Harmony: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud

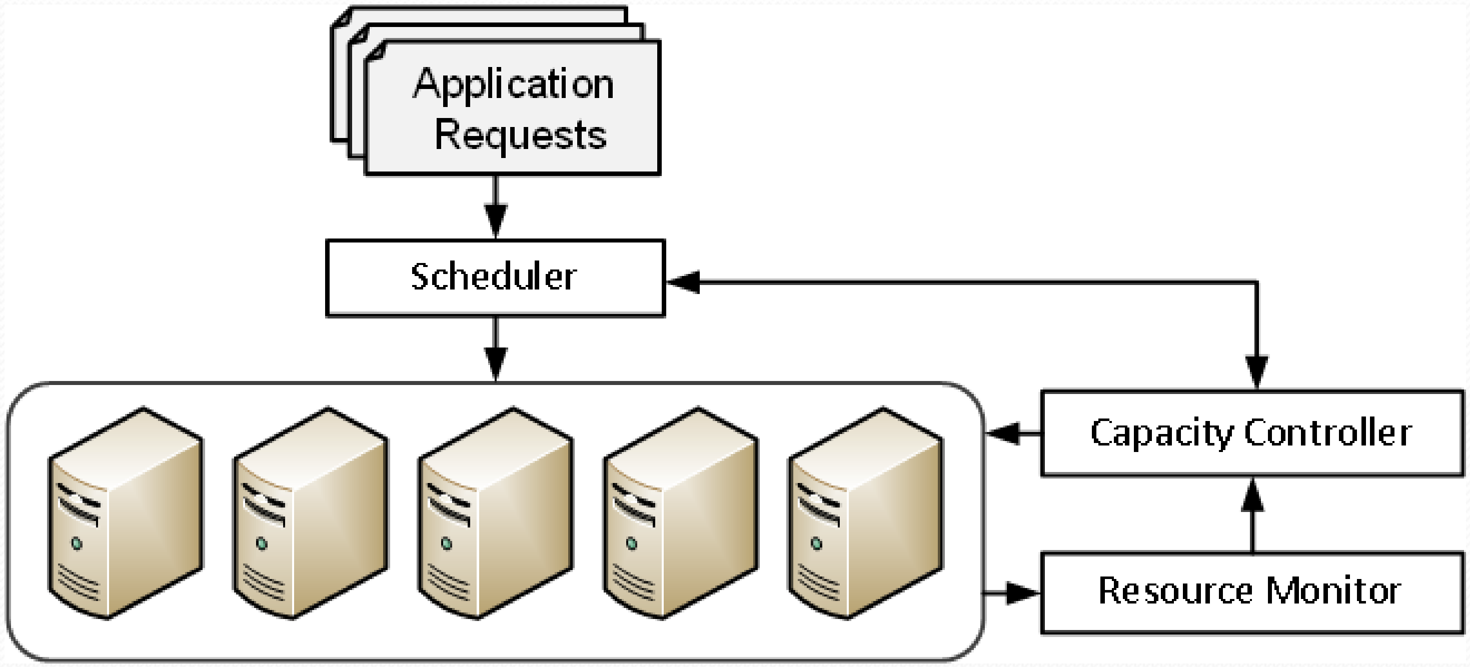
Qi Zhang[1], M. Faten Zhani[1], Raouf Boutaba[1], Joseph L. Hellerstein[2]
[1] University of Waterloo
[2] Google Inc.

IEEE International Conferences on Distributed Computing Systems (ICDCS)
Philadelphia PA USA, July 8-11, 2013

Introduction

- Data centers consume tremendous amount of energy
  - Energy costs accounts for 12%-20% of the costs of running a data center (Gartner 2011)
- A well-known technique for reducing data center energy consumption is Dynamic Capacity Provisioning (DCP)
  - Turning unused servers to save energy

Dynamic Capacity Provisioning (DCP)



- Dynamically adjusting resource capacities by turning machines on and off

Dynamic Capacity Provisioning (DCP)

- Objectives
  - Cloud user: Low scheduling (e. g. queuing) delay
  - Cloud provider: High resource utilization

- Adjusting the number of servers according to demand fluctuation
  - Too many servers causes low utilization
  - Too few servers causes high scheduling delay

- Need to consider cost of turning on and off machines
  - Wear-and-tear effect

Challenges

- Dynamic Capacity Provisioning has been studied extensively
  - Adjusting the number of server replicas to handle demand fluctuations
  - Assuming servers and resource requests are homogenous
- In many production data centers, both servers and application requests are **heterogeneous**
  - Multiple types of servers (with different capacities and energy efficiencies) coexist in a single data center
  - Resource demand, running-time and priorities vary significantly across applications
  - Not every server can schedule every application process

➢ **How to adjust the number of each type of servers to achieve low scheduling delay and high utilization over time?**

Harmony: A Heterogeneity-Aware DCP Framework

- Using clustering to divide workload into distinct types of tasks (e.g. VMs)
- At run-time, monitor the arrival of each type of tasks
- Run a control algorithm to dynamically adjust number of servers of each type

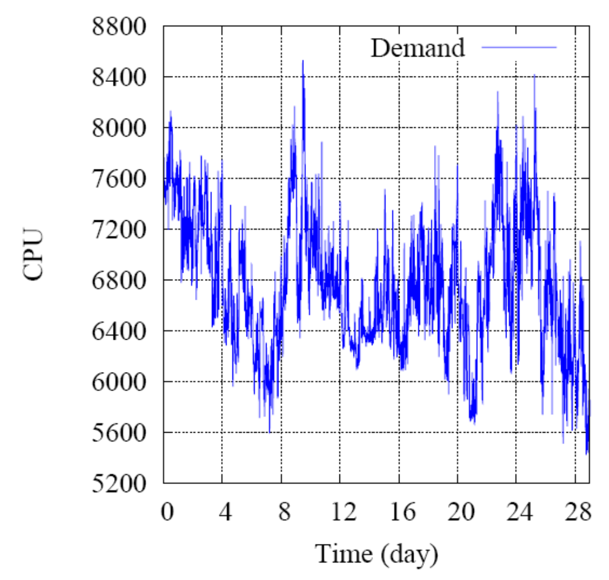
Agenda

- Introduction
- Trace Analysis
- Harmony
- Evaluation
- Conclusion

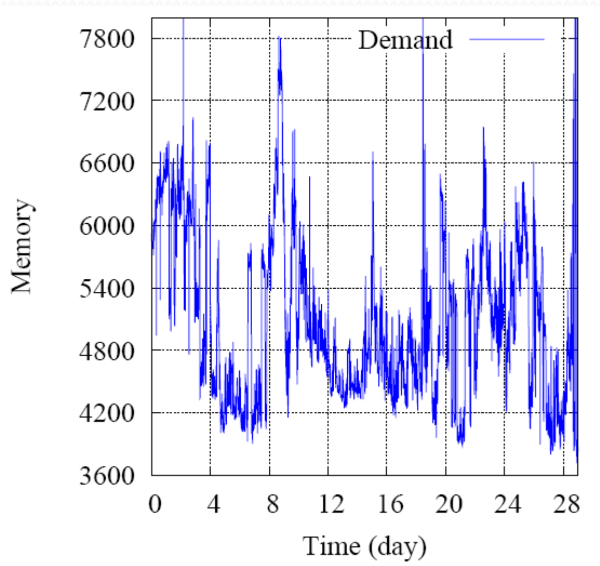
Machine and Workload Analysis

- Workload traces collected from a production compute cluster in Google over 29 days
  - ~ 12,000 machines
  - ~2,012,242 jobs
  - 25,462,157 tasks
- Applications are represented by **jobs**
  - **User-facing jobs**: e.g., 3-tier web applications
  - **Batch jobs:** e.g., MapReduce jobs
- Each job consists of one or more **tasks**
- There are 12 priorities that are divided into three priority groups: gratis(0-1), other(2-8), production(9-11)

Trace Analysis: Total Resource



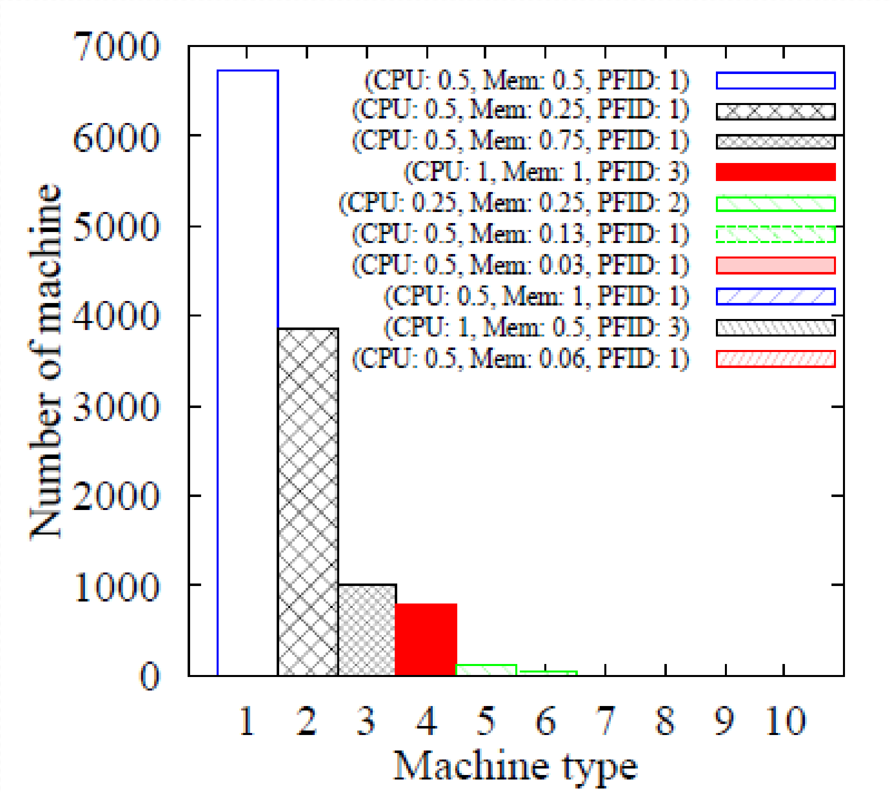
CPU Demand
over 30 days



Memory Demand
over 30 days

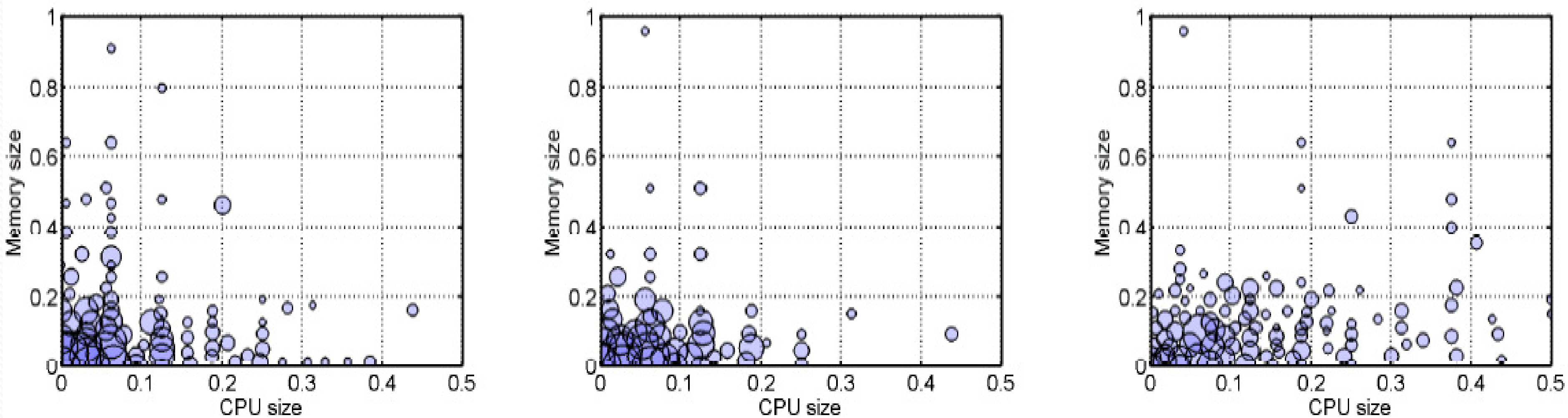
Figure: Total resource demand in Google's Cluster Data Set

Trace Analysis: Machine Heterogeneity



- 10 types of machines, some (e.g type 2 and 4) have high CPU capacity, others (e.g type 3 and 8) have high memory capacity

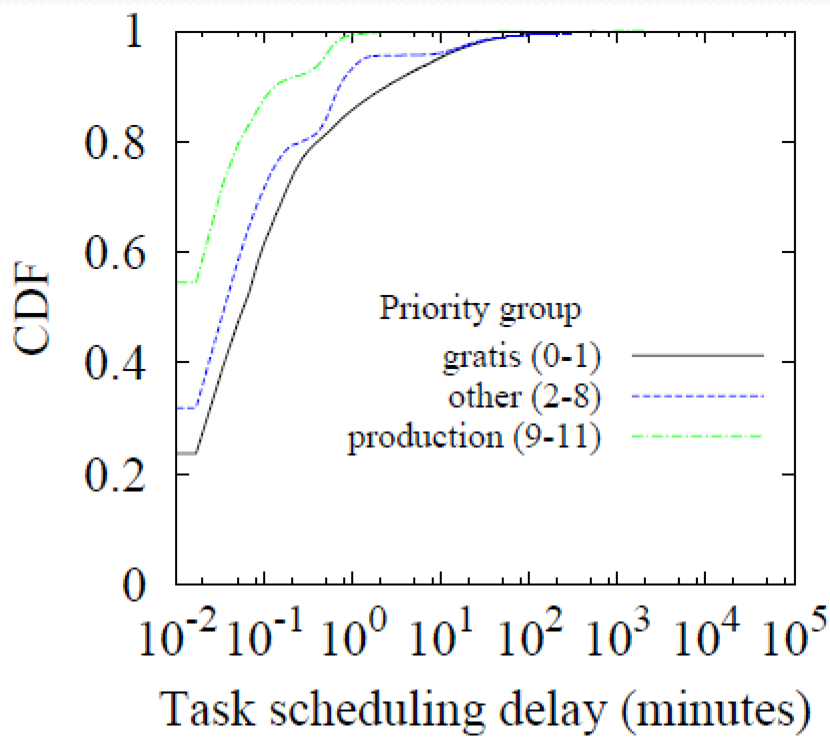
Trace Analysis: Task Size

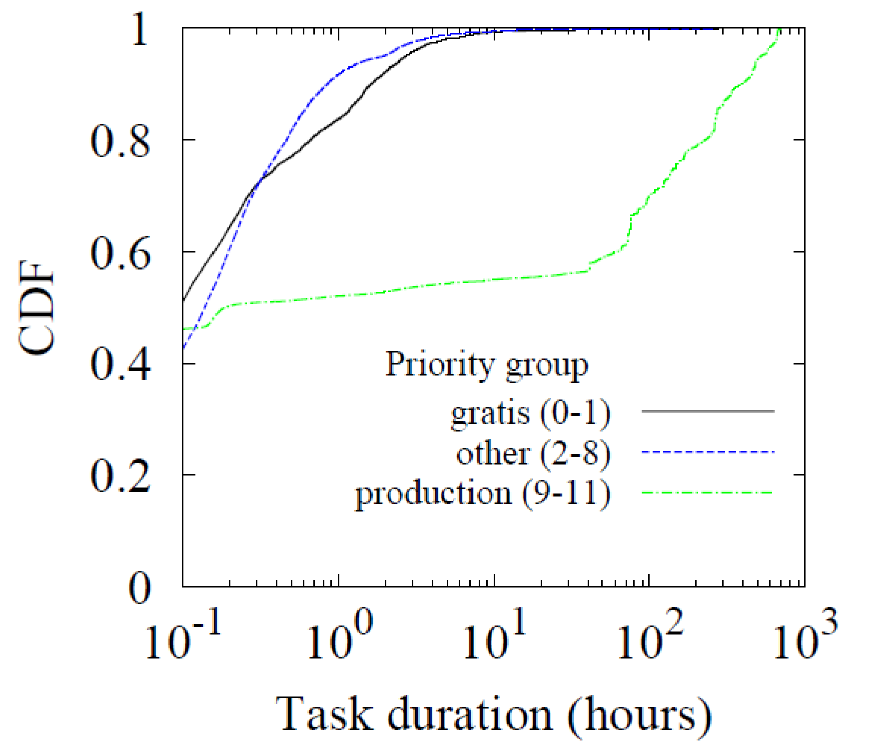


(a) Gratis (0-1) (b) Other (c) Production

- Tasks are either CPU intensive or Memory intensive
- Little correlation between CPU size and Memory size

Trace Analysis: Task Priority and Running Time





- Different groups have different scheduling delays
- Running-time across groups can differ significantly

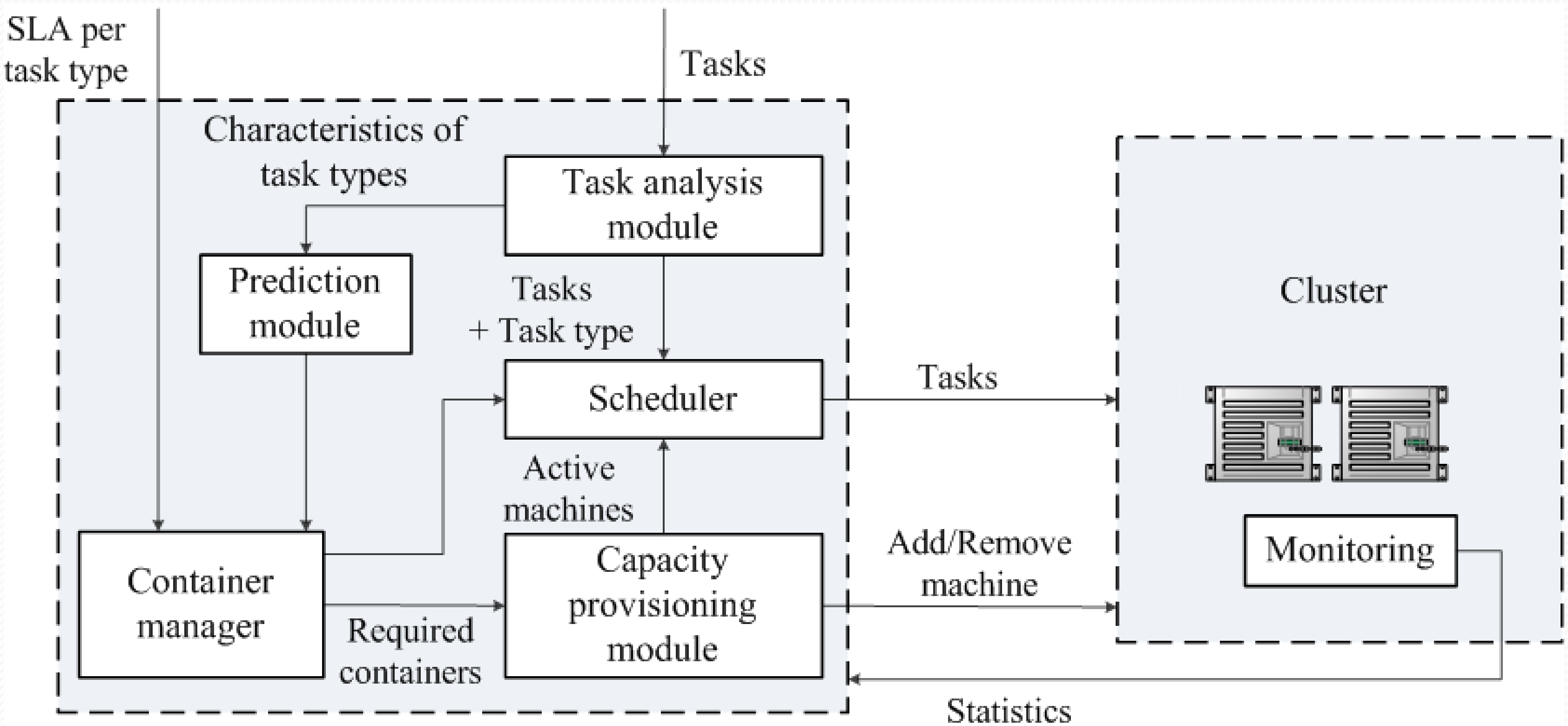
Summary

- Machines have different resource capacities
  - Some have more CPU capacities, while others have more memory capacities
- Tasks belong to different jobs have different resource requirements, running time and priorities
- Heterogeneity-awareness is important
  - Different machines are likely to have different energy characteristics
  - Scheduling CPU-intensive tasks on high memory machines can lead to inefficient schedule
  - Not every task can be scheduled on every machine

Agenda

- Introduction
- Trace Analysis
- Harmony
- Evaluation
- Conclusion

System Architecture of Harmony



Task Classification

- Classify tasks based on their size and duration using *k-means* clustering algorithm
  - First divide tasks according to priority group and running time
  - Run *k-means* for each group of tasks

- Capture the run-time workload composition in terms of arrival rate for each task class
  - First classify according task resource requirements
  - Update classification over-time

- Define *container* as a logical allocation of resources to a task that belongs to a task class
  - Use containers to reserve resources for each task class

DCP formulation

$$\max_{\delta_t^m, \sigma_t^{mn}} \quad R_T = \sum_{t=1}^{T} U_t^{perf} - E_t - C_t^{sw}$$

- where

$$U_t^{perf} = \sum_{n \in N} f^n\left(\sum_{m \in M} x_t^{mn}\right)$$ (Performance objective)

$$E_t = \sum_{m \in M} -p_t \left(z_t^m E^{idle,m} + \sum_{r \in R} \sum_{n \in N} \frac{\alpha^{mr} c^{nr}}{c^{mr}} \cdot x_t^{mn} \right)$$ (Energy cost)

$$C_t^{sw} = \sum_{m \in M} q_m |\delta_t^m|$$ (Switching cost)

- Subject to constraints

$$z_{t+1}^m = z_t^m + \delta_t^m \qquad \forall n \in N, m \in M, t \in \mathcal{T}$$ (Machine state constraint)

$$x_{t+1}^{mn} = x_t^{mn} + \sigma_t^{mn} \qquad \forall n \in N, m \in M, t \in \mathcal{T}$$ (Workload state constraint)

$$z_t^m \le N_t^m \qquad \forall m \in M, t \in \mathcal{T}$$ (Num. Machine constraint)

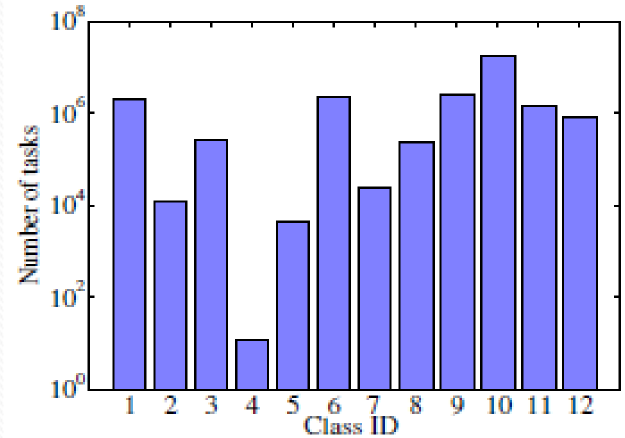
$$\sum_{n \in N} c_n^r x_t^{mn} \le z_t^m C^{mr} \qquad \forall m \in M, r \in R, t \in \mathcal{T}$$ (Capacity constraint)

Solutions

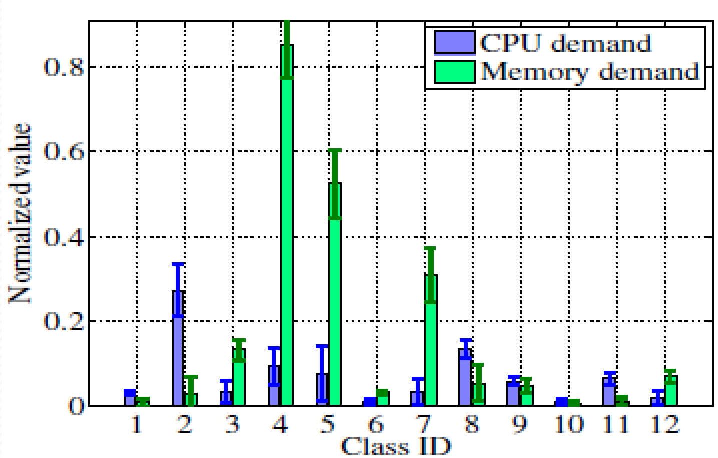
- Container-Based Provisioning (CBP)
  - Round up the number of machines to the nearest integer value
  - At run-time, schedule tasks using existing VM scheduling algorithms such as first-fit
    - Must respect the reservations computed by the algorithm
- Container-Based Scheduling (CBS)
  - Statically allocate containers in physical machines
  - At run-time, schedule tasks in containers
- Overprovisioning factor can be used to handle underestimation of resource requirements

Experiments

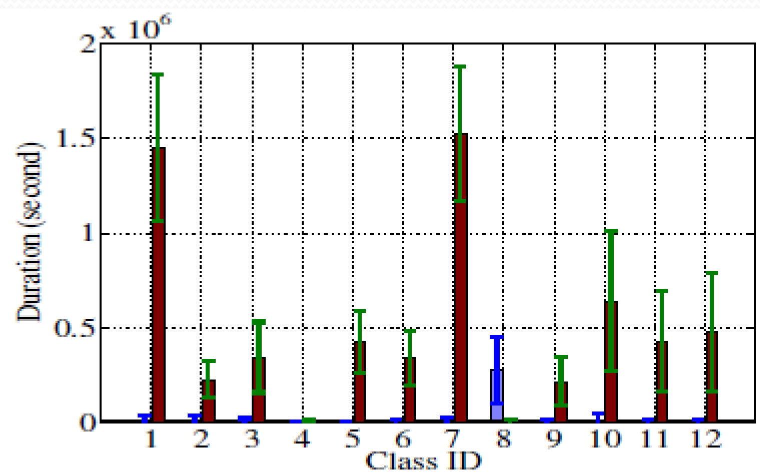
- Task classification
  - Classify tasks based on task size



Number of tasks (gratis)

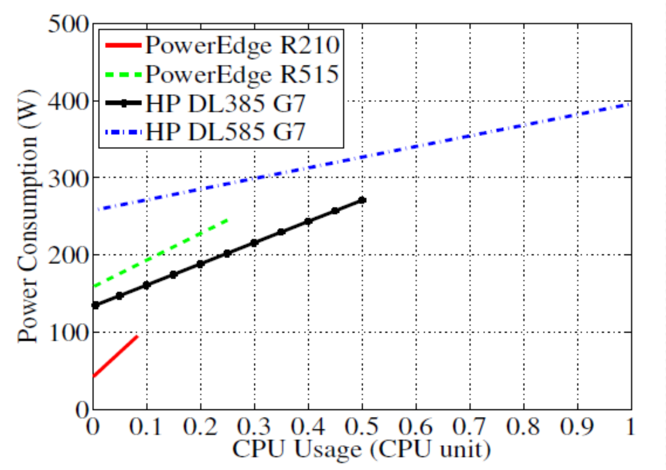


Class size (gratis)

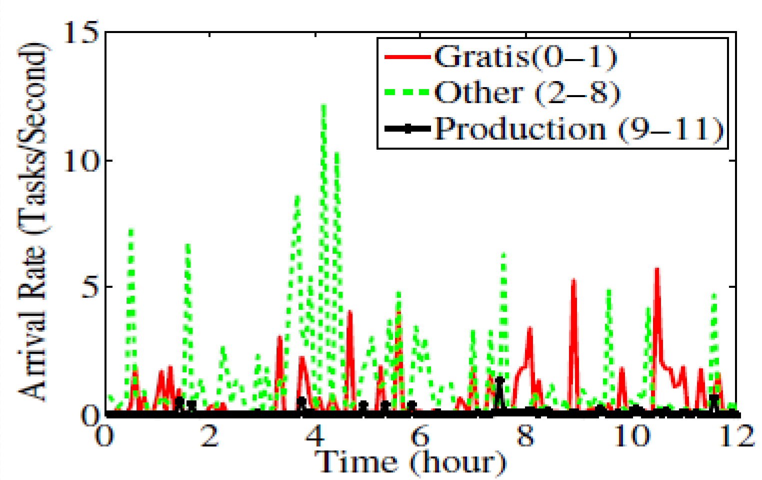


Task duration (gratis)

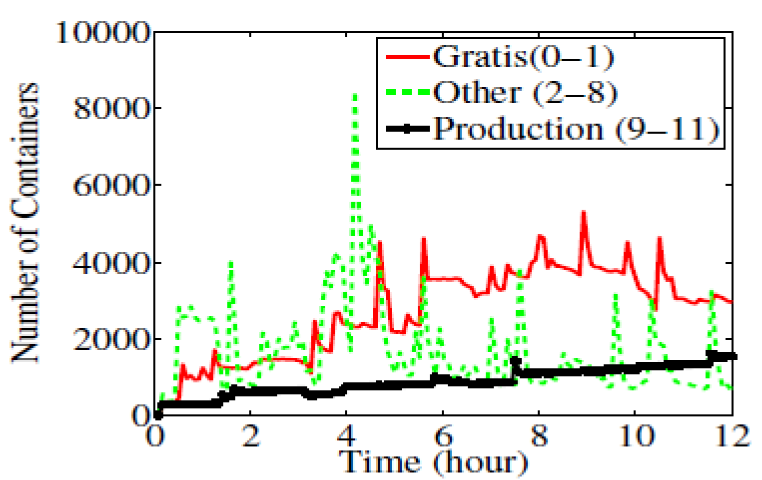
Experiments



Machine Configurations

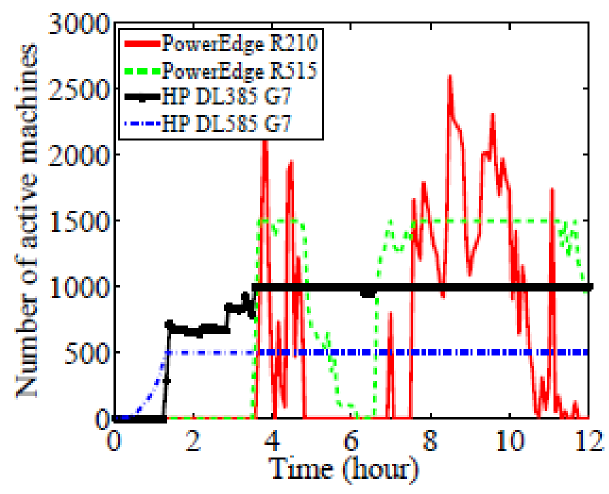


Aggregated task arrival rates

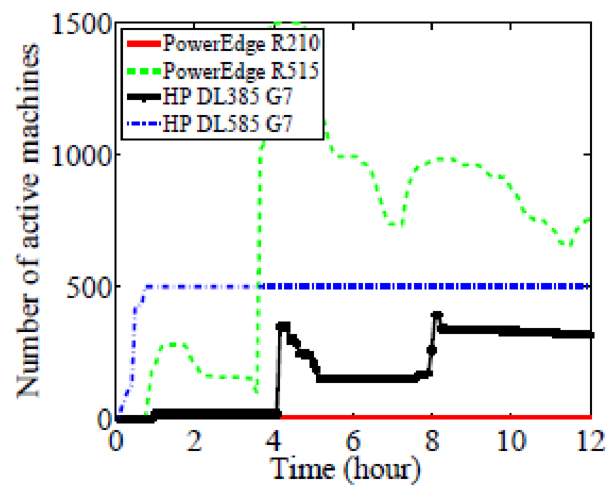


Number of containers

Experiments



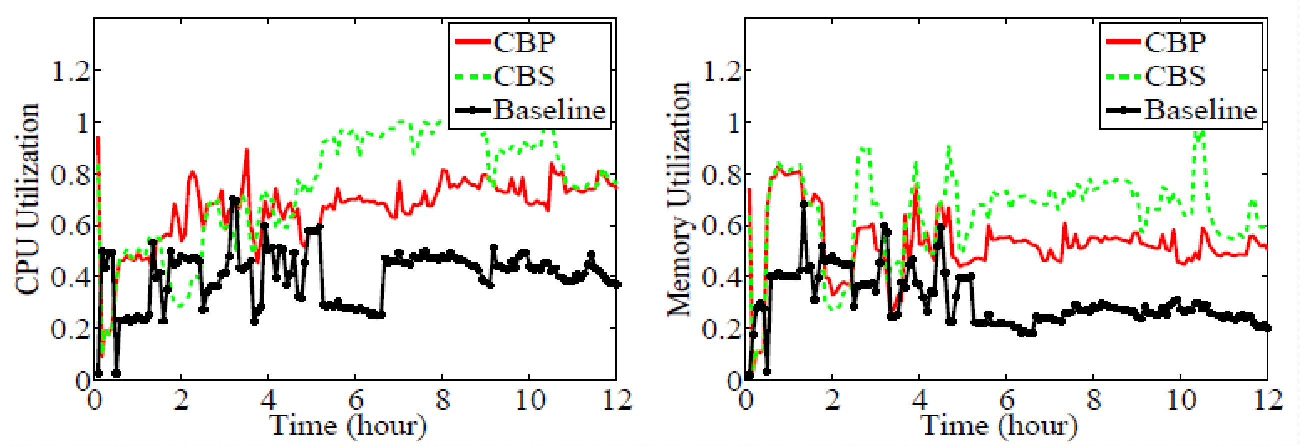
Number of machines baseline



Number of Machines CBS/CBP

- 3 types of schedulers
  - Baseline: always pick the most energy-efficient machine first
  - Container-based Provisioning
  - Container-based Scheduling

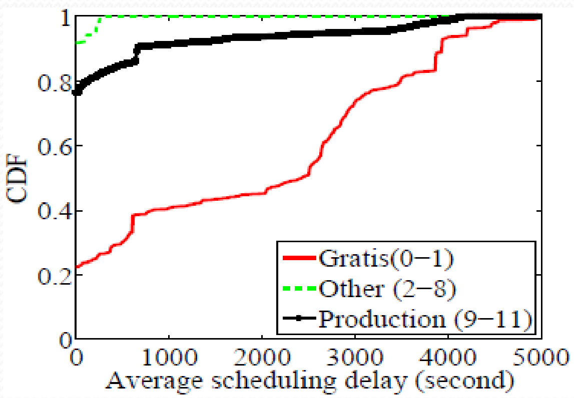
Experiments: Machine Utilization



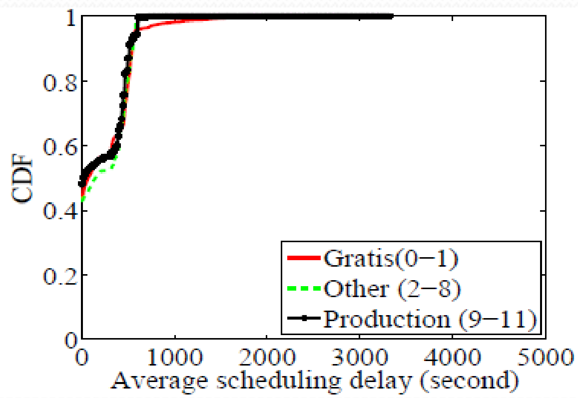
CPU Utilization   Memory Utilization

- 3 types of schedulers
  - Baseline: always pick the most energy-efficient machine first
  - Container-based Provisioning
  - Container-based Scheduling

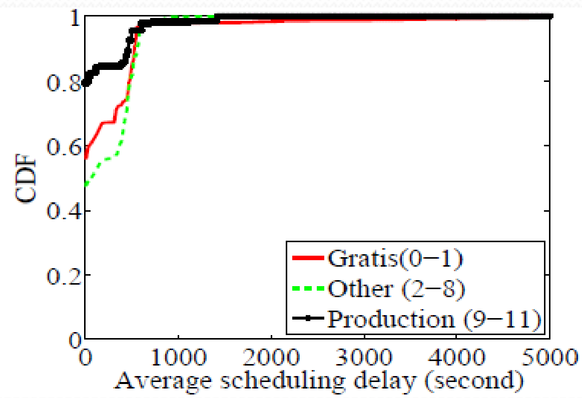
Experiments: Scheduling Delay



Baseline



Contain-based Provisioning



Contain-based Scheduling

- 3 types of schedulers
  - Baseline: always pick the most energy-efficient machine first
  - Container-based Provisioning
  - Container-based Scheduling

Conclusion

- We present Harmony, a heterogeneity-aware dynamic capacity provisioning framework
  - Dynamically adjust number of machines according to run-time task composition
- Experiments achieves much better scheduling delay and resource utilization than heterogeneity oblivious solutions
- Future work
  - Better clustering algorithms
  - Handling task placement constraints
  - Consider heterogeneous machine performances

Thank you!