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

Harrison Duong[1] Boon Thau Loo[1] Godfrey Tan[2]

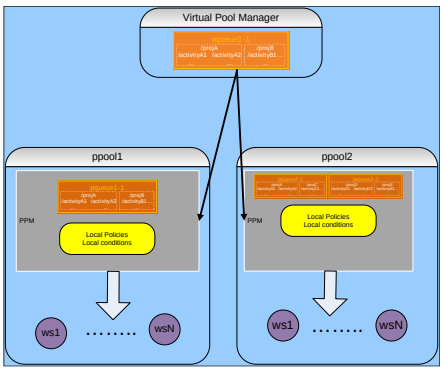
[1]University of Pennsylvania [2]Intel Corporation

Motivation

- **Cloud Computing is fast becoming a popular paradigm to harnessing a large computing capacity by many different groups**
  - Large systems on an internet scale
  - Systems that span multiple sites worldwide with tens of thousands of compute servers
  - Flash crowds of high priority jobs decreases latency of low priority jobs
  - Usage: management of each pool of compute servers can be independent
- **Goal: Utilize global resources to handle flash crowds**
  - Move jobs from one pool to others where resources are available resulting in latency improvements for lower priority job.

Introduction

- **A study into a real world Internet-scale Cloud Computing (ICCP) platform used for compute intensive tasks**
  - Used for hundreds of millions of jobs per year

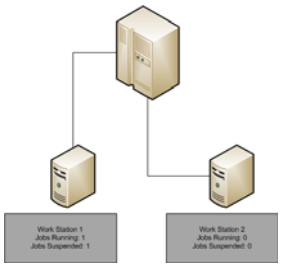


ICCP Local Site Architecture

- **Hierarchical model of machines**
  - Jobs are submitted to a Virtual Pool and gets distributed to physical pools
  - Each physical pool manager manages thousands of machines
  - Usage model dictates jobs with various levels of priorities
- **Lower-priority jobs get suspended on workstations when higher priority job comes in**

Problem

- **During high load, lower priority jobs are suspended at local workstations while global resources may be available to execute the jobs.**
  - During high load periods, jobs end up getting suspended for long periods of time.



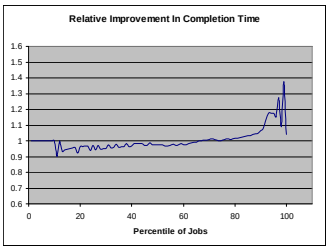
- **We explore and evaluate different solutions within an event-driven agent based simulator based on ICCP traces.**

Solution: Restart Jobs

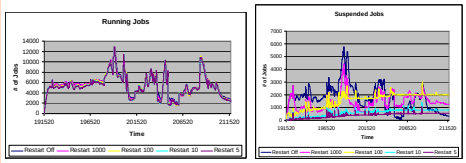
- **Restart suspended jobs on other resources available**
  - Key problem here is when to restart the job and where to restart the job
- ***Job Restart Delay* specifies a threshold of suspend time**
  - When suspend time of a job exceeds the restart delay, the job is resubmitted and reassigned to another machine
  - Purpose of this is to utilize available resources to improve latency of low priority jobs while not affecting high priority jobs
  - To understand when would be a a good threshold, we studied different times to restart the jobs.
  - Evaluation metrics: Number of suspended jobs, Job completion time
- **Experiments are conducted by varying the restart delay**
  - Experimental period extends through two weeks of measurements
  - Strawman for comparison is based on ICCP deployment where suspended jobs are not restarted
  - During this period, roughly 840,000 jobs completed with about 35,000 jobs suspended for the baseline.

Evaluation

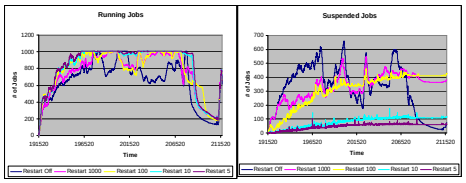
- **Relative Improvement**
  - Comparison of restart delay at 5 mins and the baseline
  - Most low priority jobs (10%) see improvements in completion time
  - We see roughly a range of 20-40% improvement on completion time of jobs for the 90 percentile jobs



- **Reassignment of Jobs**



Running and Suspended Jobs on Pool #1



Running and Suspended Jobs on Pool #2

- **Job restart delay matters**
  - From the graphs above, between restart delays, you see a difference
    - Number of running jobs on Pool 1 are similar, however restart off has more suspended jobs
    - Restart decreases lowers suspended jobs by restarting them at other lower loaded pools
- **Future Work**
  - Dynamically
    - Determine restart threshold based on real-time job statistics
  - Set limits on the number of times a particular job can be restarted to avoid repeated job restarts.
  - Implement a threshold to start execution of restart policy
    - Do not restart jobs if they have already executed for a long period of time
  - Study the effects of restarting jobs at pools on remote sites as opposed to the local site