TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Benchmarking FAAS Applications

Fundamental Cloud Architectures

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

OBJECTIVES

- Tutorials
- Class Presentations 11/28, 12/3, 12/5
- Capstone/Thesis Presentations 12/5
- 12.20pm-2:40pm, WPH, Jane Russell Commons
- Final Term Project Presentation 12/12 – spec posted
- Final Term Project Report 12/14 – spec posted
- Benchmarking for Term Projects cont'd
- Fundamental Cloud Architectures (Ch. 11, Thomas Erl)

CLASS PRESENTATIONS

- **Wednesday November 28**
- 1. Team 6 (*Rahul, Poornima, Sowmya*) Azure Cosmo DB
- 2. Team 1 (*Tanner, Ali, Khanh*) AWS Cloud Formation

- **Monday December 3rd**
- 1. Team 2 (*Derek, Milad*) Paper:
  Serverless Computing: Design, Implementation, and Performance
- 2. Team 7 (*Xiaodong, Moran, Zac*) Google BigQuery
- 3. Team 3 (*Feng, Jiaqi, Xiaola*) Azure Functions

- **Wednesday December 5th**
- 1. Team 5 (*Robert C., Jared, Raymond*) Google Cloud Functions
- 2. Team 4 (*Robert B., Jeff, Daylen*) MongoDB Atlas

BENCHMARKING FAAS APPLICATIONS

SERVICE PERFORMANCE MEASUREMENT

- Average Turnaround Time
  - Client's perspective: time delta before call until result
  - Server's perspective: time delta from function entry point to end
- Compute time: CPU usage
- Measured on the server side – Java

```
long cputime0 = ManagementFactory.getThreadMXBean().
getThreadCpuTime(java.lang.Thread.currentThread().getId());
long cputime1 = ManagementFactory.getThreadMXBean().
getThreadCpuTime(java.lang.Thread.currentThread().getId());
Long cputimedelta = (cputime1-cputime0)/1000000;
```

- How do these times relate to billed function time in CloudWatch log messages?

SERVICE PERFORMANCE - 2

- Latency
  - Formally means time between request and response
  - Typical to qualify the type of latency: "network latency"
  - Time request/response message is in transit
  - Estimate: Client's Turnaround Time – Server's Turnaround Time
  - Difference estimates round trip latency (both ways)
  - Divide by two for estimate of one-way latency

SERVICE PERFORMANCE - 3

- **Latency cont'd**
  - Other approach: Network time protocol (NTP)
  - Service for synchronizing Linux system time
  - Synchronize VM times (EC2 instances) …*good for clients*
  - Research Question: How synchronized are AWS Lambda clocks?
  - With synchronized clocks, can capture system event times:
  - CLIENT_REQ_SENT, SERVER_REQ_RCVD *to server* →
  - SERVER_RESP_SENT, CLIENT_RESP_RCVD ← *from server*

SERVICE PERFORMANCE MEASUREMENT: SEQUENTIAL

- Measure performance behavior of standalone services
- Similar to stress testing
- Sequential tests: one client, repeat test many times (callservice.sh)
- Establishes how service performs running in one environment
  - One VM, one container, no scaling
- Takes longer to collect a lot of samples
- May be more consistent as a single environment may perform more consistently than many parallel environments

- Research Question: Which type of FAAS testing provides more stable results (sequential vs. parallel)?
  - Stability measured by: standard deviation, variance

SERVICE PERFORMANCE MEASUREMENT: PARALLEL

- Concurrent tests: many clients in parallel (partest.sh)
- Concurrent tests collect performance data for many deployments in parallel
- Supports collecting a lot of data, FAST!
- Samples how ***"provisioning variation"*** impacts performance
- Example: run 1 test, 100 times with short delay between tests
- Problem: Only measures one VM, one "container"
- Fix: Run 100 tests, 1 time in parallel
- Measures many VMs, and 100 "containers"…

- **Research Question**: How does provisioning variation of FAAS infrastructure impact service performance?

SERVICE PERFORMANCE MEASUREMENT: PARALLEL - 2

- Client must be capable of generating load
- All requests must overlap to force creation of infrastructure
- Approaches:
  - (1) Make service time long – can use a laptop for 100 requests
  - (2) Use a very powerful client machine – fast CPU & network
  - (3) Use synchronized clients – separate VMs with time synchronization (optional tutorial 9)
  - HYBRID- Do both…
- Run 10x-100x batches of 100 with short delay
  - **Research Question:** How does performance vary when running on one-set of infrastructure?
  - Measures warm performance

SERVICE PERFORMANCE MEASUREMENT: PARALLEL - 3

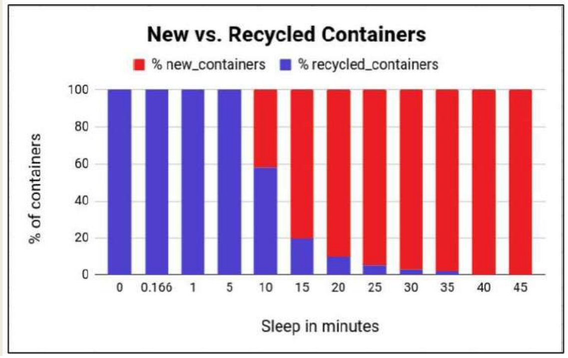
- Run 10x-100x batches of 100 with **LONG** delay…
  - Long delay is to ensure infrastructure goes COLD and is reprovisioned from scratch
  - Provides a realistic test
  - In the wild, functions will go dormant, and new infrastructure will be dynamically created on-the-fly
  - We are interested in understanding how performance might vary each time this happens
- Application based testing – AWS Lambda
  - Observed ~34% performance variance for various memory settings from 128MB to 512MB of different "generations" of infrastructure
  - An infrastructure generation is one set created in response to service demand

FORCING COLD INFRASTRUCTURE

- Possible approaches – best to worst

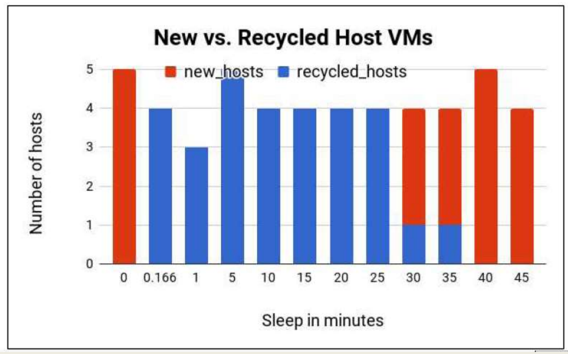
1. Wait ~45 minutes: all infrastructure (VMs & containers) are deprecated, new ones are created
2. Change VPCs / Availability Zones: forces function to be deployed to new location
   - Can run out of AZs
3. Change a parameter: (e.g. memory allocation, max runtime) – container is destroyed, but host/VM remains the same
   - Not a true cold performance test
4. Redeploy new version of code: container is destroyed (?), but host/VM remain the same
5. Larger parallel request: forces creation of new infrastructure
   - Old infrastructure remains

Lambda Container Recycling



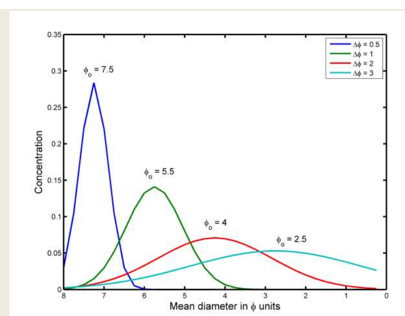
Tests from 2017

Lambda Virtual Machine Recycling



ESTABLISHING PERFORMANCE BEHAVIOR

- What does "average" look like for the service?
- One "sample" could be perfectly sunny, or very rainy
- How does service perform overall ?
- Many requests may have similar performance, a few are slow… *(skewed to the right-hand side of the graph)*



Performance may be "log normal"

November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.14

ESTABLISHING PERFORMANCE BEHAVIOR - 2

- Need to run multiple tests to "sample" how the system responds
- Goal: obtain performance measurements which compare <u>apples-to-apples</u> scenarios
- It is easy to find a pear…

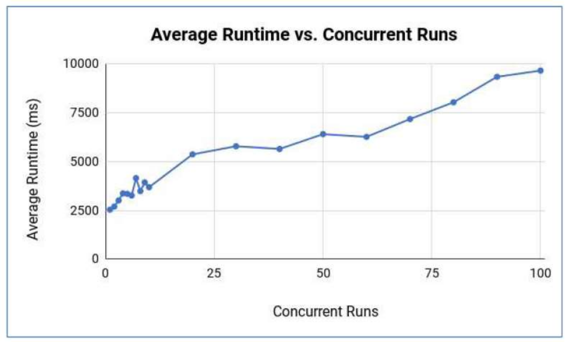
- <u>LAMBDA PEARS:</u>
- Must consider <u>state</u>: VM-cold, Container-cold, warm
- Must consider <u>server location</u>: which availability zone (AZ)?
  - Can "pin" functions to a specific AZ by running in a VPC
  - Use of VPCs add initialization overhead
  - Lambdas must negotiate private IP address on VPC (one time)

ESTABLISHING PERFORMANCE BEHAVIOR - 3

- <u>PEARS cont'd</u>
- <u>Client location</u> – Starbucks? At home? At UWT?
- Best client is an EC2 instance in an unchanging availability zone (AZ)
  - ssh to the instance from anywhere, run tests via the cloud
- Concurrent tests – <u>Changing infrastructure</u>
  - 100 parallel requests: can receive different distributions of containers-to-VMs
  - Each infrastructure-set can exhibit different performance characteristics depending on the workload
- <u>Resource Contention</u> from co-located users
  - May vary due to time of day
  - How can these conditions be replicated?

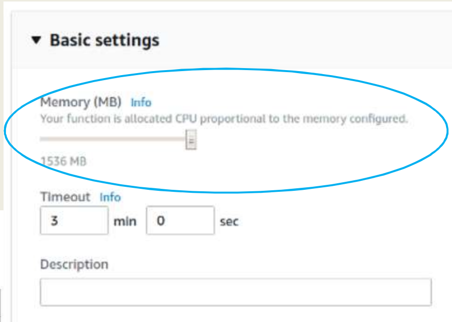
SCALE-UP PERFORMANCE

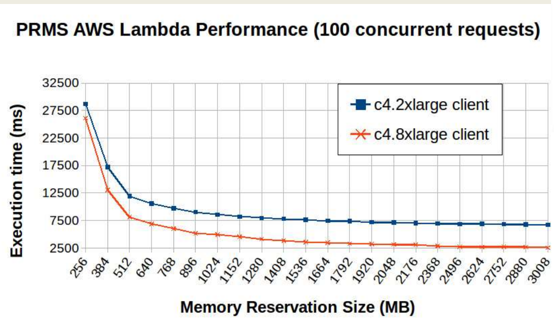
- How does performance change when increasing the number of concurrent clients?
- What is the “STEP” of the scale-up?
- STEP by 1 – add 1 new client each round
- STEP by 10 – add 10 new clients each round



MEMORY VS. PERFORMANCE

- CPU power (% allocation) on AWS Lambda is coupled to memory reservation size
- Performance is always better at with higher RAM, but how much?

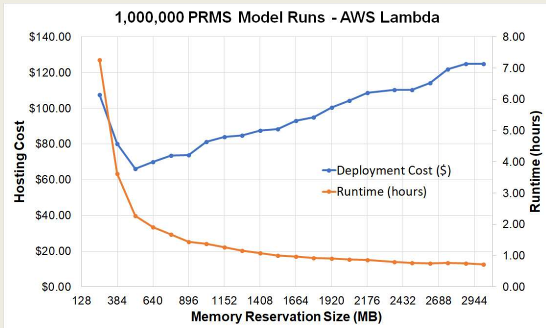




Performance boost is based on how CPU-bound the function is…

MEMORY VS. COST

- AWS Lambda performance is based on memory reservation size and run time
- Changing memory reservation size increases CPU power
- Once memory vs. performance is established can calculated memory reservation size to optimize cost
- Estimate cost for a fictional large workload e.g. 1,000,000 requests



November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.19

ETL PIPELINE SPECIFICS

- **Transform Service**
- Can calculate data processing throughput: rows per second
- Given different file sizes (e.g. 100, 1000 10000 rows) what is the throughput?
- **Research Question**: How does the size of the client data payload relate to data processing throughput? (rows/second)
- Are smaller or larger files faster to process?
- *E.g. what is the price per ounce? (gram)*

- **Load Service**
- What is the data throughput (rows/second) in loading SQL backend with data?
- How does load performance relate to transformation speed?

November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.20

ETL - 2

- **Query Service**
- To benchmark query service performance, should select a few standard queries, and repeat them using different sizes of databases
- Aggregation queries: GROUP BY to sum(), average(), count()
- Filter queries: WHERE [column] = (value)

- Filtering is fast
- Aggregation can be slower
- Joining is slower, but not really applicable for our 1-table ETL database
- Nested query (select * from (select * from …))

ETL - 3

- **Comparisons of interest**

- Service composition: T-Transform L-Load Q-Query
  - Fully decomposed, fully composed, others:
  - [T] [L] [Q], [T L] [Q], [T] [L Q], [T L Q]
- Application flow control
  - Alternate forms: laptop controller, Lambda controller sync, Lambda async, Step function
- Database backend
  - Amazon Aurora RDS, vs. SQLite

- How does these alternate configurations impact performance (sequential, parallel, scale-up), application hosting costs?

PROJECT CONCLUSIONS

- At the end, groups should have implemented a multi-service mini-application
- There should be at least:
  - The base implementation (akin to the "control" group)
  - EXAMPLE: [TRANSFROM] [LOAD] [QUERY] as separate services
- Then there should be a comparison implementation
- Research Question:
  - What is the performance and cost implications for the competing implementations? How did performance/cost change?
  - Performance measures: turnaround time, compute time, throughput (rows/sec), latency

PROJECT CONCLUSIONS - 2

- For performance tests, reports should describe the test configurations
- Availability zones, client type (VM, ec2 instance type), # of requests, # of batches
- Try to capture every detail so the test could be replicated to confirm results
- Developing test scripts makes it easy to replicate experiments exactly
- Can include "practical" perspectives
- Lessons learned from building the applications and implementing the tests

FUNDAMENTAL CLOUD ARCHITECTURES

November 28, 2018

L17.25

FUNDAMENTAL CLOUD ARCHITECTURES

- Common foundational cloud architectural models
- Exemplify common configurations of cloud-based application deployments
- Architectures describe cloud provisioning of: Compute, disk, and network resources

November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.26

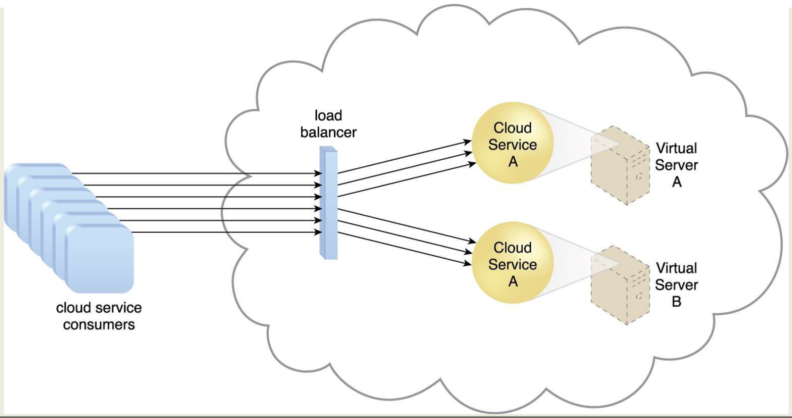
FUNDAMENTAL CLOUD ARCHITECTURES - 2

- **Workload distribution architecture**: load balancing
- **Resource pooling architecture**: resource pools
- **Dynamic scalability architecture**: auto-scaling
- **Elastic resource scalability architecture**: vertical scaling
- **Service load balancing architecture**: load balancing for cloud/web services
- **Cloud bursting architecture**: hybrid cloud
- **Elastic disk provisioning architecture**: thin vs. thick disk provisioning
- **Redundant storage architecture**: duplicate storage devices across data centers

WORKLOAD DISTRIBUTION ARCHITECTURE

- Horizontally scaled IT resources
- Add/remove resources per tier
- Load balancer distributes workload among providers
- Goal is to reduce IT resource:
  - Over-utilization
  - Under-utilization
- Sophisticated load balancing algorithms / run-time logic
  - Support resource management
  - Workload distribution

WORKLOAD DISTRIBUTION ARCHITECTURE - 2



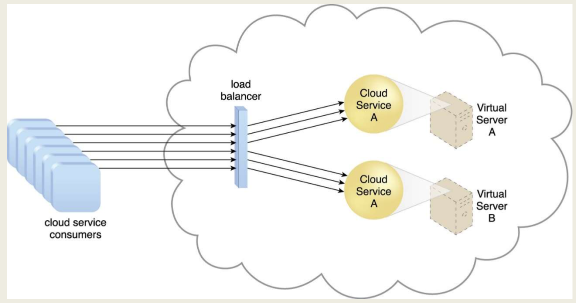
Redundant copies of the Cloud Service are implemented on both Virtual Servers. The load balancer intercepts service requests and directs them to either virtual server to ensure even workload distribution.

WORKLOAD DISTRIBUTION ARCHITECTURE - 3

- Can be applied to any IT resource
  - Virtual servers
  - Cloud storage devices
  - Cloud services
- Specializations of this architecture
  - Service load balancing (upcoming…)
  - Load balanced virtual server architecture *balancing # of VMs per host…*
  - Load balanced virtual switches architecture *Increasing virtual network bandwidth w/ additional physical uplinks*

WORKLOAD DISTRIBUTION ARCHITECTURE - 4

- Does this architecture encapsulate high availability?
  - Redundancy
  - Fault tolerant
  - Fail-over



- Is the load balancer fault tolerant?
- How could the load balancer be made fault tolerant?

HIGH AVAILABILITY LOAD BALANCING

- Active / passive mode
  - Pair of load balancers are configured
  - Primary load balancer distributes traffic
  - Second load balancer operates in listening mode
  - Secondary load balancer step-ins in if primary fails
  - Achieves high availability
- Active / active mode
  - Two or more servers aggregate traffic load at the same time
  - User sessions are “locked” to one load balancer
  - Session is cached, requests are routed to same resource provider
  - If user request goes to other load balancer, it doesn’t know how to route request – would need to query other load balancer… ***slow!***
  - If one LB fails, is the other sufficient to route traffic?

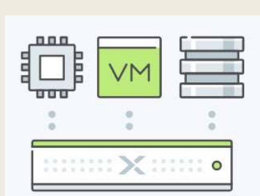
WORKLOAD DISTRIBUTION ARCHITECTURE - 5

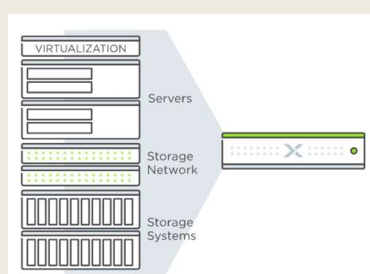
- Other common elements of this architecture:
- **Audit monitor**: logs user requests as needed
- **Cloud usage monitor**: logs server utilization
- **Hypervisor**: virtual machines may need to be distributed
- **Logical network perimeter**: workloads distributed within
- **Resource cluster**: compute cluster resources to implement architecture
- **Resource replication**: concept of generating new resources in response to demand

November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.33

RESOURCE POOLING ARCHITECTURE

- Identical IT resources are grouped and maintained
- System ensures they remained synchronized
- EXAMPLE: Hyper-converged server infrastructure
- Nutanix: https://www.nutanix.in/hyperconverged-infrastructure/



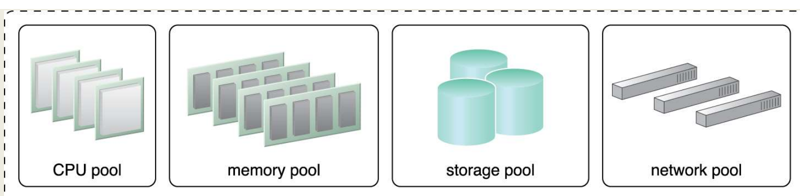


November 28, 2018	TCSS562: Software Engineering for Cloud Computing [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.34

RESOURCE POOLING ARCHITECTURE - 2

- Resource Pools:
- Physical server pool / Virtual server pool
  - Preconfigured with OS/applications, ready for immediate use
- Storage pool
  - File-based, block-storage entities, with or without data, ready for use
- Network pool
  - Virtual firewall devices or network switches for redundant connectivity, load balancing, link aggregation
- CPU pool, Memory pool
  - Allocated to virtual servers

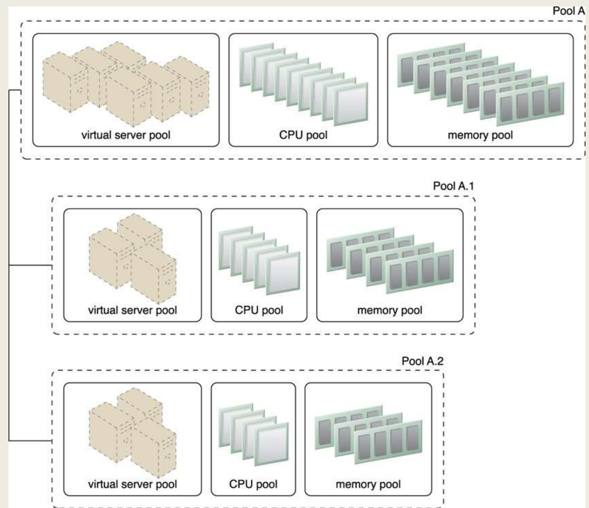
SAMPLE RESOURCE POOL



- Resources pools can be used to provide virtual devices
- Virtual server(s)
  - Consumes CPU and memory from pool
- Virtual disk(s)
  - Aggregate "just a bunch of disks" (JBoD) to provide disk(s) with required capacity, IOPS requirements, latency
- Virtual network
  - Aggregate physical network resources to provide virtual network devices which are isolated, with necessary bandwidth, and capacity

RESOURCE POOLING ARCHITECTURE - 2

- Nested pools:
  Use same resources, but in different quantities.
- Allow rapid instantiation of resources with identical configurations



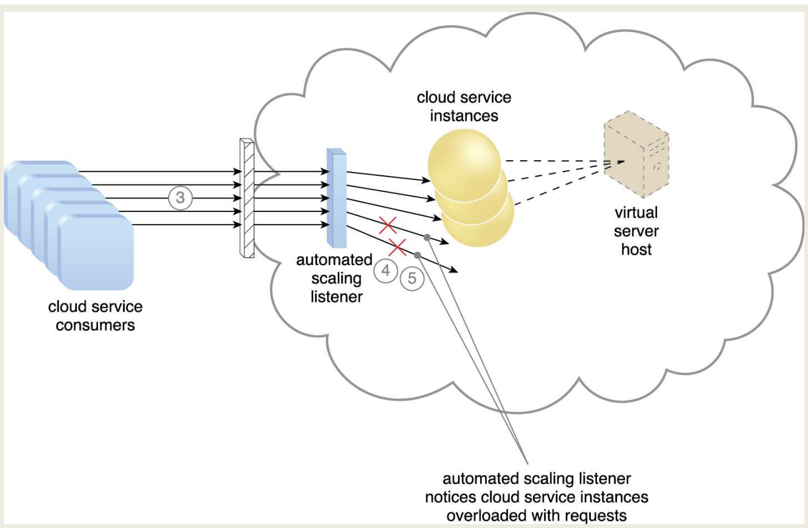
RESOURCE POOLING MECHANISMS

- Audit monitor: monitor usage to ensure legal use
- Cloud usage monitor: runtime tracking and synchronization to support management of resource pools
- Pay-per-use monitor: collects usage and billing information on how individual cloud users allocate and use resources
- Remote administration system: interfaces with backend systems to provide administration support
- Resource management system: supports administering resource pools
- Hypervisor, Logical network perimeter, Resource replication

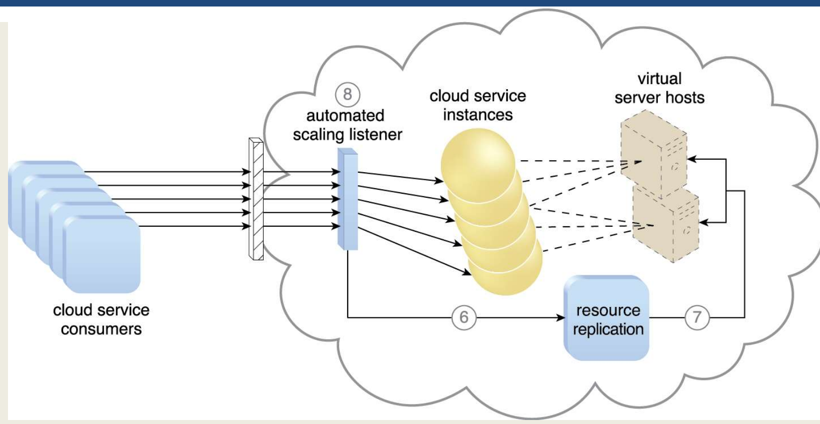
DYNAMIC SCALABILITY ARCHITECTURE

- Uses predefined scaling conditions to trigger "dynamic allocation" of IT resources from pools
- Resource allocation is adjusted dynamically based on demand
- Unnecessary resources are automatically
- <u>Automated scaling listener</u>
  - Monitors workload thresholds to determine when new resources should be added / removed using a scaling policy
  - Scaling policy – defines specifics of the scaling thresholds

DYNAMIC SCALABILITY ARCHITECTURE - 2



DYNAMIC SCALABILITY ARCHITECTURE - 3



Automatic scaling listener **triggers** creation of additional cloud service instances, which are added to pool for load balancing. *Automated scaling listener* resumes monitoring and adds and subtracts resources as required.

DYNAMIC SCALABILITY ARCHITECTURE - 4

- **Example**: AWS -Elastic Load Balancer (ELB)
  - **Classic load balancer**: application agnostic distribution of traffic across nodes
  - Uses cloud watch metrics …
  - **Application load balancer**: distributes traffic while considering unique content of requests enabling advanced routing capabilities
- ELB integrates with AWS auto scaling to dynamically provision +/- resources in response to demand
- Load balancer configuration automatically adjusted

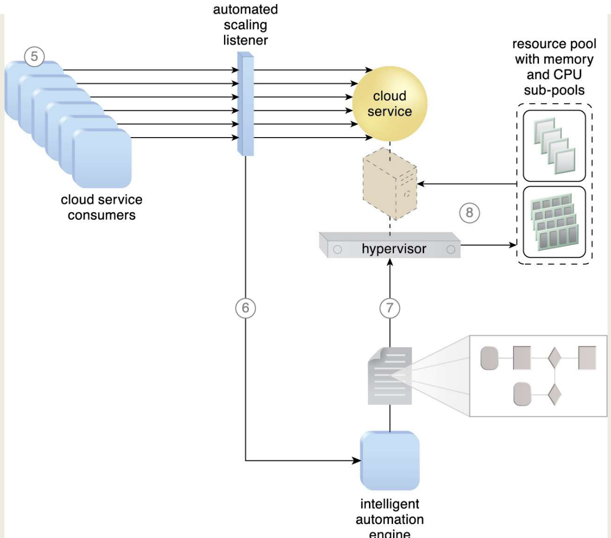
DYNAMIC SCALABILITY ARCHITECTURE QUESTIONS

- Why should load balancers / scaling listeners reroute subsequent requests for TCP sessions to the same server?
- How could "sticky" sessions impact load balancing?
- What are the advantages of classic (application agnostic) load balancing?
- For an "application load balancer" supporting "advanced routing", what features and capabilities are required of the load balancer?
- Which is more performant? Software or hardware load balancer?

ELASTIC '*RESOURCE CAPACITY*' ARCHITECTURE

- Supports dynamic provisioning of virtual servers
- Feature of public/private infrastructure-as-a-service (IaaS) clouds
- Enables reprovisioning CPUs and RAM ***(*vertical scaling*)*** to change the **SIZE** of a live virtual machine
  - Container platforms
- Ability to interact with the hypervisor and ***virtual infrastructure manager*** (VIM) to manage resources – **at runtime**
- Virtual server is monitored to increase capacity from a resource pool when thresholds are met.

ELASTIC RESOURCE CAPACITY ARCHITECTURE - 2



ELASTIC RESOURCE CAPACITY ARCHITECTURE - 3

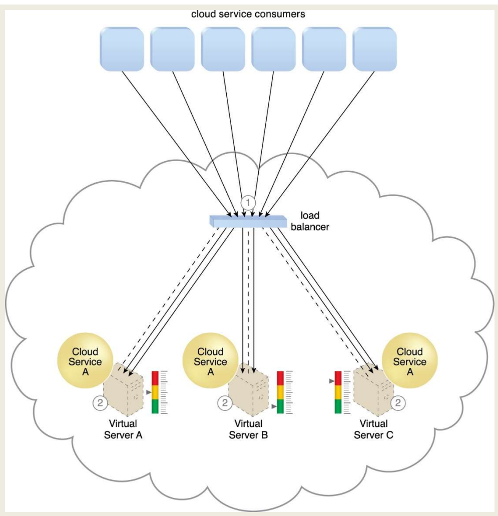
- Virtual servers may require rebooting for the changes in memory and CPU to take effect
- VIMs may automatically redistribute RAM & CPUs to VMs based on demand if rebooting is not required
- Not all Cloud VIMs or Container orchestration frameworks support/expose this feature
- Features are accessible at the hypervisor level
- Can resize # of CPUs and RAM of VMs on-the-fly by interacting directly with XEN/KVM hypervisors – via the CLI !
  - *Its preferable to recreating the VM entirely*

FUNDAMENTAL CLOUD ARCHITECTURES

- **Workload distribution architecture**: load balancing
- **Resource pooling architecture**: resource pools
- **Dynamic scalability architecture**: auto-scaling
- **Elastic resource scalability architecture**: vertical scaling
- **Service load balancing architecture**: load balancing for cloud/web services
- **Cloud bursting architecture**: hybrid cloud
- **Elastic disk provisioning architecture**: thin vs. thick disk provisioning
- **Redundant storage architecture**: duplicate storage devices across data centers

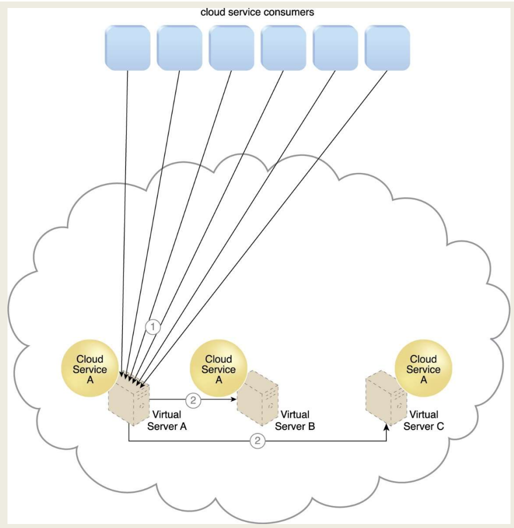
SERVICE LOAD BALANCING ARCHITECTURE

- A specialized variation of the workload distribution architecture
- Redundant deployments of cloud services are created, and load balancer distributes workloads
- The architecture we configure in tutorial #2 !
- Focuses on scaling cloud service implementations



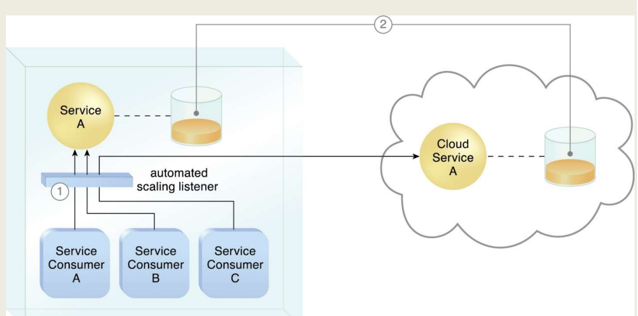
REQUEST REDISTRIBUTION

- Service redistributes request to the proper server
- "Shard" is a segment of a database hosted on a single server
- Sharding enables horizontal scaling of datasets by distributing rows across multiple servers
- Data fetch with sharding: Request is processed by application server to route request to server hosting the shard



CLOUD BURSTING ARCHITECTURE

- Burst beyond on-premise IT resources to use public cloud when predefined capacity thresholds are surpassed
- Cloud resources are pre-deployed, but in ***inactive*** state until cloud bursting occurs
- Once cloud resources are no longer needed, they are released
- Automated scaling listener is used
- Latency to the cloud should be considered



SCALING QUESTIONS

- When is vertical scaling preferable to horizontal scaling of cloud resources?
- Is cloud bursting vertical or horizontal scaling?
- Consider a private cloud with 5 host servers. What types of scaling is likely to be more important to the system administrator: horizontal or vertical scaling? Why?
- Can Docker container orchestration frameworks support horizontal scaling?
- Vertical scaling?

ELASTIC DISK PROVISIONING ARCHITECTURE

- Static allocations of fixed amounts of cloud disk space are expensive
- **Example:**
  Provision virtual Windows Server with 450GB disk
- Before OS is installed: 0 GB is used
- After OS is installed: <100 GB is used
- Customer is charged for: 450GB
- Elastic disk provisioning establishes a dynamic storage provisioning system to granularly bill a user for storage **actually used**...
- Based on “thin-provisioning” of storage

THIN VS. THICK PROVISIONING

- **Thin Provisioning**
- Only allocate storage space as it is used
- Increases potential for sharing the disk
- Introduces problem of *over-provisioning*: allocate more virtual disk space than actually exists

- **Thick Provisioning**
- Statically allocate all requested disk space
- A single user can provision the whole disk rendering it unusable by others !

THIN PROVISIONING - EXAMPLE

- Virtual box supports "thin provisioning" of virtual disks

- Disks have a maximize size, but only what is actually used is provisioned allowing the volume to grow.

- Eucalyptus EBS volume implementation
  - Disk volumes are thinly provisioned
  - Threat of over provisioning

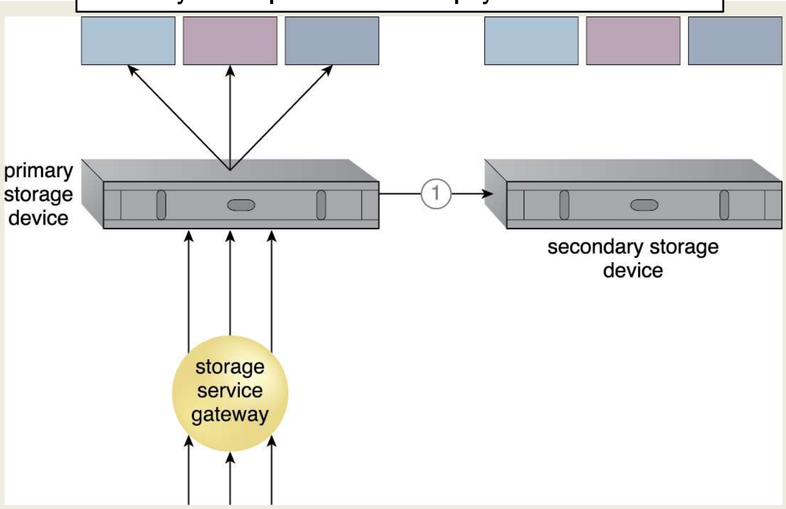
- Resizing volumes can be challenging

REDUNDANT STORAGE ARCHITECTURE

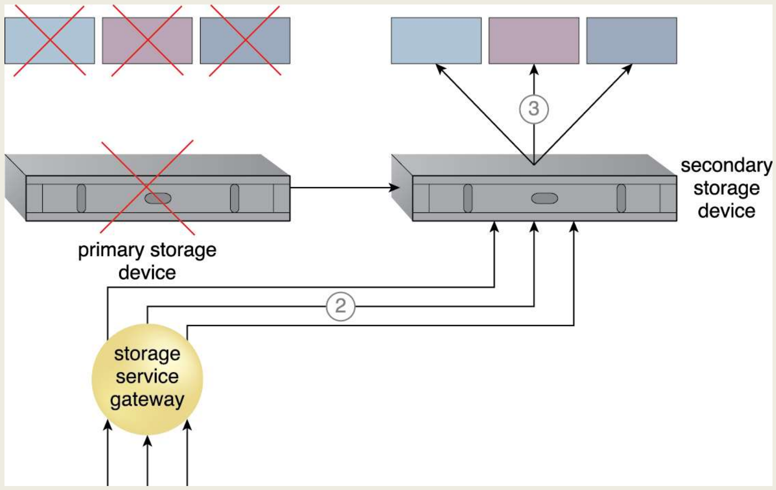
- Provide fault tolerance and improved availability of cloud storage devices
- Individual storage devices ***already*** have dual disk arrays and redundant disk controllers
- We are talking about SANs, NASs
- The idea is to replicate storage devices

REDUNDANT STORAGE ARCHITECTURE - 2

These colored blocks represent user disks. They are "Virtual" in the sense that the storage device abstracts how they are implemented with physical disks...



REDUNDANT STORAGE ARCHITECTURE - 3



REDUNDANT STORAGE ARCHITECTURE - 4

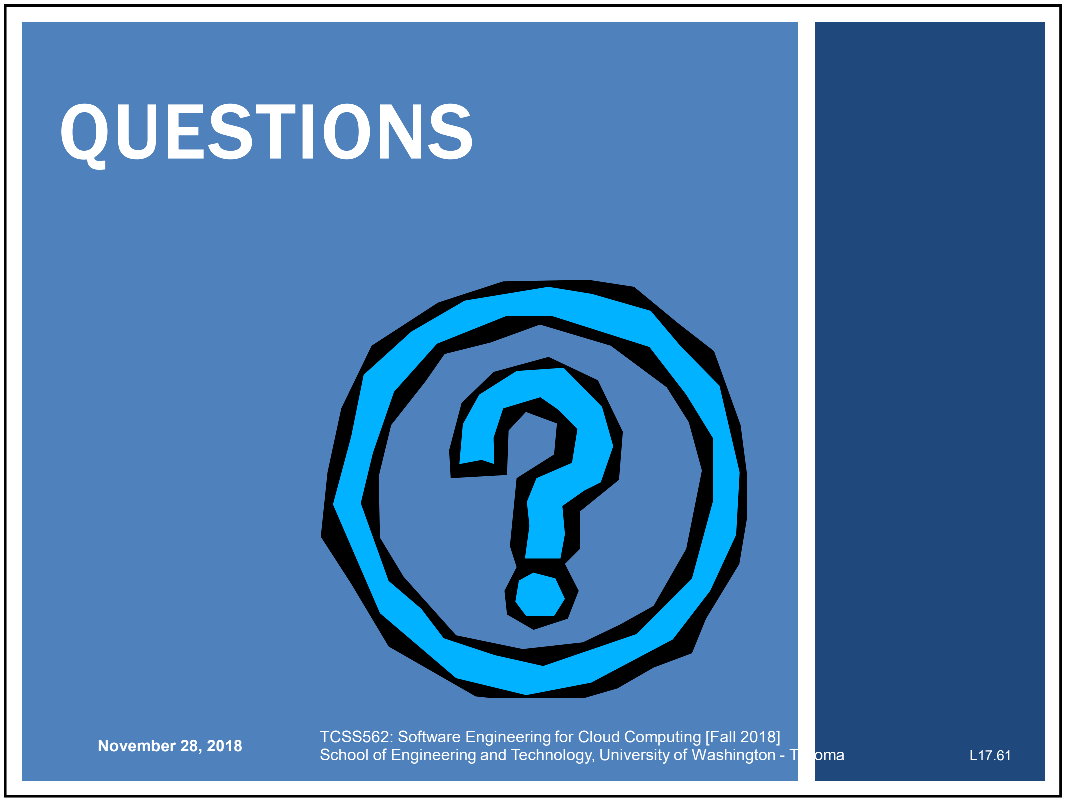
- Introduce a secondary duplicate cloud storage device that synchronizes data with the primary storage device
- Storage gateway service routes requests to second device when the primary device fails
- Secondary storage devices may be located in different physical locations

CLOUD STORAGE QUESTIONS

- If we have two identical storage devices that internally feature redundant disk arrays based on RAID 1, how many copies of the data exist?
- Besides disk space, what else does thin provisioning save?
- In addition to data redundancy, what else is gained from having multiple copies of our data?

FUNDAMENTAL CLOUD ARCHITECTURES SUMMARY

- **Workload distribution architecture**: load balancing
- **Resource pooling architecture**: resource pools
- **Dynamic scalability architecture**: auto-scaling
- **Elastic resource scalability architecture**: vertical scaling
- **Service load balancing architecture**: load balancing for cloud/web services
- **Cloud bursting architecture**: hybrid cloud
- **Elastic disk provisioning architecture**: thin vs. thick disk provisioning
- **Redundant storage architecture**: duplicate storage devices across data centers


QUESTIONS
November 28, 2018
TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - T oma
L17.61