CS15-319 / 15-619 Cloud Computing

Recitation 4

September 22, 2015

Overview

- Administrative Stuff
  - OH hours, Piazza guidelines (search before asking!)
  - Monitor AWS expenses regularly
  - Do calculations before launching services
  - Terminate your instances when not in use
  - Stopping instances still has EBS cost ($0.1/GB-Month)
  - Tag your instances within the first hour!
  - Make sure spot instances are tagged!

Piazza Posts

1. mapreduce_map_input_file

   `s3://cmucc-datasets/wikipediatraf/201508pagecount-20150801-xxx.gz`
2. Java Memory Issues

   `--args -s,mapred.tasktracker.map.tasks.maximum=`**`maximum number of simultaneous map tasks`**

   `--args -m,mapred.child.java.opts=-Xmx`**`amount of memory in MB`**
3. Posts about AWS outage. @609 and @646 9/20/2015 Affected Netflix, Nest, Reddit, IMDB, students etc.

Reflection

- Last Week's Reflection
  - Project 1.2, Quiz 2
- Theme - **Big data analytics**
  - P1.1: Sequential Analysis of 418MB wikipedia data
  - P1.2: Parallel Analysis of 69GB compressed (300 GBs uncompressed) wikipedia data
    - 1 hour with 9 large worker instances
    - Imagine how many hours it will take for one instance to complete the work
- Power of Parallel Analysis
  - Amount of work done remains the same
  - Span, however reduces significantly

Reflection

- You should have learned
  - Use smaller instances to test correctness of your code prevents you from spending over budget
  - Why MapReduce is used to analyze big data
  - How MapReduce works
  - How to write a Mapper and Reducer
  - Performance/cost tradeoff
  - How to narrow down bugs by using logs
- Don’t forget about them just yet!
  - They will still be relevant in 15619 Project and Project 4

This Week

- This Week's Schedule
  - Quiz 3 (OLI Modules 5 & 6)
    - Due on Friday, Sep 25th, 2015, 11:59PM ET
  - Project 2.1
    - Due on Sunday, Sep 27th, 2015, 11:59PM ET

OLI Module 5 - Cloud Management

Cloud Software stack - enables provisioning, monitoring and metering of virtual user “resources” on top of the Cloud Service Provider’s (CSP) infrastructure.

- Cloud middleware
- Provisioning
- Metering
- Orchestration and automation
- Case Study: Openstack - Open-source cloud stack implementation

OLI Module 6 - Cloud Software Deployment Considerations

- Programming the cloud
- Deploying applications on the cloud
- Build fault-tolerant cloud services
- Load balancing
- Scaling resources
- Dealing with tail latency
- Economics for cloud applications

Guest Lecture

Mark Russinovich
CTO Microsoft Azure
Thursday, 9/24
4:30pm in GHC 4307

QoS

What is important to a web service provider?

- Quality of Service
  - Performance - Throughput & Latency (Projects 2.1 and 3)
  - Availability - Whether the web service is present and ready for immediate use (Projects 2.2 and 3)
  - Reliability - Measured by the number of failures per month (Projects 2.2 and 3)
  - Security - Will not be a major theme in this course
- Cost
  - Cost continues to be the bottom line for all technology solutions
  - Restricts the QoS

QoS and cost are two important objectives to many web service providers.

QoS ⇒ Performance

- Performance
  - Throughput - The number requests served by a web service during a given time period
    - e.g. How many users can stream a video from a single server at a given time?
  - Latency - Round-trip time (RTT) between sending a request and receiving a response
    - e.g. Can you play *Dota* or *League of Legends* if the latency is > 300 ms?
- Goal is to maximize throughput, minimize latency and minimize cost. How?
  - Increase number of servers to increase throughput. (Project 2)
  - Upgrade server specs (increase CPU, RAM) for each machine. (Project 2)
  - Geographic replication (e.g. Content Delivery Networks). (Project 3)

Project 2 Overview

- Project 2.1 AWS APIs, Scalability & Elasticity
  - Vertically Scaling, Horizontally Scaling and Load Balancing.
- Project 2.2 Elasticity, Failure and Cost
  - Able to horizontally scale out and scale in, to maximize throughput, minimize cost.
- Project 2.3 Explore Load Balancing
  - Able to distribute load evenly amongst your servers.

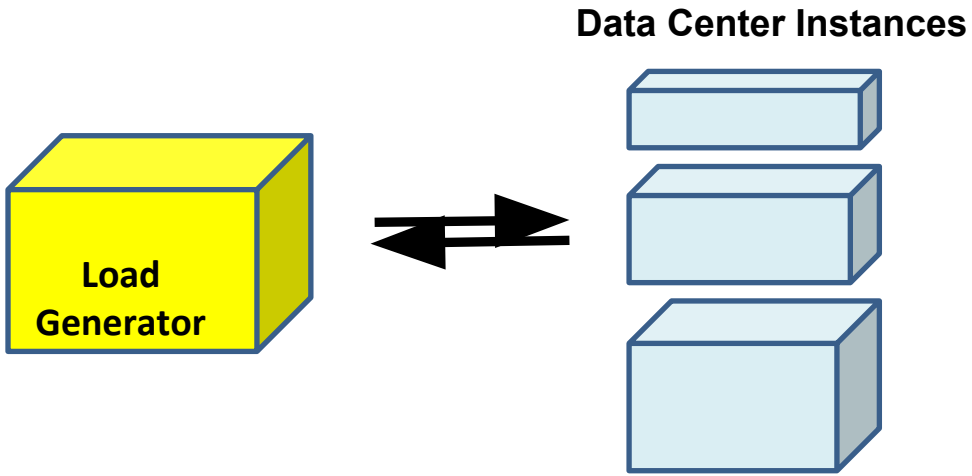
Questions So far?

Project 2.1 Learning Objectives

- Get familiar with AWS APIs for provisioning/de-provisioning instances.
- Be able to define horizontal and vertical scaling and to distinguish between them.
- Identify when to scale horizontally (fluctuating load can be handled better because of cost effectiveness) versus vertically (predictive scaling during times when increased usage is expected for consistent duration).
- Demonstrate the ability to identify the patterns of some predictable load and horizontally scale accordingly.
- Get familiar with, and understand the need for, Amazon's Elastic Load Balancer.
- Understand the limitations of Amazon's Elastic Load Balancing.

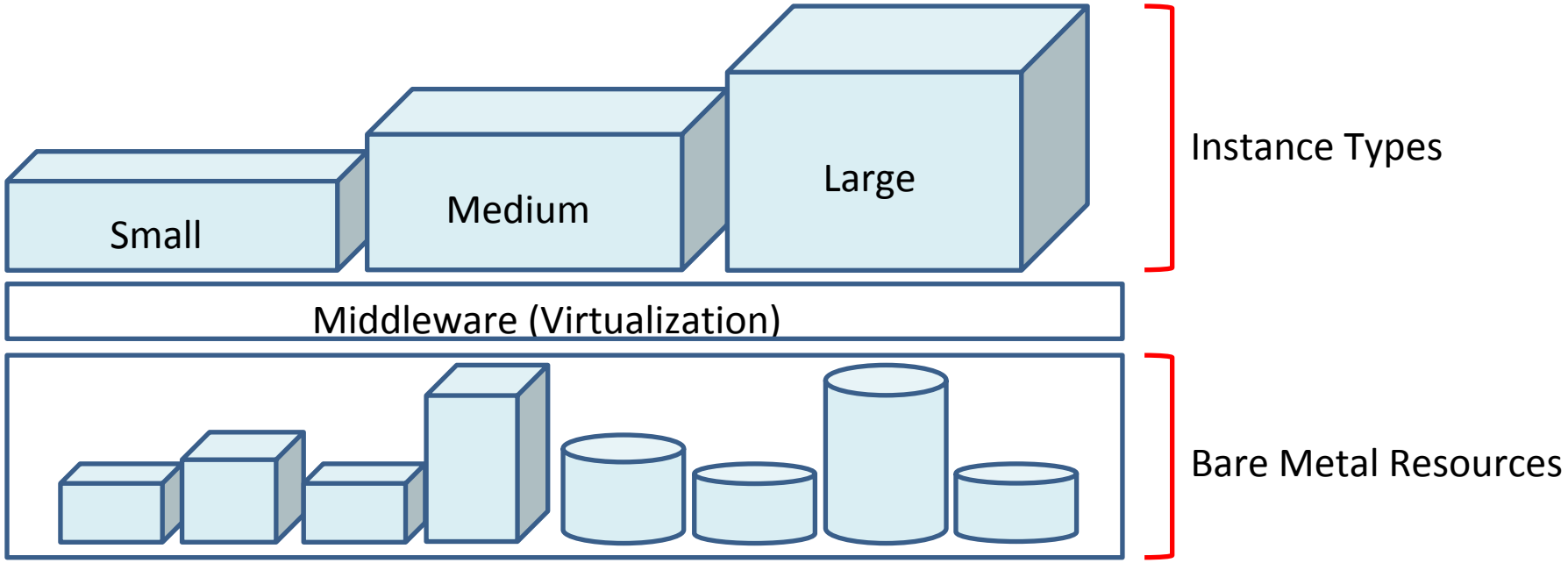
Architecture of P2.1

Load Generator: Emulates hundreds/thousands of people requesting service from the data center instances.

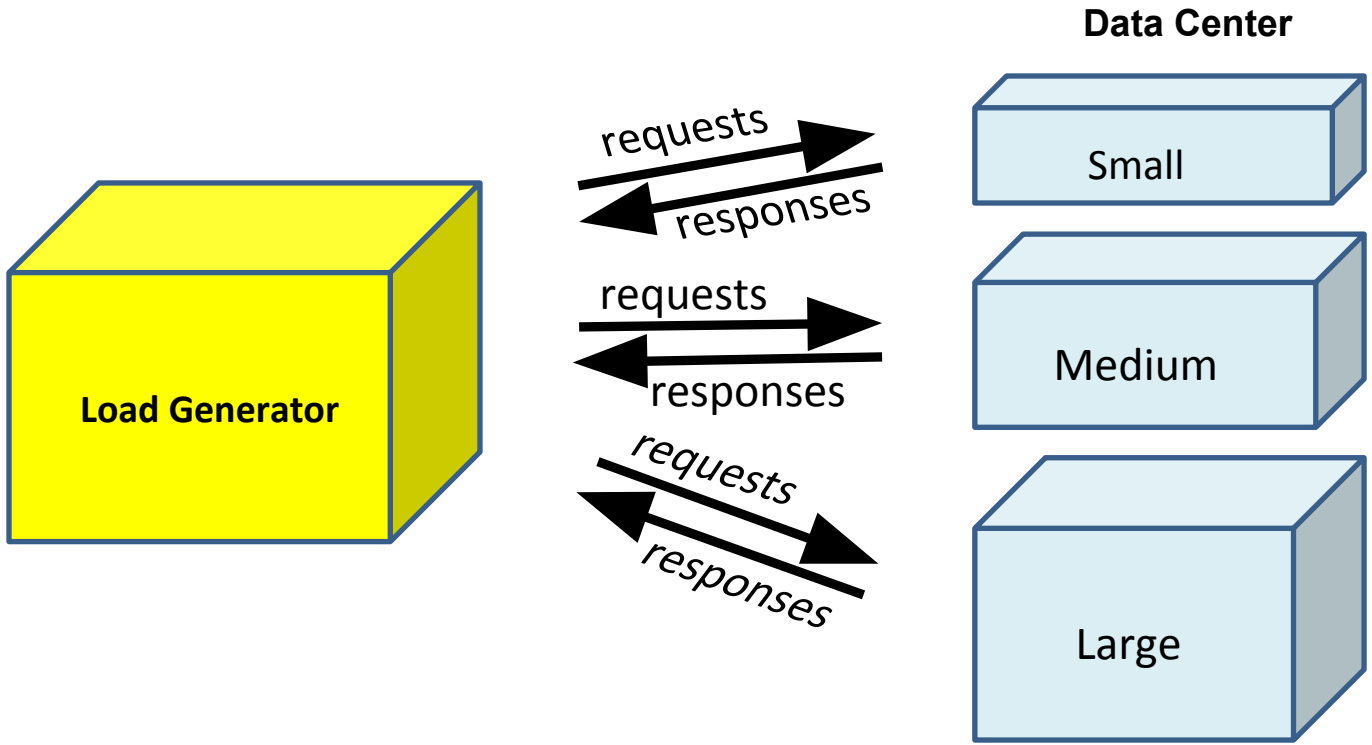


Data Center Instances: Web servers that provide service to the clients (load generator) that request for information.

Resources in Cloud Infrastructure

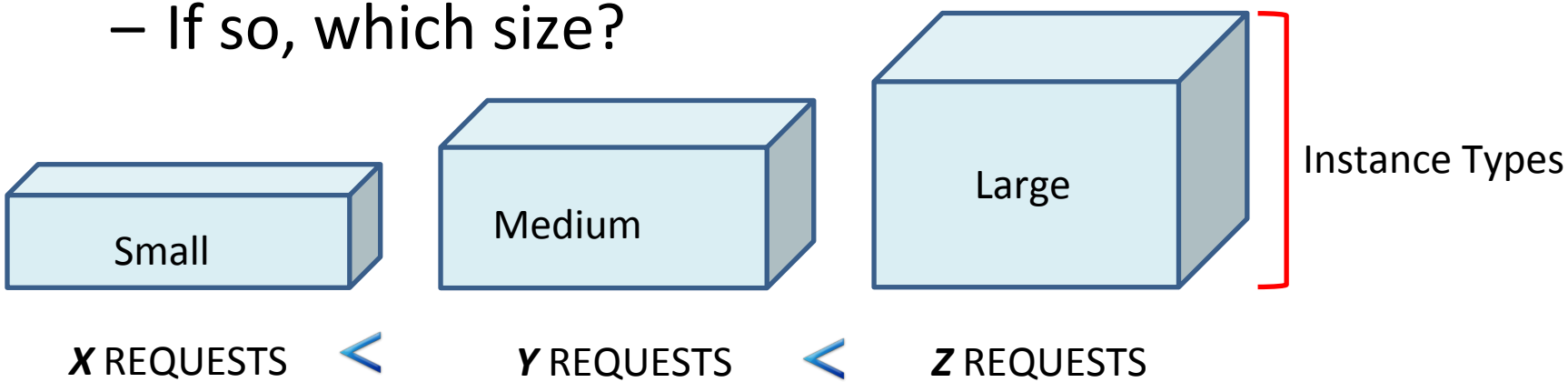


Load Testing Request & Response Flow



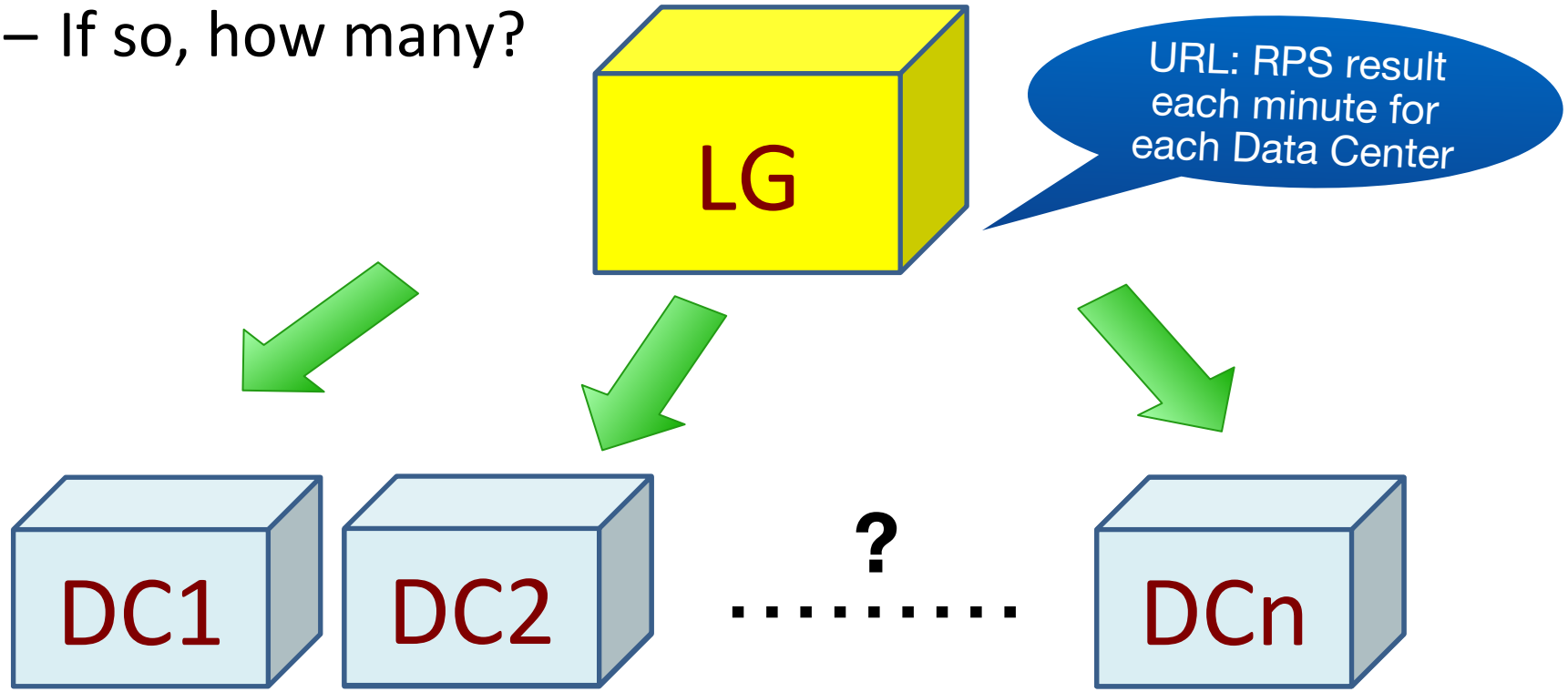
Vertical Scaling

- Would a larger instance suffice?
  - If so, which size?



Horizontal Scaling

- Would more instances suffice?
  - If so, how many?

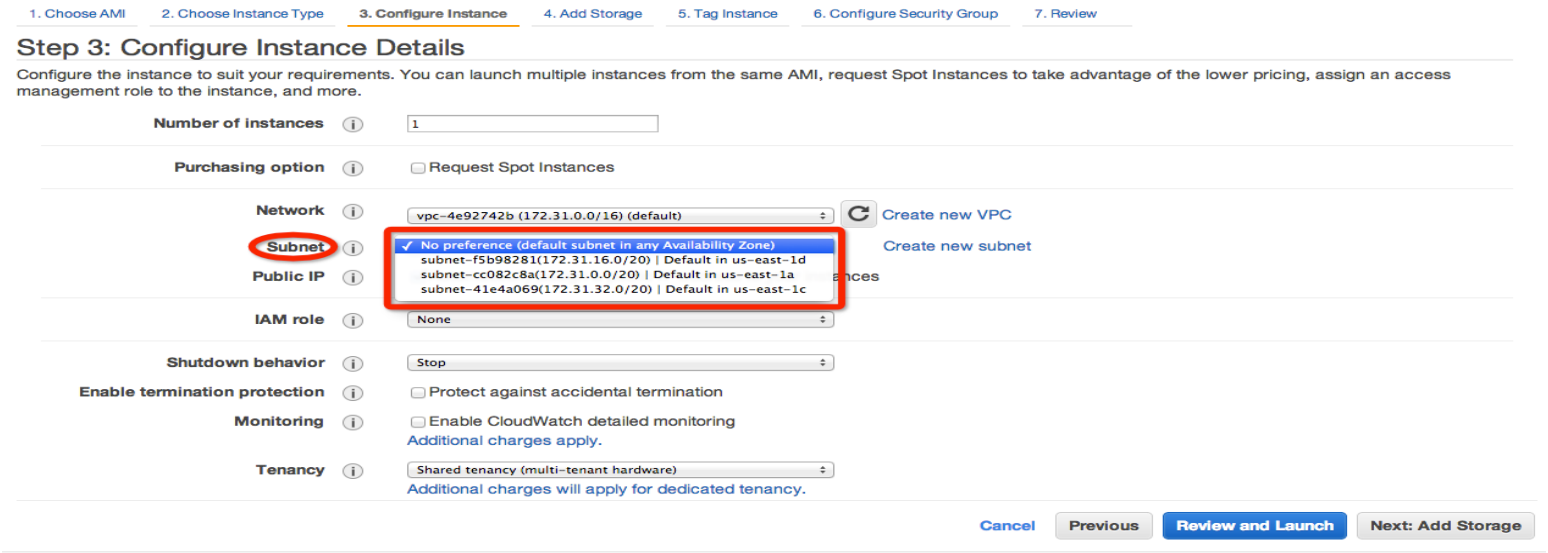


Amazon APIs

- Supported APIs
  - Command Line Interface API Tools (link)
  - AWS SDK for Java (link)
  - AWS SDK for Python (link)

Reminders for Project 2.1

- Make sure the Load Generator and Data Center VMs are in the same subnet (availability zone)



Reminders for Project 2.1 (cont'd)

- Terminate instances vs. stop instances
  - Stop will still charge for VM storage (EBS volumes)
    - Stop is a good idea when you need a break

- NEVER add your credentials in your submitted code. Someone may steal your credentials for their own use (bitcoin mining?), and you'd have to pay for it.

Penalties for 2.1

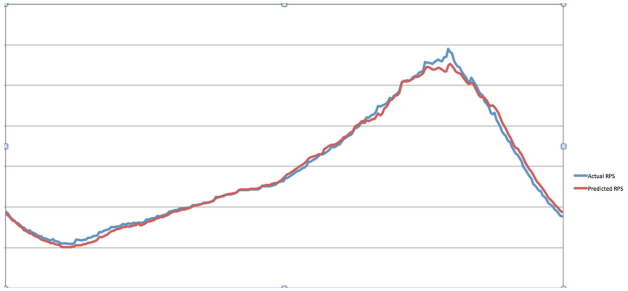
Project Grading Penalties

Besides the penalties mentioned in recitation and/or on Piazza, penalties accrue for the following:

Violation	Penalty of the project grade
Spending more than $8 for this project phase	-10%
Spending more than $16 for this project phase	-100%
Failing to tag all your instances for this project	-10%
Submitting your AWS credentials in your code for grading	-20%

Now it’s time for a demo for P2.1

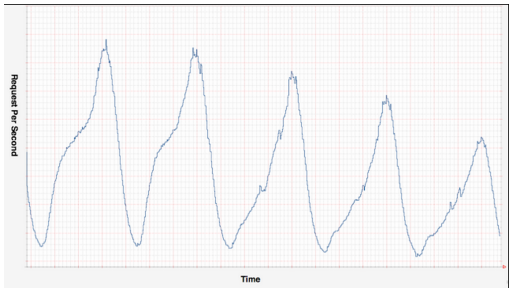
QoS ⇒ Performance ⇒ Throughput



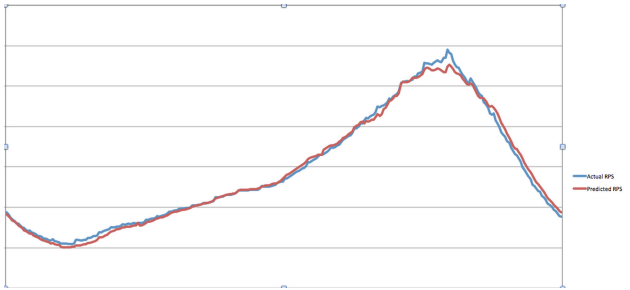
- Solution: Add more resources
  - Increase the number/size of servers
  - Looks easy, but what about cost?
- Scalability/Elasticity
  - Facebook used MySQL servers at the start, then had to solve a lot of problems with scalability because of >1 billion active users
  - Netflix experiences a varying load with time
    - Your web service can handle sudden, unpredicted load.

QoS -> Performance -> Throughput -> Elasticity

Netflix Traffic pattern



Higher Peaks on Sat/Sun,
Lower Peaks on Wed



24 hour (6-8pm peak)