Trace-driven Analysis of an Internet-scale Cloud Computing Platform

Harrison Duong* Boon Thau Loo* Godfrey Tan†
*University of Pennsylvania †Intel Corporation

1 Introduction

In recent years, cloud computing have become a popular computing paradigm, where computation is moved away from personal computers or an individual application server to a "cloud" of computers in the network. Users of the cloud only need to be concerned with the computing service being asked for, as the underlying details of how it is achieved are hidden. This method of distributed computing is done through pooling all computer resources together and being managed by software rather than a human. The popular application of cloud computing have primarily been for the deployment of web applcations by end users, e.g., Amazon's elastic compute cloud [1], and the Google/IBM's academic cluster computing initiative [2].

In this paper, we study an important use of cloud computing: that of performing compute-intensive tasks within an Internet-scale enterprise network. Our main driving application is an Internet-scale Cloud Computing Platform (ICCP) developed by a Fortune-500 company for massively parallel simulations within the company. ICCP 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. With ICCP, engineers no longer have to manually setup simulation machines, or determine where and when to schedule their simulation jobs. By pooling together machines all over the enterprise network, resources are used more efficiently as computers can be consolidated to be used for more tasks.

We present an initial analysis of job execution traces obtained over a 18 month period collected from tens of thousands of machines at ICCP. From our ICCP traces, we observe that despite only utilizing less than 40% of compute resources on average, the ICCP is not designed to handle flash crowds. There have been severe cases of slow turnaround of jobs during peak times, primarily caused by high priority jobs preempting lower priority jobs. The introduction of high priority jobs tend to be bursty, isolated to particular sites, and can last for several hours up-to-a-week.

Using a trace-driven simulator, we evaluate the potential benefits of selectively restarting suspended jobs based on current job suspend time and available resources. As cloud computing environments mature, there will be a need for the system to gracefully handle multiple classes of jobs with different priorities and SLAs. The ICCP deployment today is estimated to involve hundreds of physical pools distributed globally at different 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, and in aggregate, hundreds of millions of jobs per year. To our best knowledge, this is one of the first efforts at empirically understanding the infrastructure requirements of a cloud computing platform that is operational at a global scale.

2 Architecture and Motivation

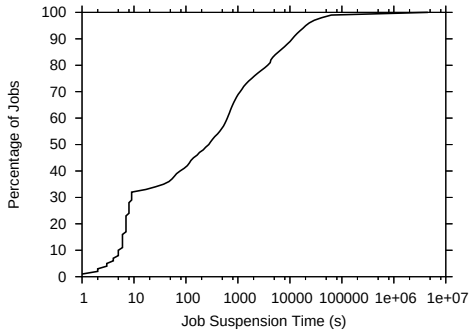

Figure 1: CDF of job suspension time

ICCP is designed as a hierarchical system consisting of multiple *sites* at different geographically locations. Each site consists of multiple *physical pools*, each consisting of thousands of multi-core machines. Each site is managed by a *virtual pool manager*, and each pool by a *physical pool manager*. Users submit jobs to a particular site, and the virtual pool manager will assign the job to a physical pool based on resource availability. The physical pool manager then decides which machine to allocate to execute the job. The requirements of the job (e.g. OS and memory requirements) may dictate which machine gets assigned to the job.

ICCP utilizes a priority-based queuing system, in which there are multiple job queues of different priority level, and weighted-fair queueing is used to ensure proportional sharing. In addition, queues can also be specified based on job requirements. In ICCP, a job is first submitted by a user to a particular queue. The virtual pool manager assigns the job to one of the physical pools in a round-robin fashion. The physical pool manager then assigns the job to an available machine. High priority jobs are allowed to preempt lower priority jobs that are running on a machine that the higher priority job requires. Once a job is on a machine, the job will stay on the machine until it completes its task. This results

in low priority jobs being suspended for long periods of time depending on how long higher priority jobs take to complete.

To illustrate the impact of high priority jobs on lower priority ones, Figure 1 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 suspension occured due to the presence of high-priority jobs. We note that 20% of all jobs are suspended for more than 5000 seconds, and the median suspension time is 275 seconds. In addition, we observe a long-tailed distribution of jobs that require more than 100k seconds to complete.

3 Initial Observations

Given that overall system utilization is only 40%, better techniques need to be developed to ensure faster turnaround of jobs. Unlike prior work on priority scheduling in distributed systems, the ICCP environment (like recent cloud computing platforms) exhibit different job characteristics. For example, in our traces, higher priority jobs tend to be bursty and hence job suspension happens in burst. Since this phenomenon can last for several hours to up-to a week, we can detect such situations and adapt our job scheduling to improve utility of low-priority jobs with little or no adverse impact to high priority jobs. 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, uneven utilization at various pools would exist.

One solution that we are currently exploring is the selective restarts (rescheduling) of suspended jobs at remote pools. Without a-priori knowledge on the execution time of each job, one challenge is in figuring out the appropriate *restart threshold*, which is the duration time a job has been suspended before a restart is initiated.

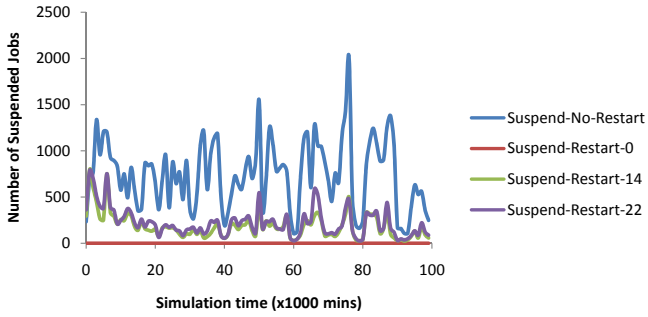

Figure 2: Number of suspended jobs over time

We utilize a trace-driven event-based simulator that takes as input job traces from ICCP collected a 18 month period, and replay these traces with different restart thresholds. We focus on the simple case of restarting jobs within a single site, and limit our analysis to the 10 week time window presented in the previous section. Figure 2 show the number of suspended jobs for various restart thresholds. *Suspend-Restart-K* restarts a job whenever its suspended time has exceeded *K* seconds, while *Suspend-No-Restart* do not restart jobs, i.e. suspended jobs resume execution when all higher priority jobs have completed execution.

We note that whenever restarts are not carried out, many jobs are suspended and stay suspended for long periods of time (as indicated by *Suspend-No-Restart*), resulting in their starvation even when resources are available in other pools. On the other hand, naïvely restarting suspended jobs immediately (*Suspend-Restart-0*) reduces the number of suspended jobs to zero, but decreases overall system throughput. In between these extremes, as the *restart threshold* is increases from 14 minutes to 22 minutes, the number of suspended jobs decreases marginally. In both cases, the number of suspended jobs and the average job suspension times are significantly lower compared to *Suspend-No-Restart*.

4 Ongoing Work

Our initial observations suggests the benefits of restarts even with the use of a fixed restart threshold. We are currently experimenting with heuristics for adaptively setting the threshold that increases with the number of restarts per job, while putting a cap on the maximum number of restarts per job. Concurrently, we are exploring techniques to predict the execution time of each job based on historical information and temporally correlating with the execution times of recent jobs. We intend to exploit the global distribution of ICCP machines to restart jobs at remote data centers while respecting administrative domain privileges across different administrative domains and the availability of input data.

Beyond job restarts, our overall research agenda centers on dynamic and adaptive algorithms that will improve aggregate utility by adapting to job behaviors at a global scale. To this end, we are exploring other aspects of scheduling jobs for improving overall turnaround time. For example, executing redundant jobs at multiple locations to tradeoff compute resources for job latency and assigning higher priority to jobs that are close to completion (or in later stages of a workflow). We also intend to incorporate job deadlines, where redundant executions and restarts become more aggressive as deadline approaches.

References

[1] Amazon Elastic Compute Cloud, Virtual Grid Computing. `http://www.amazon.com/gp/browse.html?node=201590011`.

[2] Google, IBM forge computing program. `http://www-03.ibm.com/press/us/en/pressrelease/22414.wss`.