Virtualization, Elasticity and Performance for Distributed Applications

Hong-Linh Truong
Distributed Systems Group, TU Wien

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong
@linhsolar
What this lecture is about?

- Resources and their impact on distributed systems and applications
- Virtualization
  - Resource virtualization
- Elasticity
  - Key concepts and techniques
- Performance
  - Utilizing virtualization and elasticity for some performance patterns
Some questions for DevOps

- How to have a development environment that is similar to the operational one?
- How to utilize computing resources in the best way?
- How to achieve the best performance?

Types of distributed applications

- Workflow/process style
- Data-centric pipeline style

Figure sources: http://www.cloudcomputingpatterns.org/Distributed_Application
Recall – Breakdown the complexity

Figure source: Sam Newman, Building Microservices, 2015

How to make sure that the underlying resources and infrastructures are suitable for „small autonomous services”?
Concepts of today‘s lecture

Virtualization

Elasticity

Performance
What is virtualization? A bird view

- Virtualization:
  - To abstract low-level compute, data and network resources to create \textit{virtual version} of these resources
  - Virtualization software creates and manages “virtual resources” isolated from physical resources

→ Virtualization is a powerful concept: we can apply virtualization techniques virtually for everything!

→ Virtualization is a key enabling technology for cloud computing and modern computer networks.
Virtualizing physical resources

So if we just develop „Web services“, why is it important to us?
Main types of virtualization of infrastructures for distributed apps

- **Compute resource virtualization**
  - Compute resources: CPU, memory, I/O, etc.
  - To provide virtual resources for „virtual machines“

- **Storage virtualization**
  - Resources: storage devices, harddisk, etc.
  - To optimize the usage and management of data storage

- **Network Function Virtualization**
  - Network resources: network equipment & functions
  - To consolidate network equipment and dynamically provision and manage network functions
Compute Resource Virtualization Technologies

- Physical compute resources:
  - Individual physical hosts/servers (CPU, memory, I/O),
  - Clusters and data centers
- At the low-level: two main streams
  - Hypervisor/Virtual Machine monitor
    - Virtual machines (VirtualBox, VMWare, Zen, etc.)
  - Containerization
    - Containers (Linux Containers, Docker, Warden Container, OpenVZ, etc.)
Hypervisor/Virtual Machine Monitor

Another model (Hypervisor level 1)

https://www.citrix.de/products/xenserver/tech-info.html
Containers

We do not dig into low-level techniques in virtualization, but examine

- How would virtualization techniques enable us to acquire, utilize and manage resources for our Devs and Ops of distributed applications and systems?

- How would such techniques change our software design?

- How to align resources/infrastructures with software using them
Virtual machines versus containers

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Virtual Machines</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS</td>
<td>Each VM runs on virtual hardware and Kernel is loaded into its own memory region</td>
<td>All the guests share same OS and Kernel. Kernel image is loaded into the physical memory</td>
</tr>
<tr>
<td>Communication</td>
<td>Will be through Ethernet Devices</td>
<td>Standard IPC mechanisms like Signals, pipes, sockets etc.</td>
</tr>
<tr>
<td>Security</td>
<td>Depends on the implementation of Hypervisor</td>
<td>Mandatory access control can be leveraged</td>
</tr>
<tr>
<td>Performance</td>
<td>Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS.</td>
<td>Containers provide near native performance as compared to the underlying Host OS.</td>
</tr>
<tr>
<td>Isolation</td>
<td>Sharing libraries, files etc between guests and between guests hosts not possible.</td>
<td>Subdirectories can be transparently mounted and can be shared.</td>
</tr>
<tr>
<td>Startup time</td>
<td>VMs take a few mins to boot up</td>
<td>Containers can be booted up in a few secs as compared to VMs.</td>
</tr>
<tr>
<td>Storage</td>
<td>VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run</td>
<td>Containers take lower amount of storage as the base OS is shared.</td>
</tr>
</tbody>
</table>


TABLE I
VM AND CONTAINER FEATURE COMPARISION
VM versus containers

Source: Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio:
An updated performance comparison of virtual machines and Linux containers. ISPASS 2015: 171-172
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7095802

Fig. 6. Random I/O throughput (IOPS).

Fig. 11. MySQL throughput (transactions/s) vs. CPU utilization.

Fig. 12. MySQL latency (in ms) vs. concurrency.
Examples of performance

Fig. 8. Evaluation of NoSQL Redis performance (requests/s) on multiple deployment scenarios. Each data point is the arithmetic mean obtained from 10 runs.

Fig. 10. MySQL throughput (transactions/s) vs. concurrency.

Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio: An updated performance comparison of virtual machines and Linux containers. ISPASS 2015: 171-172
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7095802
Tools, frameworks and providers: Chef, Vagrant, Amazon, Google, Microsoft, OpenStack, …
Interactions in VMs/containers provisioning and management

You focus on application development, how does it impact your work?
Examples

Source: https://docs.docker.com/engine/understanding-docker/
Examples

IBM Cloud OpenStack Services runs on OpenStack Icehouse to provide you with an environment built on the most current open standards.

Other OpenStack components included:
- Heat – for pattern orchestration
- Ceilometer – for reporting, metering

OpenStack experimental projects are not enabled by default:
- Trove
- Sahara

Source: http://www.slideshare.net/OpenStack_Online/ibm-cloud-open-stack-services
Cluster of VMs/containers

https://github.com/kubernetes/kubernetes/blob/release-1.2/docs/design/architecture.md
Virtual data centers

- On-demand virtual data centers
  - Compute nodes, storage, communication, etc.
  - Virtual data centers work like a single distributed system (e.g., a cluster)

- Challenges
  - Provision resources/nodes (using VMs or containers)
  - Configure networks within virtual data centers
  - Configure networks between virtual data centers and the outside systems
  - Deploy software into the virtual data centers
Example - Weave Net and docker

- Work with Kubernetes & Mesos as well
- Key idea: using network plug-in for containers + P2P overlay of routers in the host

Source: https://www.weave.works/docs/net/latest/introducing-weave/
Example -- DC/OS

Source: https://docs.mesosphere.com/1.8/overview/architecture/
Storage Virtualization

- Low-level storage
  - e.g., RAID (redundant array of independent disks)
- High-level, e.g., database
  - MySQL Cluster + auto-sharding

Why is it relevant to you?
What changes should we make in our apps?

Source: https://www.vmware.com/pdf/vi_architecture_wp.pdf
Is it the sysadmin task? I never see the network part in my apps. So why is it relevant to the software developer?
Why is resource virtualization interesting for distributed applications?

What are impacts of virtualization on the development and operation of distributed applications?
List of why and impact

- Server consolidation
  - Consolidating compute capabilities
- Security, fault tolerance and performance
  - Through dynamic provisioning and auto-scaling
- Cost/optimization
  - elasticity, hot deployment, etc.
- Compatibility issues
- DevOps
  - Closing the gap between real and development environments
Server Consolidation

- Cost, complexity (management)
  - Infrastructures (cooling, spaces), human resources
- Resources under utilization
How does it help me? Consolidation looks good for the sysadmin but not relevant to the software developer? What changes the developer has to do?
Microservices + partitioning

- Partition complex code into different services → easy configuration and maintenance
- But this has to be in sync with underlying resources provisioning
Security improvement

(Virtual) server and service isolation

Service instance 1
(user A)

Operating System

Physical Machine

Virtual Machine

Service 1

Operating System

Physical Machine

Service instance 2
(user B)

Operating System

Physical Machine

Virtual Machine

Service 2

Operating System
Fault tolerance and performance

How does resource virtualization help improving fault tolerance and performance?

- Possible benefits
  - Failure masking
  - Cost/optimization
    - Elasticity, hot deployment, etc.
    - Cloud bursting (combining private + public resources)
  - Improving service performance in incident management
    - E.g., spend time to fix a machine or just quickly relaunch a new one (and fix the old one later)?
Examples of cloud bursting/hybrid clouds

Bahman Javadi, Jemal Abawajy, Rajkumar Buyya, Failure-aware resource provisioning for hybrid Cloud infrastructure. Journal of Parallel and Distributed Computing, Volume 72, Issue 10, October 2012, Pages 1318-1331, ISSN 0743-7315,
Development and deployment

- Compatibility and support legacy application
- Maintenance
- Close the gap between development/test environment and real/production environments
- Simplify testing, emulating real environments, etc.
Elasticity in physics

“elasticity (or stretchiness) is the physical property of a material that returns to its original shape after the stress (e.g. external forces) that made it deform or distort is removed” – http://en.wikipedia.org/wiki/Elasticity_(physics)

- It is related to the form (the structure) of something
  - “Stress” causes the elasticity (structure deformation)
  - “Strain” measures what has been changed (amount of deformation)

- In the context of computing: given a process or a system
  - What can be used to represent “Stress” and “Strain”?
  - When does a “strain” signals a “dangerous situation”?
  - How to be elastic under dynamic “stress”?

DST 2017 37
Elasticity in computing

“Elastic computing is the use of computer resources which vary dynamically to meet a variable workload” –

“Clustering elasticity is the ease of adding or removing nodes from the distributed data store” –

“What elasticity means to cloud users is that they should design their applications to scale their resource requirements up and down whenever possible.“, David Chiu –
http://xrds.acm.org/article.cfm?aid=1734162
Elasticity in (big) data analytics

- More data → more compute resources (e.g. more VMs)
- More types of data → more activities → more analytics processes
- Change quality of analytics
  - Change quality of data
  - Change response time
  - Change cost
  - Change types of result (form of the data output, e.g. tree, table, story)
1. Demand elasticity
   Elastic demands from consumers

2. Output elasticity
   Multiple outputs with different price and quality

3. Input elasticity
   Elastic data inputs, e.g., deal with opportunistic data

4. Elastic pricing and quality models associated resources
Diverse types of elasticity requirements

- **Application user**: “If the cost is greater than 800 Euro, there should be a scale-in action for keeping costs in acceptable limits”

- **Software provider**: “Response time should be less than amount $X$ varying with the number of users.”

- **Developer**: “The result from the data analytics algorithm must reach a certain data accuracy under a cost constraint. I don’t care about how many resources should be used for executing this code.”

- **Cloud provider**: “When availability is higher than 99% for a period of time, and the cost is the same as for availability 80%, the cost should increase with 10%.”
Our focus in this course: elasticity of compute resources for distributed applications

Figure source: http://www.cloudcomputingpatterns.org/Distributed_Application

Q1: Where can elasticity play a role in these application models?

Q2: How does virtualization help implementing elasticity of resources
Elasticity implementation

- Elasticity specification
  - Constraints/Rules
- Elasticity monitoring and prediction
- Elasticity controller/adjustment:
  - Interpret specifications and monitoring data
  - Control
    - Reactive scale versus proactive scale
    - Vertical scaling (scale up/down) versus Horizontal scaling (scale out/in)
## Elasticity constraints

<table>
<thead>
<tr>
<th>Reference</th>
<th>Auto-scaling Techniques</th>
<th>H/V</th>
<th>R/P</th>
<th>Metric</th>
<th>Monitoring</th>
<th>SLA</th>
<th>Workloads</th>
<th>Experimental Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63]</td>
<td>Rules</td>
<td>Both R</td>
<td>CPU, memory, I/O</td>
<td>Custom tool, 1 minute</td>
<td>Response time</td>
<td>Synthetic Browsing and ordering behavior of customers.</td>
<td>Custom testbed (called UC Cloud) + TPC</td>
<td></td>
</tr>
<tr>
<td>[72]</td>
<td>Rules</td>
<td>H R</td>
<td>Average waiting time in queue, CPU load</td>
<td>Custom tool.</td>
<td>--</td>
<td>Synthetic</td>
<td>Public cloud, FutureGrid, Eucalyptus India cluster</td>
<td></td>
</tr>
<tr>
<td>[64]</td>
<td>Rules</td>
<td>Both R</td>
<td>CPU load, response time, network link load, jitter and delay</td>
<td>--</td>
<td>--</td>
<td>Only algorithm is described, no experimentation is carried out.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[48]</td>
<td>Rules + QT</td>
<td>H P</td>
<td>Request rate</td>
<td>Amazon CloudWatch, 1-5 minutes</td>
<td>Response time</td>
<td>Real Wikipedia traces</td>
<td>Real provider, Amazon EC2 + Httpperf + MediaWiki</td>
<td></td>
</tr>
<tr>
<td>[52]</td>
<td>RightScale + MA to performance metric</td>
<td>H R</td>
<td>Number of active sessions</td>
<td>Custom tool</td>
<td>--</td>
<td>Synthetic</td>
<td>Custom testbed, Xen + custom collaborative web application</td>
<td></td>
</tr>
<tr>
<td>[73]</td>
<td>RightScale + TS: LR and AR(1)</td>
<td>H R/P</td>
<td>Request rate, CPU load</td>
<td>Simulated</td>
<td>--</td>
<td>Synthetic Three traffic patterns: weekly oscillation, large spike and random</td>
<td>Custom simulator, tuned after some real experiments.</td>
<td></td>
</tr>
<tr>
<td>[59]</td>
<td>RightScale</td>
<td>H R</td>
<td>CPU load</td>
<td>Amazon CloudWatch</td>
<td>--</td>
<td>Real World Cup 98</td>
<td>Real provider, Amazon EC2 + RightScale (PaaS) + a simple web application</td>
<td></td>
</tr>
<tr>
<td>[96]</td>
<td>RightScale + Strategy-tree</td>
<td>H R</td>
<td>Number of sessions, CPU idle</td>
<td>Custom tool, 4 minutes</td>
<td>--</td>
<td>Real World Cup 98</td>
<td>Real provider, Amazon EC2 + RightScale (PaaS) + a simple web application</td>
<td></td>
</tr>
<tr>
<td>[81]</td>
<td>Rules</td>
<td>V R</td>
<td>CPU load, memory, bandwidth, storage</td>
<td>Simulated</td>
<td>--</td>
<td>Synthetic</td>
<td>Custom simulator, plus Java rule engine Drools</td>
<td></td>
</tr>
<tr>
<td>[77]</td>
<td>Rules</td>
<td>V R</td>
<td>CPU load</td>
<td>Simulated, 1 minute</td>
<td>Response time</td>
<td>Real ClarityNet</td>
<td>Custom simulator</td>
<td></td>
</tr>
</tbody>
</table>

Table rows are as follows: (1) The reference to the reviewed paper. (2) A short description of the proposed technique. (3) The type of auto-scaling: horizontal (H) or vertical (V). (4) The reactive (R) and/or proactive (P) nature of the proposal. (5) The performance metric or metrics driving auto-scaling. (6) The monitoring tool used to gather the metrics. The remaining three fields are related to the environment in which the technique is tested. (7) The metric used to verify SLA compliance. (8) The workload applied to the application managed by the auto-scaler. (9) The platform on which the technique is tested.


DST 2017
<xml>

<rules

  enabled="true">

  <constraintRules>

    <rule name="Default" description="Always active"
          enabled="true" rank="1">
      <actions>
        <range min="2" max="5" target="RoleA"/>
      </actions>
    </rule>

    <rule name="Peak" description="Active at peak times"
          enabled="true" rank="100">
      <actions>
        <range min="4" max="6" target="RoleA"/>
      </actions>
      <timetable startTime="08:00:00" duration="02:00:00">
        <daily/>
      </timetable>
    </rule>
  </constraintRules>

  <reactiveRules>

    <rule name="ScaleUp" description="Increases instance count"
          enabled="true" rank="10">
      <when>
        <greater operand="Avg_CPU_RoleA" than="80"/>
      </when>
      <actions>
        <scale target="RoleA" by="1"/>
      </actions>
    </rule>

    <rule name="ScaleDown" description="Decreases instance count"
          enabled="true" rank="10">
      <when>
        <less operand="Avg_CPU_RoleA" than="20"/>
      </when>
      <actions>
        <scale target="RoleA" by="-1"/>
      </actions>
    </rule>
  </reactiveRules>

  <operands>
    <performanceCounter alias="Avg_CPU_RoleA"
      performanceCounterName="\Processor(_Total)\% Processor Time"
      aggregate="Average" source="RoleA" timespan="00:45:00"/>
  </operands>

</rules>

</xml>

Source: https://msdn.microsoft.com/en-us/library/hh680881%28v=pandp.50%29.aspx
High level elasticity control in SYBL (http://tuwiendsg.github.io/iCOMOT/)

#SYBL.CloudServiceLevel
Cons1: CONSTRAINT responseTime < 5 ms
Cons2: CONSTRAINT responseTime < 10 ms
WHEN nbOfUsers > 10000
Str1: STRATEGY CASE fulfilled(Cons1) OR fulfilled(Cons2): minimize(cost)

#SYBL.ServiceUnitLevel
Str2: STRATEGY CASE ioCost < 3 Euro: maximize( dataFreshness )

#SYBL.CodeRegionLevel
Cons4: CONSTRAINT dataAccuracy>90% AND cost<4 Euro


A quick check: if you want to allow the developer to specify elasticity in his/her source code, e.g., Java, what would be your solution?
General concept: Lifecycle of applications and elasticity

Elasticity specification

Control processes

Control operation

Deployment process
Elasticity Prediction Function
Elasticity Adjustment Function
Elasticity Primitive Operations

Static Description
Runtime View 1 (Elasticity Space)
Runtime 2 (Elasticity Space)

Operation Time

Monitoring information

Requirement trigger
Process control
Behavior change

Cloud-specific Management Function specific APIs
VIRTUALIZATION AND ELASTICITY FOR IMPLEMENTING PERFORMANCE PATTERNS
Design for handling failures

- Resource failures
  - Problems with CPUs, networks, machines, etc.
  - → other dependent services failures
- **Scopes**: with an enterprise, within a data center, across multiple sites, across multiple infrastructures provided by different providers, etc.
- Our design must be ready to handle such failures
- Using virtualization and elasticity techniques to deal with issues
- Relying on best practices
Examples of best practices when using Amazon services

- Using Elastic IPs
- Utilize resources from multiple zones
- Maintain Amazon virtual machines
- Use Amazon Cloudwatch for monitoring
- Automatically make snapshots of VMs
- Automatically backups

Source: https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
Recall this case

Change the way to handle client requests outside the service and within the service
Which are possible solutions?

- Throttling
- Queue-based load leveling within the service
- Multiple instances and queues
- Multiple instances and elastic resources
- Circuit breaker to deal with failures
- You name it
Throttling

Disable too many access and disable unessential services

Code: http://www.django-rest-framework.org/api-guide/throttling/#how-throttling-is-determined
Example

Custom domain indicates the tenant for this user

http://surveys.adatum.com
Adatum
5 requests per second

http://surveys.fabrikam.com
Fabrikam
10 requests per second

http://surveys.contoso.com
Contoso
150 requests per second

Multi-tenant "Surveys" application

Meter the number of requests per second

Web role

Error: "Throttled"

Using tasks and queue-based load leveling pattern

Examples of queue-based load leveling pattern

Using multiple instances of services and queues

How do we control these instances in an efficient way?

Load balancing and elastic resources -- recall

Figure source: http://queue.acm.org/detail.cfm?id=1971597
Load balancing and elastic resources -- Concepts

- Using load balancer for a group of resources

Load balancer can monitor instances and send requests to healthy instances but what if we still need more instances?

- Auto-scaling

Examples

Amazon services

Google (from console.cloud.google.com)

They are programming tasks

Sources: http://docs.aws.amazon.com/autoscaling/latest/userguide/policycreating.html
# Examples from Amazon services

## Increase Group Size

<table>
<thead>
<tr>
<th>Name:</th>
<th>AddCapacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute policy when:</td>
<td>AddCapacityAlarm</td>
</tr>
<tr>
<td></td>
<td>Edit Remove</td>
</tr>
<tr>
<td></td>
<td>breaches the alarm threshold: CPUUtilization &gt;= 80 for 300 seconds</td>
</tr>
<tr>
<td></td>
<td>for the metric dimensions: AutoScalingGroupName = my-asg</td>
</tr>
<tr>
<td>Take the action:</td>
<td>Add ▼ 30 percent of group ▼ when 80 &lt;= CPUUtilization &lt; +Infinity</td>
</tr>
<tr>
<td>Add step ▼ 1</td>
<td></td>
</tr>
<tr>
<td>Instances need:</td>
<td>300 seconds to warm up after each step</td>
</tr>
</tbody>
</table>

Create a simple scaling policy ▼ 1

## Decrease Group Size

<table>
<thead>
<tr>
<th>Name:</th>
<th>DecreaseCapacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute policy when:</td>
<td>DecreaseCapacityAlarm</td>
</tr>
<tr>
<td></td>
<td>Edit Remove</td>
</tr>
<tr>
<td></td>
<td>breaches the alarm threshold: CPUUtilization &lt;= 40 for 300 seconds</td>
</tr>
<tr>
<td></td>
<td>for the metric dimensions: AutoScalingGroupName = my-asg</td>
</tr>
<tr>
<td>Take the action:</td>
<td>Remove ▼ 2 instances ▼ when 40 &gt;= CPUUtilization &gt; -Infinity</td>
</tr>
<tr>
<td>Add step ▼ 1</td>
<td></td>
</tr>
</tbody>
</table>

Create a simple scaling policy ▼ 1

```bash
aws autoscaling attach-load-balancers --auto-scaling-group-name my-asg --load-balancer-names my-lb
```

Sources: [http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html](http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html)
Circuit breaker pattern

- What if service operations fail due to unexpected problems or cascade failures (e.g. busy → timeout)
  - Let the client retry and serve their requests may not be good

→ Circuit breaker pattern prevents clients to retry an operation that would likely fail anyway and to detect when the operation failure is resolved.
Circuit breaker pattern

http://martinfowler.com/bliki/CircuitBreaker.html

Open Case Study for recap

- Multiple topics
- Amount of data per topic varies
- Should not have duplicate data in database

• Should I use docker? VMs?
• Where elasticity can be applied?
• Topic/data distribution to ingest clients?
Summary

- Modern distributed applications should consider underlying computing resources
  - Incorporate features to leverage virtualization and elasticity at runtime through programming tasks
- Elasticity and virtualization enable robust, efficient and reliable distributed applications
- They can also simplify the development and operation activities.
- Do exercises by examining examples in this lecture → e.g., providing your dockers for next year students
Further materials

- https://www.computer.org/web/the-clear-cloud/content?g=7477973&type=blogpost&urlTitle=performance-patterns-in-microservices-based-integrations


Thanks for your attention

Hong-Linh Truong
Distributed Systems Group, TU Wien
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong
@linhsolar