CloudCmp: Shopping for a Cloud Made Easy

Ang Li Xiaowei Yang

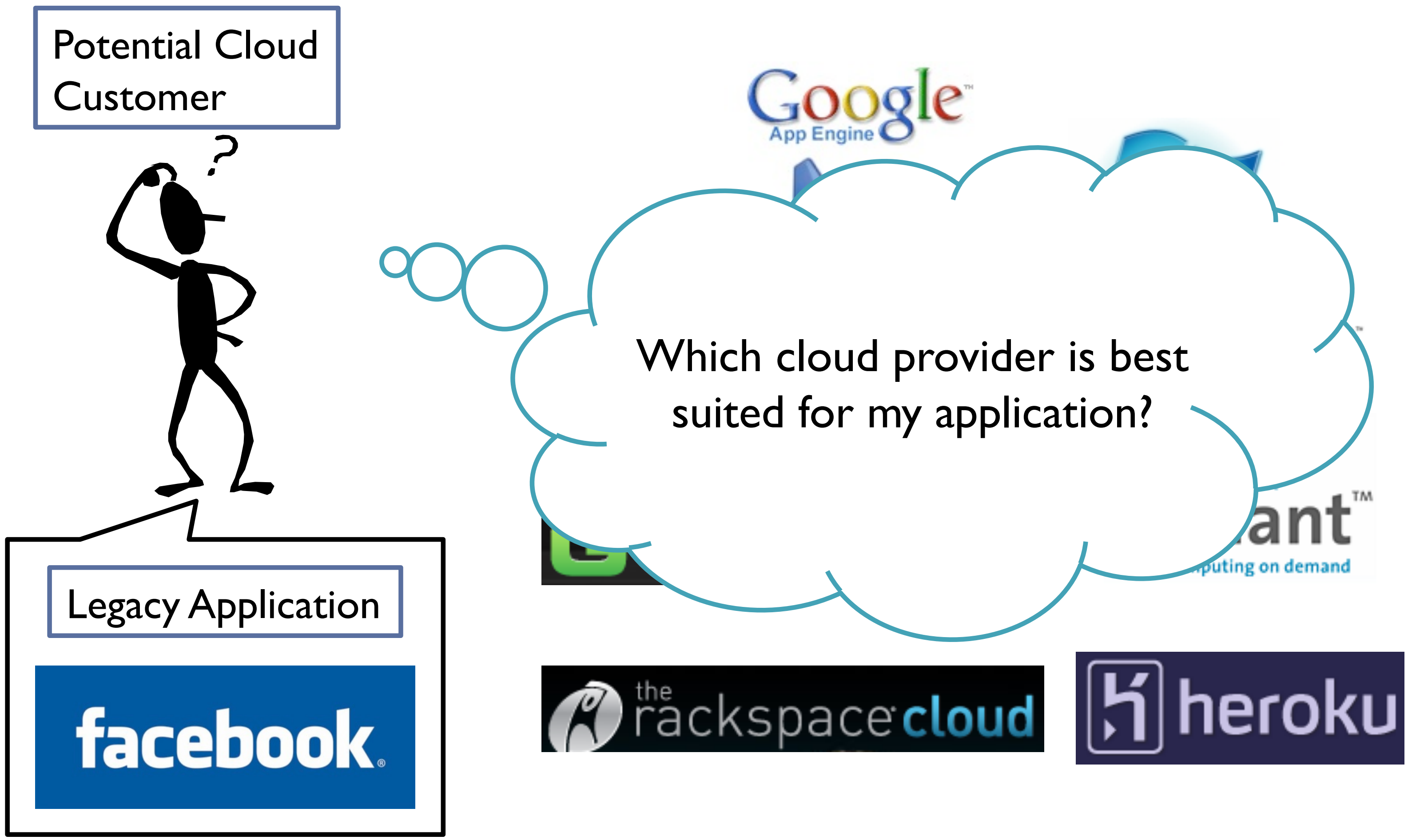
Duke University

Srikanth Kandula Ming Zhang

Microsoft Research

Motivation

- Cloud computing is gaining popularity



Answering this question is not trivial

- Reason #1: clouds have different service models
  - Infrastructure-as-a-Service
    - Virtual machines with customized guest OSes
    - Applications run on virtual machines using OS APIs
  - Platform-as-a-Service
    - Sandbox environment with specific platform APIs
  - A mixture of both
    - E.g., Amazon AWS

Unclear how to compare clouds with different service models

Answering this question is not trivial

- Reason #2: clouds offer different charging schemes
  - Pay per instance-hour
    - How many instances are allocated and how long each one is used
    - Charged regardless of utilization
  - Pay per CPU cycle
    - How many CPU cycles are consumed by the application
    - An idle application incurs no cost

Prices of different clouds are not directly comparable

Answering this question is not trivial

- Reason #3: applications have different characteristics
  - Storage intensive
    - E.g., backup services
  - Computation intensive
    - E.g., scientific computing, data processing (MapReduce, Dryad)
  - Network latency sensitive
    - E.g., online web services

One/few application benchmarks may not represent all types of applications

Answering this question is not trivial

- Reason #4: high overhead to port application to clouds
  - Different and incompatible APIs
    - Especially true for PaaS providers
  - Configuration and data migration
    - Time-consuming
    - Privacy concern

CloudCmp: help customers pick cloud

- The ultimate goal:

  Estimate the performance and costs of an application on a cloud without actually deploying it

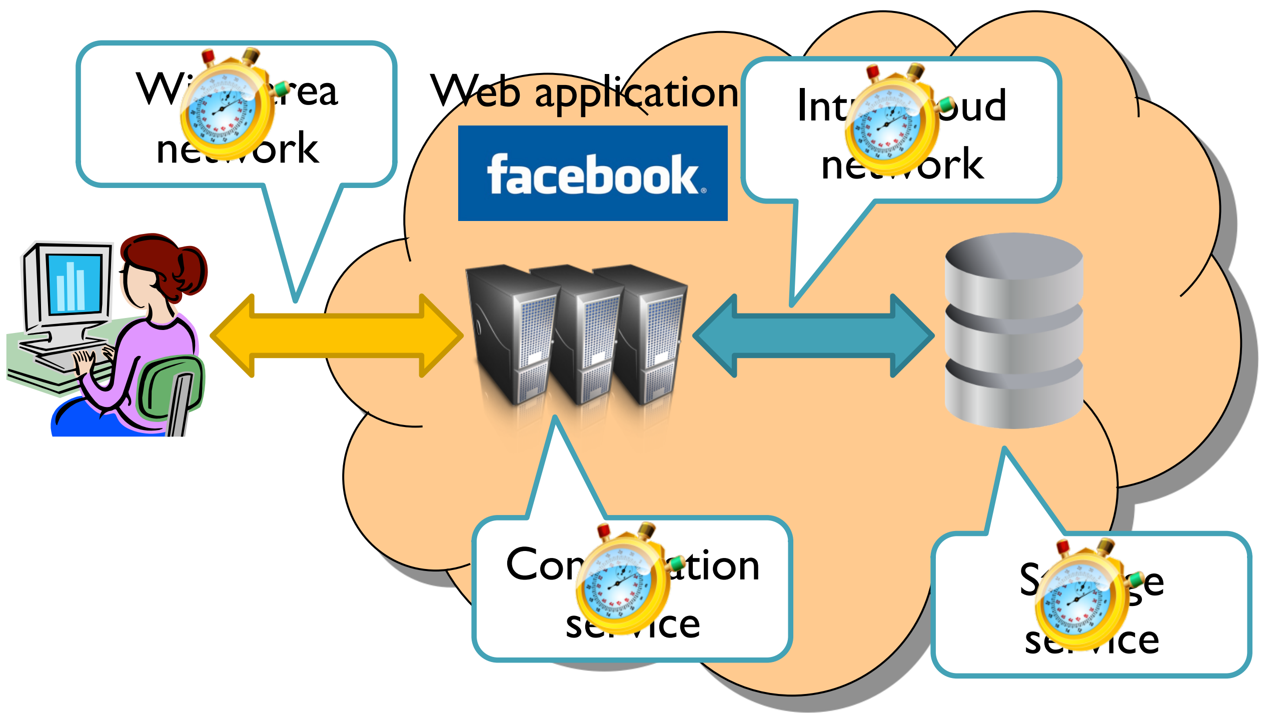
√ Application-specific
√ Little/no deployment overhead
√ Help understand performance-cost trade-off

Outline

- Proposed design of CloudCmp
  - Identify common services
  - Benchmark services
  - Capture application workload
  - Predict performance and costs
- Challenges
  - How to design the benchmarking tasks
- Benchmarking results
  - Correlate well with actual application performance
- Conclusion

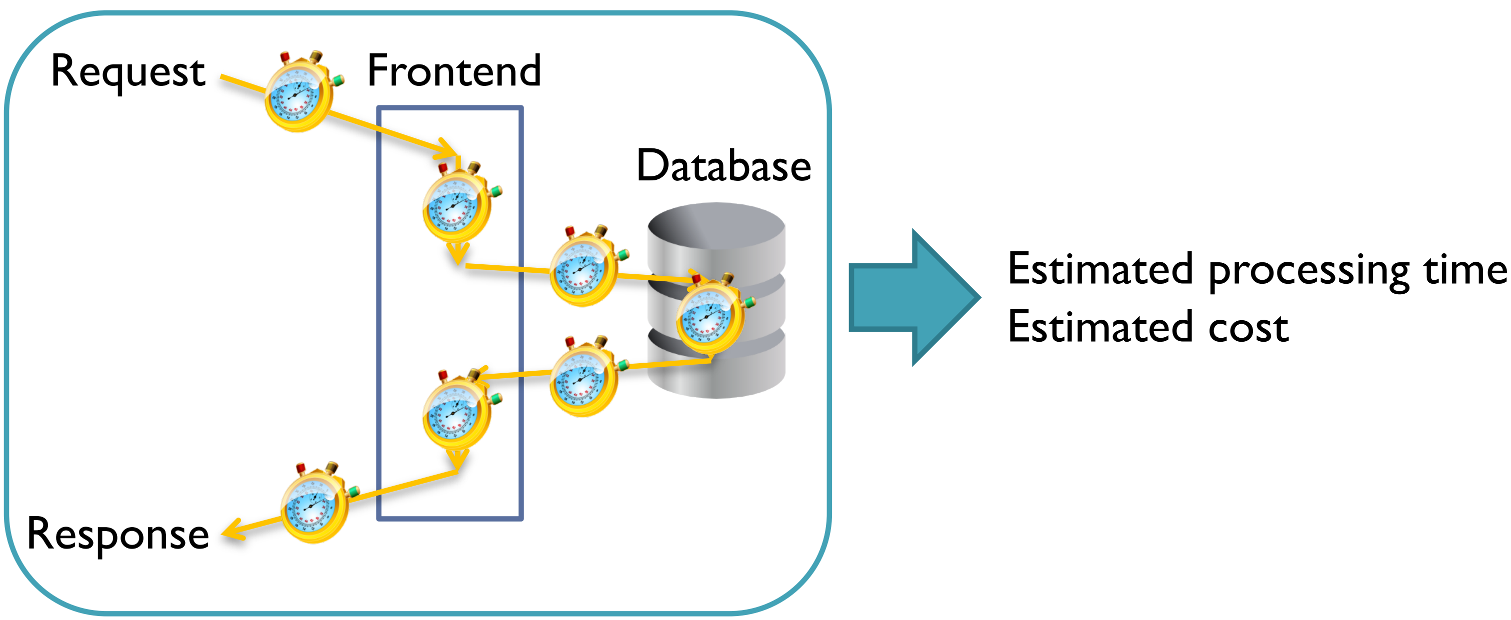
How does CloudCmp work?

- Step 1: identify the common cloud services
- Step 2: benchmark the services



How does CloudCmp work?

- Step 3: capture realistic application workload
  - Extract the execution path of each request
- Step 4: estimate the performance and costs
  - Combine benchmarking results and workload information



Challenges

- How to design the benchmarking tasks?
  - Fair and representative
- How to accurately capture the execution path of a request?
  - An execution path can be complex, across multiple machines
- How to estimate the overall processing time of an application
  - Applications can be multi-threaded

Challenges

- How to design the benchmarking tasks?
  - Fair and representative
- How to accurately capture the execution path of a request?
  - An execution path can be complex, across multiple machines
- How to estimate the overall processing time of an application
  - Applications can be multi-threaded

Designing benchmarking tasks: computation

- Java-based benchmarking tasks
  - CPU/memory/disk I/O intensive
  - Same byte-code on different providers
    - Minimize the bias introduced by different compilers/interpreters
- Measure the cost per task
  - Pay per instance-hour
    - Compute using the per hour price and the task running time
  - Pay per CPU cycle
    - Obtain the CPU cycles using cloud APIs

Designing benchmarking tasks: storage

- Test common storage operations
  - Insert/fetch/query
  - Test against tables with different sizes
- Measure each operation's latency and cost

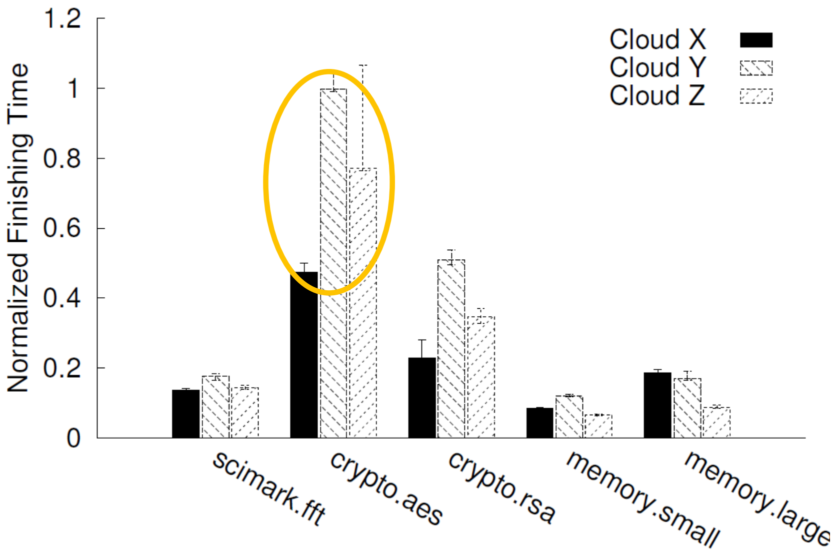
Designing benchmarking tasks: network

- Intra-cloud network
  - Measure the TCP throughput and latency between two randomly chosen instances
- Wide-area network
  - Measure the latency from vantage points on PlanetLab
    - Vantage points are chosen from diverse locations

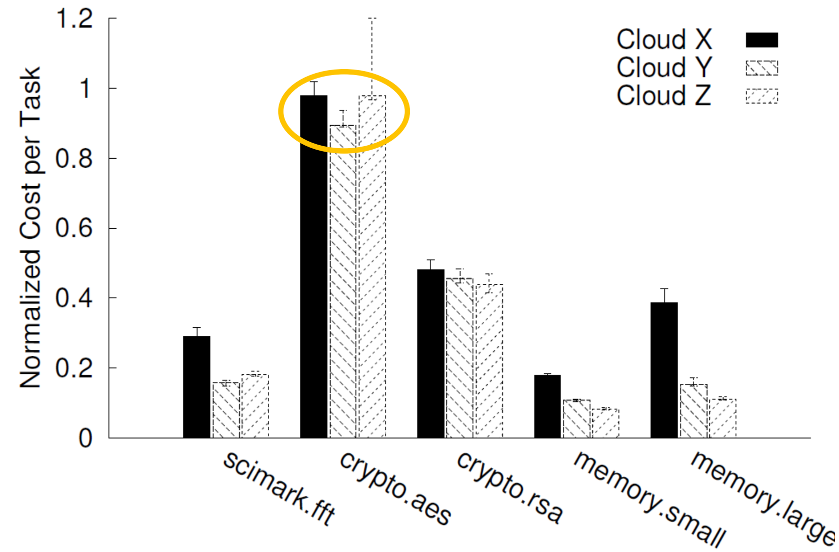
Benchmarking results

- Measure three popular cloud providers
  - One PaaS, two IaaS with storage APIs
  - Names of the clouds are removed due to legal concerns
    - Referred to as Cloud X, Y, and Z

Results: computation



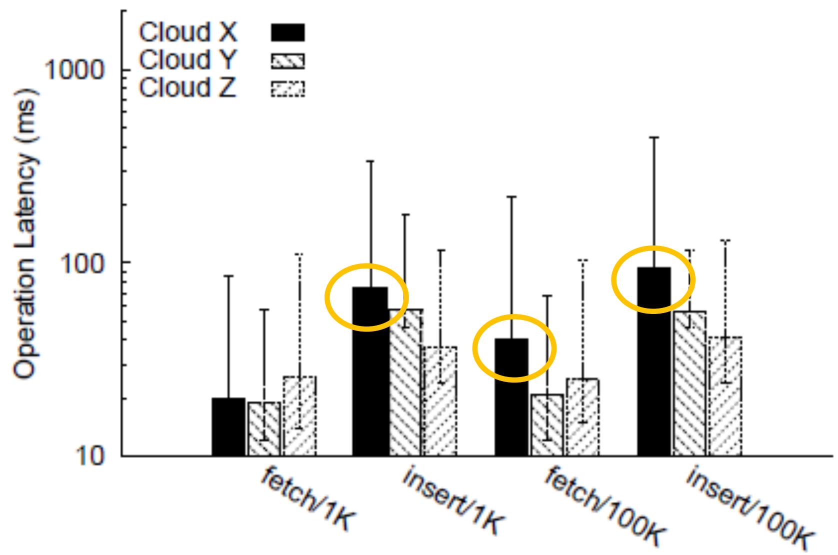
Benchmark finishing time



Benchmark running cost

At similar pricing points, different clouds can offer greatly diverse performance

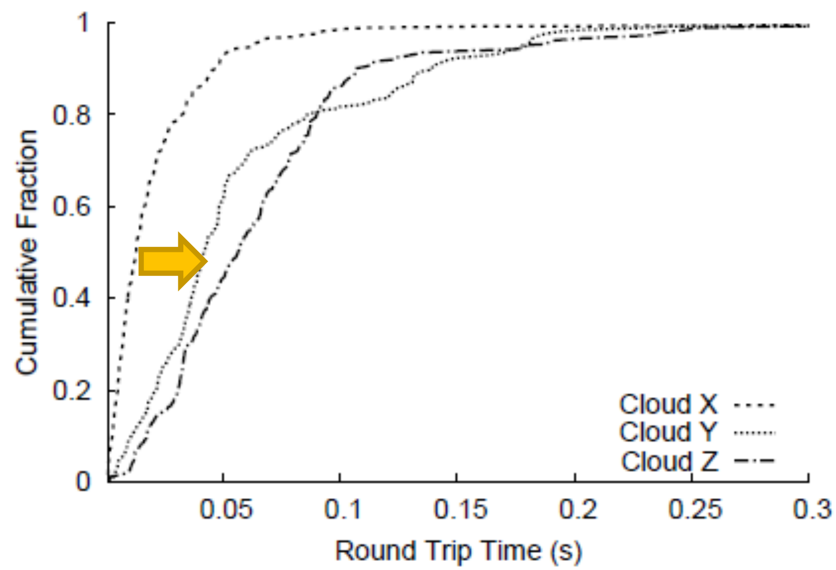
Results: storage



- Despite X's good performance in computation, its storage service can be slower than the others

Results: wide-area delivery network

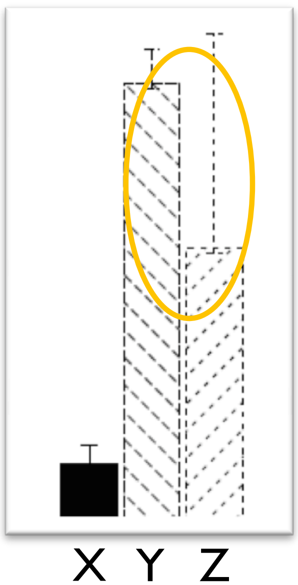
- Minimum latency to the closest data center

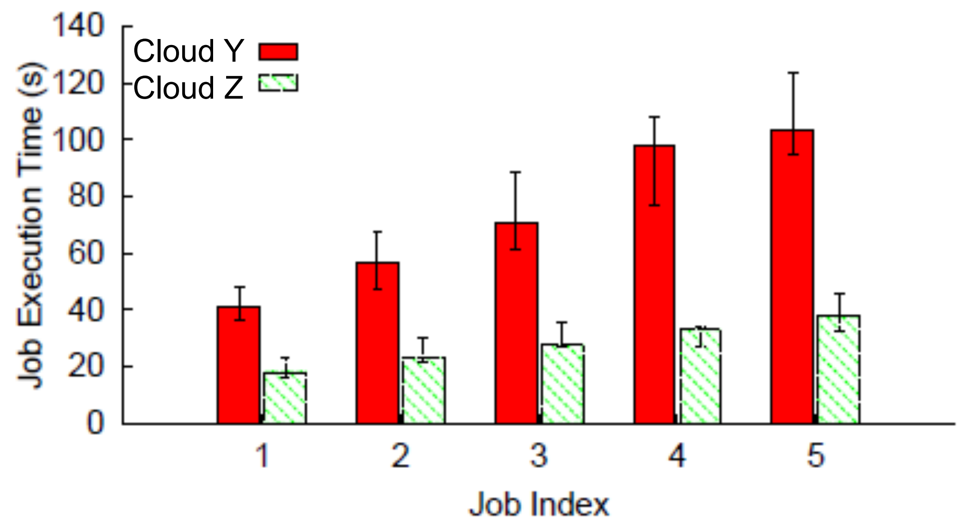


- On average, X's wide-area network latency can be up to 80% shorter than that of the others

Benchmarks are relevant to actual application performance

- Deploy real application on different clouds
  - BLAST: distributed, computation intensive





Future work: to estimate the exact time and costs using the benchmarking results

Conclusion

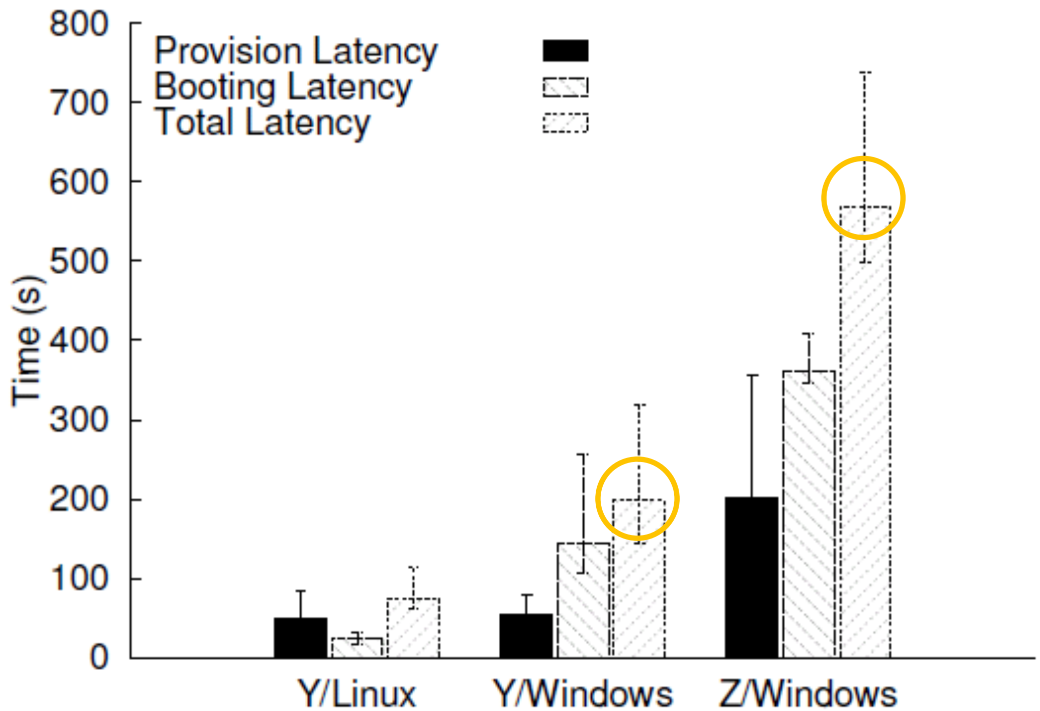
- Choosing the best-suited cloud is non-trivial
- CloudCmp aims to help compare cloud providers without actual deployment
  - Application-specific
  - Little deployment overhead
  - Estimate both performance and costs
- We think CloudCmp can be useful in practice
  - Clouds offer diverse performance
  - No cloud aces all services
  - Benchmarking results correlate well with actual application performance

Thank you

- Questions?
- http://cloudcmp.net

Backup slides

Scaling Latency



- The scaling latencies of different providers vary significantly (Z's latency is more than twice as high as Y's)
- The choice of operating system can affect scaling performance as well

Capture execution path accurately

- Blackbox tools to infer causal relationship
  - Do not require modifying the application
  - vPath [Tak09]
    - Exploit the common programming model of web applications
  - //Trace [Mesnier07]
    - A more general approach using the throttling technique

Estimate the overall processing time

- Simulate the execution process
  - Similar to the technique used in WebProphet [Li2010]
  - Estimate the time spent on each component using benchmarking results
  - Simulate the execution with the constraints of the causal relationships
    - E.g., component A depends on component B, then A can only be executed after B has finished