

ECE4574 – Large-Scale SW Development for Engineering Systems

Lecture 15 – Cloud-based Architectures

Creed Jones, PhD

Course Updates

- Project
 - Sprint 2, week 2
- Next Quiz (6) is Wednesday, November 1

Key Topics for Today

- Cloud Services
- Key Factors
 - Scalability - Vertical vs. Horizontal
 - Data consistency
 - Network latency
- Some Architectural Patterns for the Cloud
 - Horizontally Scaling Compute pattern
 - Queue-centric Workflow pattern
 - MapReduce pattern
 - Database Sharding pattern
 - Multisite Deployment pattern

Much of today's lecture comes from *Cloud Architecture Patterns* by Bill Wilder

- Wilder, B. (2012). Cloud architecture patterns. O'Reilly Media.
- Available through VT library.



The convenience of cloud computing comes from the concept of delivering services

Software as a Service (SaaS)

- Software is licensed and provided over the web – apps run on the cloud and are accessed via web protocols – often through a browser
- GoogleDocs, Salesforce, Slack

Well-thought-out architectures allow applications to maximize the benefits of cloud services, keeping in mind some important attributes

- Scalability
- Data Consistency
- Ease of Use

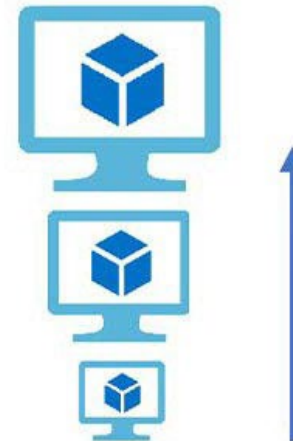
The architecture of a software system using cloud services can be *public, private, hybrid or multicloud*

- **Public cloud architecture:** the cloud services and physical infrastructure are owned and operated by a third-party cloud service provider.
 - shared resources with other companies
- **Private cloud architecture:** a dedicated cloud that is owned and managed by your organization; privately hosted on-premises.
 - more expensive and requires more IT expertise to maintain.
- **Hybrid cloud architecture:** uses both public and private cloud services, allowing workloads to move between environments.
 - often used to retain flexibility.
- **Multicloud architecture:** uses cloud services from multiple cloud providers.
 - driven by economics and/or technical features (or familiarity).

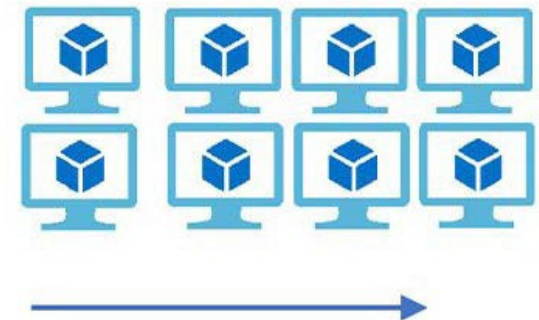
Scalability is the ability to increase the capacity of a software system – whether measured in users, transactions, data volume or other quantity

- A cloud-based system runs on a set of *computing nodes*, which emulate (but may not directly correspond to) a single physical computer
- Vertical scaling increases computing capacity by increasing the resources and capability of existing nodes
- Horizontal scaling increases computing capacity by increasing the number of nodes
- Of course, both can be done

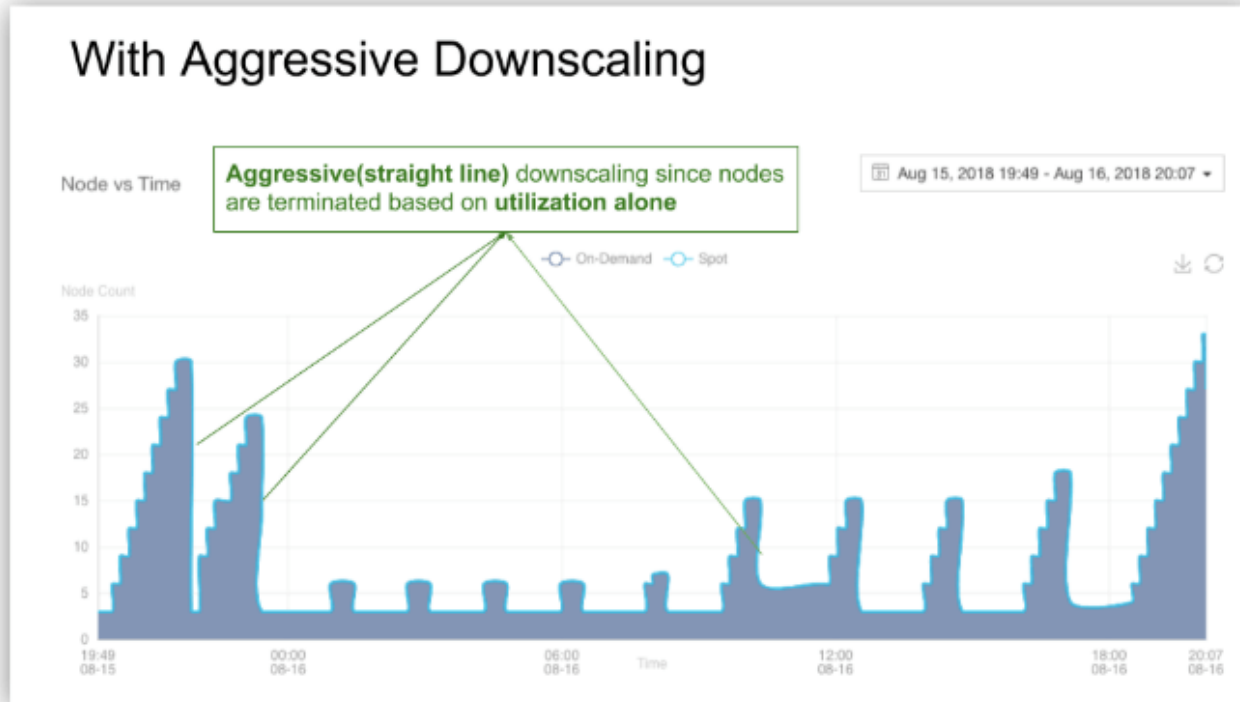
Vertical Scaling
(Increase size of instance (RAM , CPU etc.))



Horizontal Scaling
(Add more instances)



Important point – cloud-based systems lend themselves to easier upscaling and (especially) downscaling than physical systems

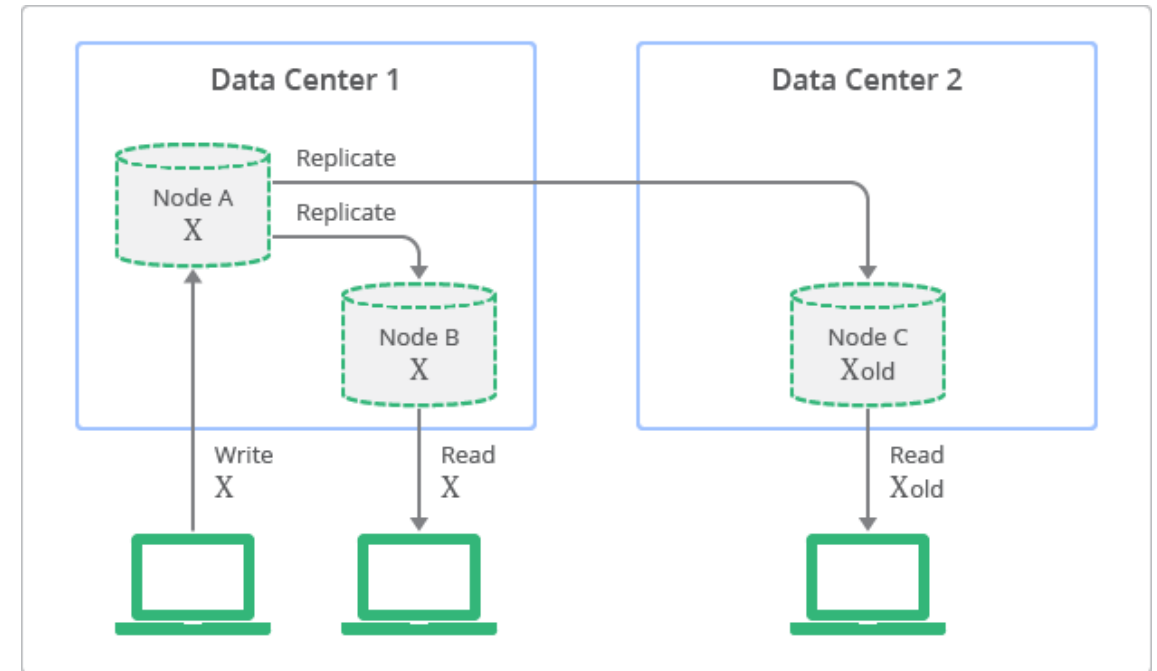


- There are various policies for downscaling
- Higher scale costs more, of course
- But, if we downscale too aggressively, it's more difficult to adjust to sudden surges in demand
 - and latency can suffer
- Depends on pricing model
- This is one of a class of problems in dynamic resourcing
 - networking and operating systems face similar challenges

www.gubole.com

Data Consistency is an important consideration in cloud and other distributed systems

- In cloud systems, it's likely that more than one data center will contain storage associated with a particular process
 - Cloud providers do this to allow for network and server downtime
- But what if data X_{old} is written to both servers (replication), and at some later time, when Node C is down, data is updated to X?
 - Subsequent reads (after Node C is back up) will have ambiguous values

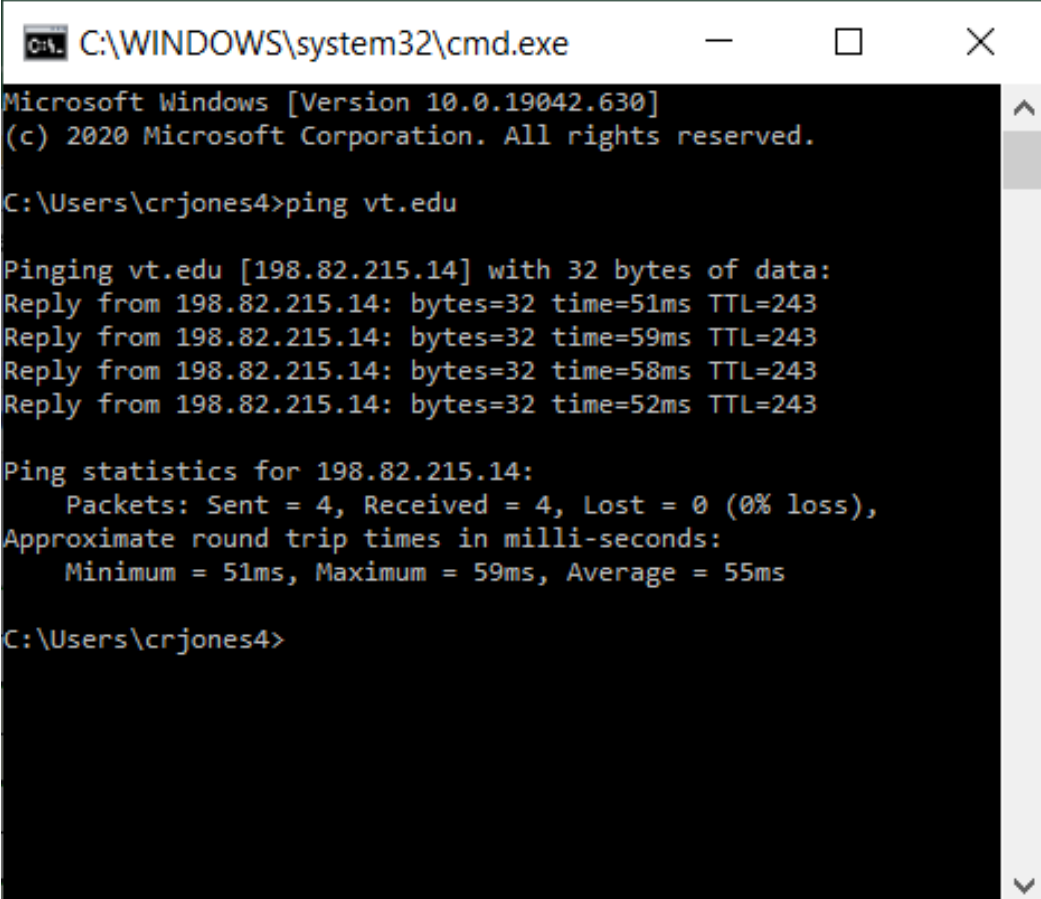


Data Consistency is an important consideration in cloud and other distributed systems

- Client-side consistency measures are things like “read-your-writes”, serialization of reads and/or writes, and strategies to not assume consistency in all cases
 - *Eventual consistency* is the guarantee that – given enough time – any access will retrieve the most recently updated value (can we live with this?)
- Server-side consistency are modes that describe under what conditions one can expect data consistency
 - N: nodes where replicas of the data value resides
 - W: how many replicas must update to consider the update “done”
 - R: how many replicas are contacted as part of a read
 - Set W, R and N based on optimization of reading or writing
 - $W+R>N$: we always have strong consistency
 - $W+R\leq N$: internal updating strategies to reduce latency (usually $R=1$ in this case)

Perceived network latency is composed of actual latency and other data-manipulation times

- Network latency is measurable using ping
- Perceived latency also includes data decompression time and other preprocessing
- Perceived latency can be reduced by predictive fetching



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.630]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\crjones4>ping vt.edu

Pinging vt.edu [198.82.215.14] with 32 bytes of data:
Reply from 198.82.215.14: bytes=32 time=51ms TTL=243
Reply from 198.82.215.14: bytes=32 time=59ms TTL=243
Reply from 198.82.215.14: bytes=32 time=58ms TTL=243
Reply from 198.82.215.14: bytes=32 time=52ms TTL=243

Ping statistics for 198.82.215.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 51ms, Maximum = 59ms, Average = 55ms

C:\Users\crjones4>
```

Using cloud services as our architectural basis has some great advantages, but also some disadvantages to be aware of

☺ Cost

- Lower than owned compute resources (usually!)

☺ Scalability

- Faster and cheaper than alternatives

☺ New technology is readily available

☺ Flexibility in technology

- Middleware choices, for example

☺ Built-in features

- Backup
- Management

☹ Security

- Data loss or compromise
- Denial of service
- Hacking

☹ High-quality connectivity is required

- Some uses don't allow this
- Direct and severe impact of connectivity issues

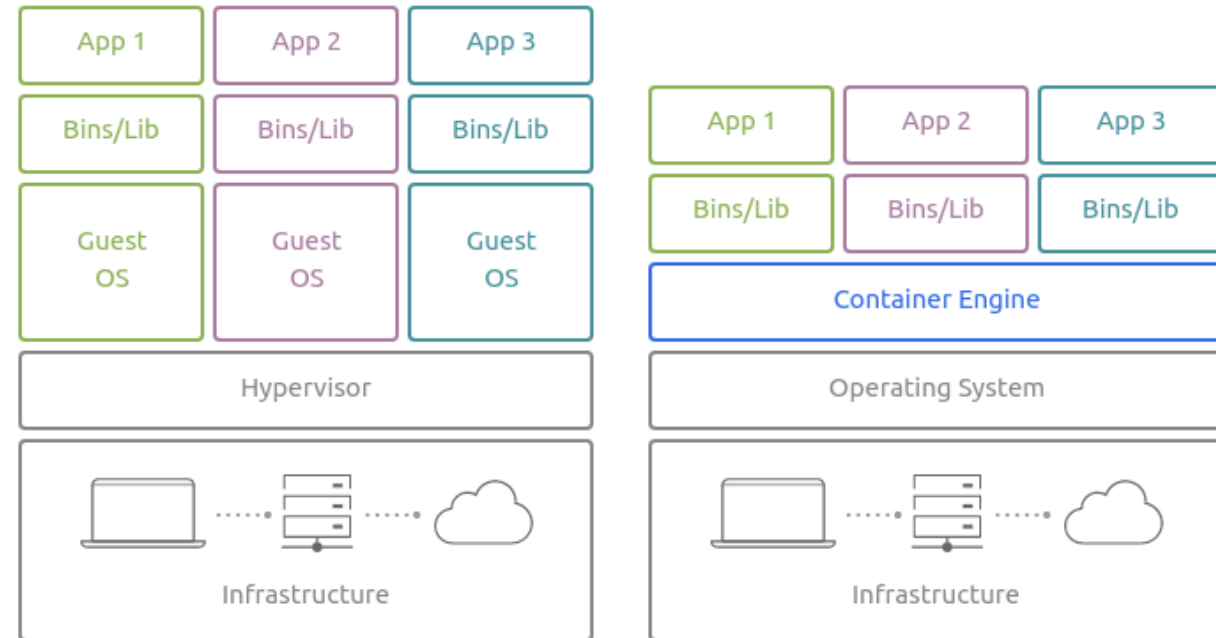
☹ Reliance on third parties

SOME ARCHITECTURAL PATTERNS FOR THE CLOUD

What are containers?

Containers are a technology that allows the user to divide up a machine so that it can run more than one application (in the case of process containers) or operating system instance (in the case of system containers) on the same kernel and hardware while maintaining isolation among the workloads. Containers are a modern way to virtualise infrastructure, more lightweight approach than traditional virtual machines: all containers in single host OS share the kernel and other resources, require less memory space, ensure greater resource utilisation and shorter startup times by several orders of magnitude.

- Remember the cloud concept of *containers* – a portion of a computer's resources (CPU cores, memory, interfaces) that can be allocated to a process
- Key features of cloud services (scaling, data consistency, fault tolerance) are built on strategies that manipulate containers

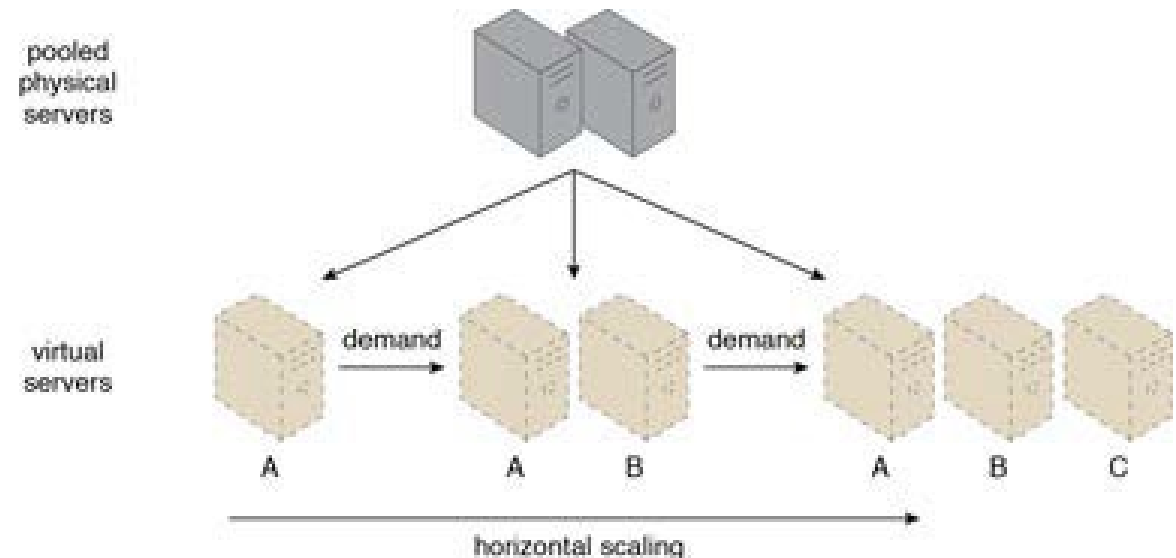


Machine Virtualization

Containers

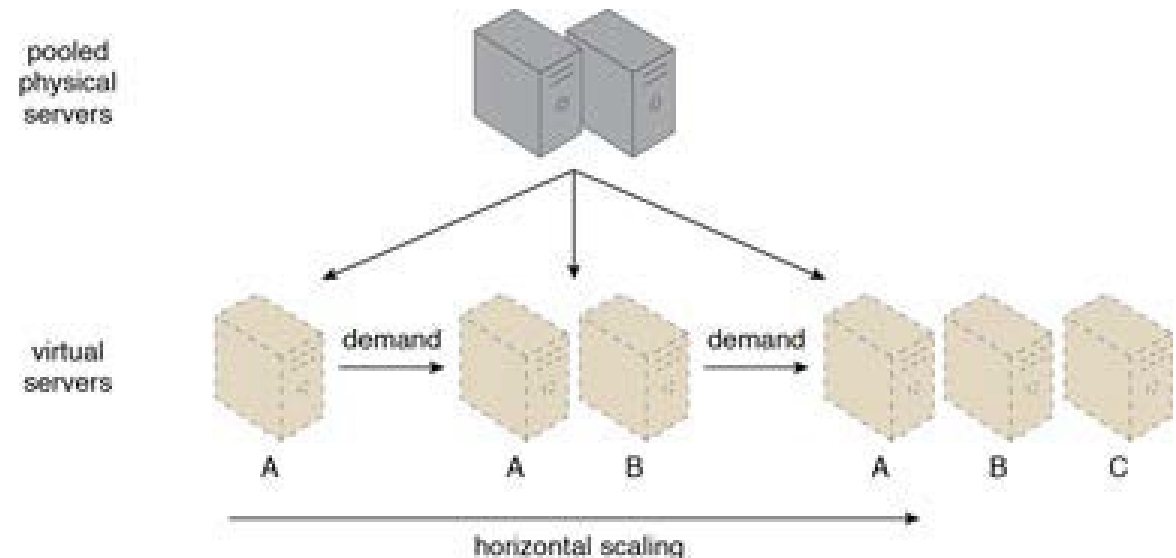
The Horizontally Scaling Compute pattern simply implements reversible scaling – to save money at non-peak times

- The Cloud provider being used will have standard methods for scaling in both directions
- Upscaling is in response to load
- Key parameters influence the promptness and magnitude of downscaling
- The *load balancer* component allocates requests to nodes
 - See <https://www.codeproject.com/Articles/294350/MVC-on-Azure-for-Beginner>



There are some common requirements for using the Horizontally Scaling Compute

- Since web requests from a particular client can go to many different servers, sessions will be stateless
 - Can use browser cookies to convey some state information
 - Requires some common-access storage solution coupled to the compute nodes
- Most platforms (Azure, for example) don't yet support sticky sessions in the load balancer component
 - or it's pretty new

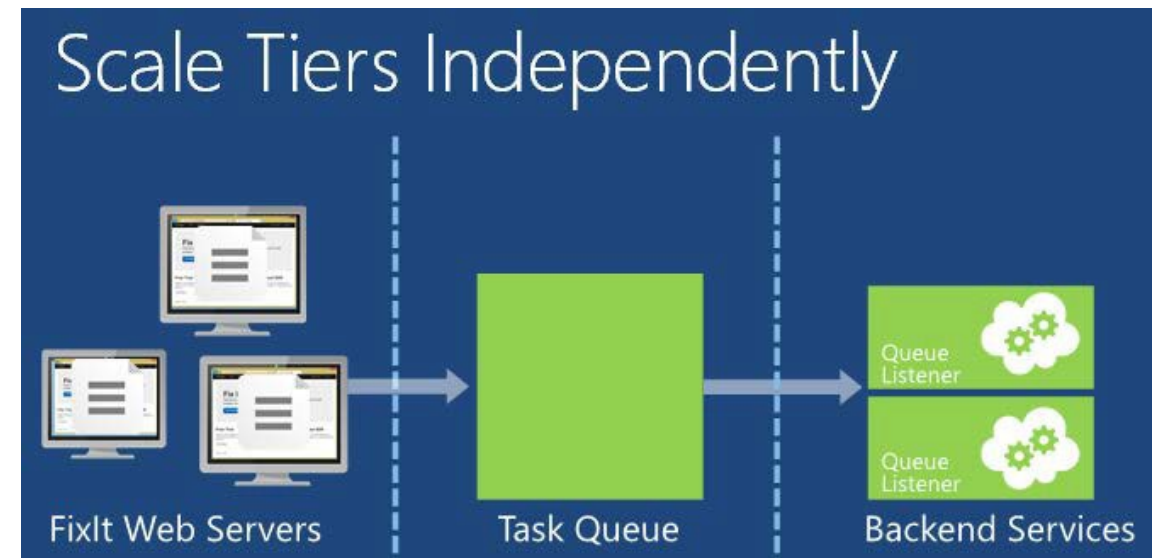


In the Queue-centric Workflow pattern, web service and backend tiers are scaled independently

“When the application gets a request, it puts a work item onto a queue and immediately returns the response. Then a separate backend process pulls work items from the queue and does the work.

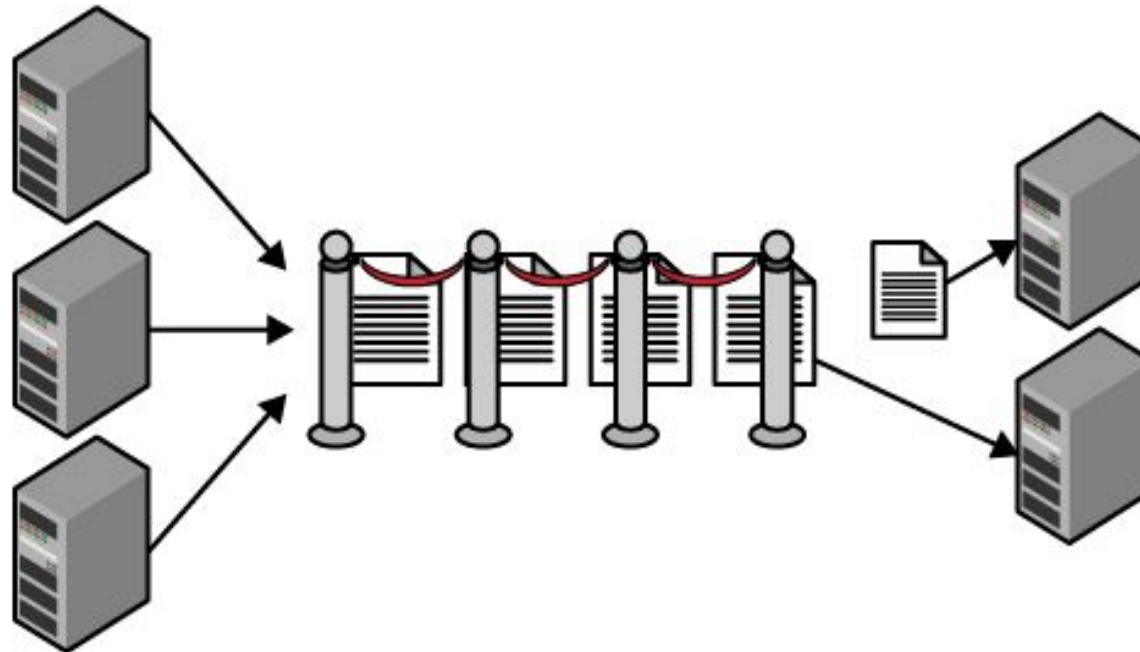
The queue-centric work pattern is useful for:

- Work that is time consuming (high latency).
- Work that requires an external service that might not always be available.
- Work that is resource-intensive (high CPU).
- Work that would benefit from rate leveling (subject to sudden load bursts).”



<https://docs.microsoft.com/en-us/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/queue-centric-work-pattern>

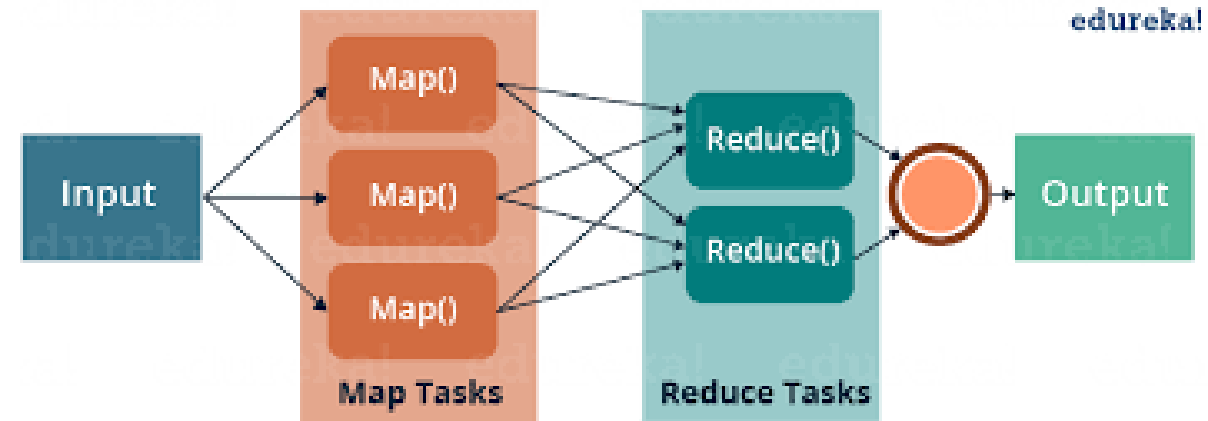
Note that the entire system must be sized to meet strict latency requirements, but the QCW pattern is ideal for addressing varied throughput needs



"The web tier adds messages to the queue. The service tier removes and processes messages from the queue. The number of command messages in the queue fluctuates, providing a buffer so that the web tier can offload work quickly, while never overwhelming the service tier. The service tier can take its time, only processing new messages when it has available resources."

The MapReduce pattern is common for large data sets; requests are translated (Map), then assigned based on keys (Shuffle) and processed (Reduce)

- Well suited for problems where the processing is highly parallelizable
 - engineering applications
 - media
 - machine learning and analytics
- Available in Google Cloud, AWS and Azure
- Implemented in the Hadoop big data platform



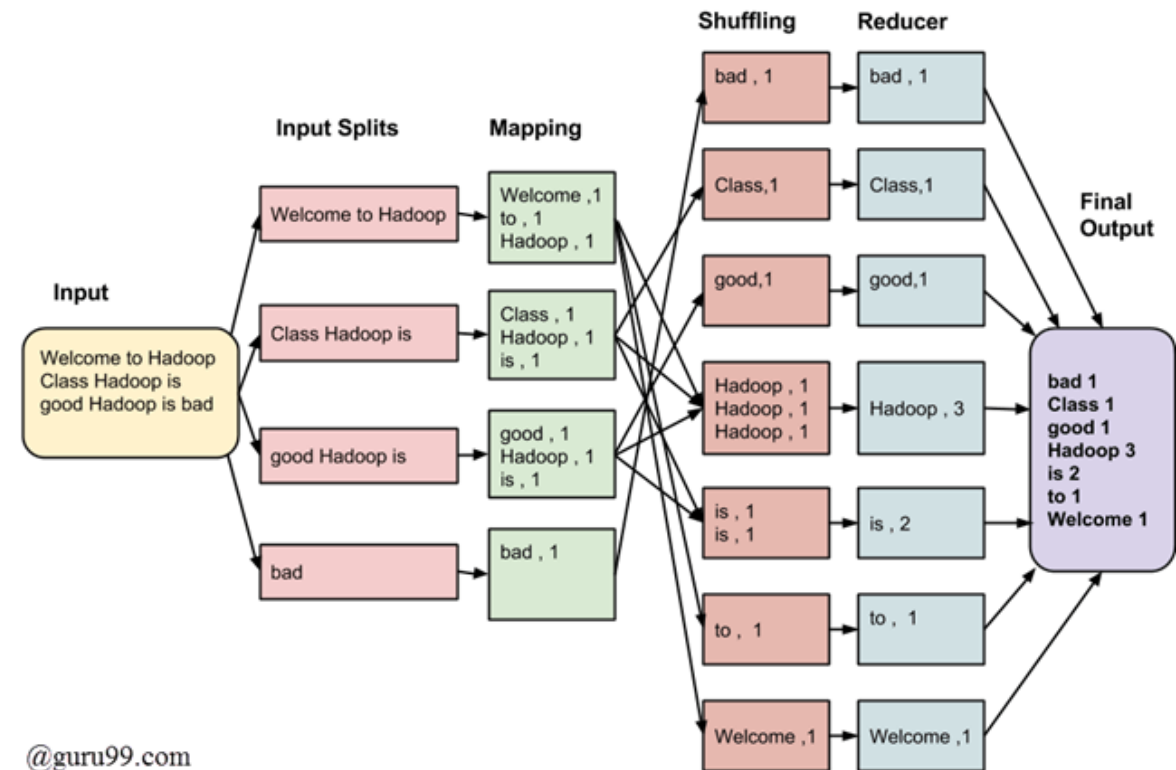
“With MapReduce, rather than sending data to where the application or logic resides, the logic is executed on the server where the data already resides, to expedite processing.”

Map

- The input data is first split into smaller blocks. Each block is then assigned to a mapper for processing.

Reduce

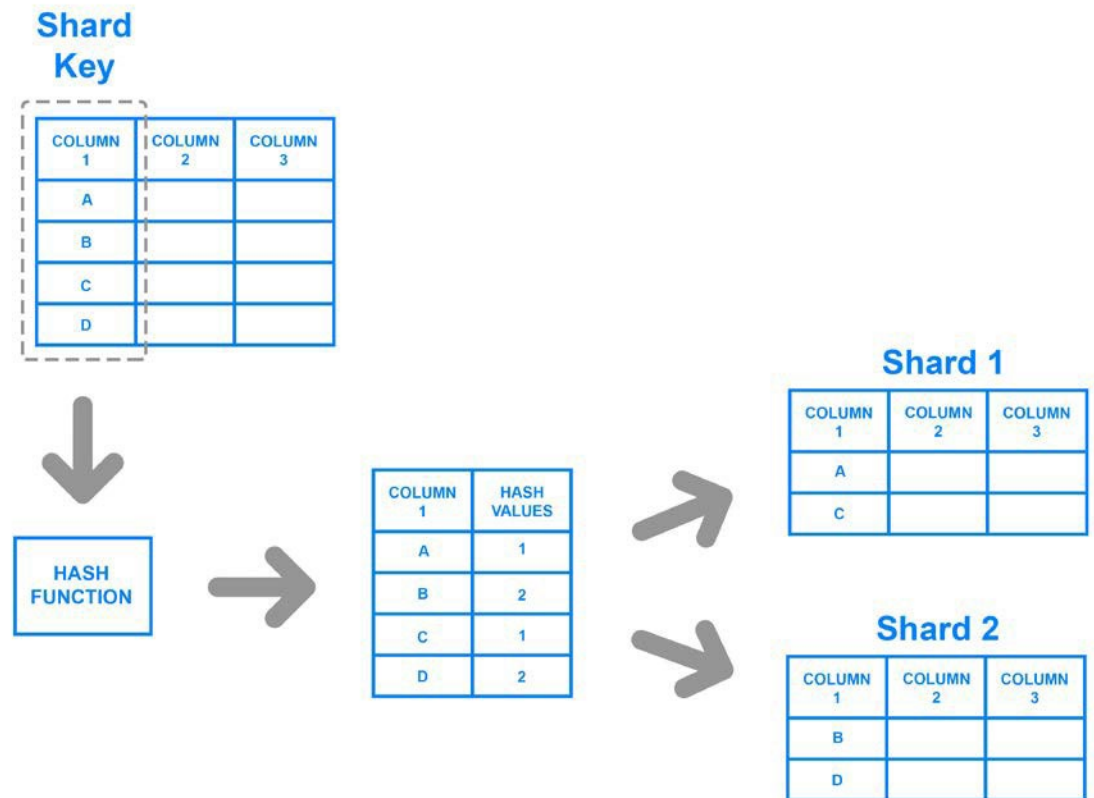
- After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.



@guru99.com

In the Database Sharding pattern, a large database is divided into subsets called *shards*

- Each shard has the same DB schema as the original
- *Most* rows appear in exactly one shard
- Shards are autonomous
 - Shards don't interact with other shards
- Indexing and reference data tables are replicated in all shards



In the Database Sharding pattern, a large database is divided into subsets called *shards*

“Sharding allows you to scale your database to handle increased load to a nearly unlimited degree by providing increased read/write throughput, storage capacity, and high availability.

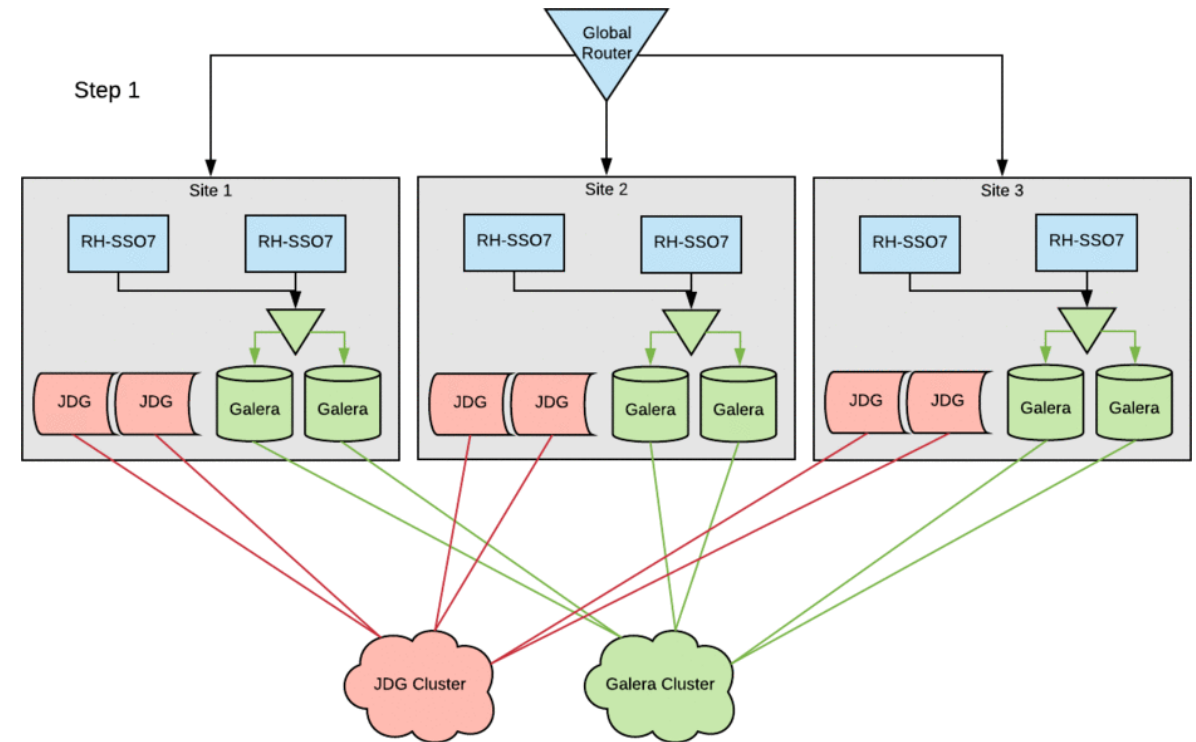
- Increased Read/Write Throughput — By distributing the dataset across multiple shards, both read and write operation capacity is increased as long as read and write operations are confined to a single shard.
- Increased Storage Capacity — Similarly, by increasing the number of shards, you can also increase overall total storage capacity, allowing near-infinite scalability.
- High Availability — Finally, shards provide high availability in two ways. First, since each shard is a replica set, every piece of data is replicated. Second, even if an entire shard becomes unavailable since the data is distributed, the database as a whole still remains partially functional, with part of the schema on different shards.”

<https://www.mongodb.com/features/database-sharding-explained>

The Multisite Deployment pattern deploys key software applications to multiple data centers to reduce latency

“It’s ideal for situations where:

- Users are not clustered near any single data center, but form clusters around multiple data centers or are widely distributed geographically
- Regulations limit options for storing data in specific data centers
- Circumstances require that the public cloud be used in concert with on-premises resources
- Application must be resilient to the loss of a single data center”



Key Topics for Today

- Cloud Services
- Key Factors
 - Scalability - Vertical vs. Horizontal
 - Data consistency
 - Network latency
- Some Architectural Patterns for the Cloud
 - Horizontally Scaling Compute pattern
 - Queue-centric Workflow pattern
 - MapReduce pattern
 - Database Sharding pattern
 - Multisite Deployment pattern