Virtual Machine Placement in Cloud Environment

Dharmesh Kakadia

Advisor : Prof. Vasudeva Varma

Search and Information Extraction Lab

International Institute of Information Technology, Hyderabad

July 4, 2014

Outline

1. Introduction to Cloud and Scheduling
2. Dynamic SLA aware Scheduler
3. Network aware Scheduler
4. Wrap up

Cloud Computing

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]

[1] NIST Definition of Cloud Computing

Scheduling : History

The word scheduling is believed to be originated from a latin word *schedula* around 14th Century, which then meant *papyrus strip*, slip of paper with writing on it. In 15th century, it started to be used as mean *timetable* and from there was adopted to mean scheduler that we currently use in computer science.

Scheduling in computing, is the process of deciding how to allocate resources to a set processes. [2]

[2] Source : Wikipedia

Motivation

- The resource arbitration is at the heart of the modern computers.
- Can not afford ineffective resource management at cloud-scale.
- New challenges/opportunities due to
  - Virtualization
  - Consumption patterns
  - New workloads

Scheduling, it turns out, comes down to deciding how to spend money.[3]

[3] Towards a cloud computing research agenda. K. Birman et al. SIGACT'09

Scheduling

In simple notation, scheduling can be expressed as

$$Map < VM, PM >= f(Set < VM >, Set < PM >, context)$$

context can be

- Performance Model
- Heterogeneity of Resources
- Network Information

Problem

How to come up with function f ?

Problem

How to come up with function f ? That,

- Saves energy in data center while, maintaing SLAs
- Improves network scalability and performance
- Saves battery of mobile devices
- Saves cost in multi-cloud environment

Problem

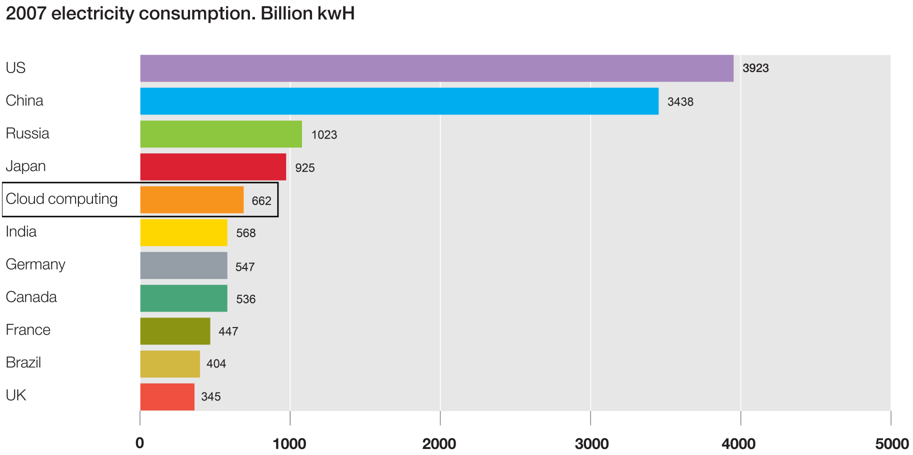
How to come up with function f ? That,

- **Saves energy in data center while, maintaing SLAs**
- **Improves network scalability and performance**
- **Saves battery of mobile devices**
- **Saves cost in multi-cloud environment**

Outline

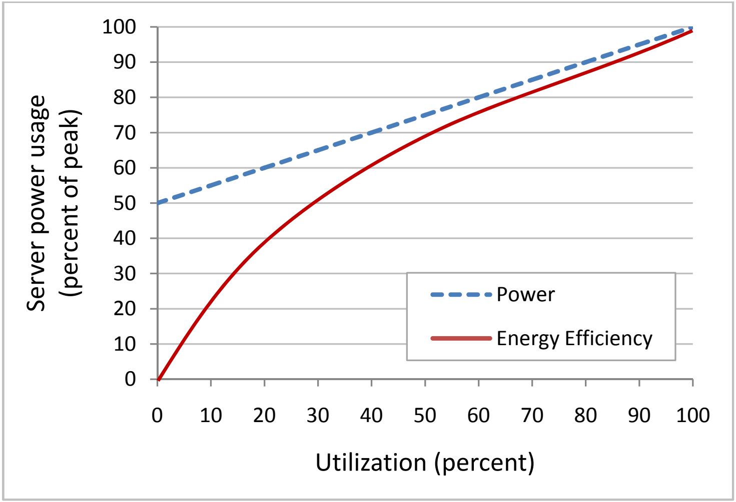
1. Introduction to Cloud and Scheduling
2. Dynamic SLA aware Scheduler
3. Network aware Scheduler
4. Wrap up

ELectricity Usage by Cloud Data Center



Source : Greenpeace Dirty Cloud Report

Server Power Characteristics



Goal

- Maintaining SLA guarantees while effectively saving the power consumed by the data center.
- Consolidate virtual machines effectively based on the resource usage.
- Maximize utilization of physical machines and put them to standby mode migrating VMs on to other physical machines.

Utilization Model

$$ResourceVector(RV) = < E_{cpu}, E_{mem}, E_{disk}, E_{bw} >$$

where

$$E_x = \frac{x \ used \ by \ VM}{max \ x \ capacity \ of \ PM} \tag{1}$$

Based on multiple resources viz. CPU, memory, disk and network as a single measure, U given as,

$$U = \alpha \times E_{cpu} + \beta \times E_{mem} + \gamma \times E_{disk} + \delta \times E_{bw}$$

where, $\alpha, \beta, \gamma, \delta \in [0, 1]$ And,

$$\alpha + \beta + \gamma + \delta = 1$$

Similarity Calculation

Based on Cosine similarity

Method 1 - Based on **dissimilarity** (lower the better) between RV of the incoming VM and RV_{PM}.

$$\textbf{similarity} = \frac{RV_{vm}(PM) \cdot RV_{PM}}{\|RV_{vm}(PM)\| \|RV_{PM}\|}$$

Method 2 - Based on **similarity** (higher the better) between RV of the incoming VM and PM_{free}.

$$\textbf{similarity} = \frac{RV_{vm}(PM) \cdot PM_{free}}{\|RV_{vm}(PM)\| \|PM_{free}\|}$$

Allocation Algorithm(VMs to be allocated)

for all VM $\in$ VMs to be allocated **do**

  for all PM $\in$ Running PMs **do**

    $similarity_{PM} = calculateSimilarity(RV_{vm}(PM), RV_{PM})$

    add $similarity_{PM}$ to *queue*

  end for

  sort *queue* ascending/descending using $similarity_{PM}$

  for all $similarity_{PM}$ in *queue* **do**

    $target_{PM} =$ PM corresponding to $similarity_{PM}$

    if U after allocation on *target* PM $< (U_{up} - buffer)$ **then**

      allocate(VM, *target* PM)

      return SUCCESS

    end if

  end for

  return FAILURE

end for

Scale-up Algorithm

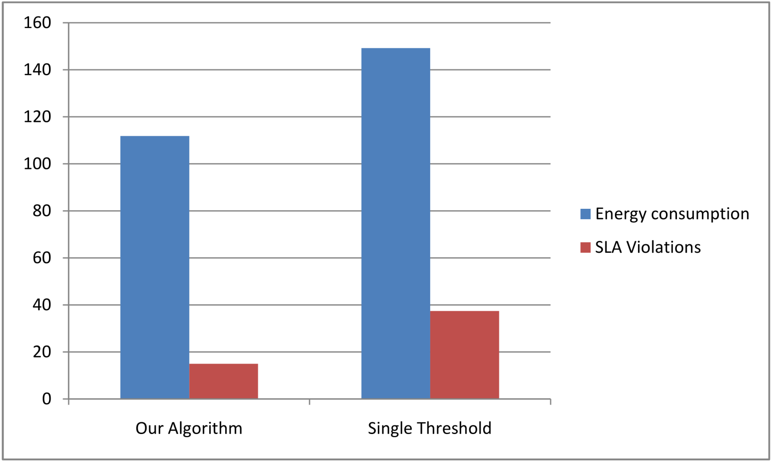
Scale up()
if $U > U_{up}$ **then**
  VM = VM with *max* U on that PM
  Allocation Algorithm(VM)
end if

if Allocation Algorithm fails to allocate VM **then**
  target PM = add a standby machine to running machine
  allocate(VM, *target* PM)
end if

Scale-down Algorithm

```
Scale down Algorithm()
if U < U_down then
    {if U of a PM is less than U_down}
    Allocation Algorithm(VMs on PM)
end if
```

Results : Energy and SLAs

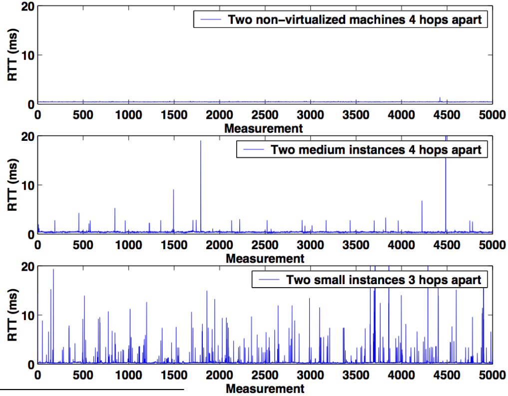


- $\sim$ 21% energy savings
- $\sim$ 60% less SLA violations

Outline

Network Performance in Cloud

- In Amazon EC2, TCP/UDP throughput experienced by applications can fluctuate rapidly between 1 Gb/s and zero.
- Abnormally large packet delay variations among Amazon EC2 instances. [4]



[4] G. Wang et al. *The impact of virtualization on network performance of amazon ec2 data center.* (INFOCOM'2010)

Scalability

- Scheduling algorithm has to scale to millions of requests
- Network traffic at higher layers pose signifiant challenge for data center network scaling
- New applications in data center are pushing need for traffic localization in data center network

Problem

VM placement algorithm to consolidate VMs using network traffic patterns

Subproblems

- *How to identify?* - cluster VMs based on their traffic exchange patterns
- *How to place?* -placement algorithm to place VMs to localize internal datacenter traffic and improve application performance

How to identify?

VMCluster is a group of VMs that has large communication cost (c_{ij}) over time period T.

How to identify?

VMCluster is a group of VMs that has large communication cost (c_{ij}) over time period T.

$$c_{ij} = AccessRate_{ij} \times Delay_{ij}$$

$AccessRate_{ij}$ is rate of data exchange between VM_i and VM_j and $Delay_{ij}$ is the communication delay between them.

VMCluster Formation Algorithm

$$AccessMatrix_{n\times n} = \begin{bmatrix} 0 & c_{12} & \cdots & c_{1n} \\ c_{21} & 0 & \cdots & c_{2n} \\ \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & \cdots & 0 \end{bmatrix}$$

c_{ij} is maintained over time period T in moving window fashion and mean is taken as the value.

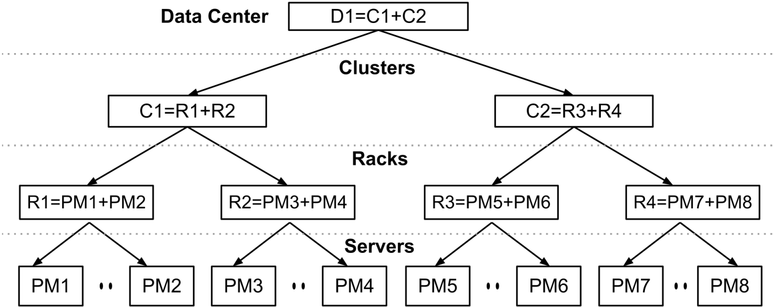
for each row $A_i \in$ AccessMatrix **do**
  if maxElement$(A_i) > (1 + opt_threshold) * avg_comm_cost$ **then**
    form a new VMCluster from non-zero elements of A_i
  end if
end for

How to place ?

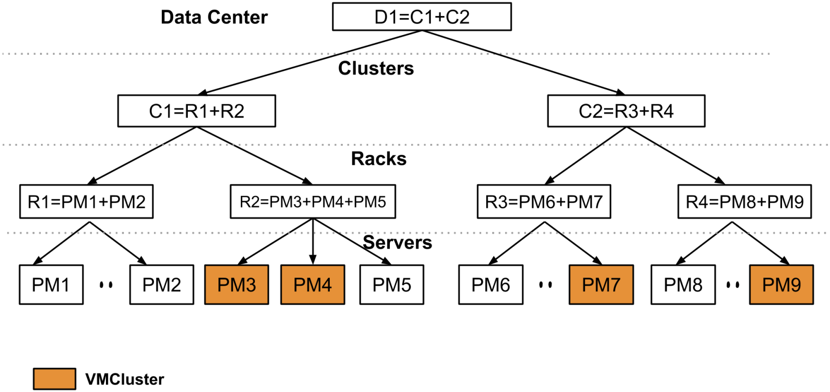
- Which VM to migrate?
- Where *can* we migrate?
- Will the the effort be worth?

Communication Cost Tree

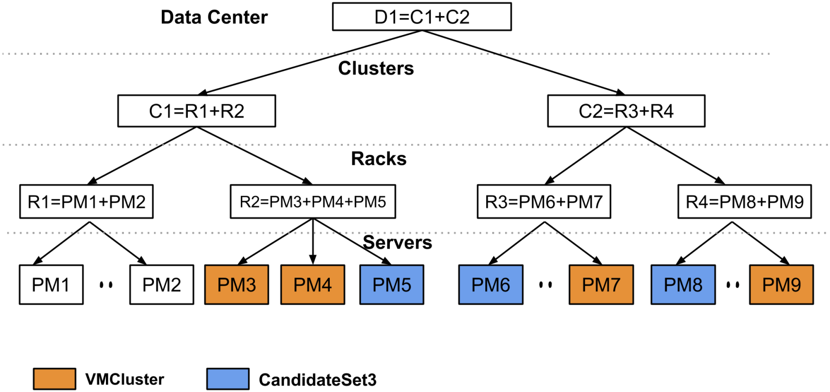
- Each node represents cost of communication of devices connected to it.



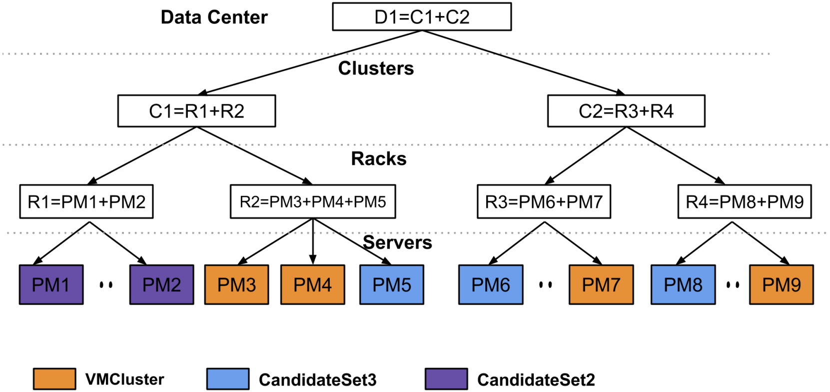
Example : VMCluster



Example : *CandidateSet*$_3$



Example : $CandidateSet_2$



How to place ?

How to place ?

Which VM to migrate?

$$VMtoMigrate = \arg\max_{VM_i} \sum_{j=1}^{|VMCluster|} c_{ij}$$

How to place ?

Which VM to migrate?

$$VMtoMigrate = \arg\max_{VM_i} \sum_{j=1}^{|VMCluster|} c_{ij}$$

Where *can* we migrate?

$$\begin{aligned} CandidateSet_i(VMCluster_j) = \{c \mid & \text{ where c and } VMCluster_j \\ & \text{have a common ancestor at level } i\} \\ & - CandidateSet_{i+1}(VMCluster_j) \end{aligned}$$

How to place ?

Which VM to migrate?

$$VMtoMigrate = \arg\max_{VM_i} \sum_{j=1}^{|VMCluster|} c_{ij}$$

Where *can* we migrate?

$$\begin{aligned} CandidateSet_i(VMCluster_j) = \{c \mid & \text{ where c and } VMCluster_j \\ & \text{have a common ancestor at level } i\} \\ & - CandidateSet_{i+1}(VMCluster_j) \end{aligned}$$

Will the the effort be worth?

$$PerfGain = \sum_{j=1}^{|VMCluster|} \frac{c_{ij} - c'_{ij}}{c_{ij}}$$

Consolidation Algorithm

- Select the VM to migrate
- Identify CandidateSets
- Select destination PM, check if
  - Destination will be overloaded
  - Gain is significant

Consolidation Algorithm

for $VMCluster_j \in$ VMClusters **do**
  Select $VMtoMigrate$
  for i from leaf to root **do**
    Form $CandidateSet_i(VMCluster_j - VMtoMigrate)$
    for $PM \in candidateSet_i$ **do**
      if UtilAfterMigration(PM,$VMtoMigrate$) $<$overload_threshold AND PerfGain(PM,$VMtoMigrate$) $>$ significance_threshold **then**
        migrate VM to PM
        continue to next VMCluster
      end if
    end for
  end for
end for

Experimental Evaluation

We compared our approach to traditional placement approaches like Vespa [1] and previous network-aware algorithm like Piao's approach [2].

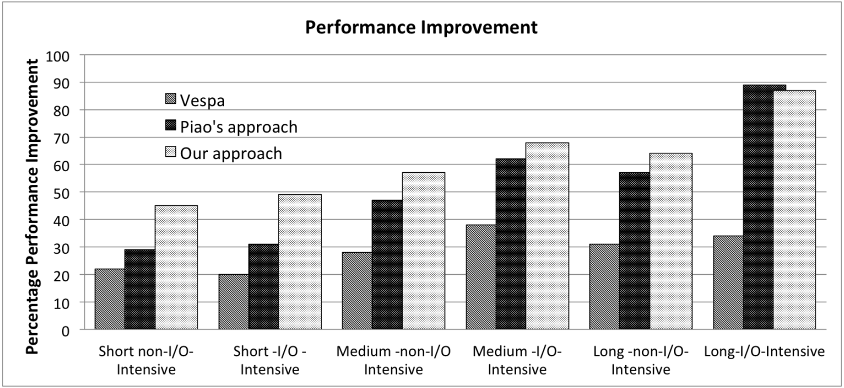
- Extended NetworkCloudSim [3] to support SDN.
- Floodlight
- The server properties are assumed to be HP ProLiant ML110 G5 (1 x [Xeon 3075 2660 MHz, 2 cores]), 4GB) connected through 1G using HP ProCurve switches.
- Traces from three real world data centers, two from universities (uni1, uni2) and one from private data center (prv1).

Trace Statistics

Traces from three real world data centers, two from universities (uni1, uni2) and one from private data center (prv1).

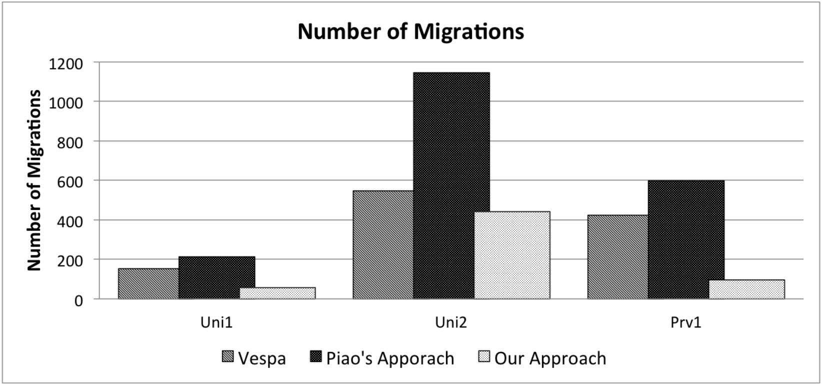
Property	Uni1	Uni2	Prv1
Number of Short non-I/O-intensive jobs	513	3637	3152
Number of Short I/O-intensive jobs	223	1834	1798
Number of Medium non-I/O-intensive jobs	135	628	173
Number of Medium I/O-intensive jobs	186	864	231
Number of Long non-I/O-intensive jobs	112	319	59
Number of Long I/O-intensive jobs	160	418	358
Number of Servers	500	1093	1088
Number of Devices	22	36	96
Over Subscription	2:1	47:1	8:3

Results : Performance Improvement



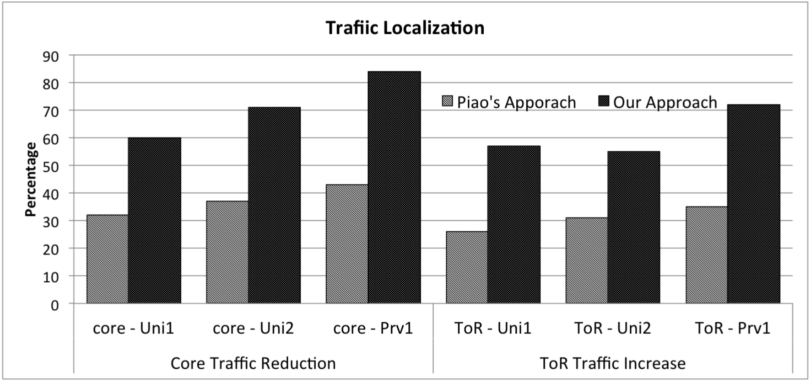
- I/O intensive jobs are benefited most, but others also share the benefit
- Short jobs are important for overall performance improvement

Results : Number of Migrations



- Every migration is not equally beneficial

Results : Traffic Localization



- 60% increase ToR traffic (vs 30% by Piao's approach)
- 70% decrease Core traffic (vs 37% by Piao's approach)

Results : Complexity – Time, Variance and Migrations

Measure	Trace	Vespa	Piao's approach	**Our approach**
Avg. scheduling Time (ms)	Uni1	504	677	**217**
	Uni2	784	1197	**376**
	Prv1	718	1076	**324**
Worst-case scheduling Time (ms)	Uni1	846	1087	**502**
	Uni2	973	1316	**558**
	Prv1	894	1278	**539**
Variance in scheduling Time	Uni1	179	146	**70**
	Uni2	234	246	**98**
	Prv1	214	216	**89**
Number of Migrations	Uni1	154	213	**56**
	Uni2	547	1145	**441**
	Prv1	423	597	**96**

Conclusion

- Network aware placement (and traffic localization) helps in Network scaling.
- VM Scheduler should be aware of migrations.
- Think rationally while scheduling, you may not want *all* the migrations.

Outline

Recap

Explored scheduling in environments where,

- Energy Efficiency and SLAs are important
- Extreme heterogeneous in terms of resource capabilities and network
- High Network communication

Future Directions

- Performance modeling for cloud apps
- Performance predictions for different configurations (cloud/app)
- Combining special subsystems like storage with scheduling
- Study of scheduling tradeoffs

Thank you

Related Publication

1. **Dynamic Energy and SLA aware Scheduling of Virtual Machines in Cloud Data Centers.** Dharmesh Kakadia, Radheyshyam Nanduri and Vasudeva Varma. Unpublished manuscript.
2. **MECCA: Mobile, Efficient Cloud Computing Workload Adoption Framework using Scheduler Customization and Workload Migration Decisions.** Dharmesh Kakadia, Prasad Saripalli and Vasudeva Varma. In *MobileCloud '13*.
3. **Energy Efficient Data Center Networks - A SDN based approach** Dharmesh Kakadia and Vasudeva Varma. In *I-CARE'12*.
4. **Optimizing Partition Placement in Virtualized Environments.** Dharmesh Kakadia and Nandish Kopri. *Patent P13710918*.
5. **Network-aware Virtual Machine Consolidation for Large Data Centers.** Dharmesh Kakadia, Nandish Kopri and Vasudeva Varma. In *NDM collocated with SC'13*.
6. **MultiStack.** http://MultiStack.org

Backup Slides

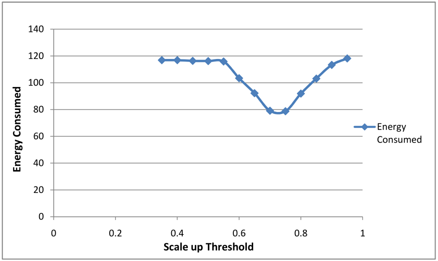
Discussion

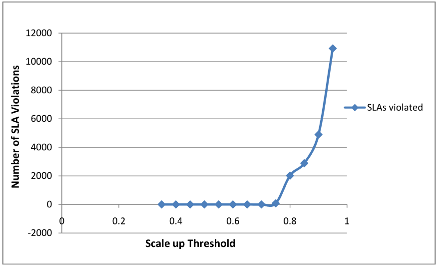
- Scale-up/down is triggered based on observation over a period of time, to avoid unstable behavior.
- Predict utilization on destination machine, to avoid SLA violation and unstable behavior.
- Use Buffers - to help guard against wrong decisions.
- Percentage (not absolute) utilization means algorithms work unchanged for heterogeneous data centers.
- Pick least recently used machine while scale up - all machines used uniformly - avoids hotspot.
- Difference between U_{up} and U_{down} should be sufficiently large to avoid jitter effect.

Simulation and Algorithm Parameters

Parameter	**Value**
Scale-up Threshold, U_{up}	[0.25, 1.0]
Scale-down Threshold, U_{down}	[0.0 to 0.4]
buffer	[0.05 to 0.5]
Similarity Threshold	[0, 1]
Similarity Method	Method 1 or 2
Number of physical machines	100
Specifications of physical machines	Heterogeneous
Time period for which resource usage of VM is logged for exact RV_{vm} calculation, Δ	5 minutes

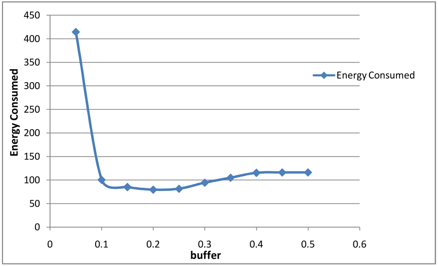
Results : Effect of U_{up}

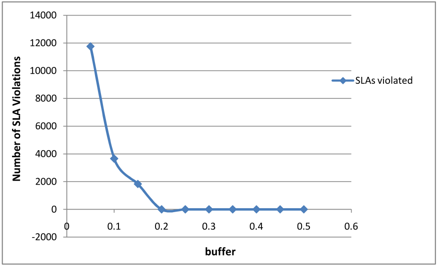




- U_{up} should not be too high or too low (optimal around 0.70-0.80)
- high U_{up} means a lot more SLA violation
- If U_{up} is low, Scale-up algorithm will run more than necessary machines.

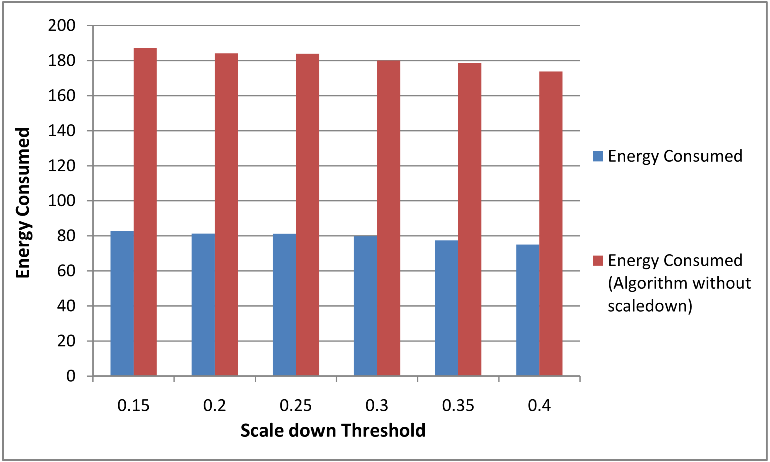
Results : Effect of *buffer*





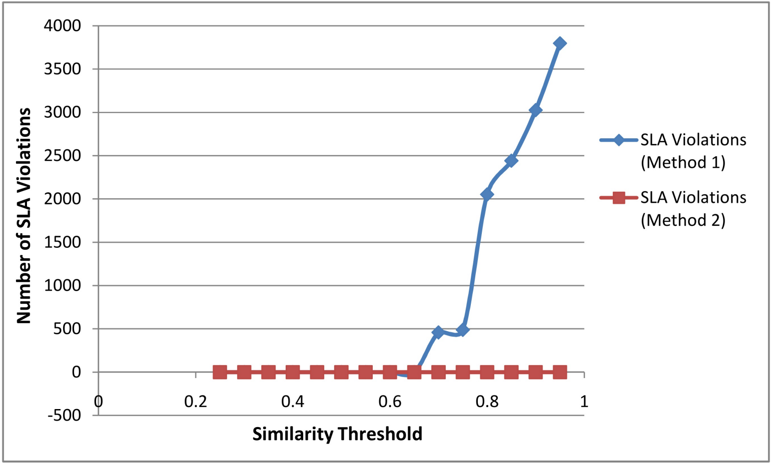
- Buffer has benefits
- Keep *buffer* only what is required
- Beware of too high values, will lead to less consolidation

Results : Effect of scale down

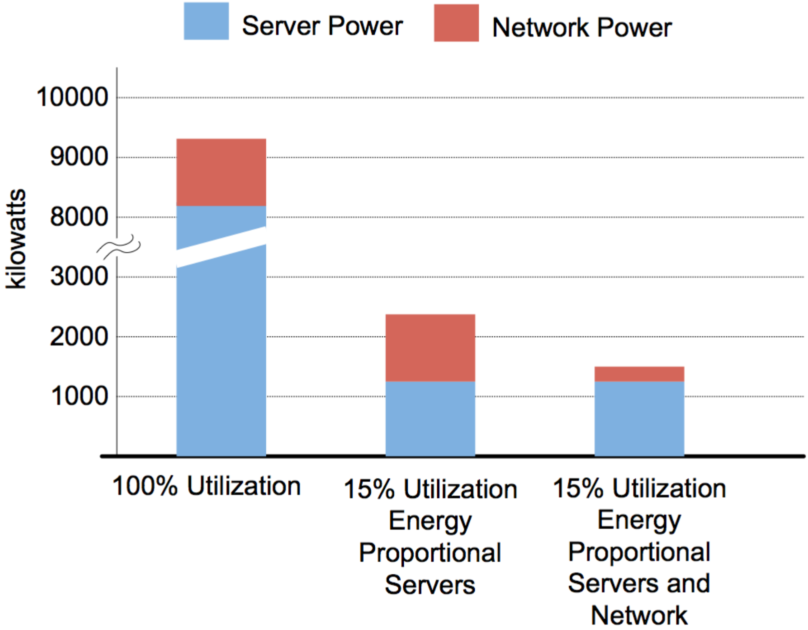


- 50% energy savings

Results : SLA : Similarity or Dissimilarity



- Similarity is better than dissimilarity



The variance in delay as number of flows grows

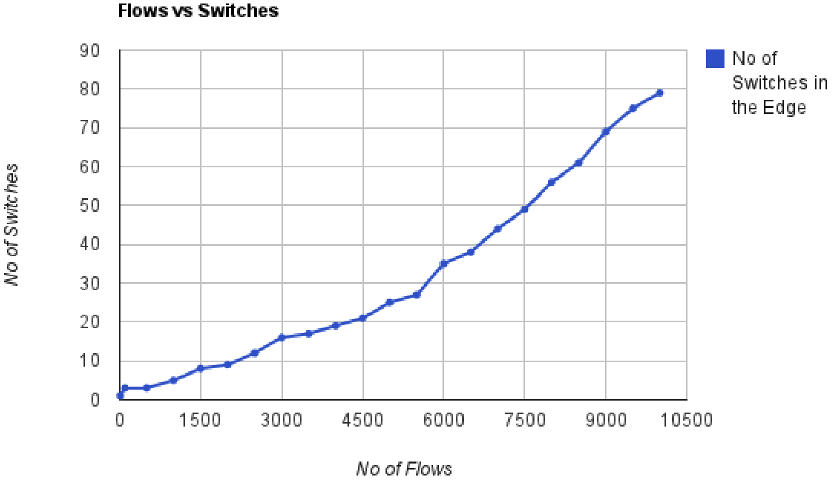
Consolidation Algorithm

```
Update traffic metrics using SDN counters
for each Switch s in S such that Utilization(s) ¡ threshold θ over time t do
    if canMigrate(s, S-s)) then
        pFlows = prioritizeFlows(s)
        incrementalMigration (pFlows)
        Poweroff(s)
    end if
end for
```

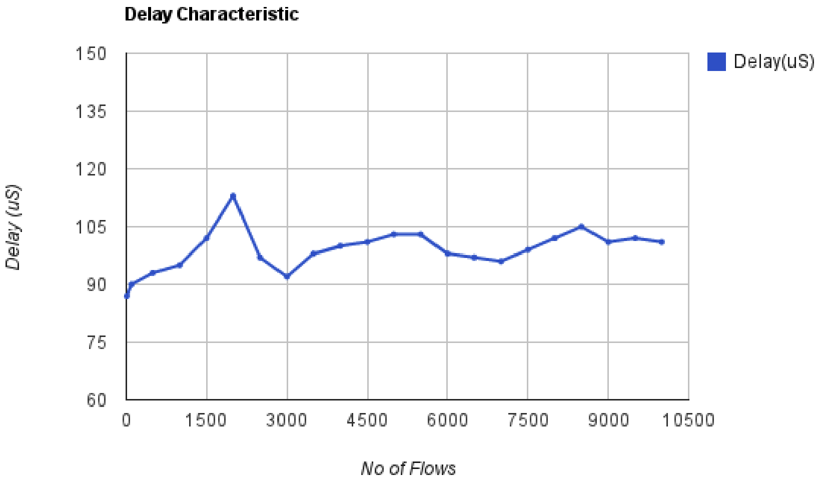
Simulation Setup

Parameter	Value
Number of Hosts	2000
Number of Edge Switches	100
Topology	FatTree
Link Capacity	100 MBPS
Switch booting time	90 sec
Number of Ports per Switch	24

Results : # switches required



Number of active switches as the number of Flows grows almost linearly



The variance in delay as number of flows grows

Current Mobile Cloud Landscape

By 2016, 40% of Mobile apps will use cloud back-end services. [5]

- cloud-enabled Apps
  - Dropbox, Evernote, Instagram, ...
  - Siri, Google Voice, ...
  - Kindle, ...
- Traditional Apps
  - GIMP
  - Firefox
  - Games

[5] http://www.gartner.com/newsroom/id/2463615

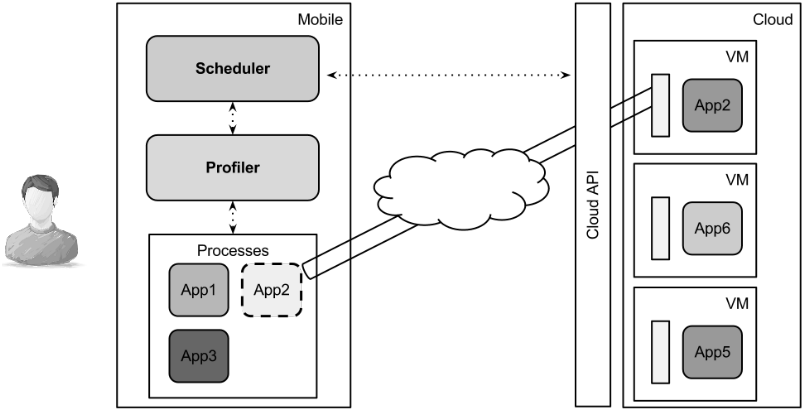
Mobile Cloud Opprtunity

- Mobile devices are becoming powerful, but rich applications are more and more hungry for resources.
- Cloud has *infinite* resources.
- Cloud is programmable.
- Always ON.
- Only a handful apps are leveraging cloud.

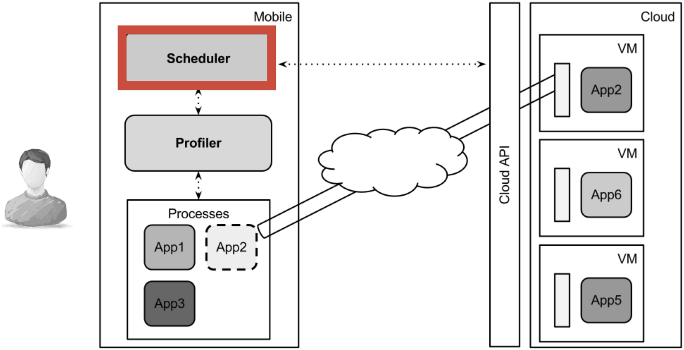
Motivation

- Observation : Many apps are not cloud-aware, but can be migrated.
- Can we create a Mobile cloud framework that leverage cloud resources,
  - Without making app cloud-aware
  - Without annoying user
  - Adaptive
  - Personalized
  - Works autopilot mode

Environment & Assumptions

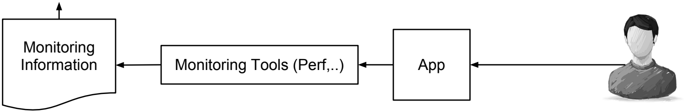


Environment & Assumptions

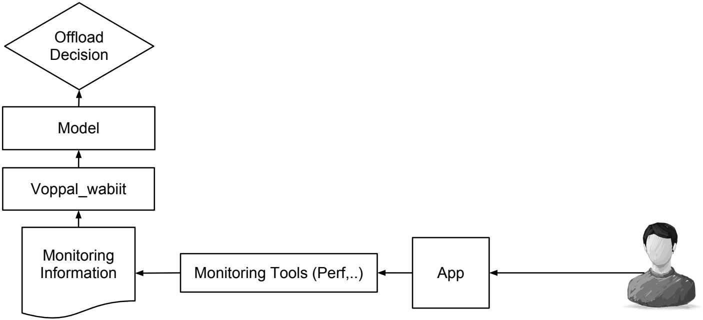


When to offload application to cloud?

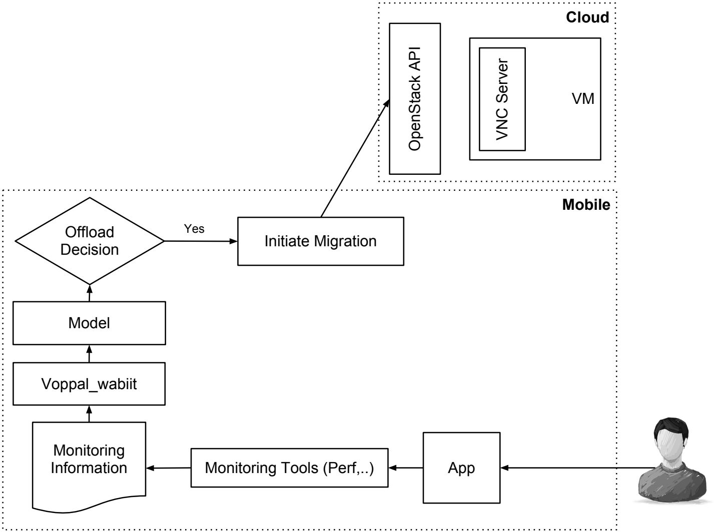
Workflow : App launch



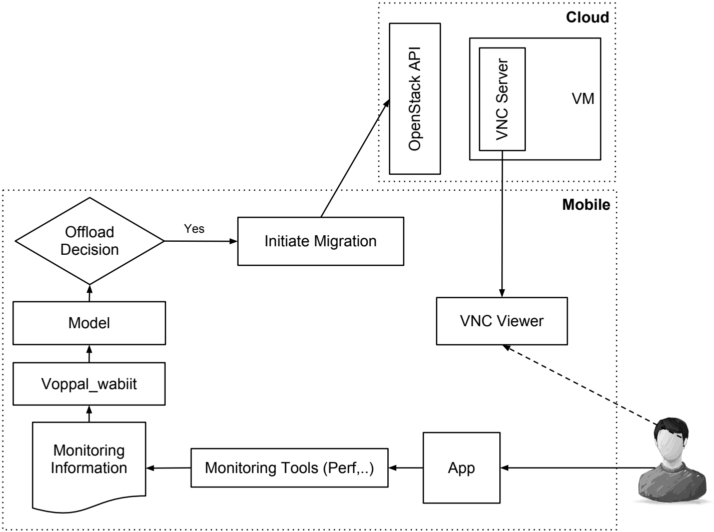
Workflow : Offload Decision



Workflow : Initiating Migration



Workflow : Remoting



Offloading Decision

if $Gain_p \geq significance\_threshold$ **then**
  Execute the p remotely on cloud.
else
  continue executing p locally.
end if

significance_threshold controls aggressiveness

Performance Gain

Feature Gain,

$$f_i = \frac{(m_i - c_i)}{m_i}$$

m_i : cost of running the application on mobile device $(0 - 1)$
c_i : cost of running the application on cloud device $(0 - 1)$

Performance Gain,

$$Gain = \frac{\sum(w_i \times f_i)}{\sum w_i}$$

w_i : weight of i the feature gain, normalized to unity

Learning Algorithm

- Gain as regression problem with squared loss function learned in an online setting
- Used vowpal wabbit [6] : fast online learning toolkit
- Features :
  - High level features
  - App features
  - Network features
  - Other Apps
  - Device static features
  - Cloud provider features

[6] https://github.com/JohnLangford/vowpal_wabbit/

Dynamic Features

- High level features : comprise of features that are concerned to user. Includes battery status, date and time, user location (moving/stable), etc.
- Application features : capture application usage habits including frequency of usage of the application, stretch of usage, use of local and remote data, etc.
- Network Status : network condition between cloud and mobile device. Includes bandwidth, latency and stability.
- Resource usage by other applications running on device : combined vector of all individual applications.

Non-Dynamic Features

- Device Configuration : capture all the hardware and software configuration of the device.
  - cpu frequency
  - cpu power steps
  - operating frequency, etc.
- Cloud Configuration: This captures characteristics of the cloud provider.
  - monetary cost
  - provider performance statistics

Evaluation

- A virtual machine running android as a mobile device
- Linux traffic control utility (tc) is used to simulate various network condition
- Used OpenStack as IaaS cloud provider

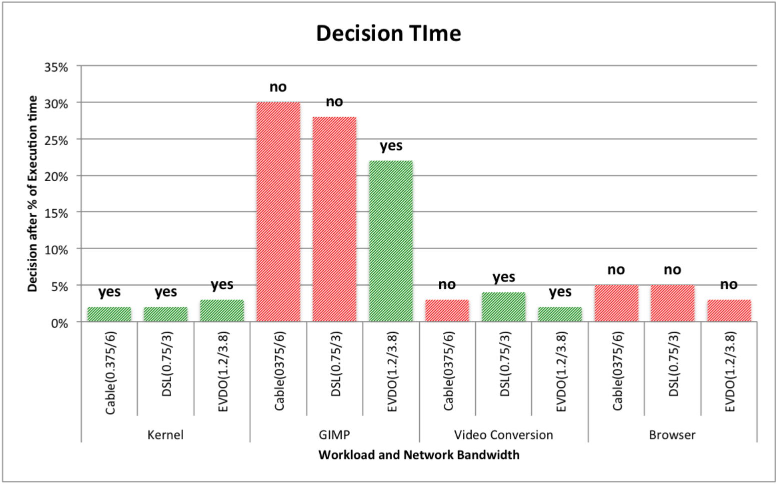
Property	Value
Cloud Operating System	Ubuntu 12.04(kernel 3.2)
Cloud VM configuration	4 GB, 2.66GHz
Device Operating System	Android 4.2
Device Configuration	1GB, 1.5 GHz

Workloads

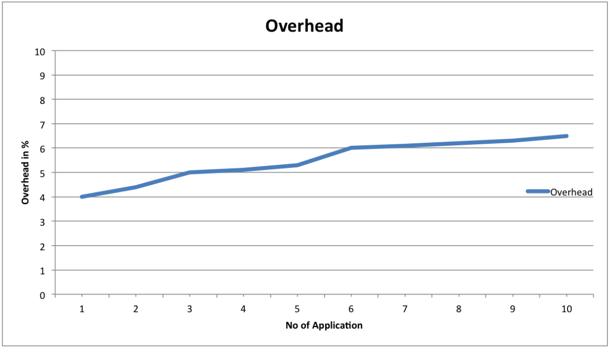
- Representative of normal user interaction
- Applications with varying resource utilization and duration
- On varying Network speed : cable(0.375/6), DSL(0.75/3) and EVDO(1.2/3.8)

Workload	Description	Characteristics
Kernel	kernel download + build	long + resource intensive
GIMP	Image editing + applying image filters	interactive + little intensive
Video conversion	download & convert a (500MB) video	short + resource intensive
Browser	browsing 5 sites	interactive

Results : Decision and Time taken



Results : Overhead



- Measured as % increase in the resource utilization with and without running our system.
- Overhead between 4–7 %

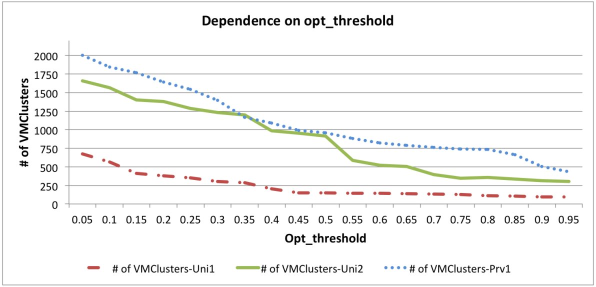
Conclusion

A Mobile cloud scheduler that is

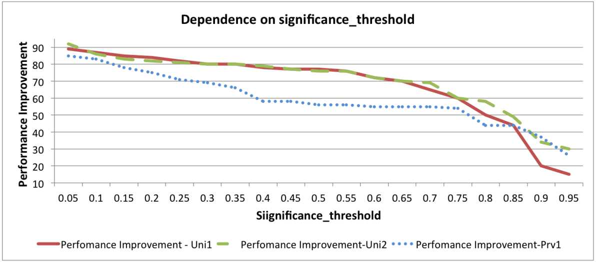
- Context-aware
- Adaptive to various workloads automatically
- Personalized
- Easy to use

and uses learning algorithm for system optimization

Results : Sensitivity to parameters



After 0.6, traffic pattern controls #VMCluster



All the improvements will be discarded as *insignificant* if *significance_threshold* is very high

Problem

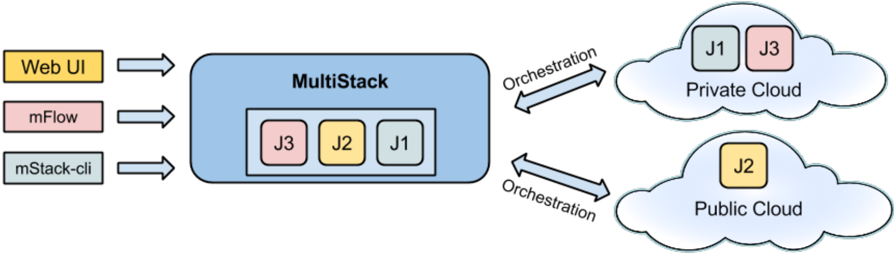
- Cloud market place is fragment.
- Very little (and only superficial) inter-operability. Each cloud is very different (Architecture/SLA/Abstraction/API/...).
- Likely to stay like this, due to conflict of interests.
- Can lead to lock-in, Data-loss, Cost increase.
- Many new applications have bursty nature.

MultiStack : Multi Cloud Big Data Research Platform

- Think as OS for Multiple Clouds.
- To identify problems and evaluate solutions to multicloud platform.
- More challenging than data center scheduling.
- Big data as the first use case.

Overview

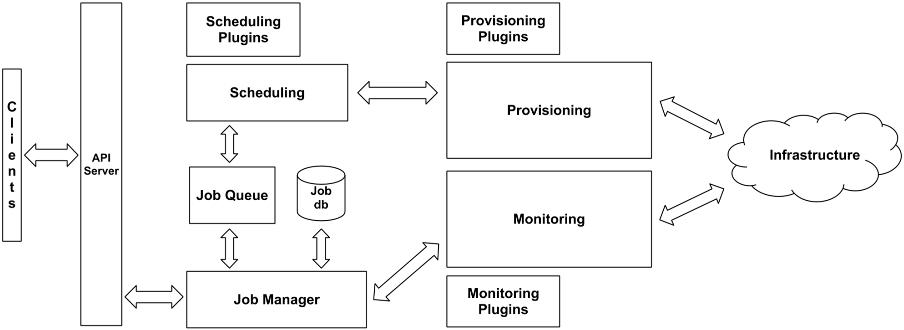
MultiCloud : Ability to use resources from multiples clouds seamlessly.



MultiStack : Services

- Resource Management
- Migration
- Monitoring
- Identity and Authentication
- Data Management
- Billing

MultiStack : Architecture



Progress so far

- Base Platform
- Simple capacity based scheduler
- Provisioning on AWS and OpenStack
- Deployment Hadoop clusters
- Manual scaling of clusters

Immediate features in pipeline

- Auto Scaling
- Ability to run across multiple cloud providers
- Priority based Job scheduling for minimizing cost and completion time
- Performance optimization with storage integration
- Client Tools
- More frameworks (Spark, Hive, Pig, Oozie, Drill, MLlib,..)
- Other Schedulers (Autoscaling, Spot-instances, Job profile based)