Automated Control for Elastic Storage

by: Harold C. Lim, Shivnath Babu, Jeffrey S. Chase

Synopsis by: Parnian Najafi Borazjani

Outline

- Introduction
- System overview
- System architecture and modeling methodologies
- Evaluation
- Contribution and related work
- Discussions and future work

Introduction

- Why Elastic Storage?
- Handle exponential growth
- A referenced Facebook Application went from 25,000 users to 250,000 users within three days
- Up to 20,000 new users per hour at peak

Introduction

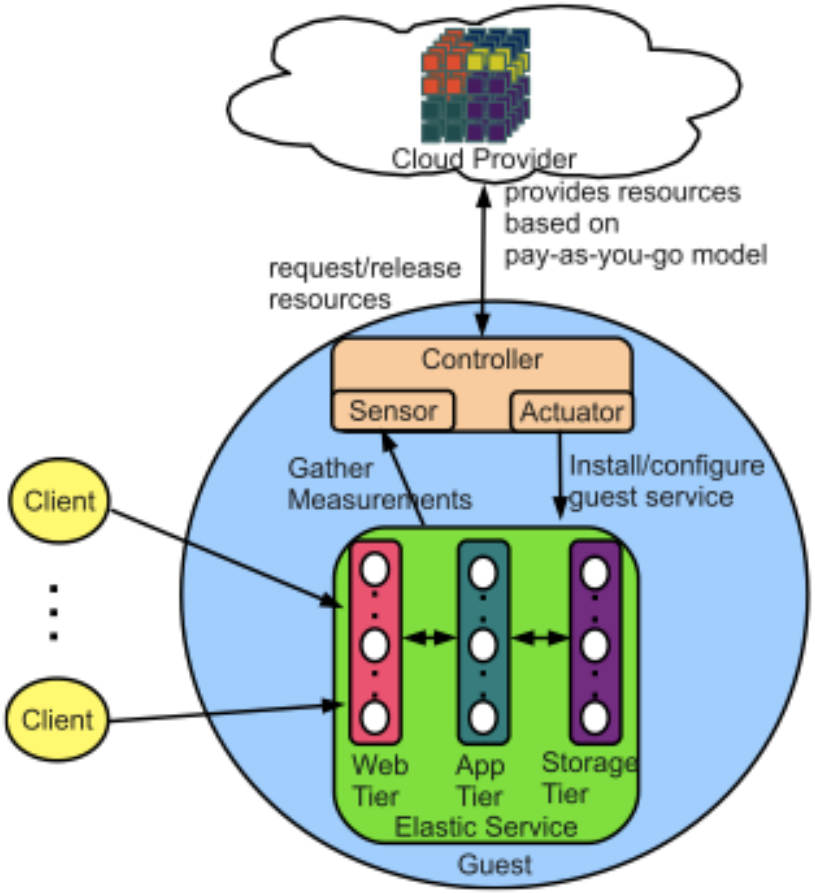
- Amazon's EC2 is a popular infrastructure as a service (IaaS)
  - Amazon Web Service (AWS) API allows customers to request, control, and release virtual server instances on demand
    - Pay as you go pricing based on per-hour usage charges
  - Increasing popularity of AWS platform
    - 9% growth in website usage from July to Aug '09
    - 181% annual growth rate

Introduction

- This paper addresses new challenges associated with scaling the storage tier in data-intensive cluster-based multi-tier services in cloud computing environment
- Employs an integral control technique called proportional thresholding to modulate the number of discrete virtual server instances in a cluster
  - Also address new challenges of actuator lag and interference due to the delay and cost of redistributing stored data on each change to the storage tier

System Overview

- Target environment — an elastic three-tier Web service hosted on server instances obtained on a pay-as-you-go basis from a cloud substrate provider
- The guest has a desired Service Level Objective (SLO) which sets a target level of acceptable performance metrics
- Response time



System Overview

- Challenges unique to storage tier:
  - State rebalancing
  - Actuator lag
  - Interference
  - Coordination of multiple interacting control elements

System Overview

- The Cloudstone application service is used as a target guest
  - Mimics a dynamic calendar events website
  - Users can browse, create, and join calendar events

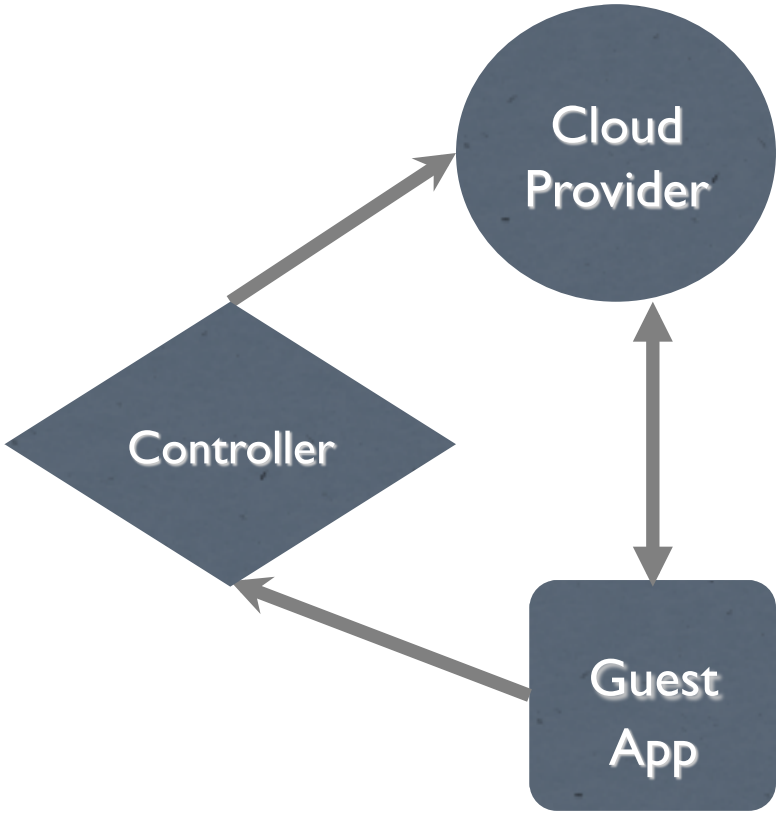
System Overview: Controller

- Runs on behalf of the guest and automates elasticity
  - Actuators : request and release instances
  - Sensors : request volume, utilization, and response time
- Combines multiple control elements into one controller with separate rules for each:
  - Resizing each tier
  - Rebalancing storage tier
- Ideal situation:
  - Controller policy can handle unanticipated workload fluctuations (e.g. flash crowds)
  - Ensure minimum payment necessary for required SLO

System Overview: Controller

- The controller runs outside of the cloud provider and is distinct from the guest application itself in cloud systems with per-instance pricing.

- Allows for application specific control policies that generalize across multiple cloud providers



System Overview: Controller

- Problem: Discrete Actuators instead of Continuous Actuators
  - The controller is limited in elasticity by the actuators exposed in the cloud provider's API
    - EC2 allocates resources in discrete units as virtual server instances of predetermined sizes (e.g. small, medium, large)
- Previous work assumes continuous actuators such as fine-grained resource entitlement on each instance

System Overview: Controller

- This paper develops a proportional thresholding technique for stable integral control with coarse-grained discrete actuators, and applies it to elastic control of the storage tier in a cloud with per-instance pricing.
  - This same technique can be used for the application tier of a multi-tier service
  - It is necessary to coordinate these controllers across tiers for an integrated multi-tier solution

System Overview: Controller

- This approach assumes that each tier exports control APIs that may be invoked to add a newly acquired storage server to the pool (join) and remove an arbitrary server from the group (leave)
  - These join and leave operations may configure the server instances in anyway necessary to attach or detach server instances to the guest application
- It is also assumed that there is a mechanism to balance load across the servers within each tier, so that request capacity scales roughly with the number of active server instances

System Overview: Controlling Elastic Storage

- Storage Tier Assumptions:
  - It distributes stored data across its servers in a way that balances load effectively for reasonable access patterns, and redistributes (*rebalances*) data in response to join and leave events
  - Robust replication; it is sufficient to avoid service interruptions across a sequence of leave events, even if a departing server is released back to the cloud before leave rebalancing is complete
  - The storage capacity and I/O capacity of the system scales roughly linearly with the size of the active server set

System Overview: Controlling Elastic Storage

- Many robust incrementally-scalable cluster storage services have been developed since the early 1990s.

- For this paper, the chosen service is the Hadoop Distributed File System (HDFS), which is based on the Google File System design and is widely used in production systems

System Overview: Controlling Elastic Storage

- Challenge 1 of 3 – Data Rebalancing
  - Adding a new storage node does not give immediate performance improvements to an elastic storage system because the node does not yet have any persistent data to serve client requests
  - Data must first be copied to the new storage node

System Overview: Controlling Elastic Storage

- Challenge 2 of 3 – Interference to Guest Service
  - Data rebalancing consumes resources that can otherwise be used to serve client requests
  - The amount of resources allocated to rebalancing affects its completion time as well as the degree of adverse impact on the guest application's performance during rebalancing
- Note that overall improvement to system performance can be achieved only through data rebalancing.
- Finding the right balance automatically is nontrivial

System Overview: Controlling Elastic Storage

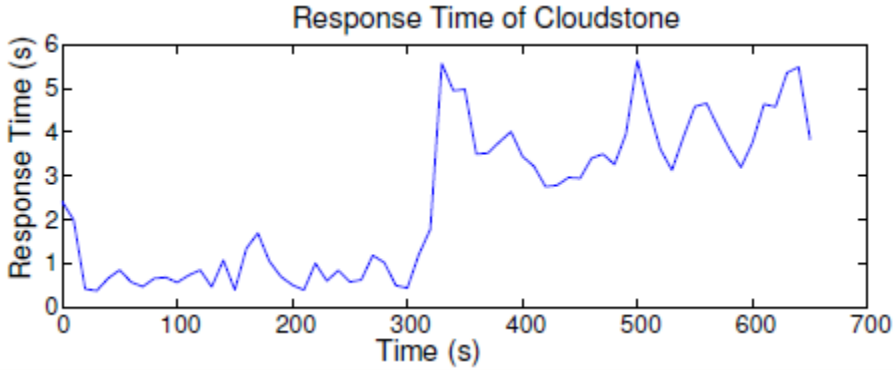
- Challenge 3 of 3: Actuator Delays
  - Even with full bandwidth allocated for rebalancing, there will always be a delay before performance improvements can be observed
    - The controller must account for this delay, or else it may respond too late or (worse) become unstable

Components of the Controller

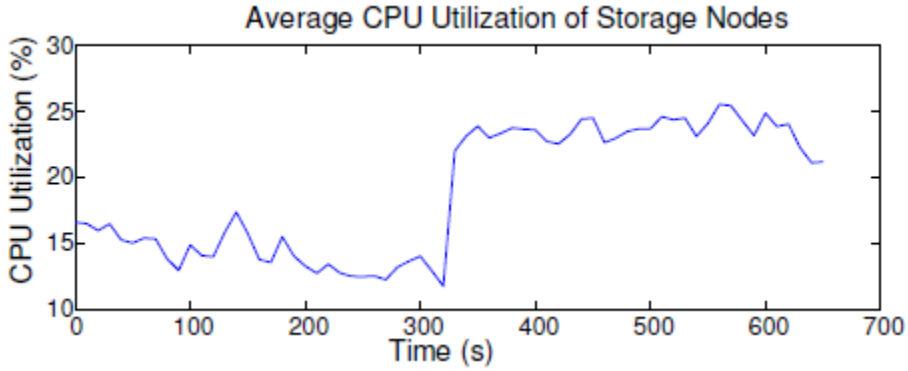
- Three components of the *elasticity controller:*
  - Horizontal Scale Controller (HSC), responsible for growing and shrinking the number of storage nodes
  - Data Rebalance Controller (DRC), responsible for controlling the data transfers to rebalance the storage tier after it grows or shrinks
  - State machine, responsible for coordinating the actions of the HSC and the DRC

System architecture - Horizontal Scale Controller (HSC)

- Actuator: The HSC uses cloud APIs to change the number of active server instances.
- Sensor: The paper uses CPU utilization on the storage nodes as the sensor feedback metric
  - It is easy to measure, and strongly correlated to overall response time of the Cloudstone



b



Modeling methodology - Controller - Integral control

- Control Policy (*K*): Integral control

$$u_{k+1} = u_k + K_i \times (y_{ref} - y_k)$$

- K_i - the integral gain parameter.
- y_k - the current sensor measurement.
- y_{ref} - the desired reference sensor measurement, which is 20% CPU utilization for 3 second average response time.
- u_{k+1}- new actuator value
- u_k - current actuator value

Modeling methodology - Controller - discrete control functions

- Because *discrete* actuators (instance number) are used in the system, the paper generates the following discrete control functions:

$$u_{k+1} = \begin{cases} u_k + K_i \times (y_h - y_k), & if\, y_k > y_h \\ u_k + K_i \times (y_l - y_k), & if\, y_k < y_l \\ u_k, & otherwise \end{cases}$$

- y_h and y_l are the higher and lower thresholds for CPU utilization y_k.
- Only when $y_k > y_h$ (under-provisioned) or $y_k < y_l$ (over-provisioned), $u_{k+1} \neq u_k$, i.e., the controller adds/removes the storage instances.

Modeling methodology - Proportional thresholding

- How to set y_h and y_l?
  - They can't be static, because for a cluster of size *N*, adding/removing a node affects *1/N* of the total capacity.
- "*Proportional thresholding*" mechanism:
  - Set $y_h = y_{ref}$, and vary y_l to vary the range.
  - *CPU* = $f(workload)$, $workload = f^{-1}(CPU)$
  - Suppose *"workload"* is the per-node workload and we have *N* instances. We get
    - $workload_l = workload_h \times \frac{N-1}{N}$
  - Suppose $y = f(workload)$, we get
    - $y_l = f(workload_l) = f(f^{-1}(y_h) \times \frac{N-1}{N})$

System architecture - Data Rebalance Controller (DRC)

- The DRC rebalances the layout of data in the system after the number of storage nodes grows or shrinks.
- Rebalancing is a cause of actuator delay and interference.
- Tuning knob of HDFS rebalancer:
  - *Bandwidth **b*** allocated to the rebalancer.
- Select *b* to control the tradeoff between *lag* and *interference*.
  - Big *b* - fast rebalance, serious impacts on normal service.
  - Small *b* - slow rebalance, not very disruptive to normal service.

Modeling Methodology - Modeling the impacts of *b*

- The paper employed multi-variate regression to decide b:
  - The time to completion of rebalancing ($Time$) as a function of the bandwidth throttle (b) and size of data to be moved(s): $Time = f_t(b, s)$.
  - The impact of rebalancing on service response time ($Impact$) as a function of the bandwidth throttle (b) and per-node workload (l): $Impact = f_i(b, l)$.
- Values of s and l are measured by sensors in DRC.

Modeling Methodology - Balancing between *lag* and *interference*

- The Data Rebalance Controller poses the choice of b as a cost-based optimization problem:
  - $Cost = \alpha Time + \beta Impact$
  - $= \alpha f_t(b, s) + \beta f_i(b, l)$
  - The ratio of α/β can be specified by the guest based on the relative preference towards *Time* over *Impact*.

System architecture - State machine

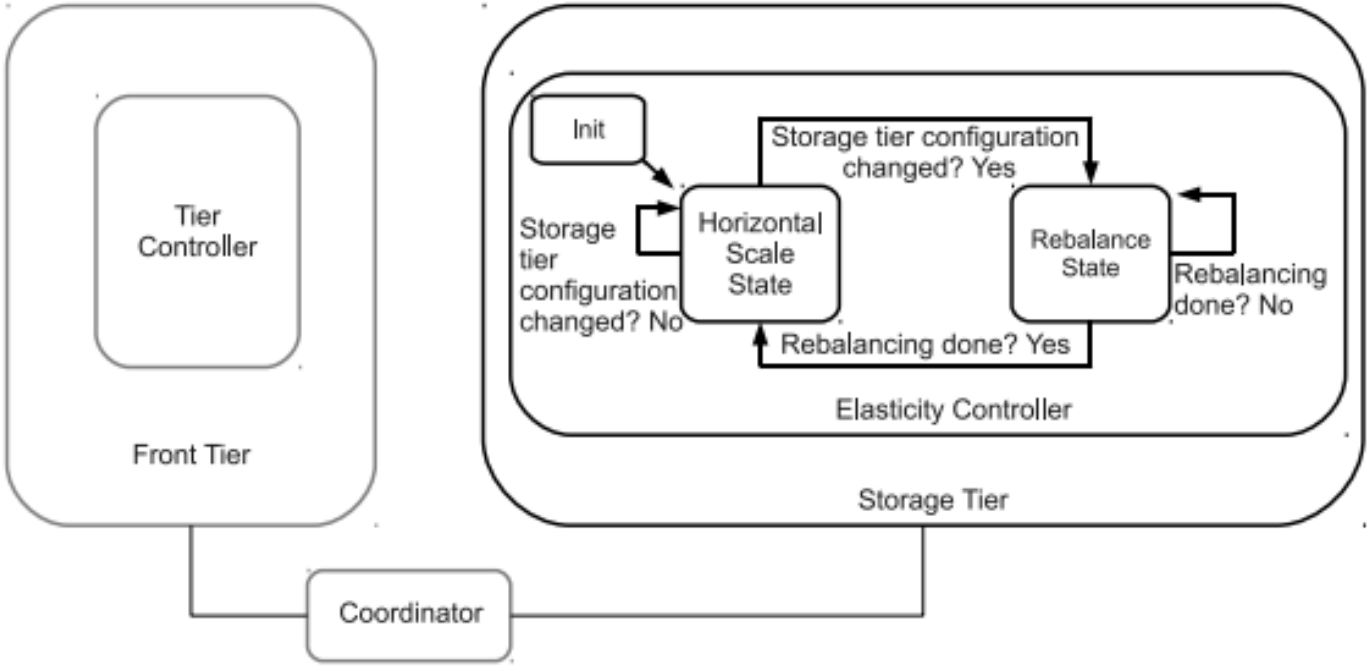
- Recall that:
  - *Horizontal Scale Controller (HSC)* is used to increase/shrink the number of storage nodes
  - *Data Rebalance Controller (DRC)* is used to rebalance the storage after the changes in storage node size

- They have mutual dependencies:
  - After HSC adds a new storage node, the system cannot obtain full service until DRC completes rebalancing.
  - When one component is taking actions, the noise will be introduced to the sensor measurements of the other one.

- To preserve stability during adjustments, a state machine is employed to coordinate HSC and DRC to manage their mutual dependencies.

System architecture - State machine

- The following diagram shows the internal state machine of the elasticity controller in the storage tier.



Evaluation - Experimental Testbed

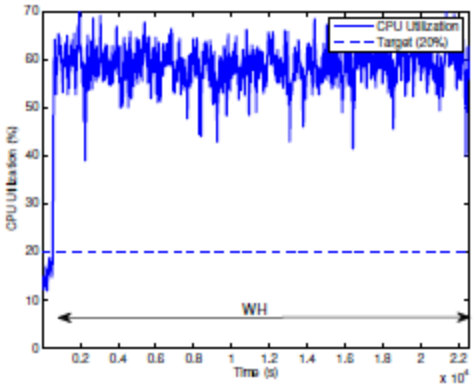
- The paper employs CloudStone to run with GlassFish as the front-end application server tier.
  - CloudStone: a flexible Web 2.0 benchmark generator
  - GlassFish: an open source application server project
- HDFS is used for the storage
  - HDFS is modified to expose the rebalancer's bandwidth throttle *b* as an actuator to the external controller.
- The paper implements a local ORCA cluster as the cloud infrastructure provider
  - ORCA: A resource control framework that provides a resource leasing service; guests can lease resources from a substrate resource provider, such as a cloud provider

Evaluation - Experimental Testbed

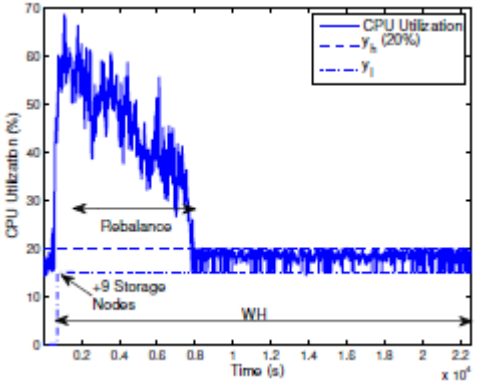
- The experimental service cluster:
  - A group of servers running on a local network.
- To fully explore the effects of the storage tier:
  - Other tiers are statically over-provisioned.
- The storage tier nodes:
  - Dynamically allocated virtual machine instances
  - They all have fixed resource configurations:
    - 30 MB disk space; 512 MB RAM; single disk arm; 2.8 GHz CPU.
- HDFS is preloaded with at least 36 GB data.

Evaluation - Controller Effectiveness

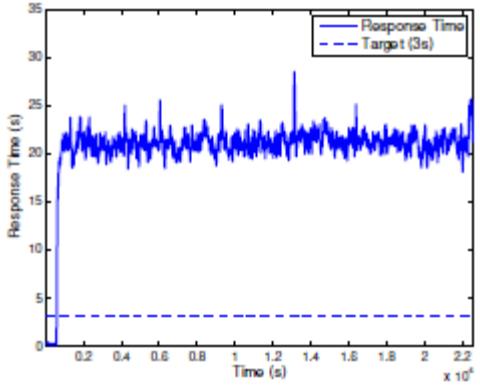
- Static and dynamic resource previsioning to load burst of 10 times at $t = 0.06 \times 10^4 s$.



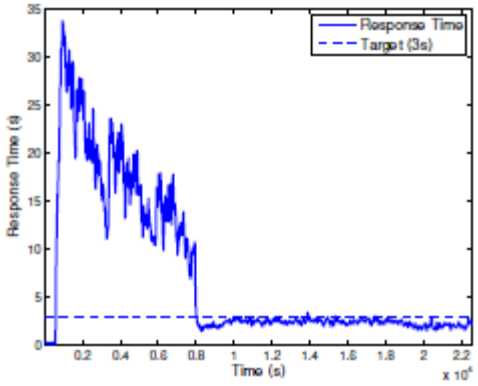
a1. CPU utilization - static



a2. CPU utilization - dynamic



b1. Response time - static



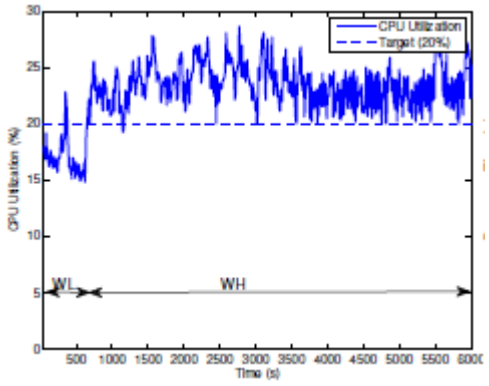
b2. Response time - dynamic

Target response time: 3 seconds.
Target CPU utilization: 20%.

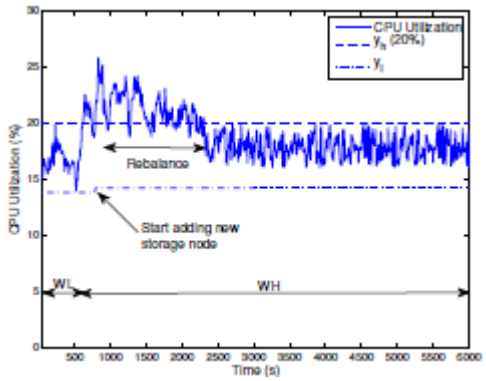
See from the figures:
1. Dynamic provisioning is able to adapt to the load burst.
2. Instance creation and data rebalancing has cost and delay on effect.

Evaluation - Controller Effectiveness

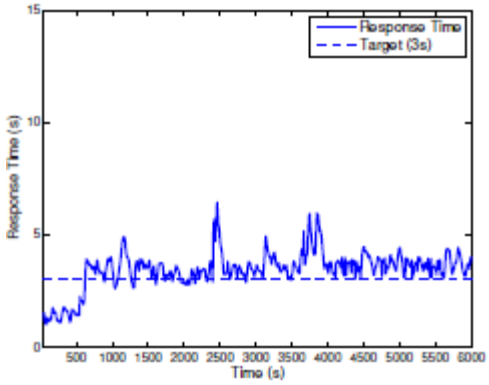
- Static and dynamic resource previsioning to small load increase of 35% at $t = 0.06 \times 10^4 s$.



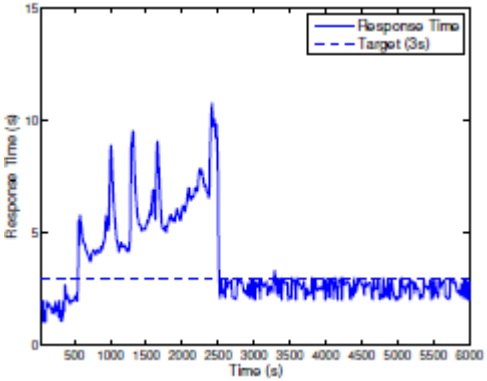
a1. CPU utilization - static



a2. CPU utilization - dynamic



b1. Response time - static



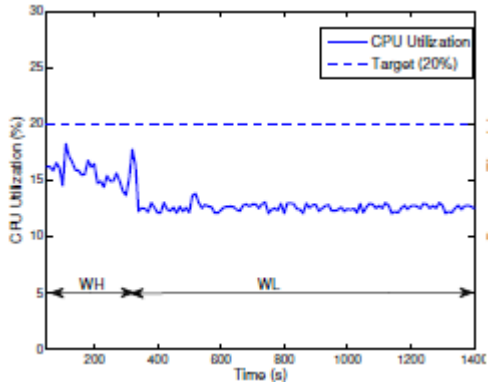
b2. Response time - dynamic

Target response time: 3 seconds.
Target CPU utilization: 20%.

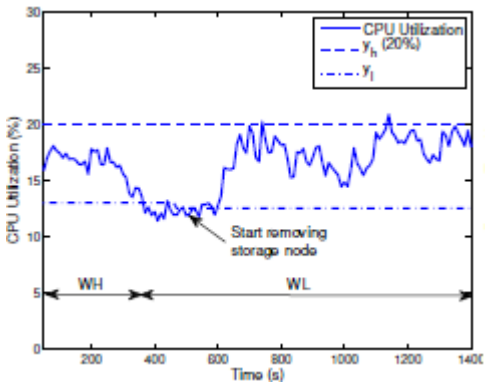
See from the figures:
1. Dynamic provisioning is alert enough to adapt to the small load increase.
2. The cost and delay of node creation/rebalancing are smaller than the prev.

Evaluation - Resource Efficiency

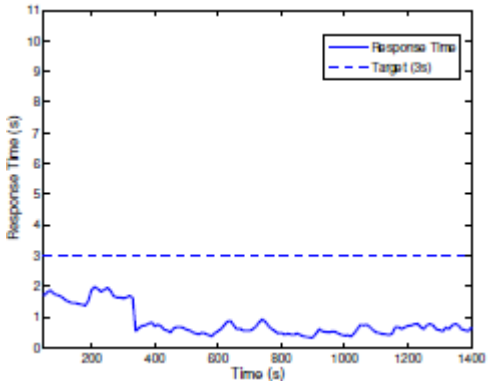
- Static and dynamic resource previsioning to load decrease of 30% at $t = 370s$.



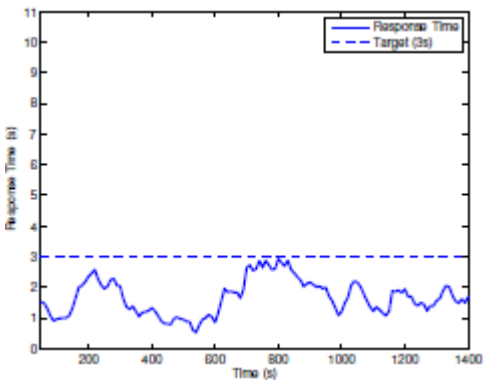
a1. CPU utilization - static



a2. CPU utilization - dynamic



b1. Response time - static



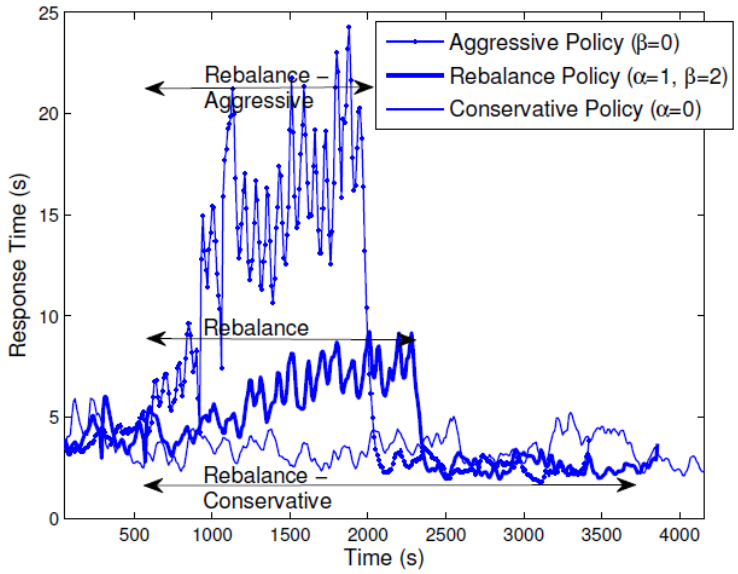
b2. Response time - dynamic

Target response time: 3 seconds.
Target CPU utilization: 20%.

See from the figures:
1. Shrinking the storage size has much lower cost/ delay than increasing it.
2. During resizing process, There are almost no SLO violations.

Evaluation - Comparison of Rebalance Policies

- Recall that:
  - $Time = f_t(b, s)$, monotone decreasing function of b.
  - $Impact = f_i(b, l)$, monotone increasing function of b.
  - And we want to optimize for the cost function:
    - $Cost = \alpha Time + \beta Impact = \alpha f_t(b, s) + \beta f_i(b, l)$



Contribution and related work

- This paper is the first to address the problem of automated control for elastic storage in cloud computing.
- SCADS is a related work dealing with dynamically scaling a storage system.
  - It uses machine learning to predict resource requirements.
- Padala *et al.* proposed a decoupled architecture (between guest and cloud provider) for cloud computing.
  - They did not consider the actuator constraints.
- Aqueduct uses a feedback controller to throttle the rebalancing bandwidth usage to ensure the SLOs will not be violated.
  - The rebalancing may be able to use very little bandwidth.

Discussions and future work

- Make the resource configuration of newly created storage instances *tunable*.
  - Resizing storage size by adding/removing storage instances with flexible resource configuration.
  - Optimizing the system by exploring the capacity and efficiency of individual storage instances, rather than storage instance amount.
  - This requires investigating the performance of storage nodes under different setups: disk size, CPU frequency, RAM size, etc.

Questions?