



深圳理工大学  
SHENZHEN UNIVERSITY OF  
ADVANCED TECHNOLOGY



中国科学院深圳先进技术研究院  
SHENZHEN INSTITUTE OF ADVANCED TECHNOLOGY  
CHINESE ACADEMY OF SCIENCES

# 云计算技术与工程

## 虚拟化数据中心管理

徐敏贤

副研究员

云计算研究中心

中国科学院深圳先进技术研究院

<http://www.minxianxu.info/vcc>

云来山更佳，云去山如画，山因去晦明，云共山高下。  
——（元）张养浩

# 内容回顾



- Q1: 云计算为什么需要负载均衡?

- 
- Distribute the workloads equally on available resources
  - Provide continuous service in case of failure of services' components
  - Minimize the resource time of tasks and improve resource utilization
  - Provide scalability and flexibility for those application whose size may increase in future
  - Provide priority to jobs that need instant execution as compared with another jobs

- Q2: 静态负载均衡 (static load balancing) 和动态负载均衡 (dynamic load balancing) 分别是什么意思? 有什么优缺点?

## ■ Static

- Follow a fixed set of rules that are not dependent on the system's current state
- E.g. Min-Min, Max-Min, Round-Robin, Shortest Job First.....

## ■ Dynamic

- Consider the current state of the system and make a decision
- Agent-based, dynamic threshold, markov chain, ant colony optimization.....

- Q3: 负载均衡中有哪些策略 (Load Balancing Policies) ? 这些策略的目标是什么?

## ■ Selection Policy

- Identify the tasks that should be transferred from one node to another based on the migration overhead

## ■ Location Policy

- Determine the computing nodes that are underloaded or free and transfers tasks to them for processing

## ■ Transfer Policy

- Discover the circumstances in which tasks are required to transfer from a local node to another local/remote node

## ■ Information Policy

- Keep all resource information in the system

## 第一部分

集群管理

## 第二部分

资源超卖

## 第三部分

异构与干扰

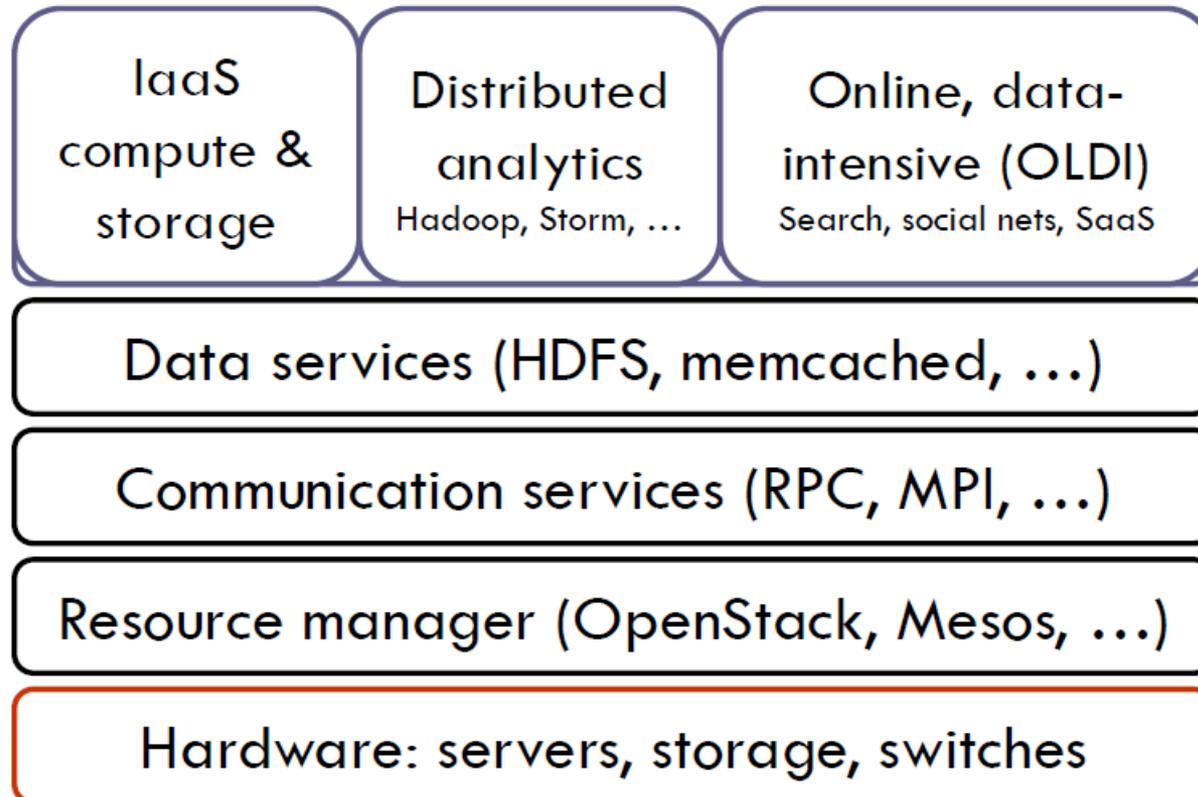
## 第四部分

集群自动化工具

注：本章内容选自 [《The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines》](#) 一书第二版



# Datacenter Software Stack



- **Server configuration & package management**
  - What should get installed where?
  - Popular tools: puppet, chef, ansible, salt, docker, ...
- **Application naming/discovery**
  - How should services find each other?
  - Popular tools: static IPs/hostnames, DNS, eureka, serf, skyDNS, ...
- **Application deployment**
  - What should run where?
  - How it should be started, stopped, monitored?
  - Popular tools: mesos, openstack +cloud foundry, Borg, kubernetes, torque, capistrano ...

# Cluster Management

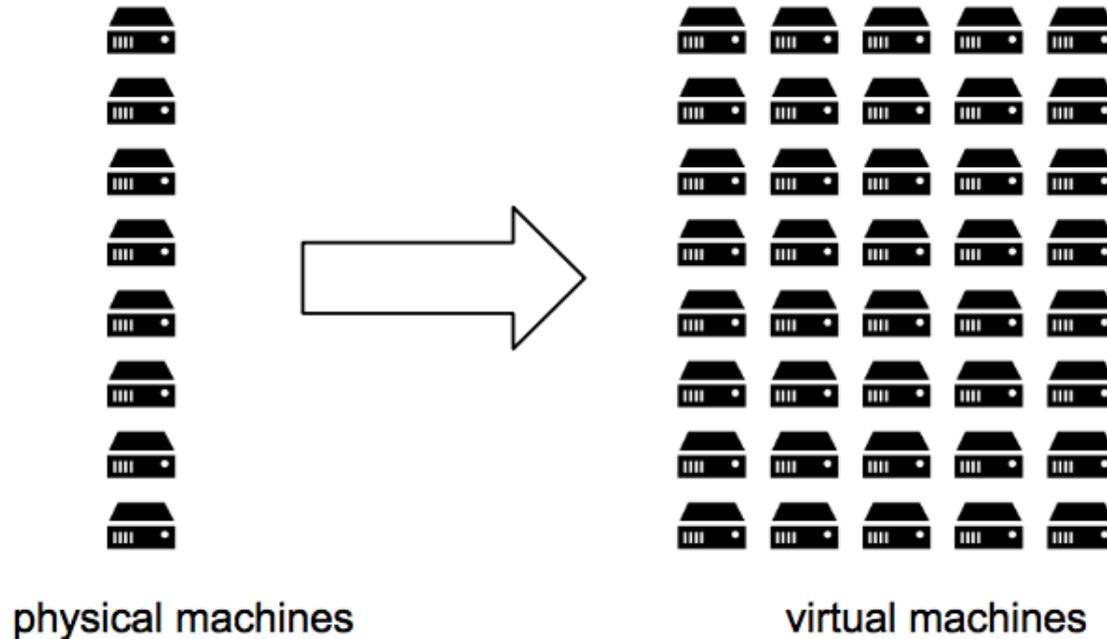


- Server configuration & package management
  
- Application naming
  
- Application deployment
  - How many and which resources should an app get?
  - App performance vs resource utilization

# VMs are Orthogonal to this Discussion



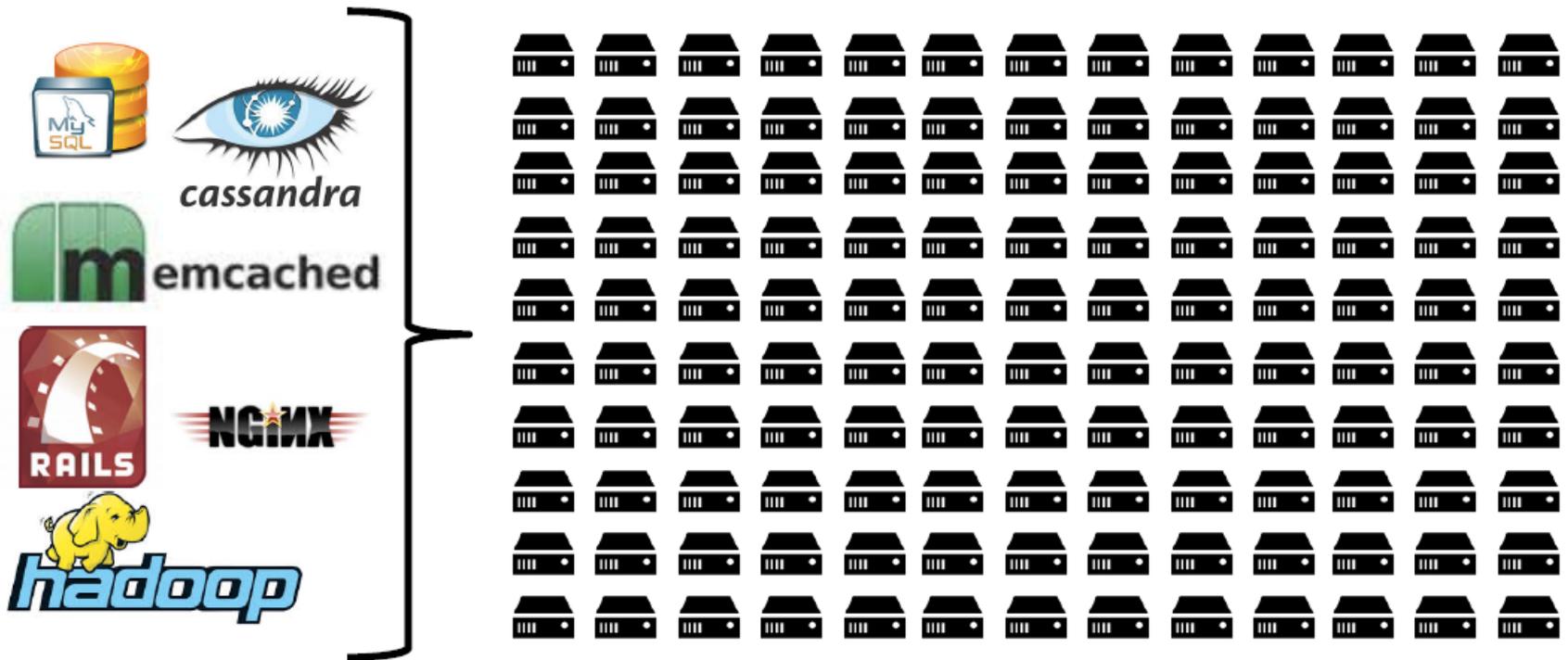
- VMs provide flexibility (encapsulation, migration, ...)
- But provisioning & assignment challenges remain



# Resource Assignment Options



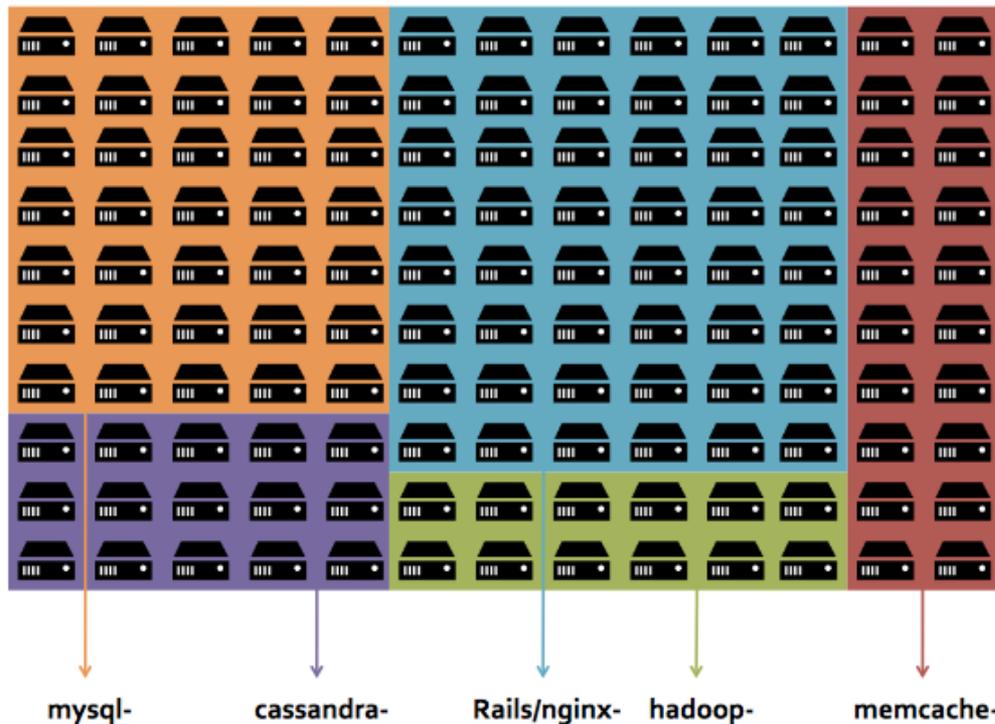
- How do we assign resources to apps?
- Two major options: private vs shared assignment



# Private Resource Assignment



- Each app receives a private, static set of resources
- Also known as static partitioning



# Private Resource Assignment



## ■ Advantages

- Simplicity
- Performance isolation (?)
- Security isolation (?)
- Allows for Hardware specialization

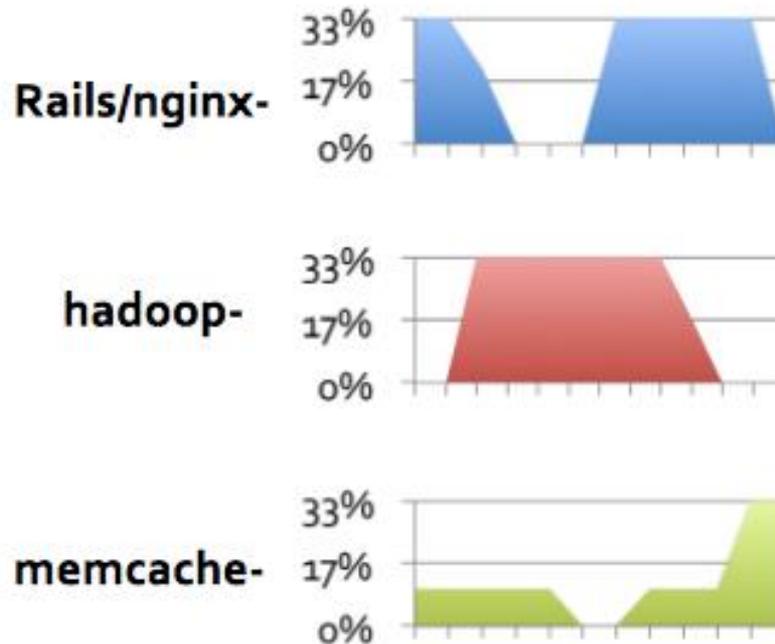
Server types @ Facebook

Standard Systems	Type I	Type II	Type III	Type IV	Type V	Type VI
CPU	High 2 x EN2670	Low 1 x 6128HE (AMD)	Medium 2 x X5650	Medium 2 x X5650	Low 1 x L5630	High 2 x EN2660
Memory	Low 16GB	High 144GB	High 144GB	Medium 48GB	Low 18GB	High 144GB
Disk	Low 250GB	Low 250GB	High IOP 6 x 600GB SAS +2x1.3TB Flash	High 12 x 3TB SATA	High 12 x 3TB SATA	Medium 1TB SATA
Services	Web, Chat, Ads	Memcache, Ads	Database	Hadoop	Photos, Video	Multifeed, Search

# Private Resource Assignment



- Challenges
  - Failures & maintenance
  - Low utilization





- Important and related issues
  - Reducing tail latency
  - Load balancing
- Key for user-facing, scale-out workloads
  - E.g., web-serving, search, etc.

# Load Balancing



- Assuming multiple machines that run a webserver
  
- How would you divide work between them?
  - ...
  - ...
  - ...

# Tail Latency Challenges



- The problem is variability
  - Queueing
  - Local and global resource sharing
  - Background daemons & maintenance tasks
  - Power management, garbage collection, ...
- Remember: even rare events are problematic
  - If requests touch 100s of servers
  
- Can we eliminate all sources of variability?

# Reducing Tail Latency

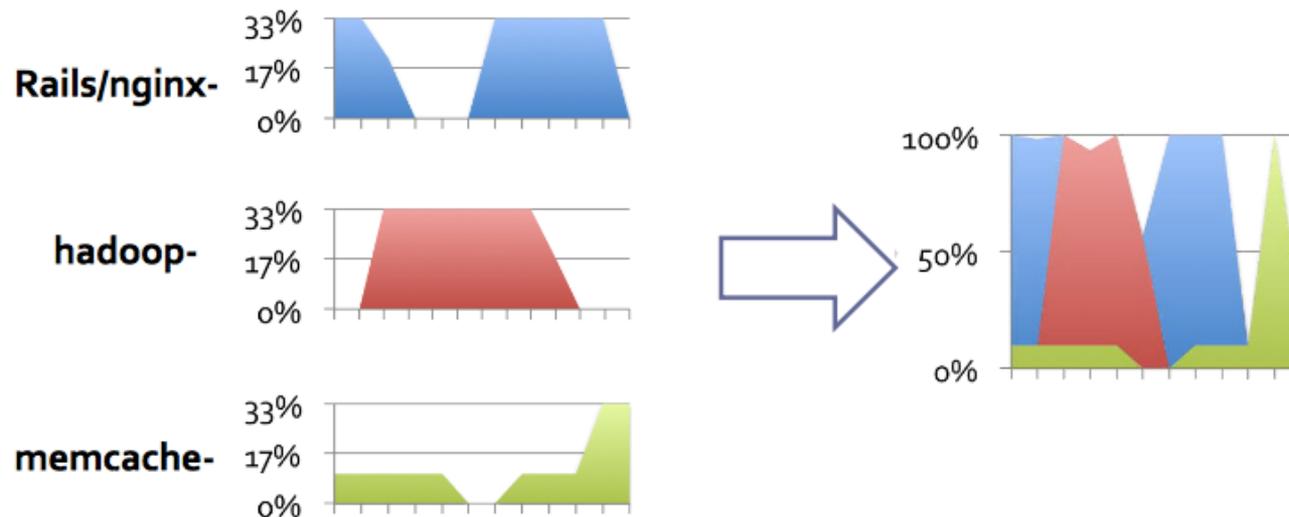


- Reduce queueing (reduce head of line blocking)
- Separate different types of requests
- Coordinate background activities
- Micro-sharding & selective replication
- Latency induced probation, canary requests
- Tied requests to replicas
- Hedged requests to replicas
  
- See *The Tail @ Scale* paper for details

# Shared Resource Assignment



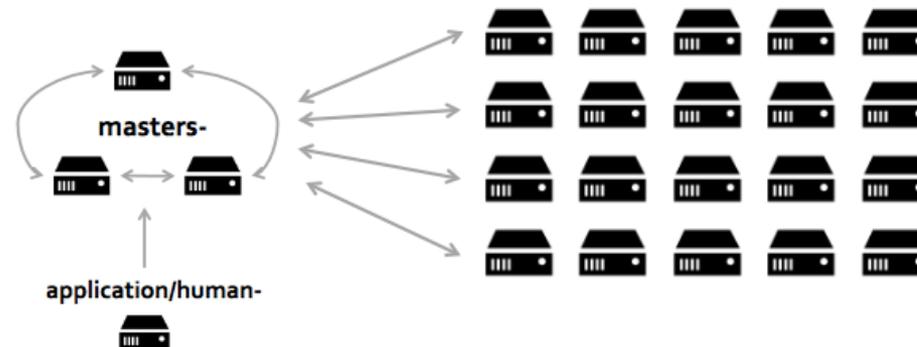
- Shared resources: flexibility → high utilization
  - Common case: user-facing services + analytics on same servers
  - Also helps with failures, maintenance, and provisioning



# Shared Cluster Management



- The manager schedules apps on shared resources
  - Apps request resource reservations (cores, DRAM, ...)
  - Manager allocates and assigns specific resources
    - Considering performance, utilization, fault tolerance, priorities, ...
    - Potentially, multiple apps on each server
  - Multiple manager architectures (see Borg paper for example)



# Cluster Manager Designs



## ■ Centralized:

- A single manager responsible for the entire cluster
- Borg [Eurosys'15] by Google, Quasar [ASPLOS'14] by Stanford, Torque
- Advantages/Disadvantages?

## ■ Two-level:

- One central master and multiple framework schedulers
- Mesos [NSDI'11] by Apache, YARN [SOCC'13] by MS and facebook
- Advantages/Disadvantages?

## ■ Distributed:

- Multiple concurrently operating sched. agents (partitioned or shared state)
- Omega [Eurosys'13] by Cambridge, Sparrow [SOSP'13] by Berkeley
- Advantages/Disadvantages?

↑  
shape the final version of the paper. Finally, we thank the reviewers from HotCloud 2012, OSDI 2012, NSDI 2013, and SOSP 2013 for their helpful feedback.

# Challenges of Shared Clusters?



## ■ Advantages:

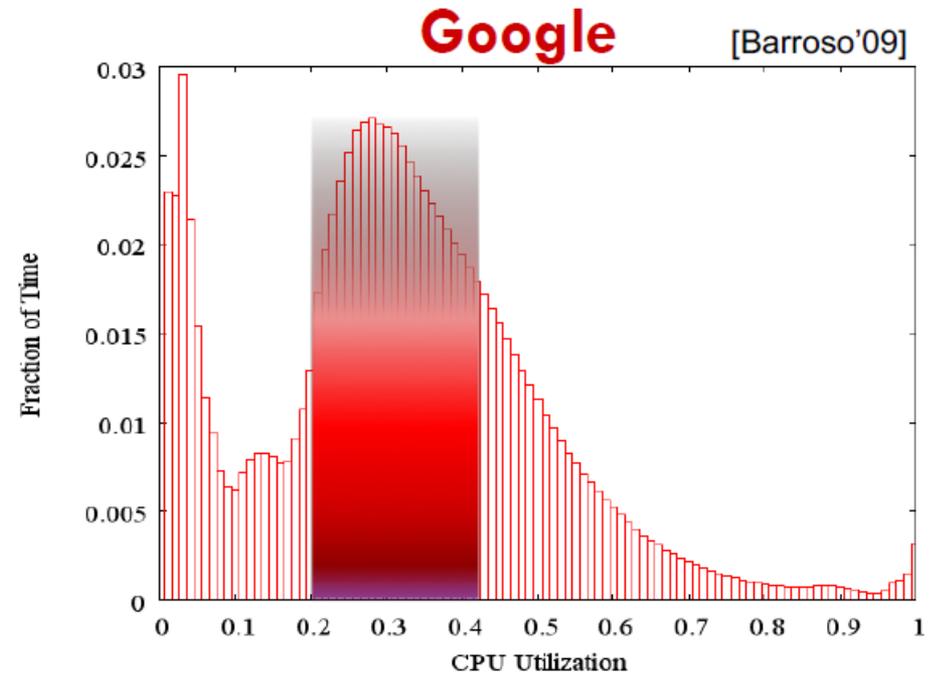
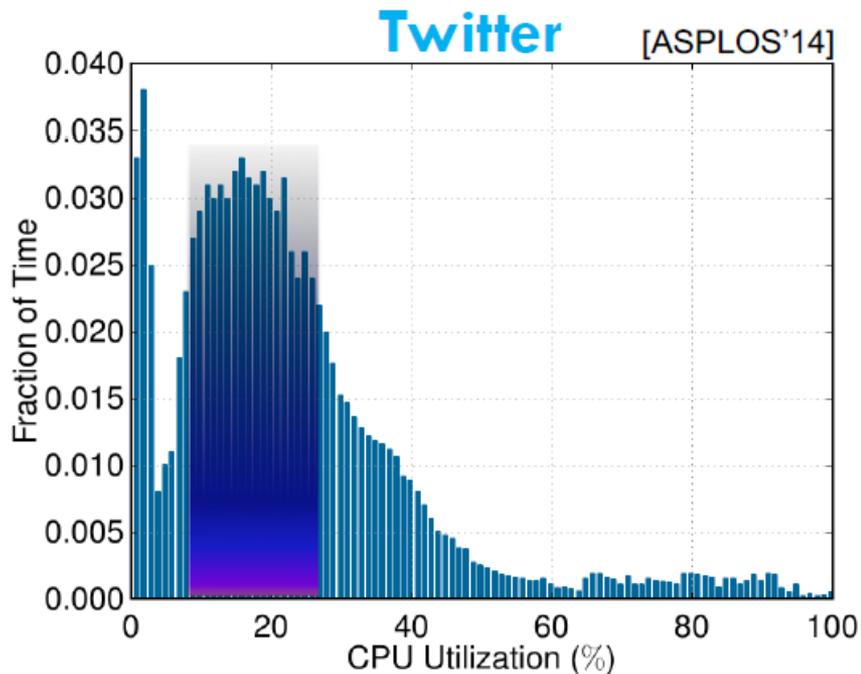
- Sea-of-resources: each app can request only what it needs
- Better resource sharing → higher utilization → resource efficiency
- Limited heterogeneity → cost efficiency, ease of maintenance

## ■ Disadvantages?

# Shared Clusters Can Be Underutilized Too



- Dynamic cluster management in both cases
  - Twitter uses Mesos and Google uses Borg
  - What is the problem?



# Datacenter Underutilization...

---

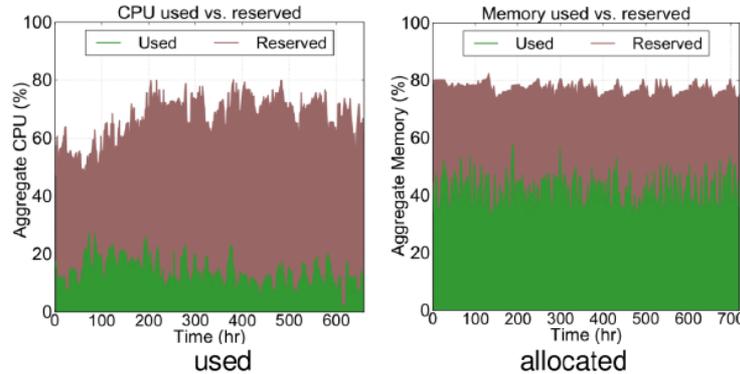


Is the cluster manager's fault!

# Overprovisioning & Resource Usage

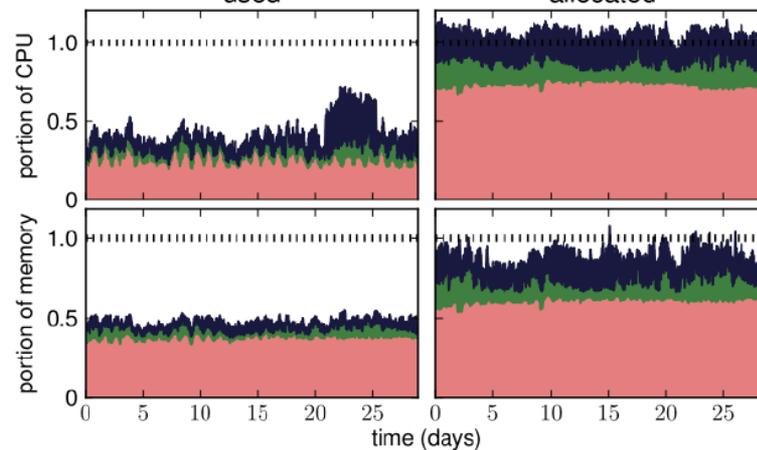


Twitter



[ASPLOS'14]

Google



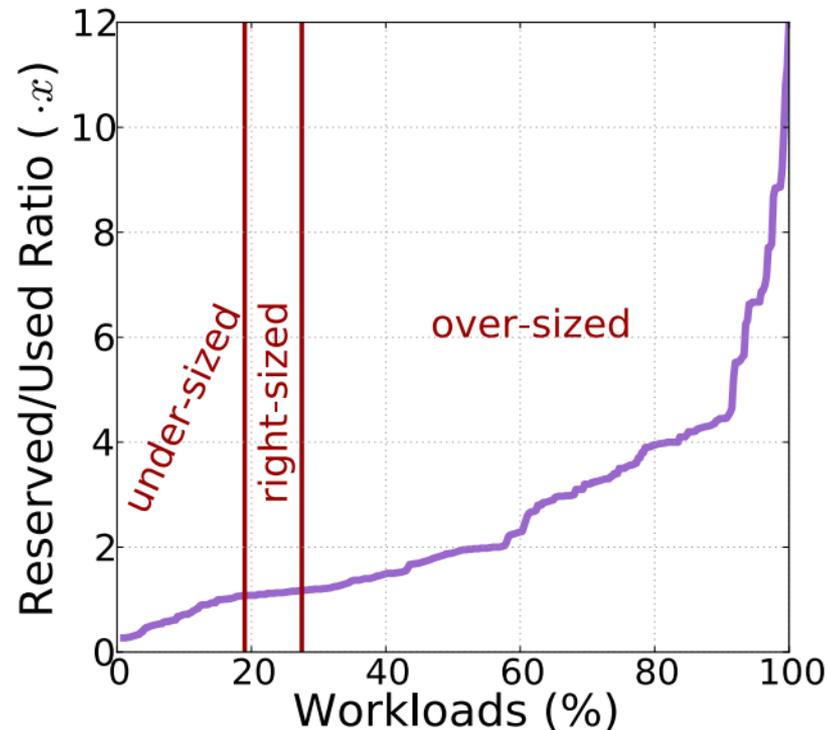
Production – high priority  
Medium priority  
Gratis

[Reiss'12]

# A Common Cure: Overprovisioning



- Twitter: 20% of tasks under-sized and **70% of tasks over-sized** 3-5x CPU and 2x memory overprovisioning



# Datacenter Underutilization..

---



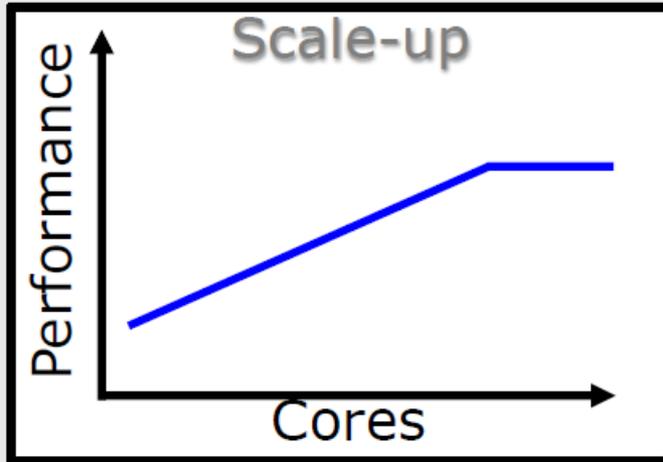
Is the user's fault!  
(not really...)

# Resource Management is Hard

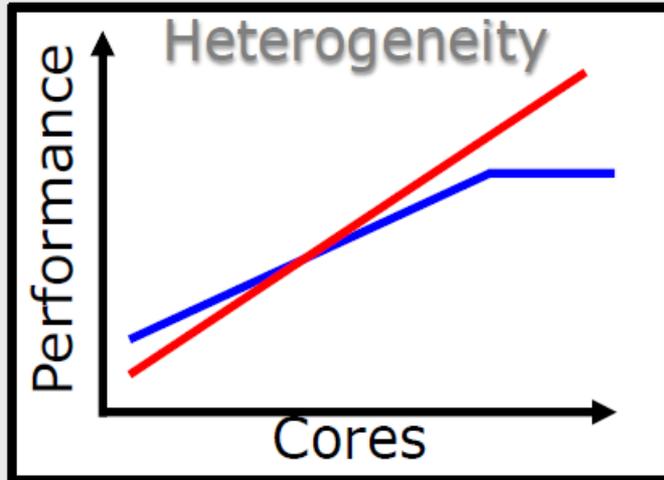
---



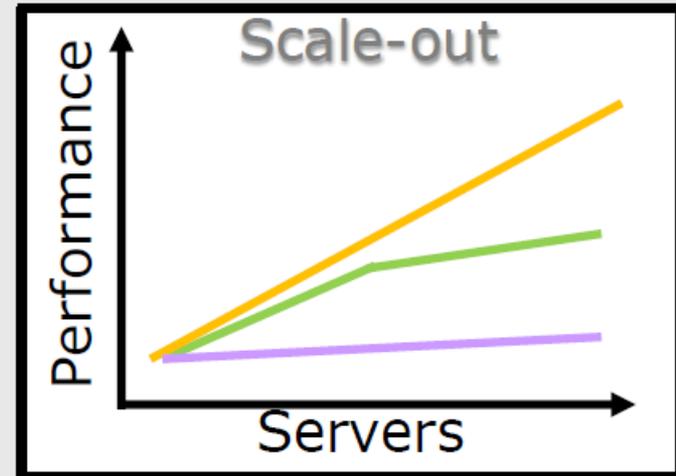
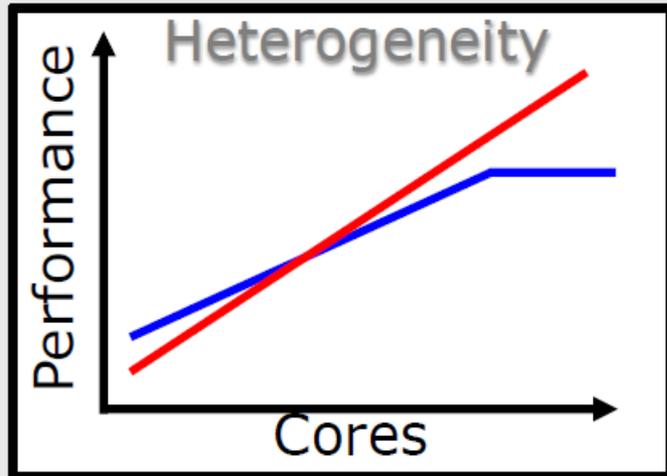
# Performance Depends on



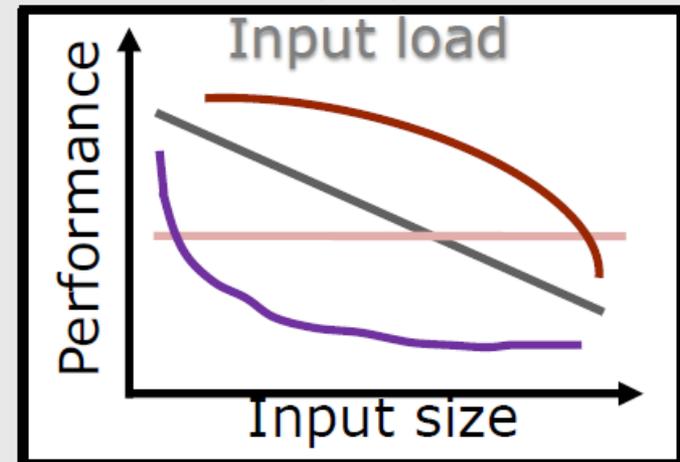
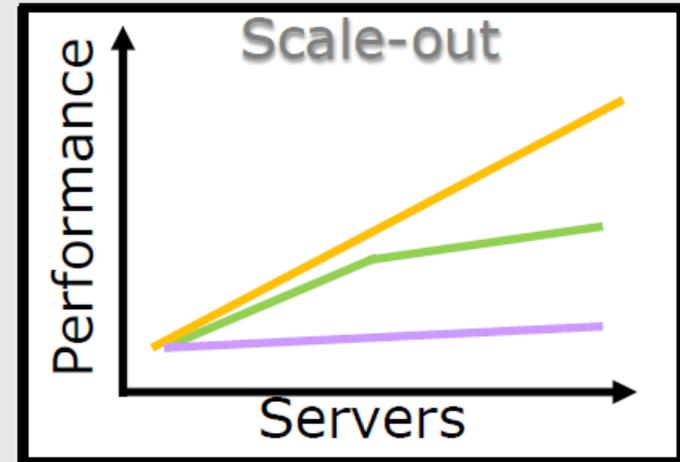
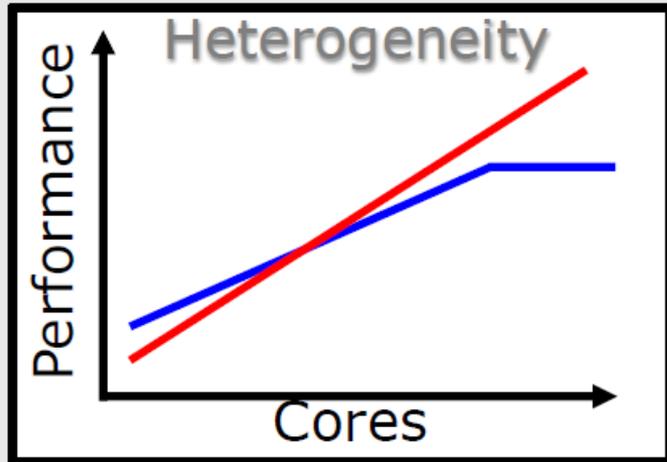
# Performance Depends on



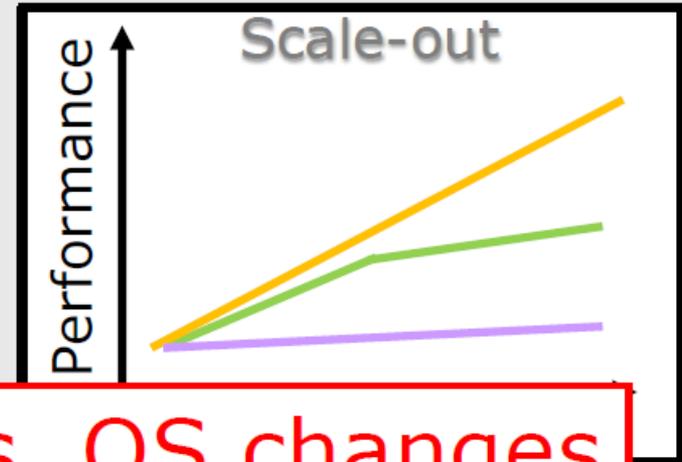
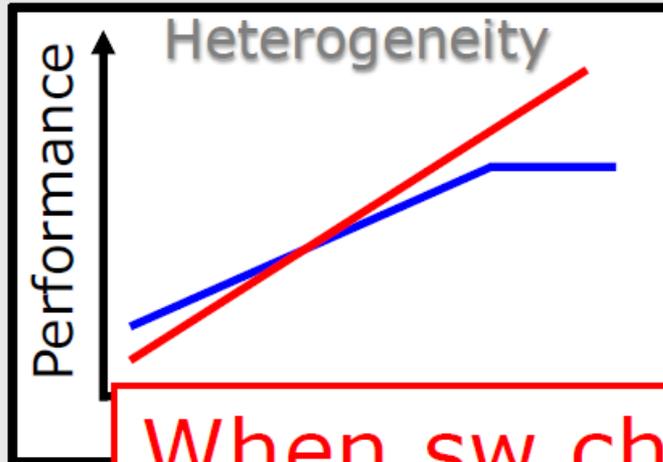
# Performance Depends on



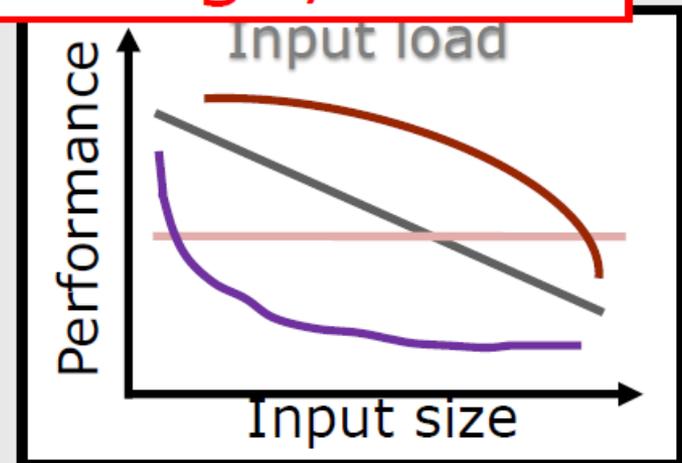
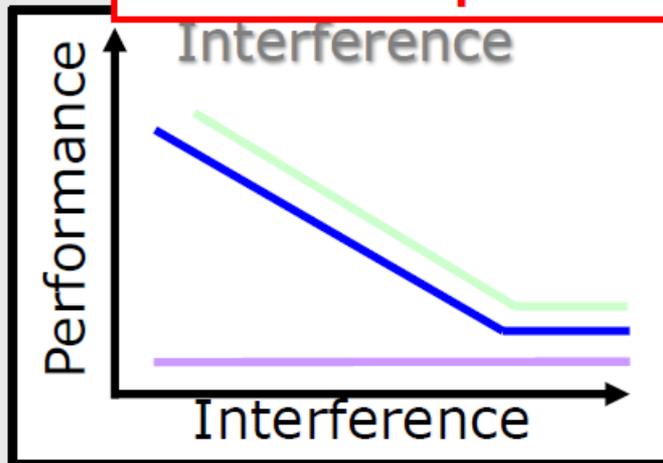
# Performance Depends on



# Performance Depends on



When sw changes, OS changes when platforms change, etc.

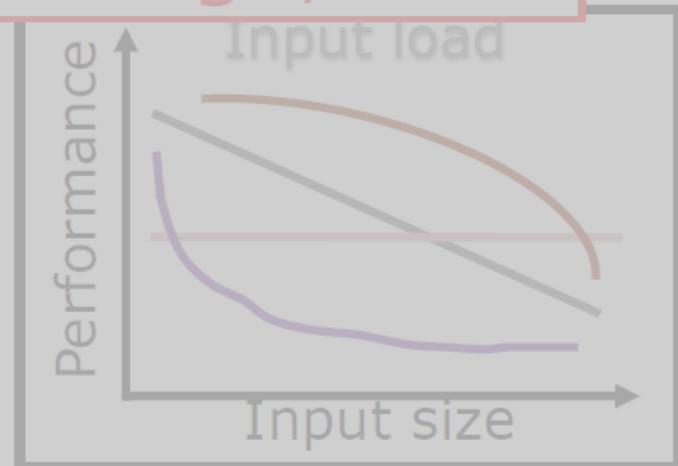


# Performance Depends on



**Overprovision Reservations!**

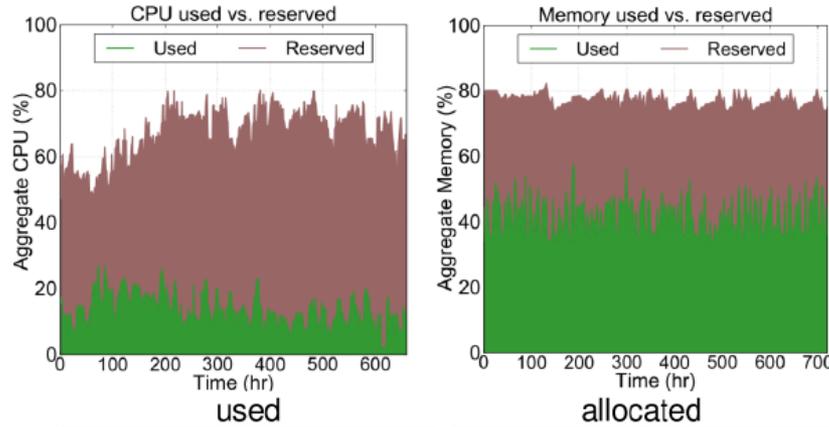
changes,  
platforms change, etc.



# Overprovisioning & Resource Usage

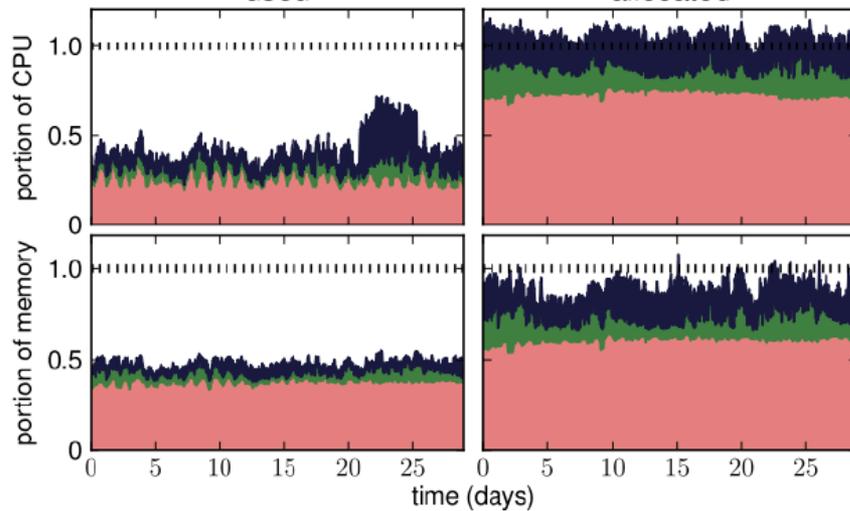


Twitter



[ASPLOS'14]

Google



Production – high priority  
Medium priority  
Gratis

[Reiss'12]

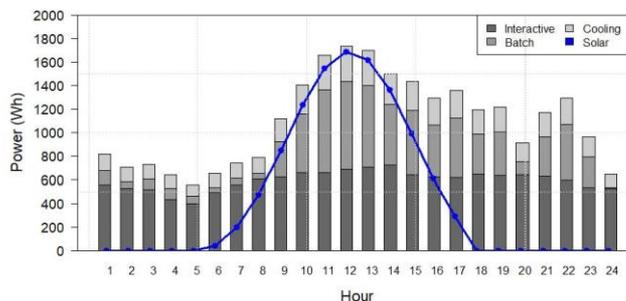
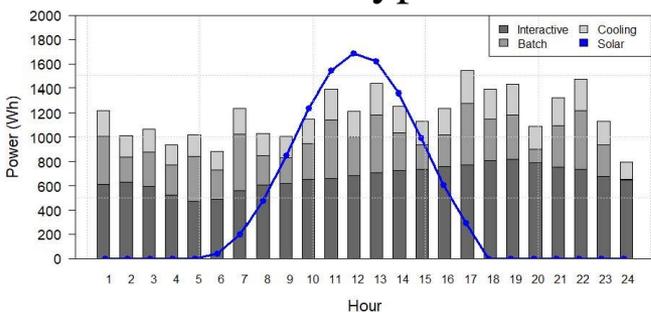
# A Self-adaptive Approach for Managing Co-located Workloads



- Providing a perspective model for multi-level adaptive resource scheduling to manage workloads and renewable energy
- Proposing a self-adaptive approach for interactive workloads and batch workloads to ensure their QoS
- Workloads Model

$$d(t) = \sum_m a_m(t) + \sum_n b_n(t)$$

- Two types of workloads:

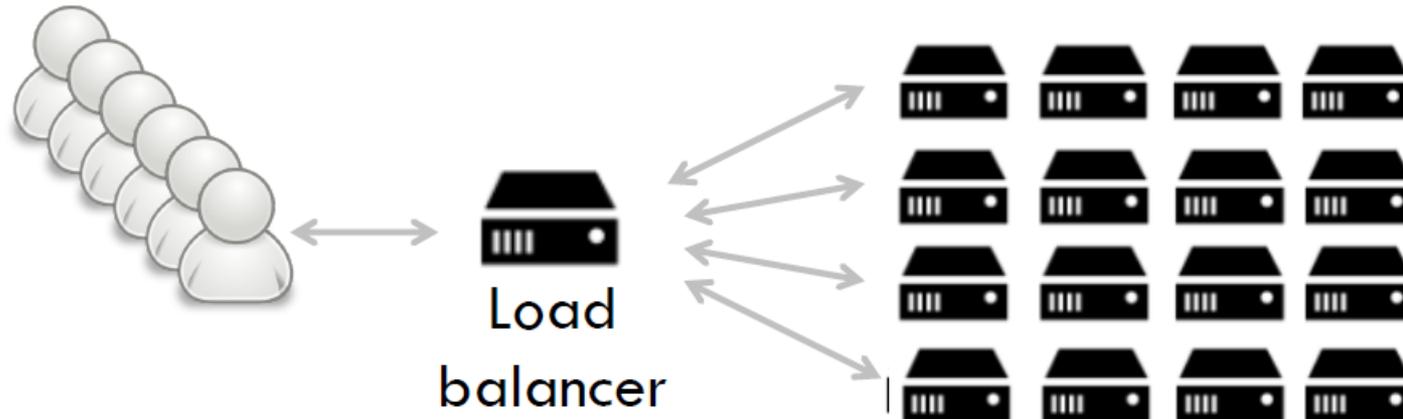


VM consolidation + host scaling + green-aware scheduling

M. Xu et al., A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing, *IEEE Transactions on Sustainable Computing (TSUSC)*, Vol. 6, No.4, 2021

# Autoscaling

- Monitor app performance or server load
  - [AWS AutoScale, Google Autopilot...]
- Adjust resources given to app
  - Add or remove to meet performance goal
  - Feedback-based control loop



- Tradeoffs
  - Monitor performance or resource utilization?
  - Key parameters for controller design?
  - Autoscale within (scale-up) or across servers (scale-out)?
  - Reactive or predictive control?
  
- Other complications?

# Load Prediction

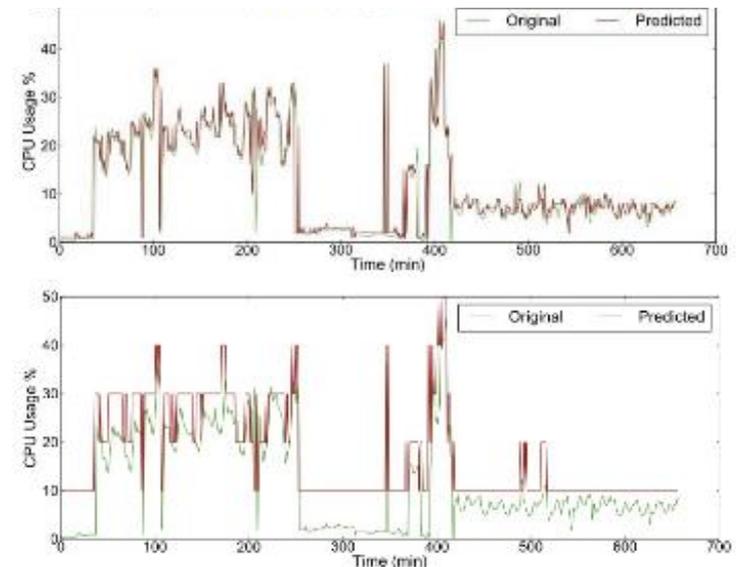
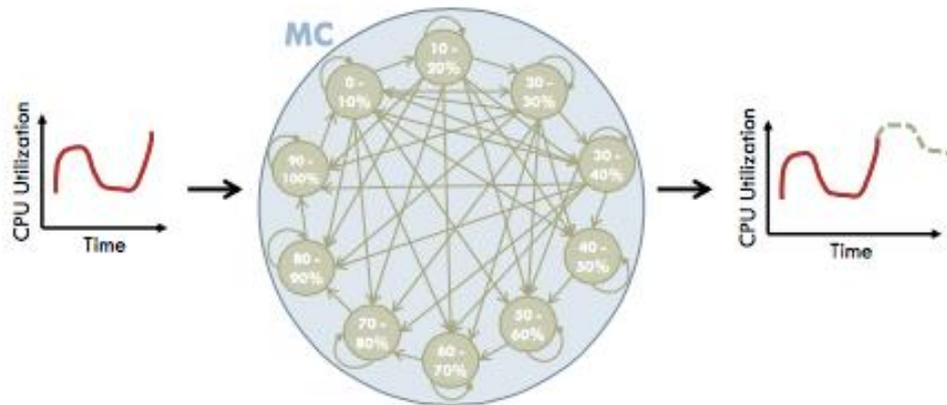


- Predict load or resource usage
  - Key in avoiding excess capacity and QoS violations
- How far ahead do you need to predict?
- Example prediction techniques
  - Moving window average
  - Exponentially weighted average
    - Predict actual load or change rate
  - Markov-chain predictor
  - Wavelet transform predictor
  - Classification predictor
  - Neural Network predictor

# Markov-chain predictor



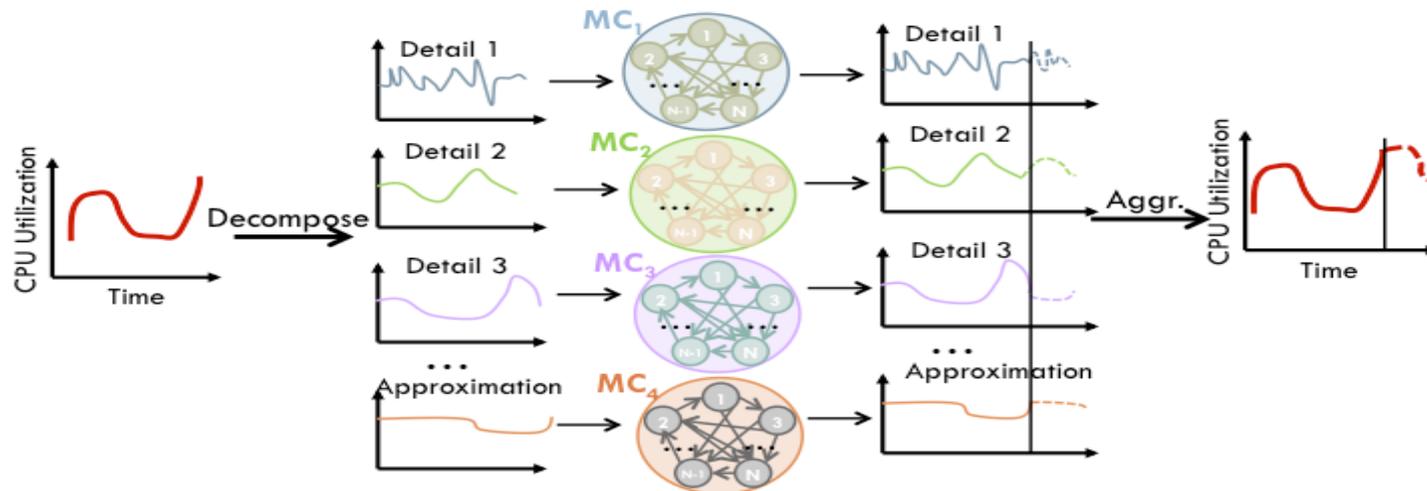
- Train state transitions on a moving window
  - Use to predict one window into the future
  - Number of states determines prediction resolution
- Fast and reasonably accurate



# Wavelet Transform Prediction



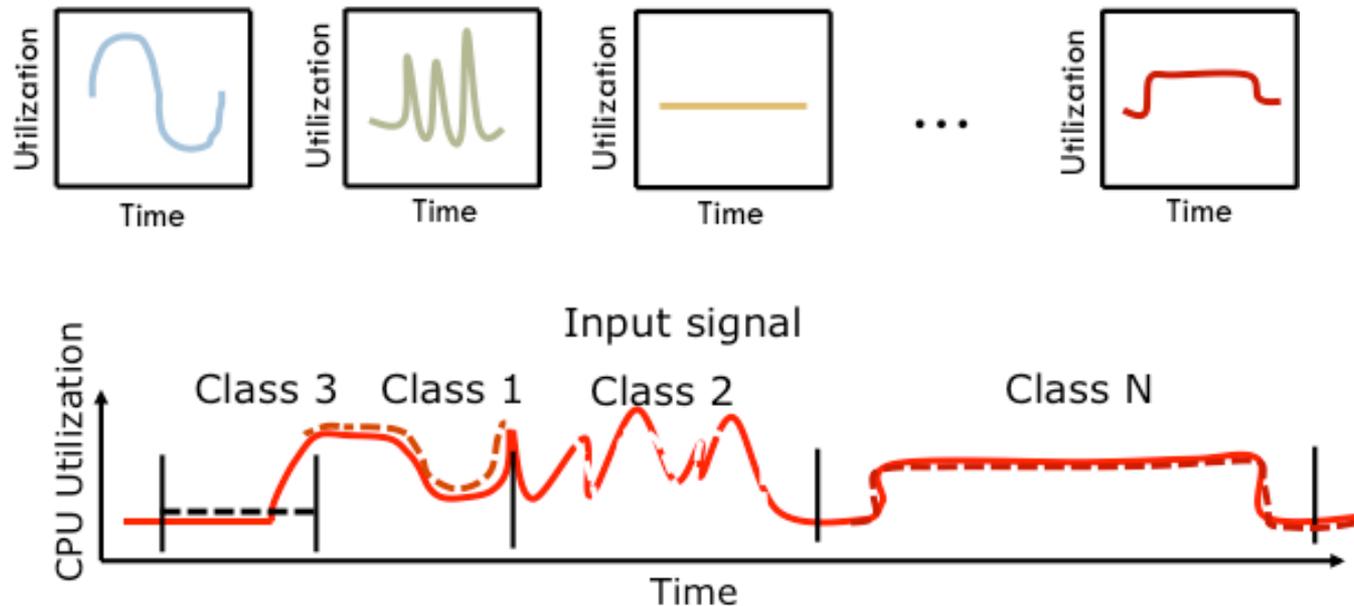
- Decompose signal into Daubechies wavelets
  - Individual Markov chains for each wavelet
  - Aggregate individual predicted wavelets → predicted signal
  - Can predict further into the future



# Classification Predictor



- Start with key load patterns (e.g., sinusoidal, spike, flat)
  - Determine which pattern fits best a moving window
- Can predict unexpected events
- Can predict further into the future

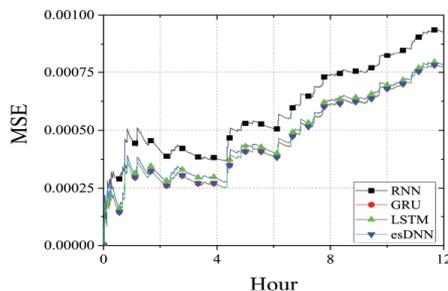
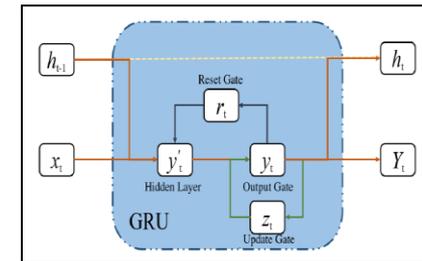


# Neural Network predictor

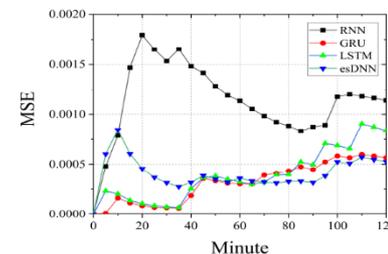
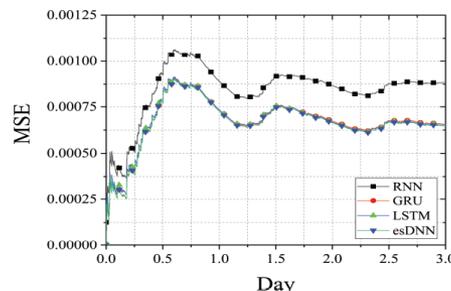


## ■ Supervised Learning + Gated Recurrent Unit

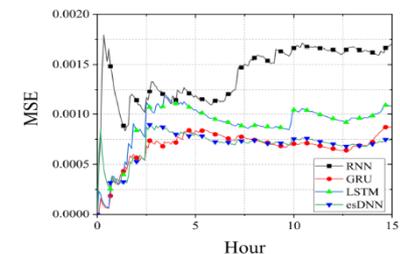
- Keep as much information as possible compared with auto-encoder and PCA
- Overcome existing limitations of gradient disappearance and explosion of LSTM and RNN
- Efficient and high accurate



**esDNN MSE with Alibaba: <0.001**



**esDNN MSE with Google: <0.002**

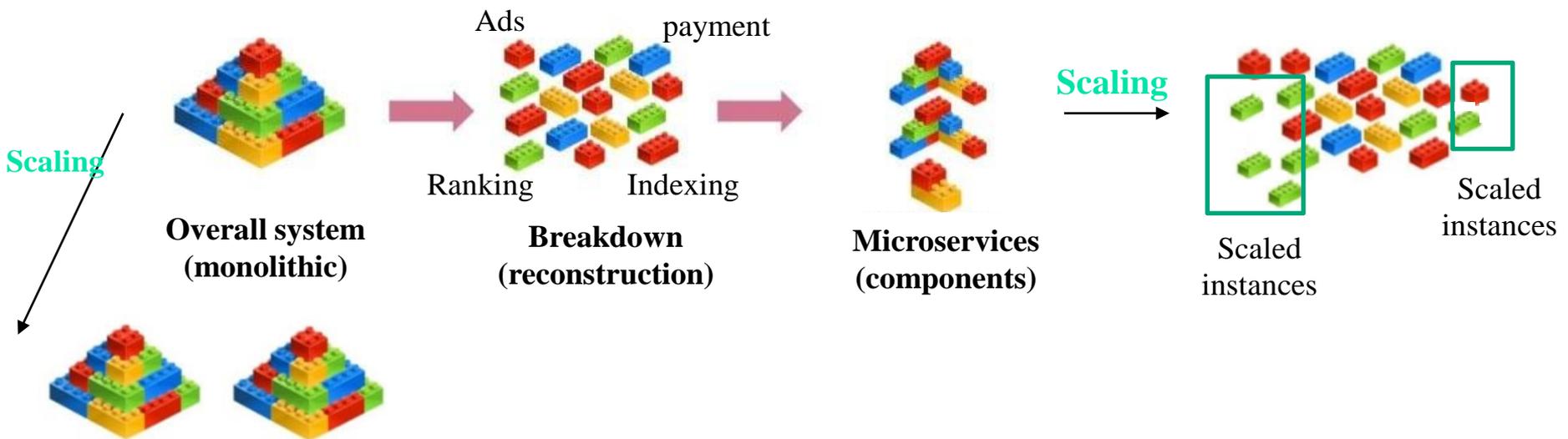


M. Xu et al., esDNN: An Efficient Approach for Multivariate Workload Prediction in Cloud Computing Environments, *ACM Transactions on Internet Technology (TOIT)*, Vol. 22, Issue 3, 2022.

# Autoscaling with microservices



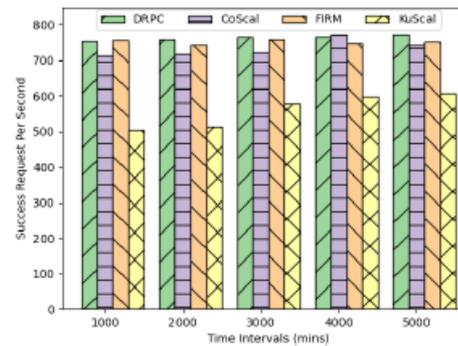
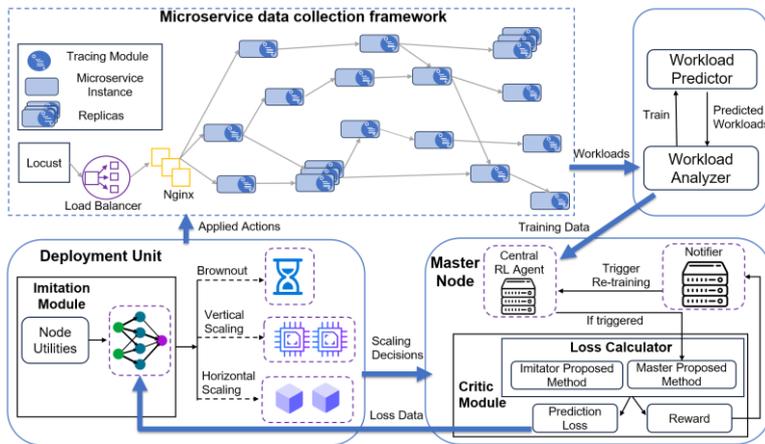
- Microservice decomposes applications into a group of **lightweight, fine-grained and self-contained** units/components
- Vertical scaling
  - Increase or decrease the capacity of a single instance/pod
- Horizontal scaling
  - Adding or removing instances of a resource



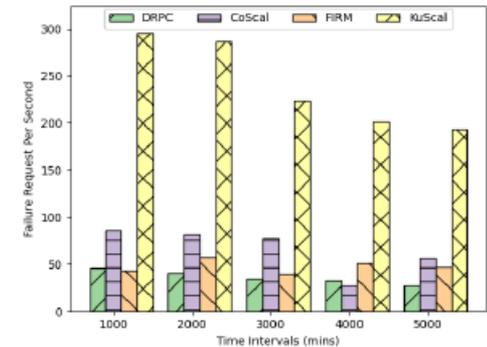
# Multi-faceted Scaling of Microservices with RL



- A performance analysis of horizontal scaling, vertical scaling, and brownout for microservices to investigate trade-offs.
- A reinforcement learning (TD3) based scaling algorithm for decision making to optimize the performance of microservices.
- CoScal reduces response time by 19%-29% and reduce failures significantly



(a) Number of success requests



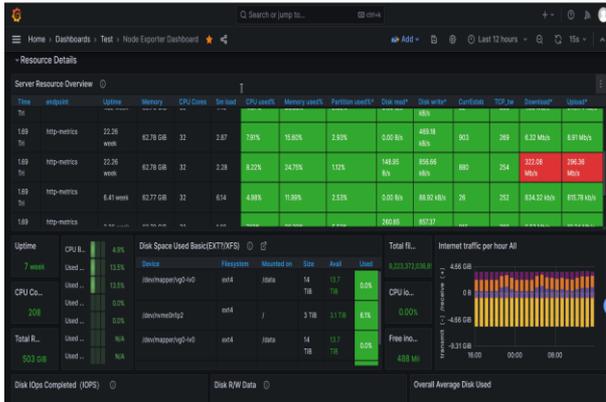
(b) Number of failure requests

M. Xu et al., DRPC: Distributed Reinforcement Learning Approach for Scalable Resource Provisioning in Container-based Clusters, *IEEE Transactions on Service Computing (TSC)*, 2024

# Prototype System



- Prototype systems have been awarded with:
  - 2022 and 2023 China Hi-Tech Fair (中国高交会) “**Outstanding Product Award**”
  - Open sourced at:
    - <https://gitee.com/siat-minxian-group/micro-auto-scaler>
    - <https://gitee.com/siat-minxian-group/es-dnn>
    - <https://gitee.com/siat-minxian-group/cloud-native-sim>



### MicroAutoScaler

MicroAutoScaler微服务弹性伸缩和自动扩展平台是由中国科学院深圳先进技术研究院云计算研究中心发起开发，旨在简化微服务应用程序运维和资源管理的平台。通过自动化伸缩应对不断变化的工作负载，提高性能，降低成本，并确保应用程序的可用性和稳定性。它是现代云原生架构的重要补充，可以显著简化微服务的部署和运维工作。该平台基于团队前期自研的基于状态感知的资源调度器StatuScale，涵盖数据收集器、状态检测器、资源调度器、反馈调节器等主要模块。

- 项目发起人: 徐敏姿 (中国科学院深圳先进技术研究院)
- 项目参与:
  - 温林峰 (中国科学院深圳先进技术研究院)
  - 吴静峰 (中国科学院深圳先进技术研究院)

### 系统组件

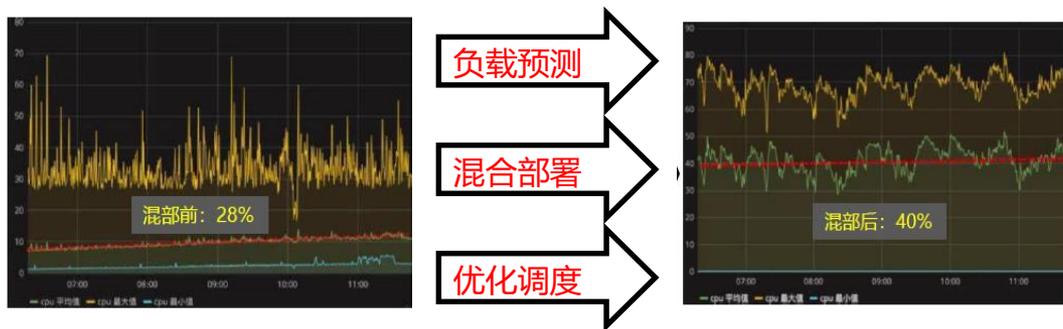
- Infrastructure of Cloud.
- Cloud resource management.
- Cloud native platforms.
- Qos suites.
- System maintenance, tools and scripts.



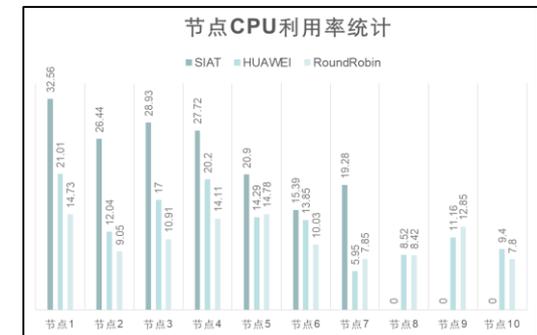
# Validations in Alibaba and Huawei



- Alibaba
  - Transferred the algorithms to Alibaba to improve resource utilization from 28% to 40%
- Huawei
  - Recently, applied successfully to 华为消费者云, improve utilization by 92%, saving 75.9% machines !



Validation in Alibaba



Validation in Huawei





# The Sources of Trouble



- HW heterogeneity
  - Different server generations and configurations
- Interference
  - In shared resources between co-scheduled applications
- Variability
  - Apps phases, datasets variability, user load variability
- Unknown apps
  - Especially in public clouds

# Recent Research Ideas in Cluster Management

---



- Goal:
  - Can we improve resource efficiency while preserving application QoS guarantees?
  - Potential: 3-5x efficiency; \$10Ms in cost savings

# Requirements



- Automate resource management
  - Large, multi-dimensional space → Leverage big data
- General solution
  - Different application types (batch, latency-critical)
  - Different types of hardware
- Cross-layer design
  - Architecture → OS → Scheduler → Application design

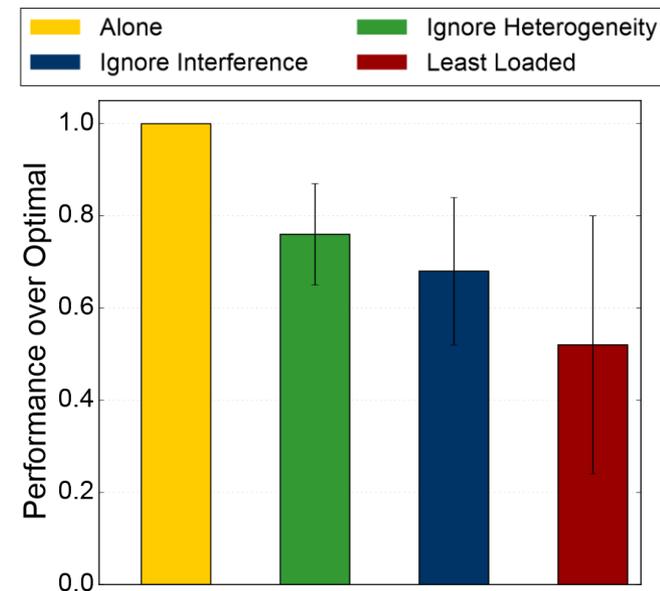
# Heterogeneity & Interference Matter

## ■ Heterogeneity

- DCs provisioned over 15 years
- Multiple server generations & configurations

## ■ Interference

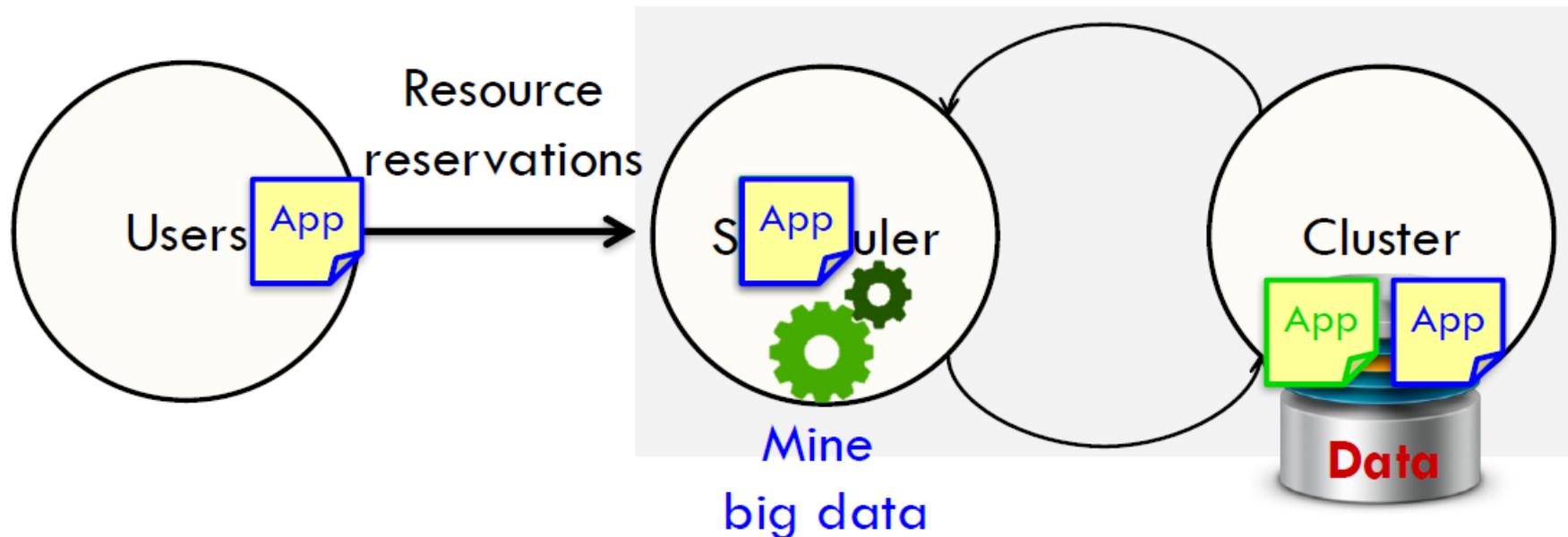
- Apps contend on shared resources
  - CPU & cache hierarchy
  - Memory system
  - Storage & network I/O



# Extracting Resource Preferences



- Naïve: exhaustive characterization
  - ~10-20 platforms x 1,000 apps
- Looks like a recommendation problem

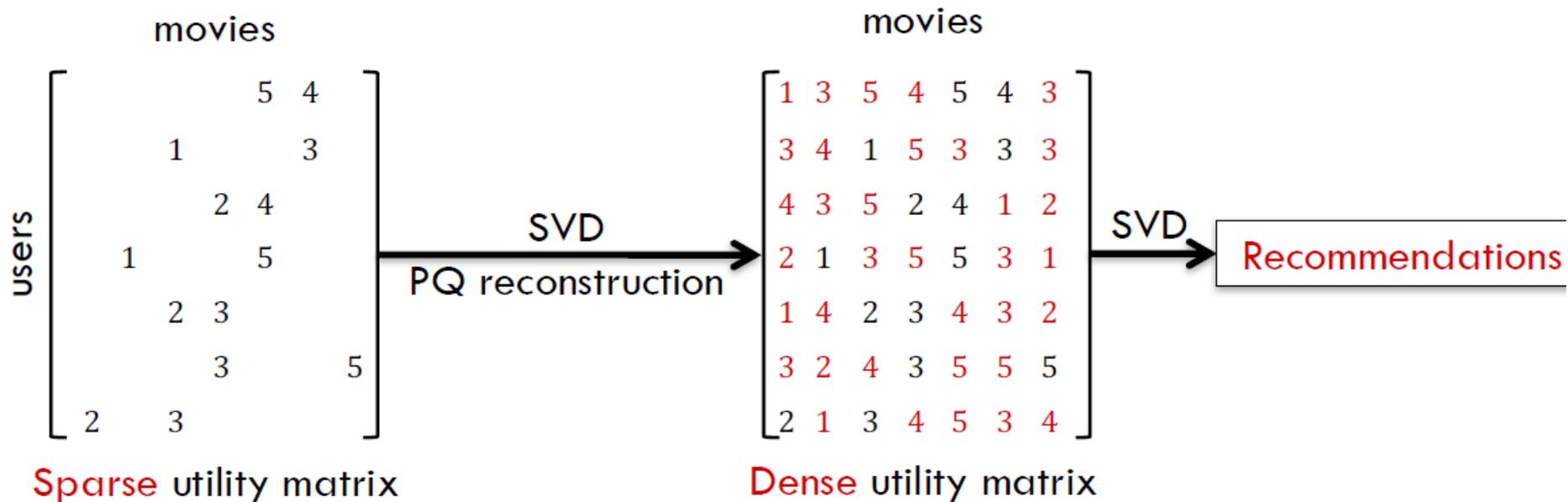


- **Content-based systems:**
  - Description of items (keywords, feature vector, etc. )
  - Profile of user preferences (history, model, user-system interaction, etc.)
- **Collaborative filtering:**
  - Uncover similarities between users and items
  - No need to know item features or explicit user preferences in advance

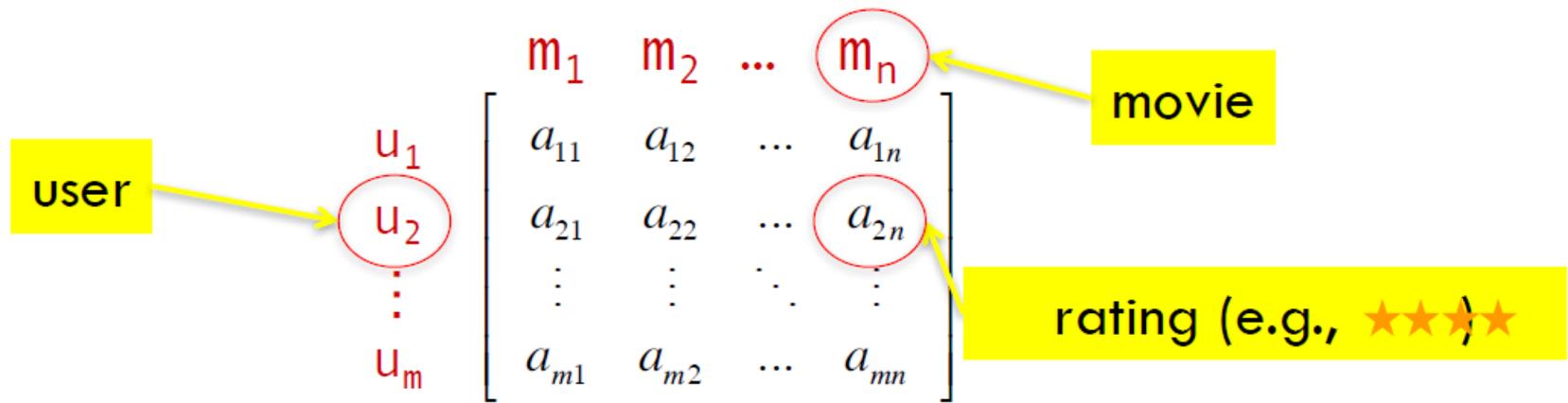
# Something familiar...



- Collaborative filtering – similar to Netflix Challenge system
  - Singular Value Decomposition (SVD) + PQ reconstruction (SGD)

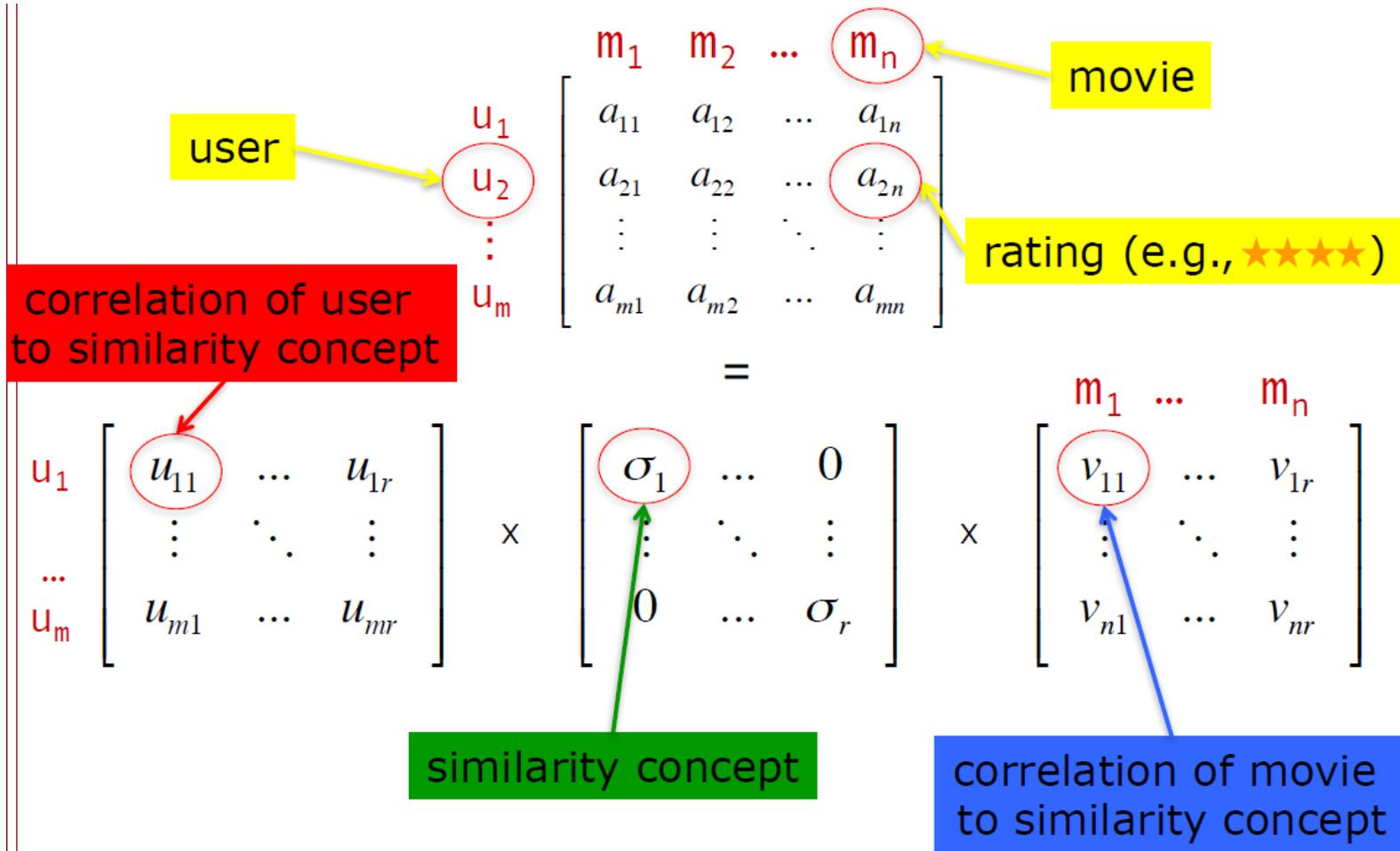


# SVD



$$\begin{matrix}
 u_1 \\
 \dots \\
 u_m
 \end{matrix}
 \begin{bmatrix}
 u_{11} & \dots & u_{1r} \\
 \vdots & \ddots & \vdots \\
 u_{m1} & \dots & u_{mr}
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \sigma_1 & \dots & 0 \\
 \vdots & \ddots & \vdots \\
 0 & \dots & \sigma_r
 \end{bmatrix}
 \times
 \begin{matrix}
 m_1 & \dots & m_n \\
 v_{11} & \dots & v_{1r} \\
 \vdots & \ddots & \vdots \\
 v_{n1} & \dots & v_{nr}
 \end{matrix}$$

# SVD



# Heterogeneity Classification



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	...	Movie M
User A	★★★★			★★★★★			
User B			★★		★★★		
⋮							
User N		★★★★					★

# Heterogeneity Classification



# Heterogeneity Classification



# Heterogeneity Classification



	Platform 1	Platform 2	Platform 3	Platform 4	Platform 5	...	Platform M
App A	1,500QPS			843QPS			
App B			458QPS		946QPS		
⋮							
App N		1,016QPS					186QPS

App performance

# Heterogeneity Classification



	Platform 1	Platform 2	Platform 3	Platform 4	Platform 5	...	Platform M
App A	1,500QPS			843QPS			
App B							
⋮							
App N							

Profiled Performance

Inferred Performance

# Heterogeneity Classification



	Platform 1	Platform 2	Platform 3	Platform 4	Platform 5	...	Platform M
App A	1,500QPS	843QPS	675QPS	843QPS	1,786QPS	...	8,675QPS
App B							
⋮							
App N							

Profiled Performance

Inferred Performance

# Heterogeneity Classification



	Platform 1	Platform 2	Platform 3	Platform 4	Platform 5	...	Platform M
App A	1,500QPS	843QPS	675QPS	843QPS	1,786QPS	...	8,675QPS
App B	987QPS	458QPS	773QPS	1,073QPS	986QPS	...	1,836QPS
⋮							
App N							

Profiled Performance

Inferred Performance

# Heterogeneity Classification



	Platform 1	Platform 2	Platform 3	Platform 4	Platform 5	...	Platform M
App A	1,500QPS	843QPS	675QPS	843QPS	1,786QPS	...	8,675QPS
App B	987QPS	458QPS	773QPS	1,073QPS	986QPS	...	1,836QPS
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
App N	9,893QPS	7,686QPS	786QPS	1,118QPS	997QPS	...	1,354QPS

Performance depends on app type:  
QPS, completion time, IPC, ...

Profiled Performance

Inferred Performance

# Interference Classification



	L1-i \$	LLC	Mem bw	CPU Int	I/O bw	...	Net bw
App A	95	81	7	56	43	...	100
App B	92	4	14	18	81	...	78
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
App N	45	49	56	11	99	...	54

Profiled Sensitivity

Inferred Sensitivity

- Toolkit tutorial: Reading after class

# Automation with Ansible

Practice with this tool!



openstack®



nectar

- Deploying complex cloud systems requires a lot of moving parts
  - Easy to forget what software you installed, and what steps you took to configure the system
  - Manual process is error-prone, can be non-repeatable
  - Snapshots are monolithic - provide no record of what has changed
- Automation
  - Provides a record of what you did
  - Codifies knowledge about the system
  - Makes process repeatable
  - Makes it programmable – “Infrastructure as Code”

## ■ Cloud-focused

Used to interact with Cloud services.

- Apache JClouds (Java-based - supports multiple clouds)
- Boto (Python - supports AWS and OpenStack)
- OpenStackClient (Python - supports OpenStack)
- CloudFormation (YAML/JSON - supports AWS, OpenStack Heat)

## ■ Shell scripts

- Bash
- Perl

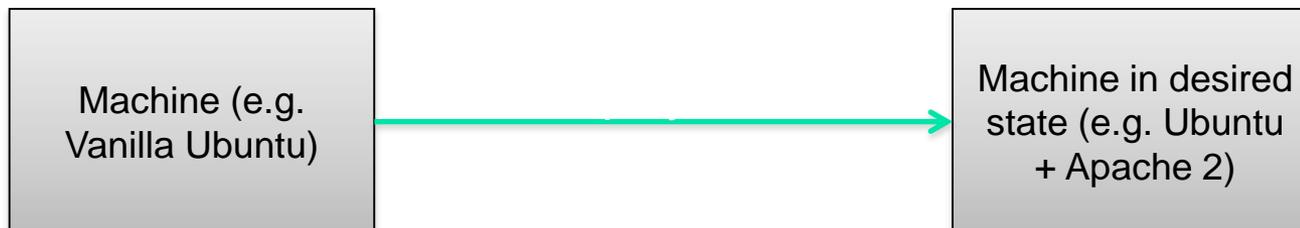
## ■ Configuration management (CM) tools

Configuration management refers to the process of *systematically* handling *changes* to a system in a way that it *maintains integrity* over time.

Automation is the mechanism used to make servers reach a desirable state, previously defined by provisioning scripts using tool-specific languages and features.

- Chef (uses Ruby for creating cookbooks)
- Puppet (uses its own configuration language)
- **Ansible (use YAML to express playbooks)**
- Fabric (Python library that uses SSH for application deployment and administration tasks)
- Terraform, SaltStack, Docker, ...

- An automation tool for configuring and managing computers
  - Finer grained set up and configuration of software packages
- Initial release: Feb. 2012
- Combines multi-node software deployment
- Ad-hoc task execution and configuration management
  - Configuring thousands of machines manually!?



# Ansible: Features



- **Easy to learn**
  - Playbooks in YAML, templates in Jinja2 etc.
  - Sequential execution
- **Minimal requirements**
  - No need for centralized management servers/daemons
  - Single command to install (*pip install ansible*)
  - Uses SSH to connect to target machine
- **Idempotent (repeatable)**
  - Executing N times no different to executing once
  - Prevents side-effects from re-running scripts
- **Extensible**
  - Write your own modules

- Supports push or pull

  - Push by default but can use cron job to make it pull

- Rolling updates

  - Useful for continuous deployment / zero downtime deployment

- Inventory management

  - Dynamic inventory from external data sources

  - Execute tasks against host patterns

- *Ansible Vault* for encrypted data

  - \$ ansible-vault create demo.yaml*

  - \$ ansible-vault decrypt demo.yaml*

  - \$ ansible-vault encrypt demo.yaml*

  - \$ ansible-vault rekey demo.yaml*

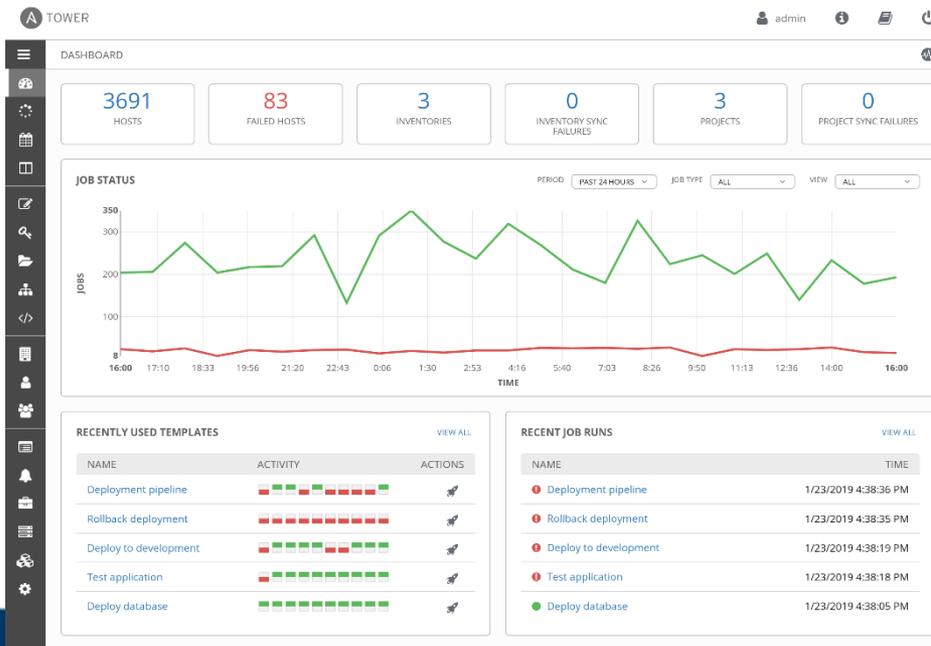
- Ad-hoc commands

Execute a one-off command against your inventory

```
$ ansible -i inventory_file -u ubuntu -m shell -a "reboot"
```

- Ansible Galaxy (<https://galaxy.ansible.com/>)

- Ansible Tower: Enterprise mission control for Ansible Dashboard, System Tracker, etc.



- Ansible Playbooks are expressed in YAML.
  - YAML: **Y**AML **A**in't **M**arkup **L**anguage
  - YAML is a human friendly data serialization standard for all programming languages.
  - YAML Syntax:  
[https://docs.ansible.com/ansible/latest/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html)
- Ansible uses Jinja2 templating for dynamic expression
  - Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates.
  - Jinja2 introduction:  
[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_templating.html#templating-jinja2](https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html#templating-jinja2)

## ■ Linux (Ubuntu)

```
$ sudo apt-get update && sudo apt-get install software-properties-common  
$ sudo apt-add-repository --yes --update ppa:ansible/ansible  
$ sudo apt-get install ansible
```

## ■ macOS

- Brew (<https://brew.sh/>)

```
$ brew install ansible
```

- Pip

```
$ sudo pip install ansible
```

## ■ Windows 10 (WSL)

- Install Windows Subsystem for Linux

<https://docs.microsoft.com/en-us/windows/wsl/install-win10>

- Install Ansible

[See guide for Linux \(Ubuntu\)](#)

## ■ Ansible documentation:

[https://docs.ansible.com/ansible/latest/installation\\_guide/intro\\_installation.html](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)

- Ansible scripts are called *playbooks*, written as simple YAML files
- Structured in a simple folder hierarchy

*Playbook folder*

```
| - variables  
|   | _vars.yml  
| - inventory  
|   | _inventory.ini  
| - roles  
|   | - defaults  
|   | - tasks  
|   |   | - task1.yml  
|   |   | _task2.yml  
|   | _templates / files  
| _playbook.yml
```

```
{  
  [webservers]  
  foo.example.com  
  128.250.0.1  
  
  [dbservers]  
  one.example.com  
  two.example.com  
}
```

## ■ Executed sequentially from a YAML file

- *hosts: webservers*

*vars:*

*package: httpd*

*tasks:*

- *name: Ensure the latest Apache is installed*

*apt:*

*name: “{{ packages }}”*

*state: latest*

- *name: Write the Apache config file*

*file:*

*src: /srv/httpd.conf*

*dest: /etc/httpd.conf*

- *name: Ensure Apache is restarted*

*service:*

*name: httpd*

*state: restarted*

{  
[webservers]  
www[01:50].example.com  
192.168.0.[1:254]  
}