BITS Pilani
Pilani Campus

Popular scaling approaches continued

Prof. Akanksha Bharadwaj
Asst. Professor, CSIS Department

SE ZG583, Scalable Services
Lecture No. 3

Agenda

Popular scaling approaches

Managing high volume transactions

- Service Replicas & load balancing
- Minimizing event processing: Command Query Responsibility Segregation (CQRS)
- Asynchronous communication
- Caching techniques: Distributed cache, global cache

Scalability features in the Cloud (AWS, Azure, Google)

- Auto-scaling
- Horizontal and vertical scaling
- Use of Load balancers
- Virtualization
- Serverless computing
- Best Practices for Achieving Scalability

Case Study

innovate achieve lead

BITS Pilani
Pilani Campus

Managing high volume transactions

Introduction to replication

- Replication is the practice of keeping several copies in different places.
- These replicas help distribute the workload and ensure that failures in one instance do not bring down the entire system.
- What is the need of replication?
  - High Availability
  - Fault Tolerance
  - Load sharing

Service Replicas

- It is a copy of the service logic that runs on one or more nodes of the cluster.
- Types of replicas: These replicas can be categorized based on their deployment model, scaling approach, and management strategy.
  - Stateful
  - Stateless
  - Active-Passive
  - Active-active
  - Leader-follower

Stateful and Stateless Replicas

- **Stateful Replicas**: Each replica maintains its own persistent state, requiring synchronization across instances.
- **Use Case**: Databases, caching services, distributed file systems.

- **Stateless Replicas**: Each replica operates independently and does not maintain any session-specific data.
- **Use Case:** API gateways, web servers, RESTful microservices.

Active-Active and Active-Passive Replicas

- **Active-Active:** All replicas handle requests simultaneously in a distributed load-sharing model.
- **Use Case**: High-traffic applications needing high availability.

- **Active-Passive:** One primary instance handles traffic, while replicas remain on standby and become active in case of failure.
- **Use Case**: Database replication, disaster recovery systems.

Leader-Follower Replica

- A replication model where a **Leader (Primary)** node handles **write operations** and **Follower (Replica)** nodes handle **read operations**. If the Leader fails, a Follower is promoted as the new Leader.
- Usecases: Databases, Caching system, Message Brokers

Load Balancing

- **Load balancing** refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*.

- If a single server goes down, the load balancer redirects traffic to the remaining online servers.

- When a new server is added to the server group, the load balancer automatically starts to send requests to it.

What are load balancers?

- A load balancer is a software or hardware device that keeps any one server from becoming overloaded.

- A load-balancing algorithm is the logic that a load balancer uses to distribute network traffic between servers

- Types Based on Deployment Location:
  - Software-based load balancers
  - Hardware-based load balancers
  - Cloud Load Balancer

Why is Load Balancing Important for Scalability?

Load balancing ensures:

- Even distribution of requests across multiple instances.
- Reduced latency and improved response time by directing traffic to less loaded servers.
- Fault tolerance and high availability by rerouting traffic in case of server failures.
- Efficient resource utilization and better cost management.

Command Query Responsibility Segregation (CQRS)

- It is a pattern that separates read and update operations for a data store.

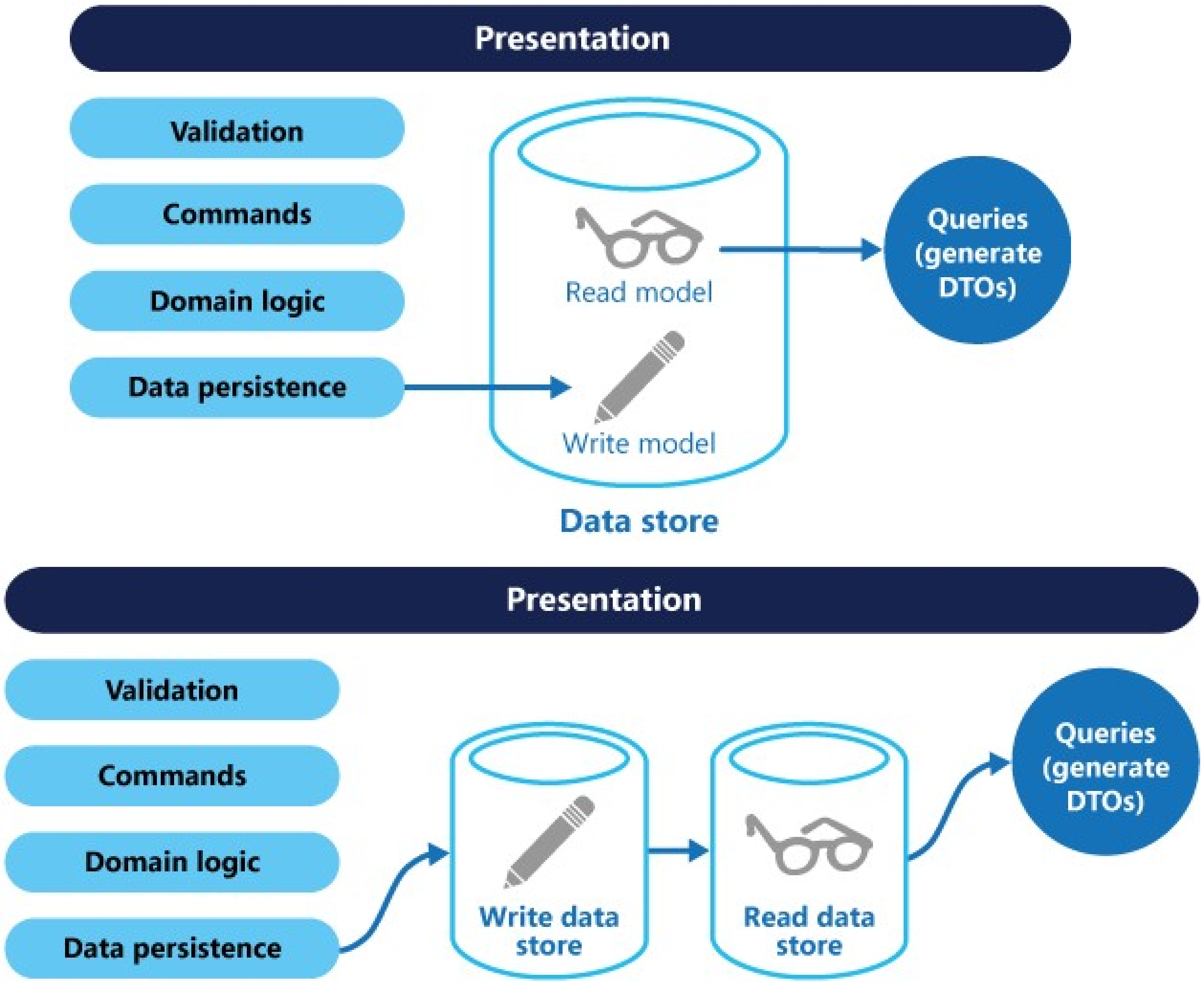
- This pattern is particularly useful in **highly scalable and distributed applications**.

Why CQRS for Scalability?

CQRS improves **scalability** by:

- Separating concerns
- Independent scaling
- Optimizing for reads and writes separately
- Supporting multiple databases
- Simpler queries

How CQRS works?



Some challenges of implementing CQRS

- Complexity
- Messaging
- Eventual consistency

What is Event sourcing?

- **Event Sourcing** is an architectural pattern where **state changes** in a system are stored as a **sequence of events** instead of modifying the current state directly.

https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

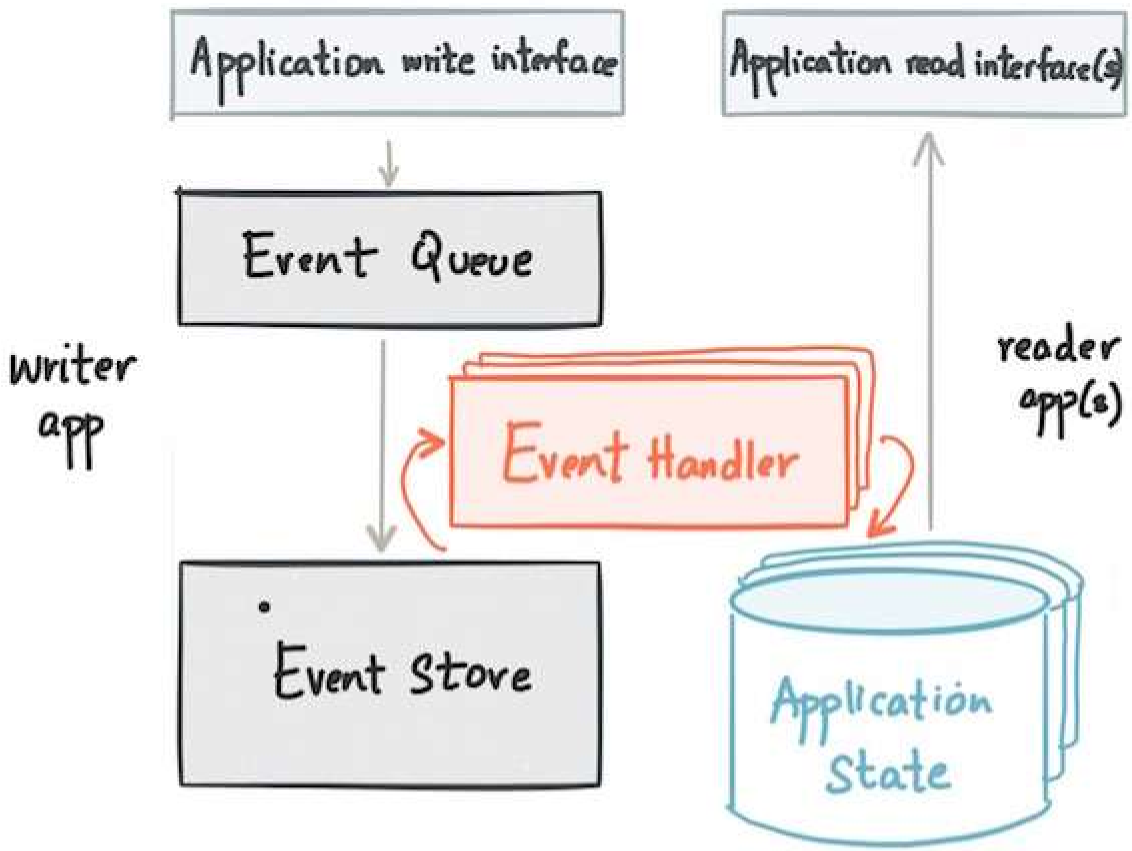
How Event Sourcing and CQRS Work Together

- **Write operations (Commands)** → Generate **immutable events** instead of updating the database directly.

- **Events are stored** in an **event store** (e.g., Kafka, EventStoreDB).

- **Read operations (Queries)** → Use a **separate, read-optimized database** that is updated asynchronously.

Event Sourcing and CQRS



https://mrwersa.medium.com/cqrs-pattern-with-kafka-streams-part-1-112f381e9b98

Architecture of CQRS with Event Sourcing

A. Command Side (Write Model)

- Handles create, update, delete operations.
- Instead of updating a relational DB, writes generate events (e.g., OrderPlaced).
- Events are stored in an event store and published to event consumers.

B. Event Store

- A log of all past events, acting as the source of truth.
- Example storage: Kafka, EventStoreDB, PostgreSQL (JSONB), DynamoDB Streams.

.

C. Query Side (Read Model)

- Events from the event store are processed to update a read-optimized database.
- Read models can use SQL (denormalized tables), NoSQL (MongoDB, Redis), or search engines (Elasticsearch).

D. Event Handlers & Projections

- As events are stored, event handlers process them to update projections (optimized views for queries).
- Example: A UserRegistered event updates a UserProfile table in a read database

Protocols for communication

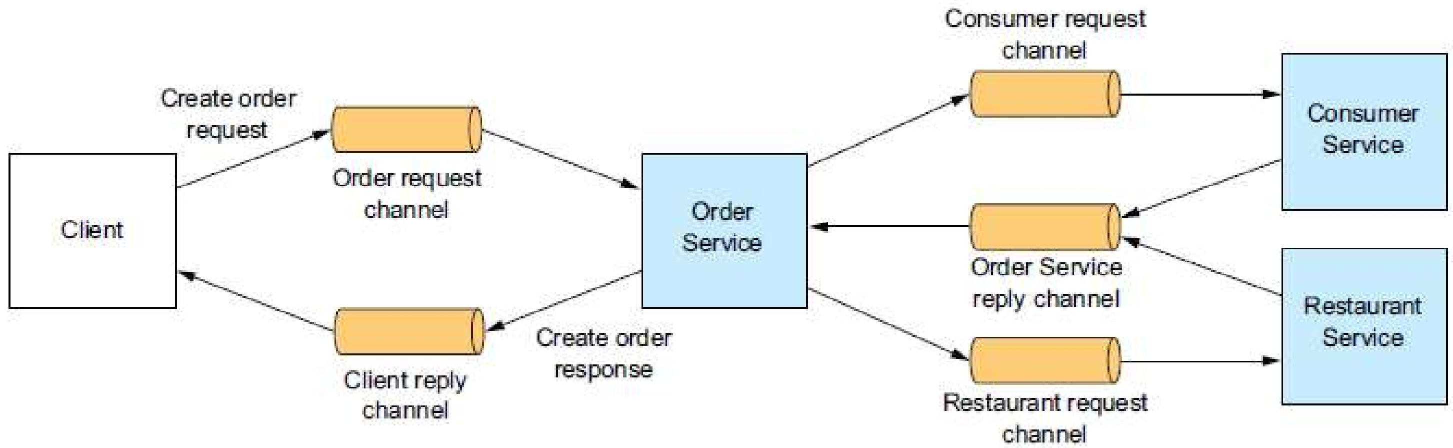
In **distributed systems,** communication between components plays a crucial role in **scalability, performance, and reliability**.

Two main communication patterns are:

- **Synchronous Communication**: The sender waits for a response before proceeding.
- **Asynchronous Communication**: The sender does not wait for an immediate response, improving system efficiency.

Asynchronous Communication

- Services communicating by exchanging messages over messaging channels.



Synchronous Vs Asynchronous

Feature	Synchronous (Blocking)	Asynchronous (Non-Blocking)
Use Case	Request-response APIs	Event-driven systems, messaging
Latency	Higher due to waiting	Lower, as operations continue
Scalability	Limited by concurrent requests	High scalability
Resilience	Prone to failures/timeouts	More resilient to failures
Complexity	Easier to implement	More complex (requires event handling)
Examples	REST APIs, RPC calls	Kafka, RabbitMQ, WebSockets

Message Broker

- It is a way of implementing asynchronous communication
- A message broker is an intermediary through which all messages flow.

Examples of popular open source message brokers include the following:

- ActiveMQ
- RabbitMQ
- Apache Kafka

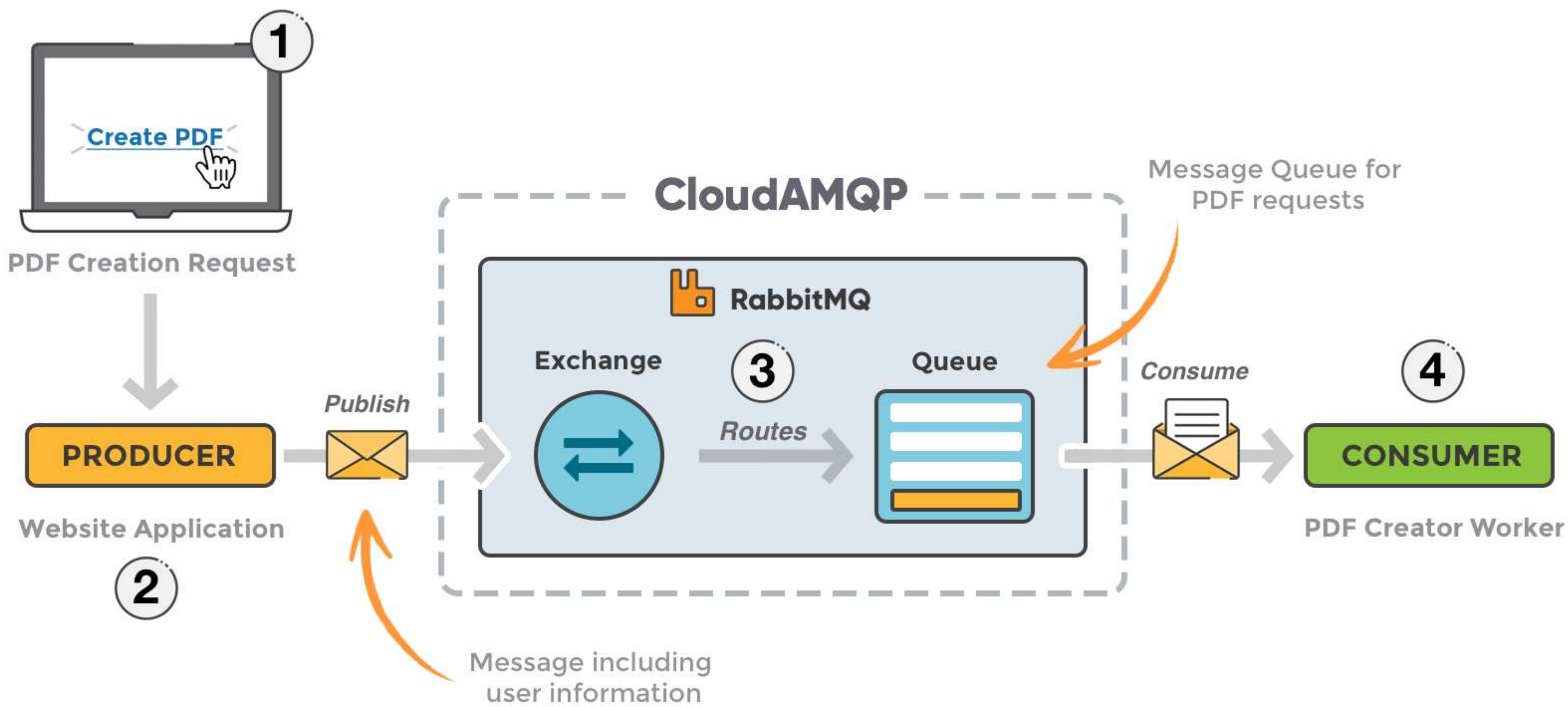
Benefits of Message Broker

- *Loose coupling*
- *Message buffering*
- *Explicit interprocess communication*
- *Resiliency*

Drawbacks of Message Broker

- *Potential performance bottleneck*
- *Potential single point of failure*
- *Additional operational complexity*

Example



https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

What is Caching?

- Caching is a critical technique in **distributed systems** to **reduce latency, improve performance, and optimize resource usage**.

- Caching allows you to efficiently reuse previously retrieved or computed data.

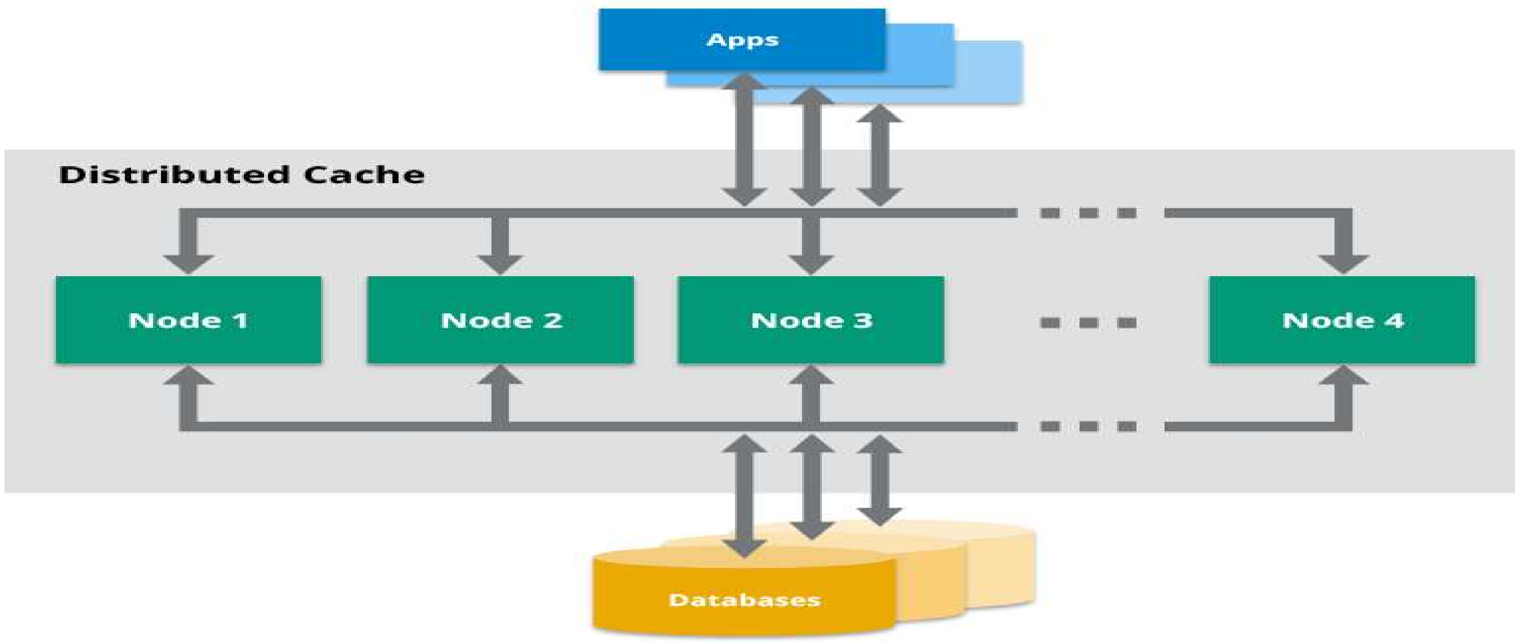
Types of cache

Different types of caching strategies are used depending on system architecture, data consistency needs, and workload patterns.

- Server side
- Client side
- Application-Level Cache
- Distributed Cache

Distributed Caches

- A distributed cache may span multiple servers so that it can grow in size and in transactional capacity.
- It is mainly used to store application data residing in database and web session data.

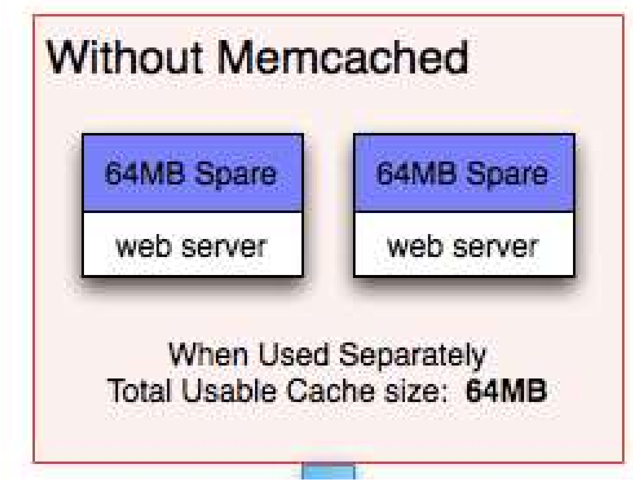


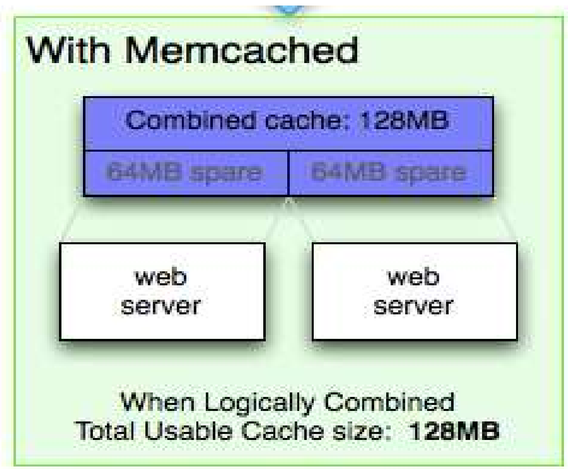
Advantages of Distributed Cache

When cached data is distributed, the data:

- Is *coherent* (consistent) across requests to multiple servers.
- Survives server restarts and app deployments.
- Doesn't use local memory.

Example: Memchached





Global Caches

- **Global Cache** refers to a caching mechanism that operates across multiple geographic regions or data centers, ensuring faster data access and improved system scalability.

Key Characteristics of Global Cache:

- ✓ Distributed Across Multiple Regions
- ✓ Reduces Latency
- ✓ Load Balancing
- ✓ High Availability

Google Global Cache (GGC)

- **Google Global Cache (GGC)** is a specialized caching system deployed by **Google in Internet Service Providers' (ISPs) networks** to improve the delivery of Google services such as:
  - YouTube
  - Google Search
  - Google Drive
  - Google Play Store
- GGC is part of **Google's Content Delivery Network (CDN)**, designed to **reduce bandwidth costs and improve access speed** by caching content **closer to users**.

innovate achieve lead

BITS Pilani
Pilani Campus

Scalability features in the Cloud

Auto-scaling

- Auto Scaling is a technique used in **cloud computing and distributed systems** to **dynamically adjust computing resources** based on real-time demand.

Types of Auto Scaling

- **Vertical Scaling** is defined as increasing a single machine's capacity with the rising resources in the same logical server or unit. .

- **Horizontal Scaling** is an approach to enhance the performance of the server node by adding new instances of the server to the existing servers to distribute the workload equally

Image: Google

Comparison

Vertical Vs Horizontal Scaling	Vertical scaling	Horizontal Scaling
Data	Data is executed on a single node	Data is partitioned and executed on multiple nodes
Data Management	Easy to manage – share data reference	Complex task as there is no shared address space
Downtime	Downtime while upgrading the machine	No downtime
Upper limit	Limited by machine specifications	Not limited by machine specifications
Cost	Lower licensing fee	Higher licensing fee

Image: Google

What is virtualization?

- Virtualization is a technology that allows multiple **virtual instances** (e.g., virtual machines, containers) to run on a single **physical machine** by abstracting hardware resources.

Types of virtualization

- Server virtualization
- Storage virtualization
- Data virtualization
- Application virtualization
- Containerization

……….

How Virtualization Improves Scalability?

- On-Demand Resource Allocation
- Efficient Load Balancing
- Multi-Tenancy Support
- High Availability & Fault Tolerance
- Cost-Effective Scaling

What is Serverless Computing?

- Serverless computing is a cloud computing model where cloud providers automatically manage the infrastructure and dynamically allocate resources as needed.

Key Characteristics of Serverless Computing

- No Server Management
- Event-Driven
- Automatic Scaling
- Pay-as-You-Go

How Serverless Computing Enhances Scalability?

- Auto Scaling Based on Demand
- Instant Scaling and Elasticity
- Cost Efficiency
- Statelessness
- Managed Load Balancing

Types of Serverless Architectures for Scalability

- **Function-as-a-Service (FaaS):** developers write individual functions that are executed in response to specific events
- **Backend-as-a-Service (BaaS):** use of managed backend services that are provided by cloud vendors

Challenges of Serverless Computing for Scalability

- ✕ Cold Start Latency
- ✕ State Management
- ✕ Vendor Lock-In
- ✕ Debugging and Monitoring

Best Practices for Achieving Scalability

- Be Stateless
- Use Load Balancers
- Auto Scaling
- Optimize Database Performance
- Use Caching Mechanisms
- Microservices Architecture
- Event-Driven Architecture
- Use Serverless Computing
- Monitor and Analyze Performance

By following these best practices, you can build highly scalable applications that handle increased demand without compromising performance or reliability.

Case Study: Amazon Prime

- Amazon Prime Video Uses AWS to Deliver Solid Streaming Experience to More Than 18 Million Football Fans
- Amazon Prime needed an architecture that could quickly scale, handle spikes, and have sufficient caching in different layers to manage the demand.
  - Scalable Cloud Infrastructure
  - Content Delivery with Amazon CloudFront
  - High-Performance Video Encoding and Storage
  - Serverless Technology
  - Real-Time Analytics with Amazon Kinesis
  - Global Availability and Reliability
  - Enhanced Security and Compliance

https://aws.amazon.com/solutions/case-studies/amazon-prime-video/

Self Study

- Amazon Prime case study: https://aws.amazon.com/solutions/case-studies/amazon-prime-video/
- Hotstar case study: https://laveena-j-21.medium.com/cloud-computing-aws-and-disney-hotstar-case-study-f4a3be4669a
- Article on serverless: https://www.simform.com/blog/serverless-architecture-guide/

References

- Book: The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise, Second Edition by Michael T. Fisher; Martin L. Abbott Published by Addison-Wesley Professional, 2015
- https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-concepts-replica-lifecycle
- https://www.citrix.com/en-in/solutions/application-delivery-controller/load-balancing/what-is-load-balancing.html#:~:text=Load%20balancing%20is%20defined%20as,server%20capable%20of%20fulfilling%20them.
- https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs#:~:text=CQRS%20stands%20for%20Command%20and,performance%2C%20scalability%2C%20and%20security.
- https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture
- https://memcached.org/
- https://www.ibm.com/docs/en/integration-bus/10.0?topic=caching-data-overview
- https://support.google.com/interconnect/answer/9058809?hl=en
- https://aws.amazon.com/
- https://www.nginx.com/resources/glossary/load-balancing/
- https://www.ibm.com/in-en/cloud/learn/virtualization-a-complete-guide
- https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/
- https://aws.amazon.com/solutions/case-studies/amazon-prime-video/