UNIVERSITY OF TARTU

INSTITUTE OF COMPUTER SCIENCE

Basics of Cloud Computing – Lecture 3

Scaling Applications on the Cloud

Satish Srirama

Mobile & Cloud Lab

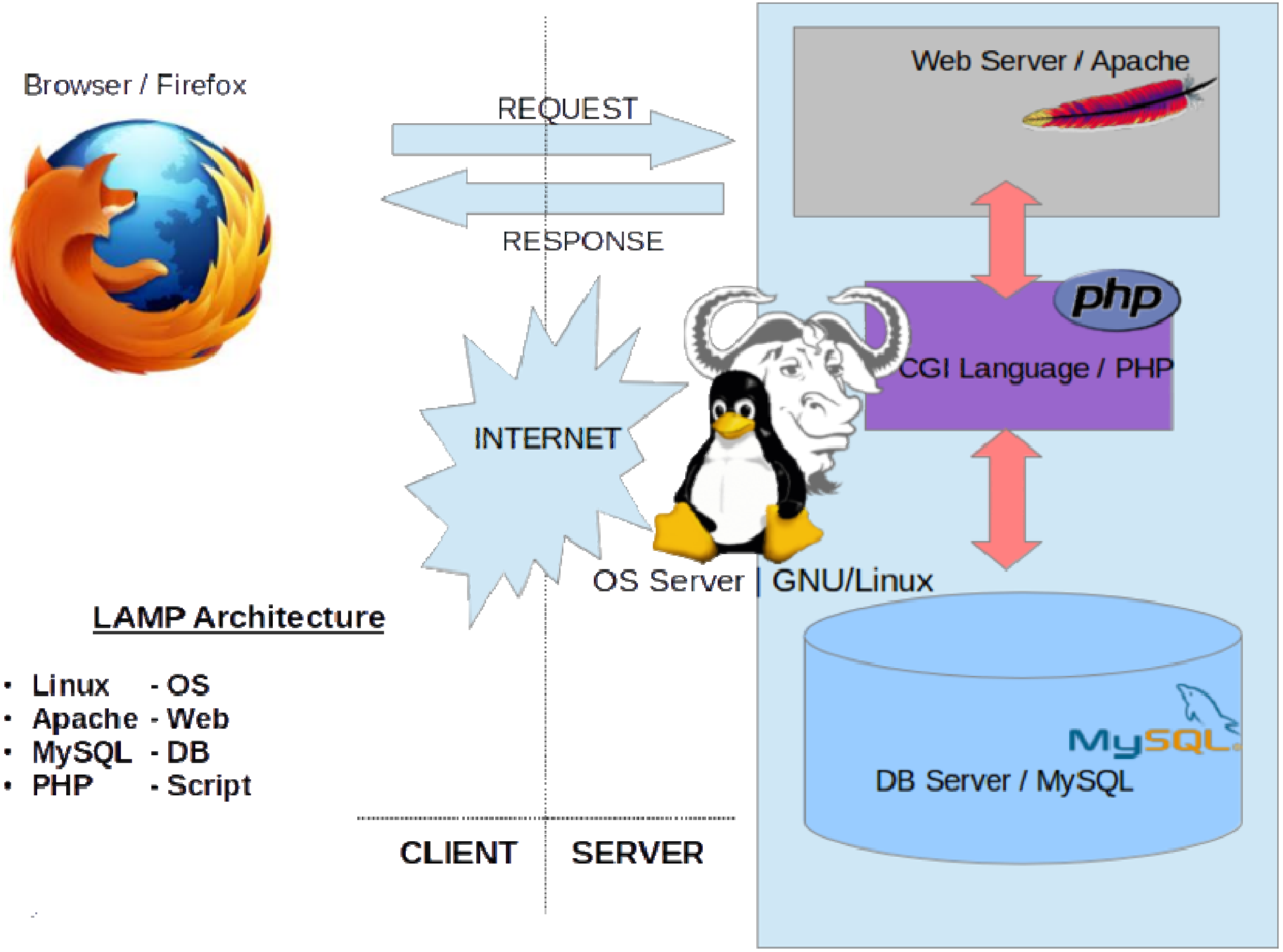
Outline

- Scaling Information Systems
- Scaling Enterprise Applications in the Cloud
- Auto Scaling
  - Amazon Auto Scale and Elastic Load Balancing
  - Open source options for developing Auto Scale solutions

Scaling Information Systems

- Fault tolerance, high availability & scalability are essential prerequisites for any enterprise application deployment
- Scalability
  - Generally nodes in information systems support specific load
  - When load increases beyond certain level, systems should be scaled up
  - Similarly when load decreases they should be scaled down

Typical Web-based Enterprise Application

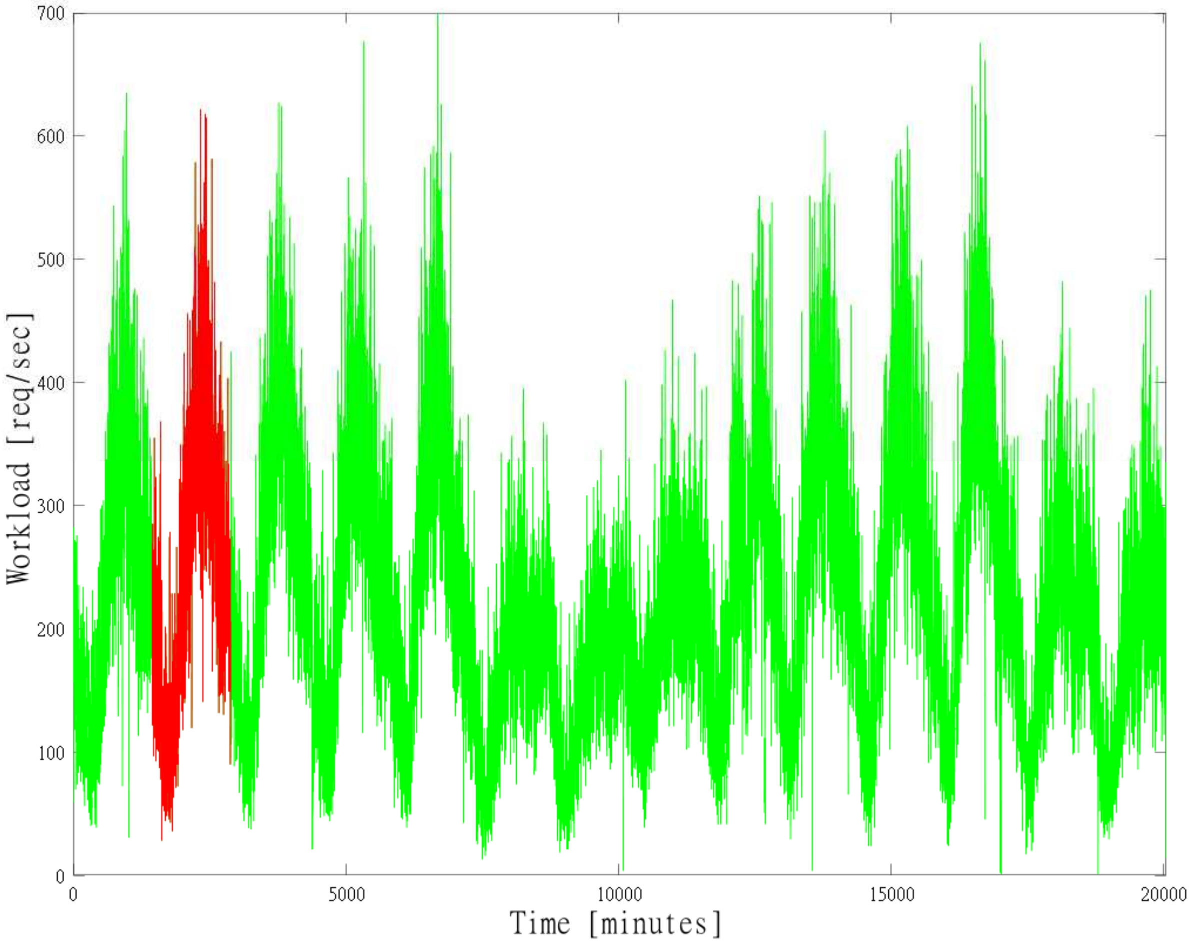


LAMP Architecture

- Linux - OS
- Apache - Web
- MySQL - DB
- PHP - Script

Source: http://en.wikipedia.org/wiki/File:LAMPP_Architecture.png

Typical Load on an Application Server



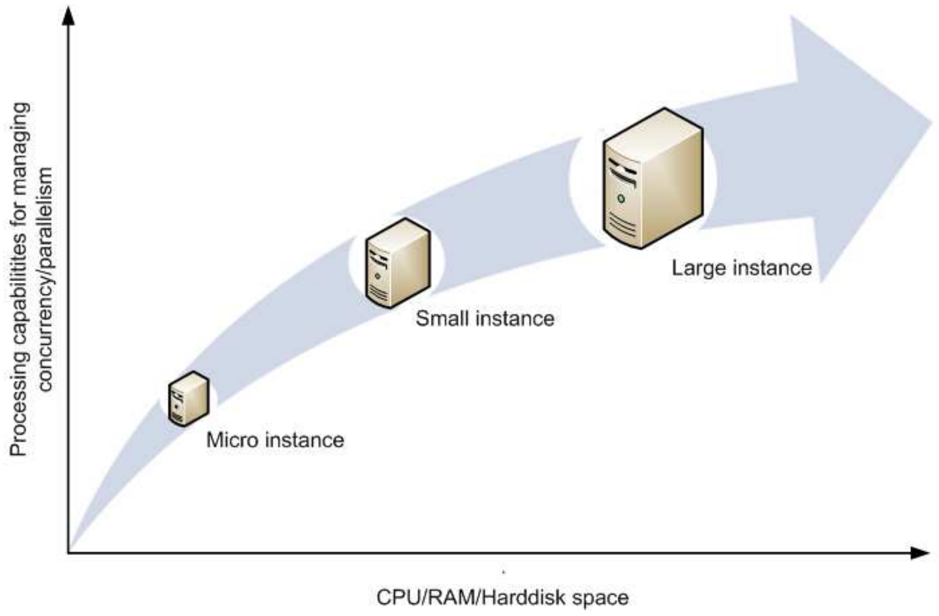
[ClarkNet traces]

Scaling Information Systems - continued

- Two basic models of scaling
  - Vertical scaling
    - Also known as Scale-up
  - Horizontal scaling
    - Aka Scale-out

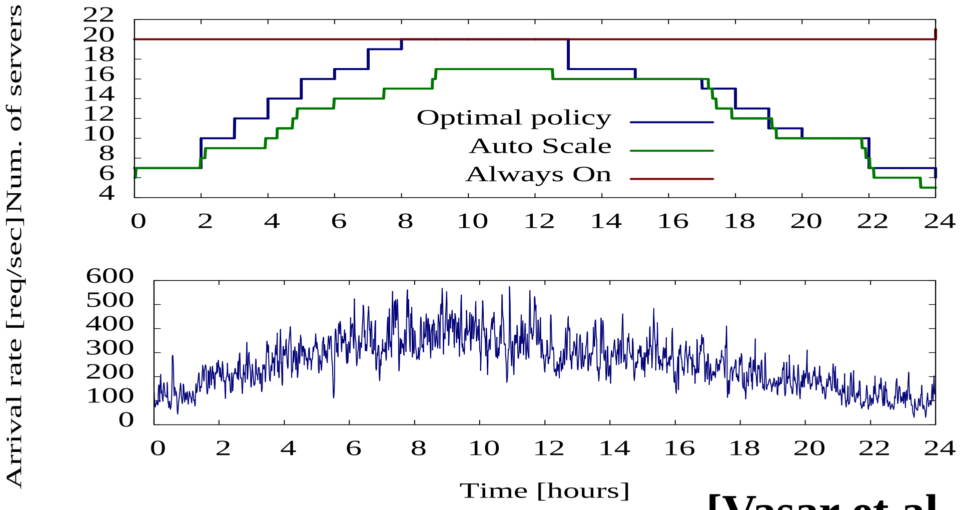
Vertical Scaling

- Achieving better performance by replacing an existing node with a much powerful machine
- Risk of losing currently running jobs
  - Can frustrate customers as the service is temporarily down



Horizontal Scaling

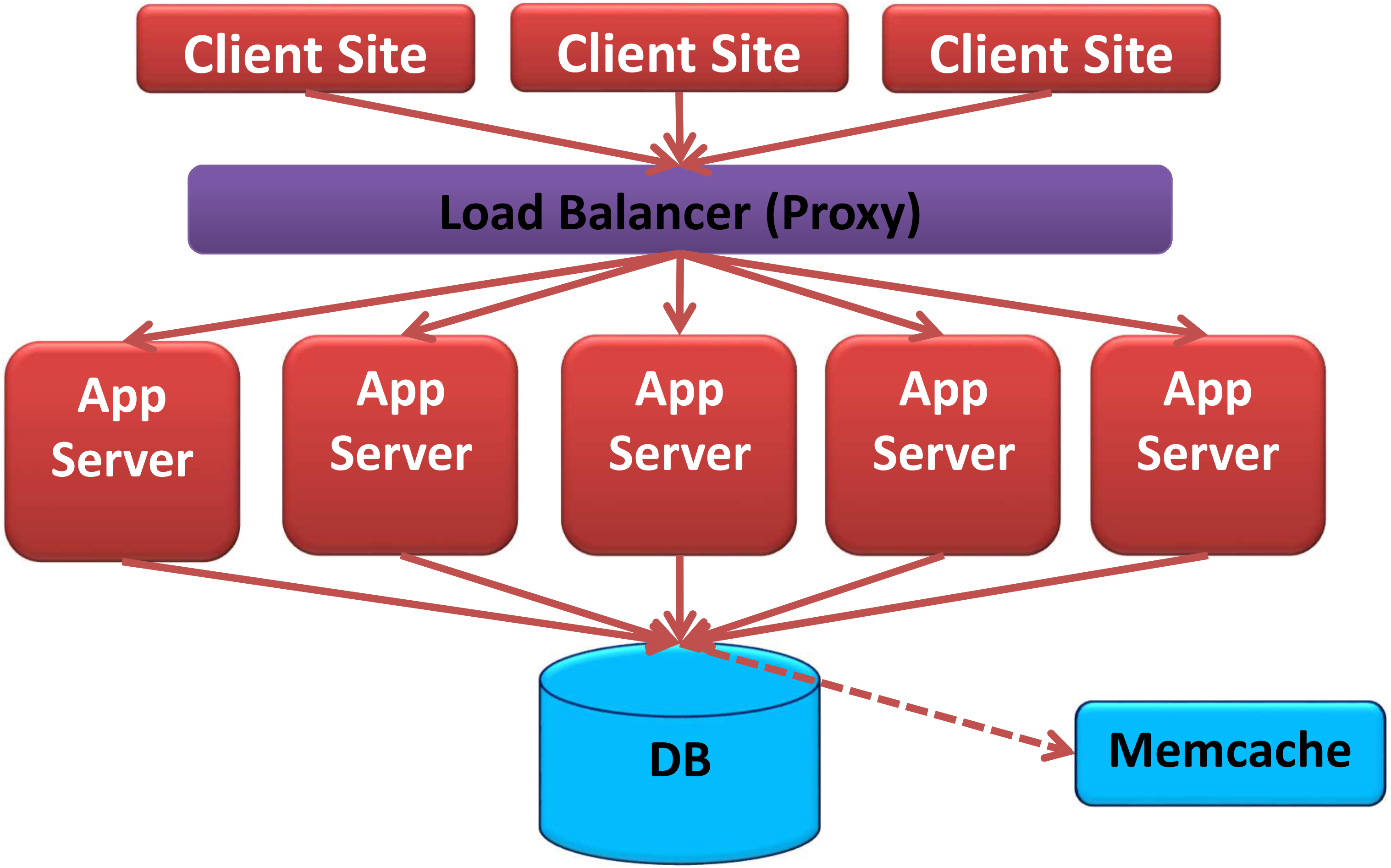
- Achieving better performance by adding more nodes to the system
- New servers are introduced to the system to run along with the existing servers



[Vasar et al, Nordicloud 2012]

Server allocation policies for different loads

Scaling Enterprise Applications in the Cloud



Load Balancer

- Load balancing has been a key mechanism in making efficient web server farms
- Load balancer automatically distributes incoming application traffic across multiple servers
- Hides the complexity for content providers
- 1+1 = 2
  - Allows server farms work as a single virtual powerful machine
- 1+1 > 2
  - Beyond load distribution, improves response time

Introduction- Types of Load Balancers

- Network-Based load balancing
  - Provided by IP routers and DNS (domain name servers) that service a pool of host machines
  - e.g. when a client resolves a hostname, the DNS can assign a different IP address to each request dynamically based on current load conditions
- Network-Layer based load balancing
  - Balances the traffic based on the source IP address and/or port of the incoming IP packet
  - Does not take into account the contents of the packet, so is not very flexible
- Transport-Layer based load balancing
  - The load balancer may choose to route the entire connection to a particular server
  - Useful if the connections are short-lived and are established frequently
- Application-Layer/Middleware based load balancing
  - Load balancing is performed in the application-layer, often on a per-session or per-request basis

Introduction- Classes of Load Balancers

- Non-adaptive load balancer
  - A load balancer can use non-adaptive policies, such as simple round-robin algorithm, hash-based or randomization algorithm
- Adaptive load balancer
  - A load balancer can use adaptive policies that utilize run-time information, such as amount of CPU load on the node
- Load Balancers and Load Distributors are not the same thing
  - Strictly speaking non-adaptive load balancers are load distributors

Load Balancing Algorithms

- Random
  - Randomly distributes load across the available servers
  - Picks one via random number generation and sending the current connection to it
- Round Robin
  - Round Robin passes each new connection request to the next server in line
  - Eventually distributes connections evenly across the array of machines being load balanced
- Least connection (Join-Shortest-Queue)
  - The system passes a new connection to the server that has the least number of current connections
- etc.

https://devcentral.f5.com/articles/intro-to-load-balancing-for-developers-ndash-the-algorithms#.UwSBpc4a5SM

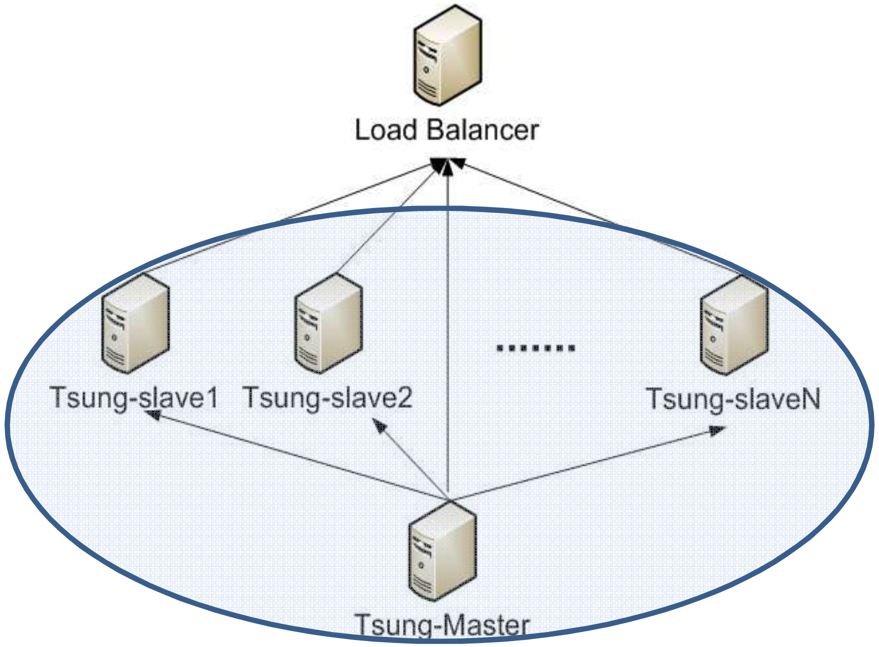
Examples of Load Balancers

- Nginx - http://nginx.org/
- HAProxy - http://haproxy.1wt.eu/
- Pen - http://siag.nu/pen/

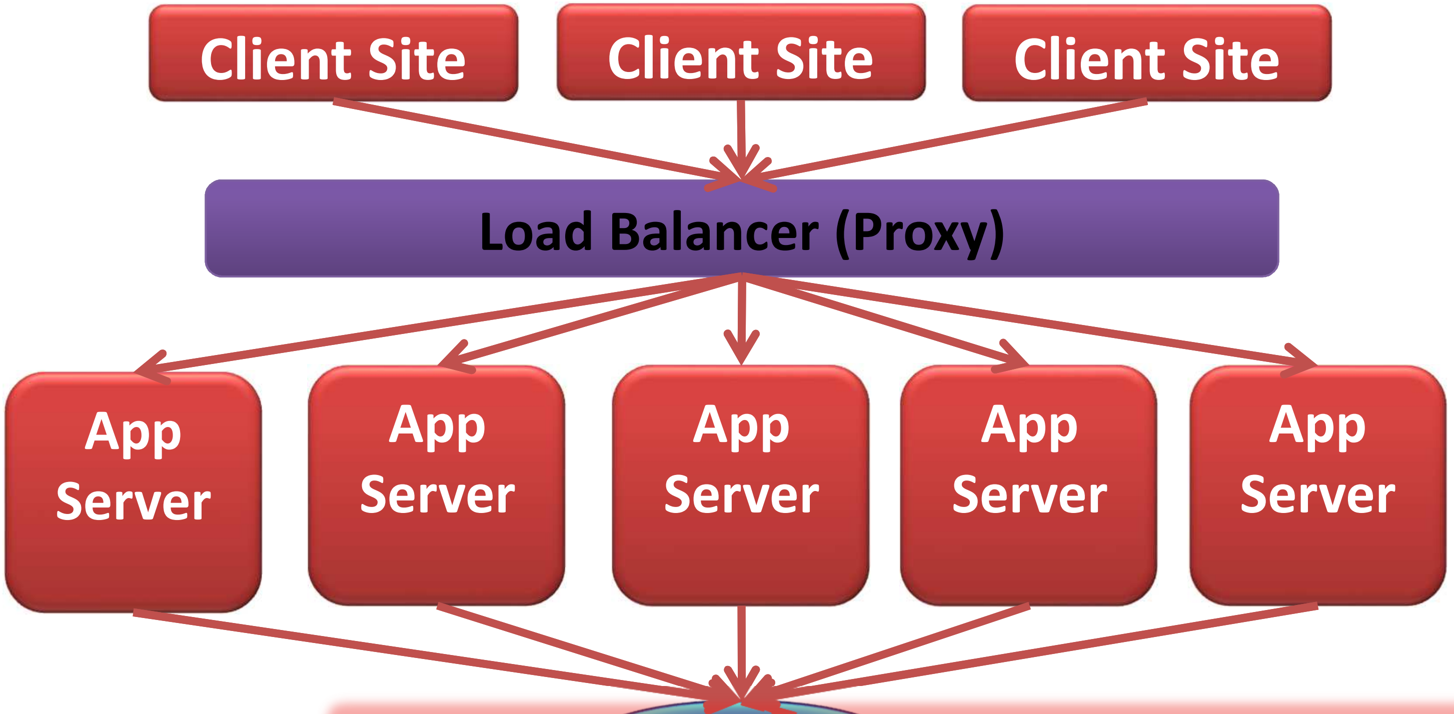
etc.

Testing the System by Simulating Load

- Benchmarking tools
  - Tsung, JMeter, etc
- Simulating concurrency is also possible
- Multiple protocols
  - HTTP, XMPP, etc.
  - SSL support

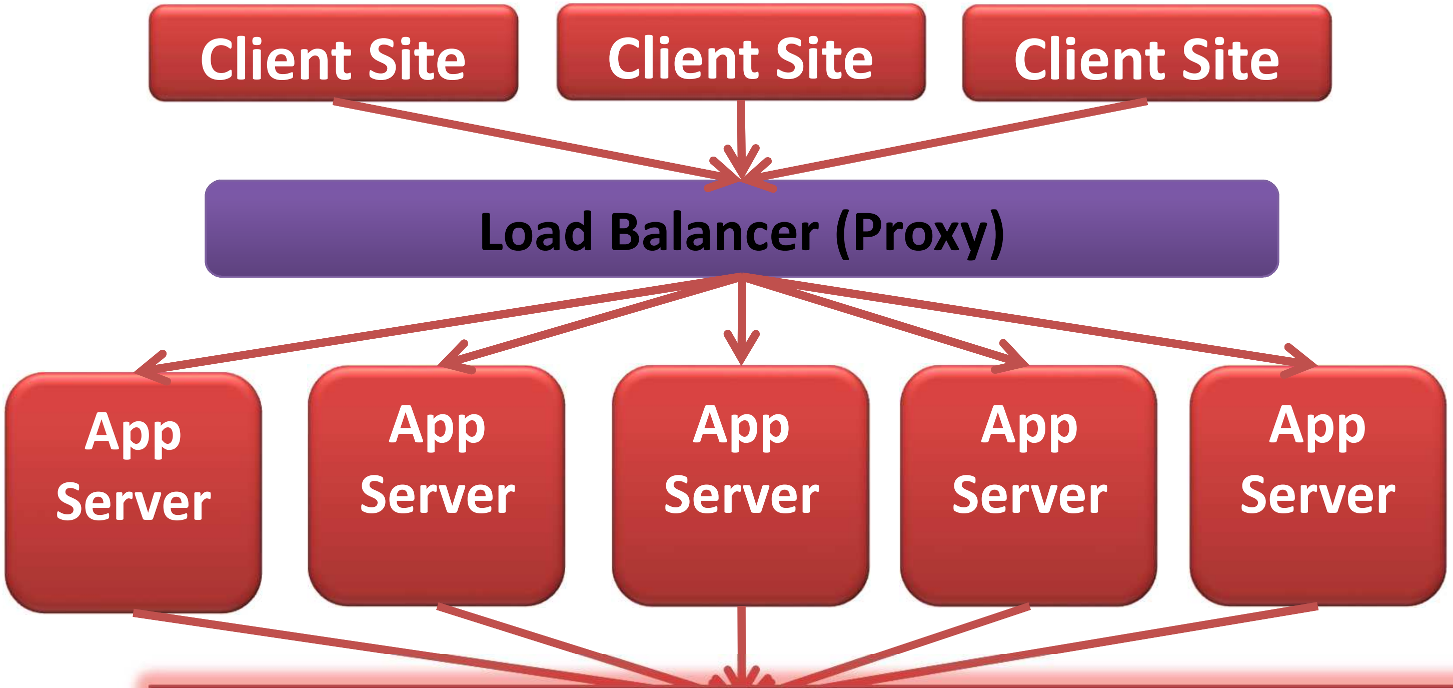


Scaling in the Cloud - bottleneck



Database becomes the Scalability Bottleneck
Cannot leverage elasticity

Scaling in the Cloud - bottleneck



Scalable and Elastic,
but limited consistency and
operational flexibility

NoSQL

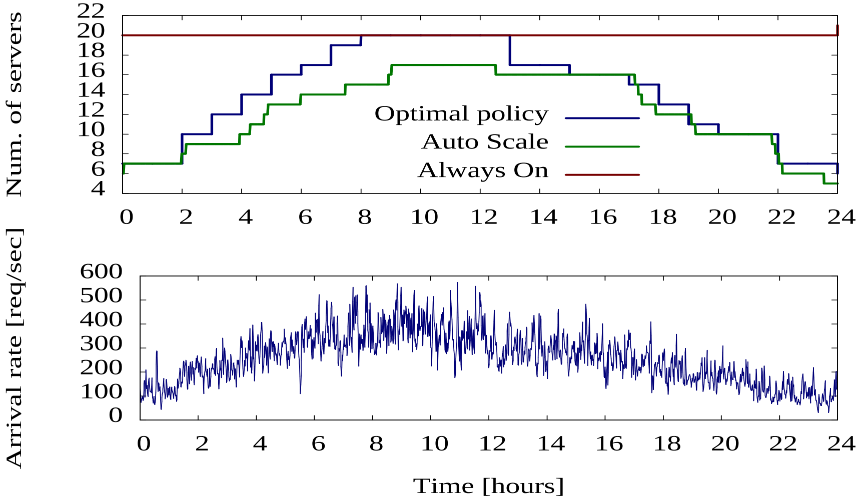
{name: "mongo", type:"DB"}

Horizontal Scaling – Further examples

- MapReduce & Hadoop
  - We will look as part of Lectures 4 and 5

AutoScale

- AutoScale allows systems to dynamically react to a set of defined metrics and to scale resources accordingly
- Providing:
  - High availability
  - Cost saving
  - Energy saving



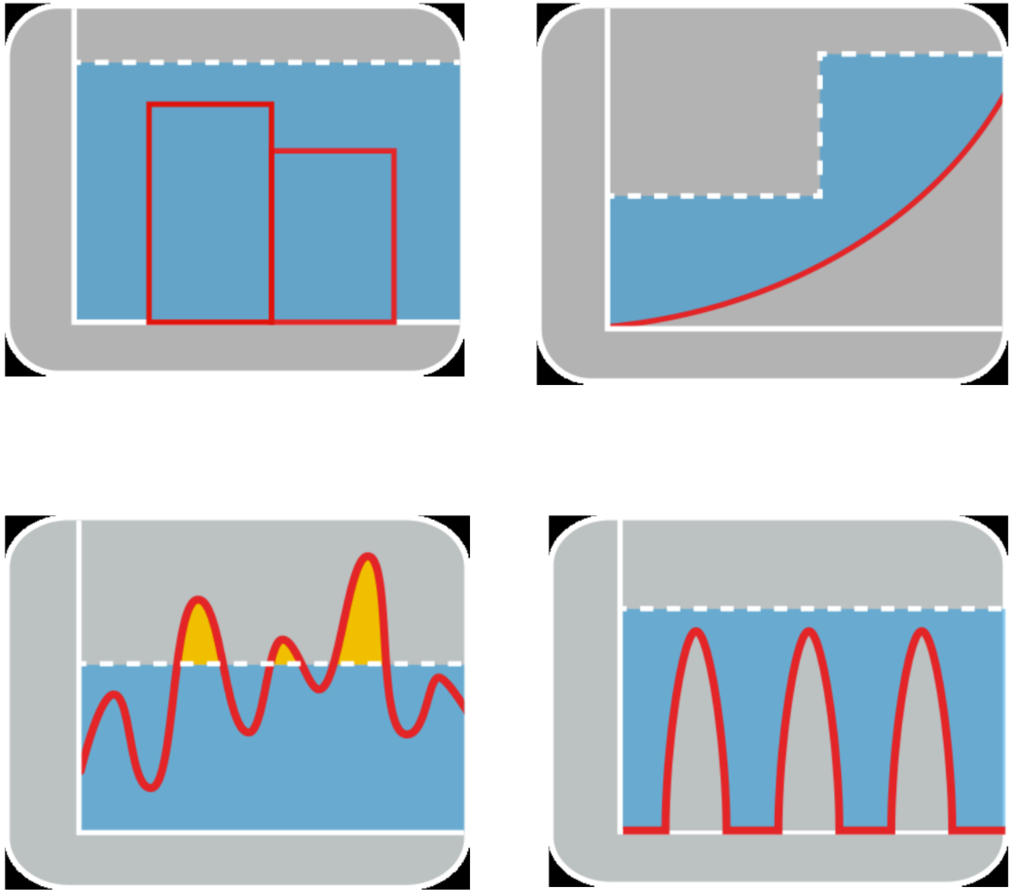
Server allocation policies for different loads

Typical Usecases

- Applications that see elasticity in their demand
- Launching a new website with unknown visitor numbers
- Viral marketing campaigns
- A scientific application might also have to scale out
  - Using 50 machines for 1 hour rather than 1 machine for 50 hours

Types of Traffic Patterns

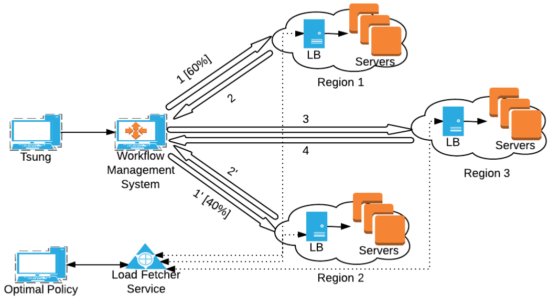
- ON & OFF
  - Analytics!
  - Banks/Tax Agencies!
  - Test environments
- FAST GROWTH
  - Events!
  - Business Growth!
- VARIABLE
  - News & Media!
  - Event Registrations!
  - Rapid fire sales
- CONSISTENT
  - HR Application!
  - Accounting/Finance



http://www.slideshare.net/lynxmanuk/autoscaling-best-practices

Auto-Scaling enterprise applications on the cloud

- Enterprise applications are mostly based on SOA and componentized models
- Auto-Scaling
  - Scaling policy **->** When to Scale
  - Resource provisioning policy **->** How to scale



- Threshold-based scaling policies are very popular due to their simplicity
  - Observe metrics such as CPU usage, disk I/O, network traffic etc.
  - E.g. Amazon AutoScale, RightScale etc.
  - However, configuring them optimally is not easy

SOA - Service-oriented architecture

AutoScaling on the cloud

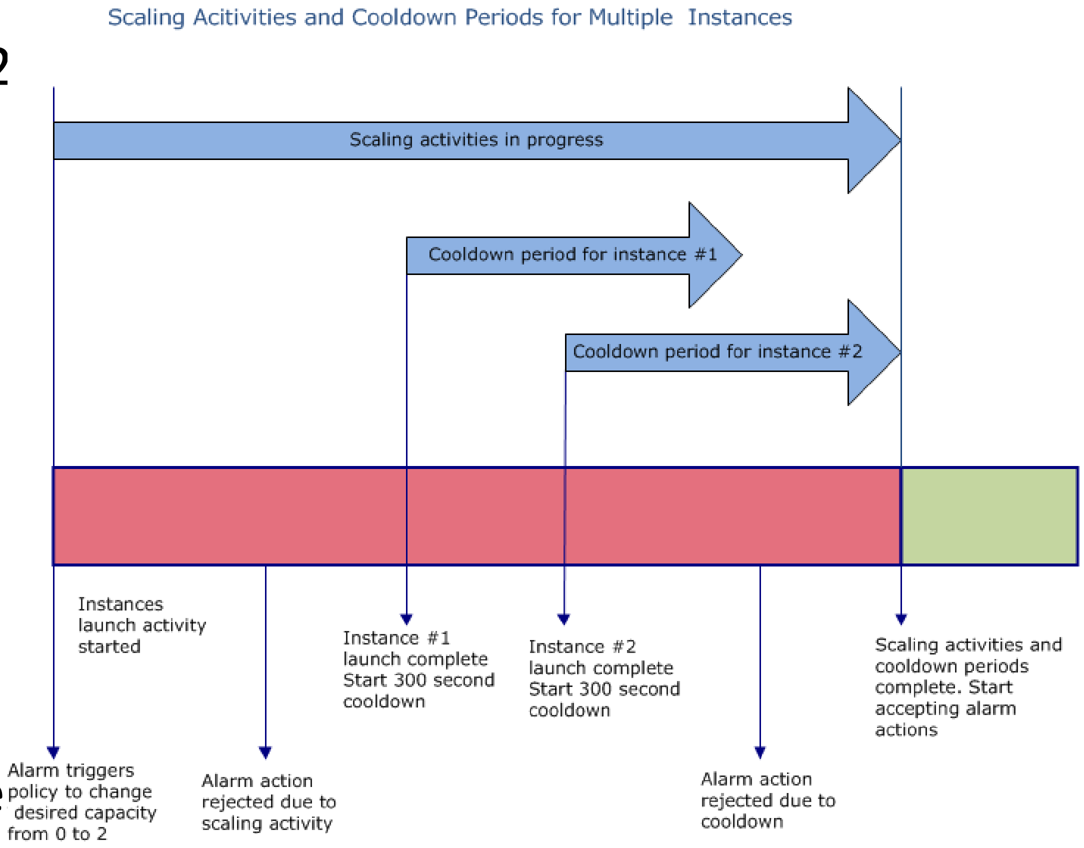
- Amazon Autoscale & Elastic Load Balance
- Vendor neutral autoscaling on cloud
  - Static Load Balancer + Resources estimation on the fly (e.g optimal heuristics) [Vasar et al, Nordicloud 2012]
  - Static Load Balancer + Optimal resource provisioning [Srirama and Ostovar, CloudCom 2014]

Amazon Auto Scaling

- Amazon Auto Scaling allows you to scale your compute resources dynamically and predictably (*scaling plan*):
  - Dynamically based on conditions specified by you
    - E.g. increased CPU utilization of your Amazon EC2 instance
      - CPU utilization of all servers on average is >75% in last 5 min, add 2 servers and average < 35% remove 1 server
  - Predictably according to a schedule defined by you
    - E.g. every Friday at 13:00:00.
- EC2 instances are categorized into *Auto Scaling groups* for the purposes of instance scaling and management
- You create Auto Scaling groups by defining the minimum & maximum no of instances
- A launch configuration template is used by the Auto Scaling group to launch Amazon EC2 instances

Amazon Auto Scaling - continued

- Auto Scaling
  - Monitor the load on EC2 instances using CloudWatch
  - Define Conditions and raise alarms
    - E.g. Average CPU usage of the Amazon EC2 instances, or incoming network traffic from many different Amazon EC2 instances
  - Spawn new instances when there is too much load or remove instance when not enough load


Scaling Acitivities and Cooldown Periods for Multiple Instances
Scaling activities in progress
Cooldown period for instance #1
Cooldown period for instance #2
Instances launch activity started
Instance #1 launch complete Start 300 second cooldown
Instance #2 launch complete Start 300 second cooldown
Scaling activities and cooldown periods complete. Start accepting alarm actions
Alarm triggers policy to change desired capacity from 0 to 2
Alarm action rejected due to scaling activity
Alarm action rejected due to cooldown

Amazon CloudWatch

- Monitor AWS resources automatically
  - Monitoring for Amazon EC2 instances: seven pre-selected metrics at five-minute frequency
  - Amazon EBS volumes: eight pre-selected metrics at five-minute frequency
  - Elastic Load Balancers: four pre-selected metrics at one-minute frequency
  - Amazon RDS DB instances: thirteen pre-selected metrics at one-minute frequency
  - Amazon SQS queues: seven pre-selected metrics at five-minute frequency
  - Amazon SNS topics: four pre-selected metrics at five-minute frequency
- Custom Metrics generation and monitoring
- Set alarms on any of the metrics to receive notifications or take other automated actions
- Use Auto Scaling to add or remove EC2 instances dynamically based on CloudWatch metrics

Elastic Load Balance

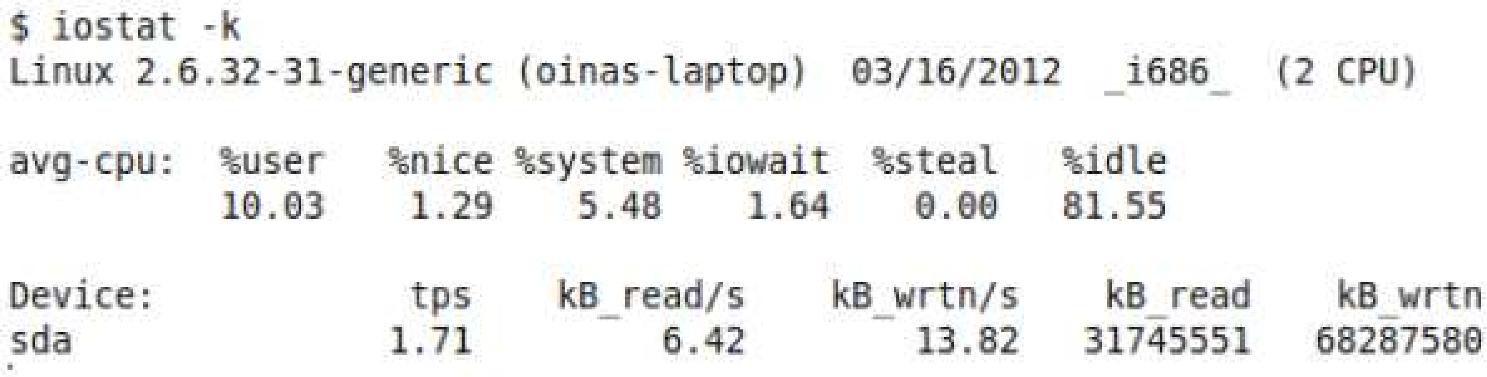
- Elastic Load Balance
  - Automatically distributes incoming application traffic across multiple EC2 instances
  - Detects EC2 instance health and diverts traffic from bad ones
  - Support different protocols
    - HTTP, HTTPS, TCP, SSL, or Custom
- Amazon Auto Scaling & Elastic Load Balance can work together

Components of an Auto Scaling system

- Load balancer
- Solutions to measure the performance of current setup
- Scaling policy defining when to scale
- Resource provisioning policy
- Dynamic deployment template

Cloud-based Performance – Open solutions

- Multiple approaches are possible
- Shell
  - – Linux utilities
    - Default
      - – free –m



```
$ iostat -k
Linux 2.6.32-31-generic (oinas-laptop)  03/16/2012  _i686_  (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          10.03    1.29    5.48    1.64    0.00   81.55

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               1.71         6.42        13.82   31745551   68287580
```

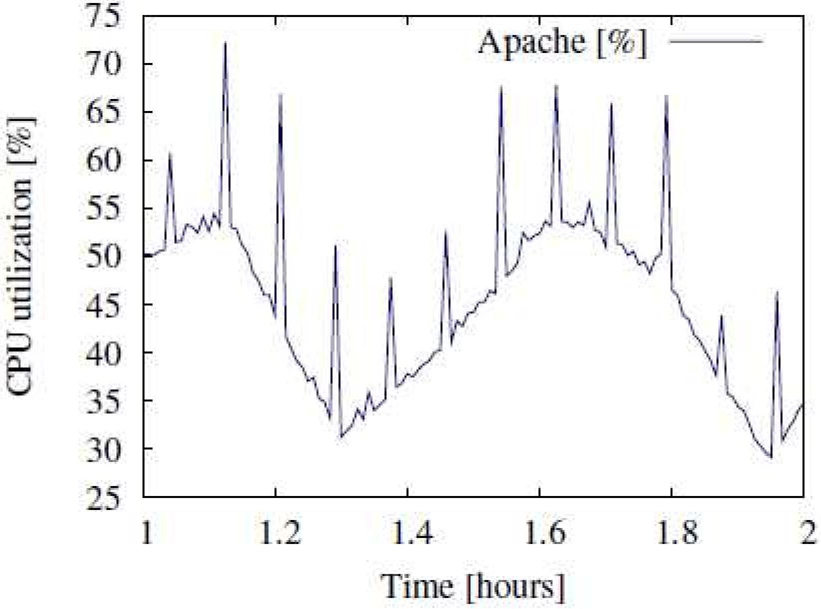
Cloud-based Performance - continued

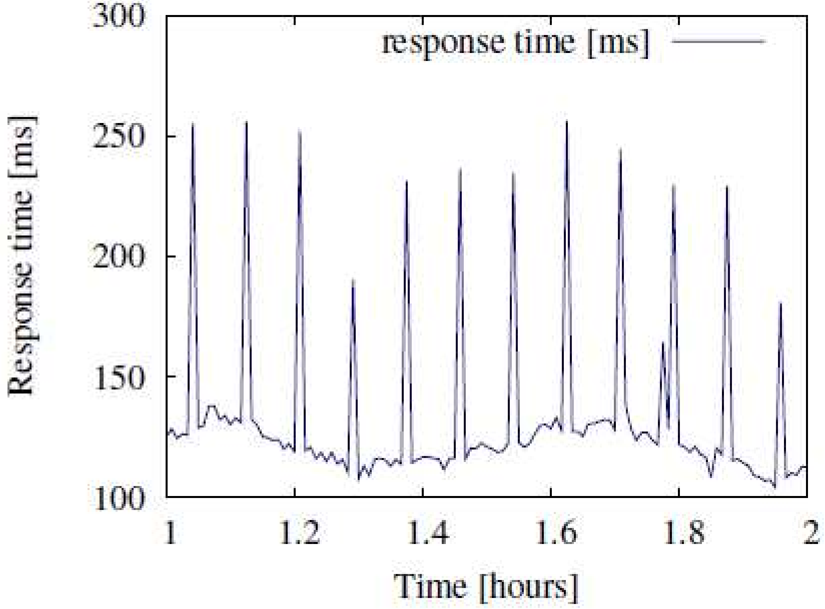
- Tools (distributed)
  - Collectd
    - RRDtool
      - Generating visual performance graphs
    - Multicast communication
    - Does not impact system performance
  - Cacti
    - RRD
    - GUI
    - Performance decreases by 20%

RRDtool - round-robin database tool

Cloud-based Performance - continued

- Cacti
  - Spikes denote gathering performance metrics





Scaling Policy

- Time based
  - Already seen with Amazon Auto Scale
    - E.g. every Friday at 13:00:00 or Feb 15th 10 more servers for Estonian tax board
  - Good for On & Off! and Consistent traffic patterns
- Reactive
  - Threshold-based scaling policies
    - E.g. CPU utilization of all servers on average is >75% in last 5 min
  - Good for Fast Growth traffic pattern
- Predictive
  - AutoScaling based on predictive traffic
    - E.g. Predicting next min load by taking mean of last 5 min load
  - Good for Variable traffic pattern

Components of an Auto Scaling system

- Load balancer
- Solutions to measure the performance of current setup
- Scaling policy defining when to scale
- Resource provisioning policy
- Dynamic deployment template

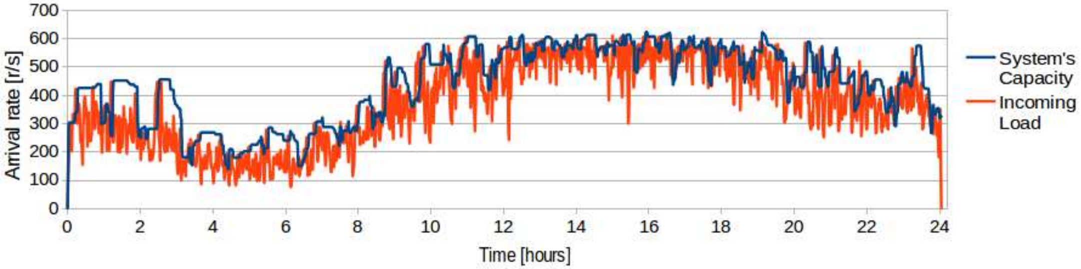
Resource provisioning policy

- Simple resource provisioning policy
  - Resources estimation based on heuristic
  - E.g. suppose a node supports ~10 rps and current setup has 4 servers and load is 38 rps
    - Assume load increased or predicted to increase to 55 rps
    - So add 2 more servers
- May not be optimal or perfect solution, but sufficient for the immediate goals

Optimal Resource Provisioning for Auto-Scaling Enterprise Applications

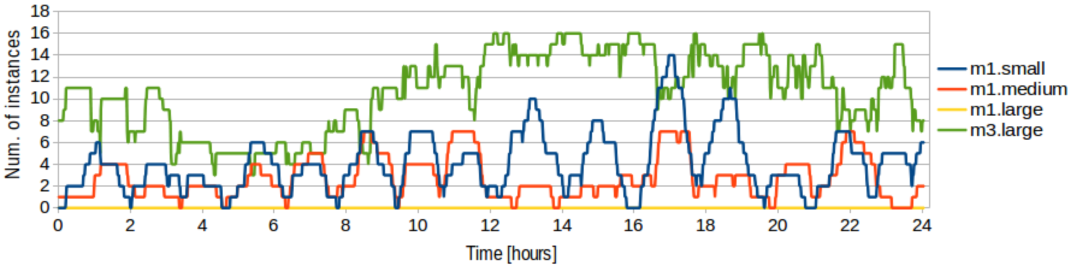
- Cloud providers offer various instance types with different processing power and price
  - Can it be exploited in deciding the resource provisioning policy?
  - Makes the policy to be aware of current deployment configuration
- Another challenge: Cloud providers charge the resource usage for fixed time periods
  - E.g. Hourly prices of Amazon cloud
- Developed an LP based optimization model which considers both the issues [Srirama and Ostovar, CloudCom 2014]

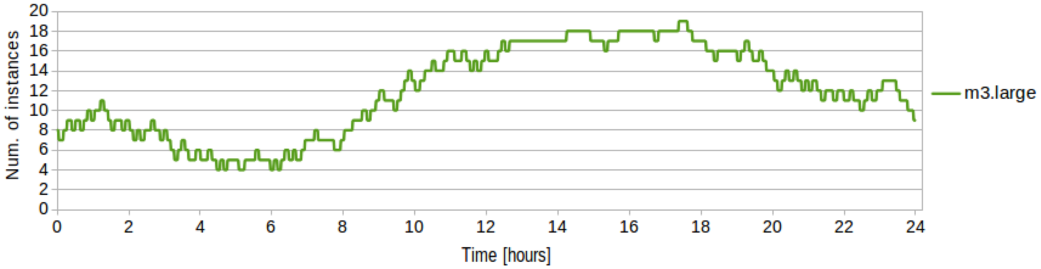
Scaling enterprise application with the optimization model



Incoming load and scaling curves of Optimization model

Instance type usage curves of Optimization model





Scaling with Amazon AutoScale

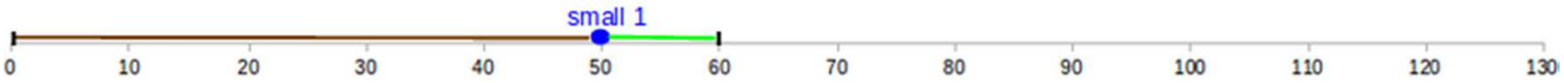
[Srirama and Ostovar, CloudCom 2014]

Optimization Model

Intuition behind instance lifetime consideration

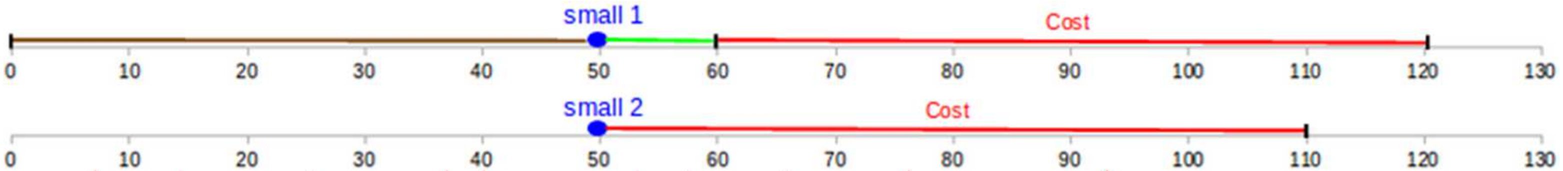
- Consider 2 instance types
  - Small instance(PW = 6r/s, Price = $0.25/h),
  - Medium instance(PW = 12r/s, Price = $0.4/h)

Load is 6r/s



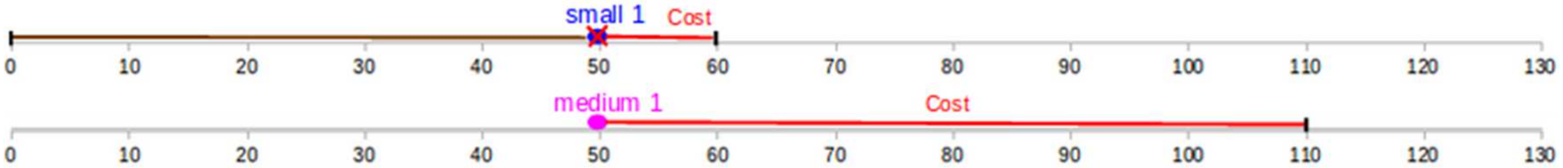
Load increases to 12r/s => ?

Solution 1



Cost = (cost of two small instances) - (10-min profit of a small instance) = 0.5 - 0.04 = $0.46

Solution 2



Cost = (cost of a medium instance) + (10-min cost of a small instance) = 0.4 + 0.04 = $0.44

- Saved cost with solution 2 : 0.46 – 0.44 = 0.02$
- So can we find this automatically?

Optimization Model

Some key definitions

- Region:
  - A task with its own independent characteristics
  - Each region can have its own capacity of instances
- Instance Type:
  - Each region can include multiple instance types
  - It is associated with processing power, price per period, capacity constraint, and configuration time
- Time bags:
  - Time interval where an instance is at a particular time
- Killing Cost:
  - Money lost when an instance is killed before it fills its paid period
- Retaining Cost:
  - The cost of the lived duration of the paid period

[Srirama and Ostovar, CloudCom 2014]

Optimization Model

- ***Cost Function:***

Cost of new instances

Configuration cost of new instances

$$Min\ (\boxed{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{r_i,t_j} * C_{r_i,t_j}} + \boxed{N_{r_i,t_j} * (CT_{r_i,t_j} * CTB_{r_i,t_j})} +$$

$$\boxed{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{q} S_{r_i,t_j,tb_k} * KC_{r_i,t_j,tb_k}} +$$

Cost of killed instances

$$\boxed{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{q} (X_{r_i,t_j,tb_k} - S_{r_i,t_j,tb_k}) * RC_{r_i,t_j,tb_k}})$$

Cost of retained instances

- ***Constraints:***

$$\sum_{j=1}^{m}(N_{r,t_j} + (\sum_{k=1}^{q} X_{r,t_j,tb_k} - S_{r,t_j,tb_k})) * P_{r,t_j} \geq W_r \longrightarrow$$

Workload constraint

$$\sum_{j=1}^{m}(N_{r,t_j} + (\sum_{k=1}^{q} X_{r,t_j,tb_k} - S_{r,t_j,tb_k})) \leq CC_r \longrightarrow$$

Cloud capacity constraint

$$N_{t_r} + (\sum_{k=1}^{q} X_{t_r,tb_k} - S_{t_r,tb_k}) \leq CCT_{t_r} \longrightarrow$$

Instance type capacity constraint

$$S_{tb_{r,t}} \leq X_{tb_{r,t}} \longrightarrow$$

Shutdown constraint

$$N_{r,t} \geq 0$$
$$S_{r,t} \geq 0$$

[Srirama and Ostovar, CloudCom 2014]

Application of the model

- Identify the scalable components in an enterprise application
- Scalable components are load tested on the planned cloud
  - To extract application specific parameters of the model
- Incoming load of each region is extracted and fed to the optimization model
  - Produces the ideal deployment configuration of the application

Evaluation of the optimization model

- The optimization model performs at least as good as Amazon AutoScale
  - Sometimes outperforms in efficiency and mostly in response times
  - Further optimizations with scaling policy can also save cost
- The model is generic and can be applied to any cloud
  - Which follows similar utility computing model
- It is also applicable to the systems which need to span across multiple clouds
- The latencies are also reasonable
  - The model could always find the optimal solution within decent amount of time

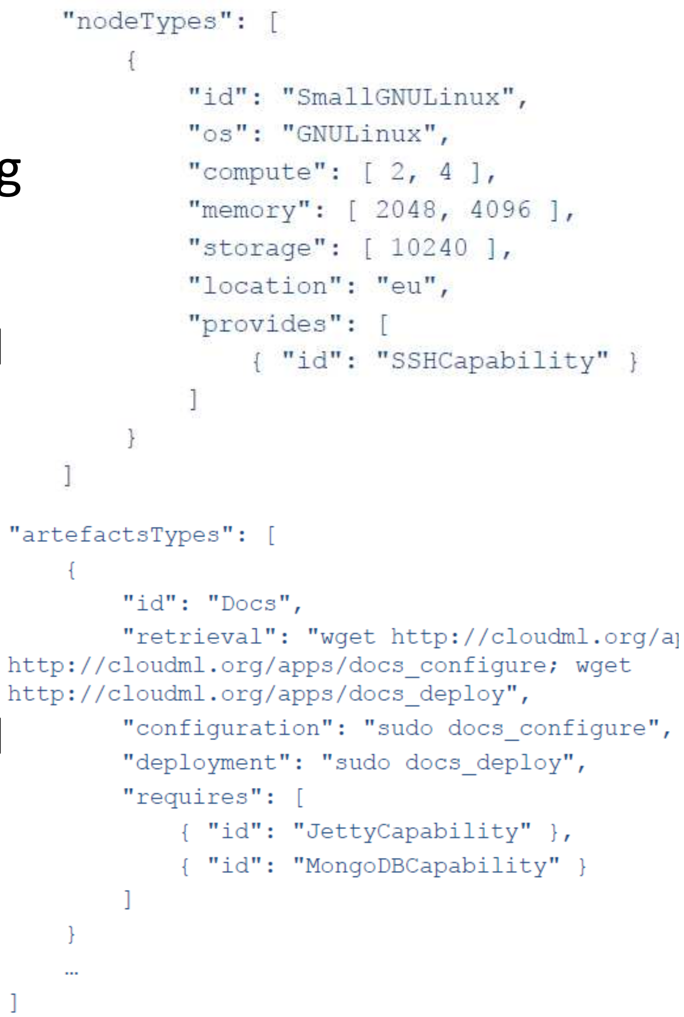
[Srirama and Ostovar, CloudCom 2014]

Components of an Auto Scaling system

- Load balancer
- Solutions to measure the performance of current setup
- Scaling policy defining when to scale
- Resource provisioning policy
- Dynamic deployment template

CloudML

- Developed in REMICS EU FP7 project
- Developed to tame cloud heterogeneity
- Domain-specific language (DSL) for modelling the provisioning and deployment at design-time
  - Nodes, artefacts and bindings can be defined
- Different means to manipulate CloudML models
  - Programmatically via Java API
  - Declaratively, via serialized model (JSON)
- Models@Runtime
  - Dynamic deployment of CloudML based mod



```
"nodeTypes": [
    {
        "id": "SmallGNULinux",
        "os": "GNULinux",
        "compute": [ 2, 4 ],
        "memory": [ 2048, 4096 ],
        "storage": [ 10240 ],
        "location": "eu",
        "provides": [
            { "id": "SSHCapability" }
        ]
    }
]
"artefactsTypes": [
    {
        "id": "Docs",
        "retrieval": "wget http://cloudml.org/a
http://cloudml.org/apps/docs_configure; wget
http://cloudml.org/apps/docs_deploy",
        "configuration": "sudo docs_configure",
        "deployment": "sudo docs_deploy",
        "requires": [
            { "id": "JettyCapability" },
            { "id": "MongoDBCapability" }
        ]
    }
    …
]
```

[Ferry et al, Cloud 2013]

TOSCA

- Topology & Orchestration Specification of Cloud Application
- By OASIS
  - Sponsored by IBM, CA, Rackspace, RedHat, Huawei and Others
- Goal: cross cloud, cross tools orchestration of applications on the Cloud
- Node Type
- Relationship Type
- TOSCA Template

https://cloudify.co/2015/07/21/what-is-TOSCA-cloud-application-orchestration-tutorial-cloudify.html

Final Thoughts on AutoScaling

- AutoScaling can be dangerous
  - E.g. Distributed Denial of Service (DDoS) attack
  - Have min-max allocations
- Choose the right metrics
  - Stay with basic metrics
    - CPU, mem, I/O disk/net etc.
  - Review autoscaling strategy with metrics
- Choose your strategy
  - Scale up early and Scale down slowly
  - Don’t apply the same strategy to all apps

This week in lab

- You work with load balancing

Next Lecture

- Introduction to MapReduce

References

- Amazon Web (Cloud) Services – documentation http://aws.amazon.com/documentation/
- Elastic Load balancing http://aws.amazon.com/elasticloadbalancing/
- Load balancing - algorithms https://devcentral.f5.com/articles/intro-to-load-balancing-for-developers-ndash-the-algorithms#.UwSBpc4a5SM
- Auto Scaling - Amazon Web Services http://aws.amazon.com/autoscaling/
- Cluet, M., Autoscaling Best Practices, http://www.slideshare.net/lynxmanuk/autoscaling-best-practices
- Ferry, N., Chauvel, F., Rossini, A., Morin, B., & Solberg, A. (2013). Managing multi-cloud systems with CloudMF. In Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (pp. 38-45). ACM.
- M. Vasar, S. N. Srirama, M. Dumas: Framework for Monitoring and Testing Web Application Scalability on the Cloud, Nordic Symposium on Cloud Computing & Internet Technologies (NORDICLOUD 2012), August 20-24, 2012, pp. 53-60. ACM.
- S. N. Srirama, A. Ostovar: Optimal Resource Provisioning for Scaling Enterprise Applications on the Cloud, The 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom-2014), December 15-18, 2014, pp. 262-271. IEEE.
- S. N. Srirama, T. Iurii, J. Viil: Dynamic Deployment and Auto-scaling Enterprise Applications on the Heterogeneous Cloud, 9th IEEE International Conference on Cloud Computing (CLOUD 2016), June 27- July 2, 2016, pp. 927-932. IEEE.
- S. N. Srirama, A. Ostovar: Optimal Cloud Resource Provisioning for Auto-scaling Enterprise Applications, International Journal of Cloud Computing, ISSN: 2043-9997, 7(2):129-162, 2018. Inderscience.