based on the recent statistics of suspend time observed by similar jobs, i.e., low priority jobs. Specifically, the average suspend time of low priority jobs, $ST^{avg}_{low,i}$, is obtained by averaging the suspend time observed by each completed low priority job over a sliding window, W.

We compute the average wait time of low priority jobs at pool k, $WT^{avg}_{low,k}$, in a similar manner and use it to estimate $E[WT_{lj,k}]$. The expected remaining run time, $E[RRT_{lj,i}]$, can be computed by subtracting the amount of time that the job had run so far from the expected run time. Finally, the expected run time, $E[RT_{lj,k}]$, can be estimated for major workload types within the IDCP environment. Figure 5 shows the average run time of jobs completed at each sampled hour aggregated by the simulation activity type of the jobs. Over the course of 24-hours, we observe that the average run time of jobs that are associated with a specific activity tends to remain stable and vary little. One reason for this relatively stable run time trend is because many simulation jobs associated with a particular function tend to have similar computational complexities and hence run time. These computational complexities may change with new chip model release, which typically happens every few days or weeks. Therefore, over a period of hours or days, the run time of jobs associated with a specific activity tend to be very similar. It is therefore practical to estimate them based on recent historic trends (in a similar way we are estimating expected wait time). We plan to do this as our next step. However, as our initial focus is to develop a simple dynamic rescheduling scheme and investigate its impact on job completion times and overall utilization, for the experiments presented in the next section, we simply assume that $RT^{avg}_{lj,k}$ is known by the system.

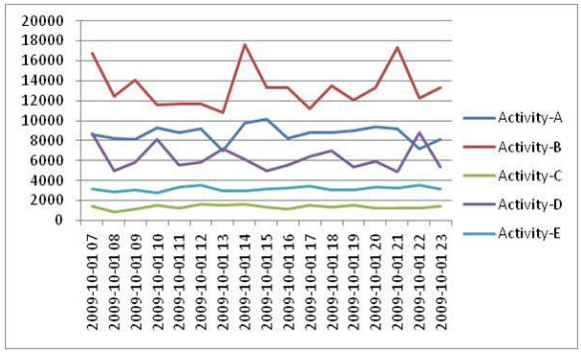

Figure 5: Average run time of jobs aggregated by their activity at each sampled hour in one of the largest physical pools.

We can now rewrite Equations 2 and 3 as follows:

$$E[RCT_{lj,i}] \leftarrow ST^{avg}_{low,i} + RRT^{known}_{lj} \quad (4)$$

$$E[RCT_{lj,k\neq i}] \leftarrow WT^{avg}_{low,k} + ST^{avg}_{low,k} + RRT^{known}_{low,k} \quad (5)$$

4. TRACE-DRIVEN EVALUATION

We present a feasibility analysis of the potential benefits of dynamic rescheduling of suspended jobs based on the approach described in the previous section. Our evaluation is carried out in a hybrid event-based and agent-based simulator called ASCA (for Agent-based Simulator for Compute Allocation) developed at Intel [11]. In order to accurately map the operational capabilities of IDCP, the simulator was developed to model various fine-grained components of IDCP such as sites, pools, queues, job requirements and priorities, virtual and physical pool managers, weighted fair-queueing, round-robin physical pool scheduling. We incorporated in ASCA our implementation of the cost-based dynamic rescheduling strategies described in Section 3.

The simulator takes as input IDCP traces, first introduced in Section 2.1, from a large IDCP site with 20 physical pools, collected over a period of one year. Note that jobs landing at this site are originated from several sites across the globe. The traces include machine configuration information across all machines including machine CPU type/speed and memory, and the state (running, suspended, etc) of every job in the system and its resource usage. The simulator samples at each minute the current states of all IDCP components and executing jobs, which are then output as logs for post-analysis. Tan *et al.* describe the implementation of this simulator in greater detail and provide detailed analysis results

to demonstrate that the simulator achieves the performance characteristics of the actual IDCP deployment in terms of utilization and job completion times [11].

4.1 Aggregate Analysis

	Restart-Off	Restart-On
Jobs Suspended	36459	59875
% Completed	97.3%	99.2%
Avg Suspend Time	1432.6	44.9
Avg Completion Time	1824.6	466.9

Table 1: Execution statistics under Restart-on and Restart-off with respect to only suspended jobs.

	Restart-Off	Restart-On
Jobs Completed	840552	841074
Avg Completion Time	305.6	259.6
Utilization (%)	42.3	43.4

Table 2: Execution statistics under Restart-on and Restart-off with respect to all jobs.

For our simulations, we focus on a 2-week period which characterizes a typical *burst* period in the IDCP environment, when jobs observe long completion time due to suspension. Several such periods, lasting from a few days to a few weeks typically occur within each quarter. During this period, the average system utilization is 42.3%.

Table 1 summarizes the completion and suspension times of jobs that are suspended at least once within that 2-week period for two scenarios: i) *Restart-Off*, the baseline scenario without the rescheduling method implemented and ii) *Restart-On*, the scenario with the rescheduling method implemented. The averaging window size for collecting statistics for expected suspension and wait time within each pool (for making cost-based decisions on rescheduling) is set to 100 minutes. The *Avg Completion Time* is the average completion time (in minutes) over all jobs completed within this period. The *Avg Suspend Time* is the average suspension time (in minutes) of finished jobs that have been suspended (and subsequently resumed) within this period. When compared to *Restart-Off*, *Restart-On* achieves a 75% reduction (from 1824.6 minutes to 466.9 minutes) in average completion times of all jobs that have been suspended at least once. Note that *Restart-On* does not reduce the number of suspensions. This is because by rescheduling suspended jobs, *Restart-On* increases the likelihood that the same job can get suspended multiple times (at different pools). However, *Restart-On* does significantly reduce the suspension time over the job's lifespan on average since in most cases the rescheduled jobs would be able to complete execution in another pool.

Table 2 summarizes under both scenarios the average completion times of all jobs including those that were never suspended. Observe that the fraction of jobs suspended is 4.3%, however, these jobs observe a high average completion time of 1824.6 minutes compared to 305.6 minutes for all jobs (see Table 2). By rescheduling only a small fraction of jobs, *Restart-On* improves upon the average completion time (per completed job) by 15% over *Restart-Off*. *Restart-On* achieves this by utilizing available resources at other pools but the resulting increase in overall system utilization is marginal (only 1.1%). The results demonstrates improving job mobility (upon suspension) can significantly improve observed latency of both suspended and non-suspended jobs even when only a small fraction of jobs is involved in rescheduling.

To explore the limits of our techniques, we emulated a high load scenario, where computing resources are reduced by half, hence increasing the occurrence of resource contention and preemption. Under the high load scenario, we observe that the average completion time for all jobs (795.6 minutes) is more than doubled compared to the actual scenario (305.6 minutes) due to increased system utilization (80.5%). Nevertheless, the rescheduling techniques are still effective at reducing completion times, resulting in 71% reduction in average completion time for all jobs that have been suspended at least once. In addition, they also outperform the original *Restart-Off* in terms of overall completion time of all jobs. In short, our rescheduling techniques are well-suited for both high and low load scenarios.

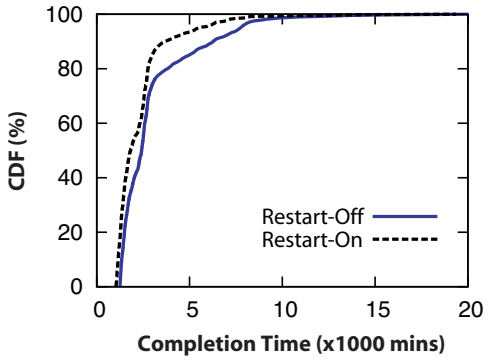

Figure 6: CDF of completion times (x1000 minutes) of top 5% of long-running jobs.

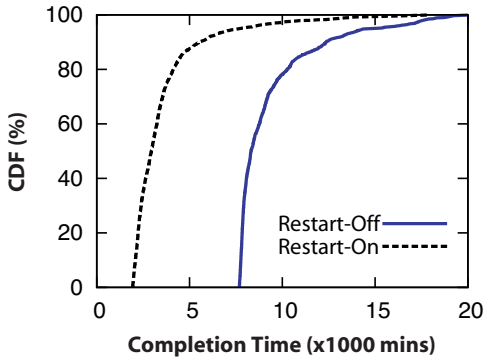

Figure 7: CDF of completion times (x1000 minutes) of top 5% of long-running jobs that have been suspended.

4.2 Per-job Analysis

While aggregate statistics are useful indications of the benefits of rescheduling, the actual benefits are best observed by analyzing individual jobs, especially pathological cases where jobs exhibit long completion times due to frequent suspensions.

Our next analysis focuses on *per-job* reduction in completion time. Focusing on long-running jobs (top 5% of jobs with highest completion times), Figure 6 and 7 show the CDF of completion times for all jobs (corresponding to the aggregate statistics in Table 2), and previously suspended jobs (Table 1).

In both cases, the reductions in completion times under *Restart-On* are significant: in terms of median completion times, the reductions are 602 minutes (25%) and 5370 minutes (65%) respectively. When considering the longest running jobs (at 95% percentile), the reductions in completion times are 2268 minutes (29%) and 7154 minutes (48%) respectively. All in all, we note that our restart strategy is able to reduce completion times of long running jobs, and the impact on jobs that would otherwise remain suspended under *Restart-Off* is particularly significant.

5. RELATED WORK

IDCP can be viewed as a real-world deployment of an Internet-scale *grid computing* [4] infrastructure. Condor [3, 6] batch processing system provides a transparent middle-ware for executing jobs on shared computational resources. To our best knowledge, IDCP's scale of deployment is larger than any existing Condor pools, both in terms of the number of jobs and machines. IDCP's scale is closer to that of p2p computational platforms such as SETI@Home, with much more stringent requirements on job completion times.

In addition to our use of job restarts, popular techniques for resource management that are potentially applicable to IDCP in future include the use of virtual machine migration [3] and redundant executions [5, 12] to trade-off utilization and completion time. Our approach is orthogonal to these techniques, and our choice of rescheduling is largely driven by current practical concerns outlined in Section 3. In addition, our recent internal evaluation of IDCP typical workloads have shown that although VM performance have improved significantly over the years [7], it has yet to go down to a low single digit before it becomes a viable option from both performance and cost perspective for the IDCP environment, where jobs tend to be memory and I/O intensive. Nevertheless, it is interesting future work to explore ways to integrate dynamic rescheduling approaches with VMs within the context of the IDCP platform.

We also note recent work around adaptive job execution techniques after the job is already executed [9]. Unlike our work, Shan *et al.* focus on opportunistic job migration when a "better" resource is discovered and do not discuss about suspended jobs. Our IDCP workloads and scale will enable us to study these issues in depth in a realistic yet challenging setting.

6. CONCLUSION AND FUTURE WORK

In this paper, we perform a trace-driven analysis of the *Intel Distributed Computing Platform (IDCP)*, an Internet-scale data center based distributed computing platform developed by Intel Corporation for running hundreds of thousands of chip simulation jobs daily. Our initial results are promising: by performing cost-based rescheduling on suspended jobs, the average job completion time is reduced by 15% over all jobs, and 75% for previously suspended jobs. Our results also illustrate that our cost-based rescheduling techniques are robust to various load conditions.

These results also have positive implications for productivity and efficiency of the IDCP environment since all jobs associated with a specific *task* can be completed in a reasonable fashion reducing the likelihood of incomplete tasks and resulting in improved productivity and/or efficiency.

We plan to continue with additional trace-driven simulation studies, where we investigate the benefits of job duplication, the relationship between utilization and job completion as a result of the interaction between our rescheduling scheme and global job scheduling schemes (e.g., at the virtual pool level). We are also exploring various techniques for predicting run time via a combination of sliding window averaging and other statistical techniques. Ultimately, we plan to evaluate our results on the IDCP platform itself.

In the public cloud computing environments like Amazon's, we think that the workload arrival process will be hard to model and predict at a fine granularity necessary for master schedulers traditionally employed to manage a set of compute servers. Just as Intel needs multiple classes of jobs, public clouds will soon need to accommodate jobs of different SLAs and thus pricing. When dealing with problems like job suspensions or various inefficiencies of the grid scheduling algorithms (largely because of the lack of workload predictability), one practical and effective approach is to make intelligent decisions from the perspective of individual jobs. Our approach for suspended jobs is just one example. Waiting jobs may decide to remove themselves from the current queues and perhaps re-queue at different pools based on their expected completion times observed at that time. In other words, scheduling decisions traditionally made exclusively by each master scheduler responsible for a set of resources could prove ineffective or detrimental to jobs a few moments later. The sheer scale of emerging computation clouds coupled with the tremendous diversity in workloads could demand us more responsive scheduling approaches that can adapt to changing conditions whenever appropriate. This may mean more intelligent jobs and less intelligent cluster schedulers, or a combination of intelligent jobs and master schedulers. We hope that our work inspires a lot more research in this area.

7. REFERENCES

[1] AMAZON ELASTIC COMPUTE CLOUD, VIRTUAL GRID COMPUTING. http://aws.amazon.com/ec2.

[2] ARMBRUST, M., FOX, A., GRIFFITH, R., AND ET AL. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, UC Berkeley, Feb. 2009.

[3] ET. AL., M. L. Checkpoint and migration of UNIX processes in the Condor distributed processing system. Tech. Rep. UW-CS-TR-1346, UW-Madison, 1997.

[4] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[5] KOOLE, G., AND RIGHTER, R. Resource allocation in grid computing. *J. of Scheduling 11*, 3 (2008), 163–173.

[6] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor - A Hunter of Idle Workstations. In *ICDCS* (1988).

[7] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *USENIX* (2005).

[8] SAMMANNA, S., TANG, T., AND LAL, V. Server virtualization using xen based vmm. In *Intel IT Technical Leadership Conference* (2008).

[9] SHAN, H., OLIKER, L., AND BISWAS, R. Job superscheduler architecture and performance in computational grid environments. *SC Conference* (2003).

[10] SRINIVAS NIMMAGADDA ET AL. High-End Workstation Compute Farms Using Windows NT. In *3rd USENIX Windows NT Symposium* (1999).

[11] TAN, G., DUZEVIK, D., BUNCH, E., ASHBURN, T., WYNN, E., AND WITHAM, T. Agent-based simulator for compute resource allocation. In *Intel IT Technical Leadership Conference* (2008).

[12] ZAHARIA, M., KONWINSKI, A., JOSEPH, A. D., KATZ, R., AND STOICA, I. Improving mapreduce performance in heterogeneous environments. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2008).