In-Memory Data Grids

Mini Workshop Oct 5, 2018

Agenda

1. Introduction to In-Memory Data Grid
2. In-Memory Data Grid Architecture
3. Possible Applications & Use Cases
4. Hands-on HazelCast: Using an Open Source In-Memory Data Grid

What is In-Memory Data Grid (IMDG)?

- IMDG provide a **lightweight, distributed, scale-out in-memory object store** — the data grid. Multiple applications can concurrently perform transactional and/or analytical operations in the low-latency data grid, thus minimizing access to high-latency, hard-disk-drive-based or solid-state-drive-based data storage. IMDGs maintain data grid durability across physical or virtual servers via replication, partitioning and on-disk persistence. Objects in the data grid are uniquely identified through a primary key, but can also be retrieved via other attributes. [By Gartner]

What is In-Memory Data Grid (IMDG)?

- IMDG is a **data structure** that resides entirely in **RAM** (random access memory) and is distributed among **multiple nodes** (servers) over a computer network.
- Because of recent development in 64-bit and multi-core architectures it possible to store terabytes of data completely in RAM

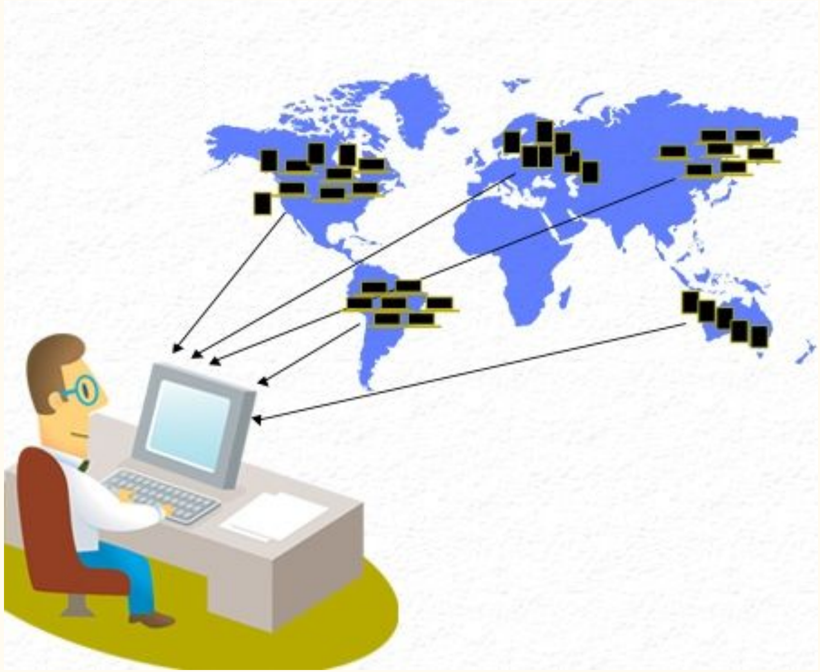
Why In-Memory Data Grid (IMDG)?

An in-memory data grid supports:

- **High throughput low-latency** application.
- **Horizontally scalable**
- **Highly-available data storage layer**
- **Resilience data model**

What is Data Grid

A data grid is a data architecture that gives individuals or groups of clients the ability to access, modify and transfer huge amounts of geographically distributed data

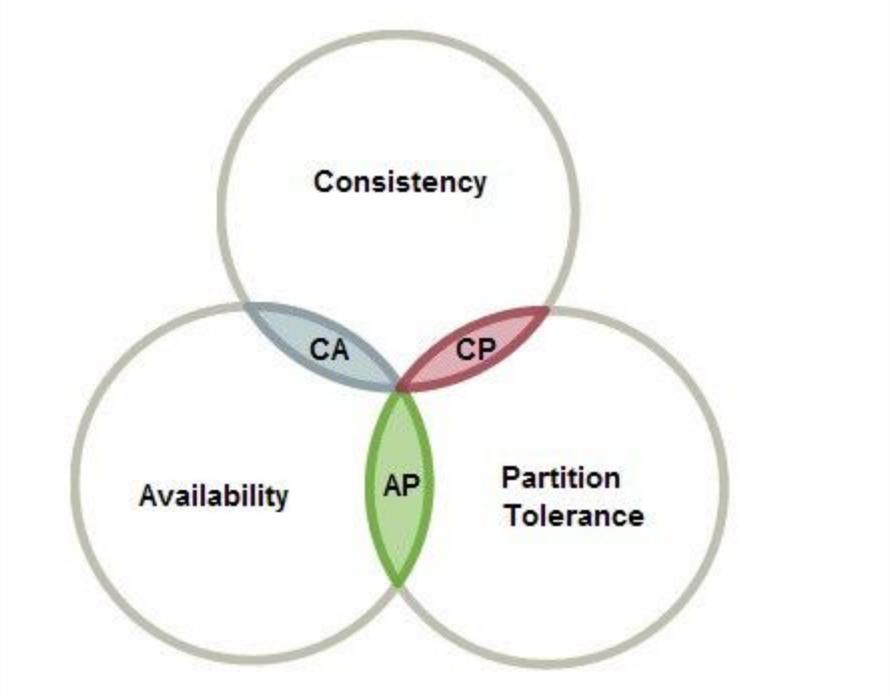


When Not to Use an In-Memory Data Grid

- The amount of data is small/tiny
- Low-Latency is not a hard requirement
- In your application you could **NOT** make a trade off’s between consistency, availability, and partition tolerance.

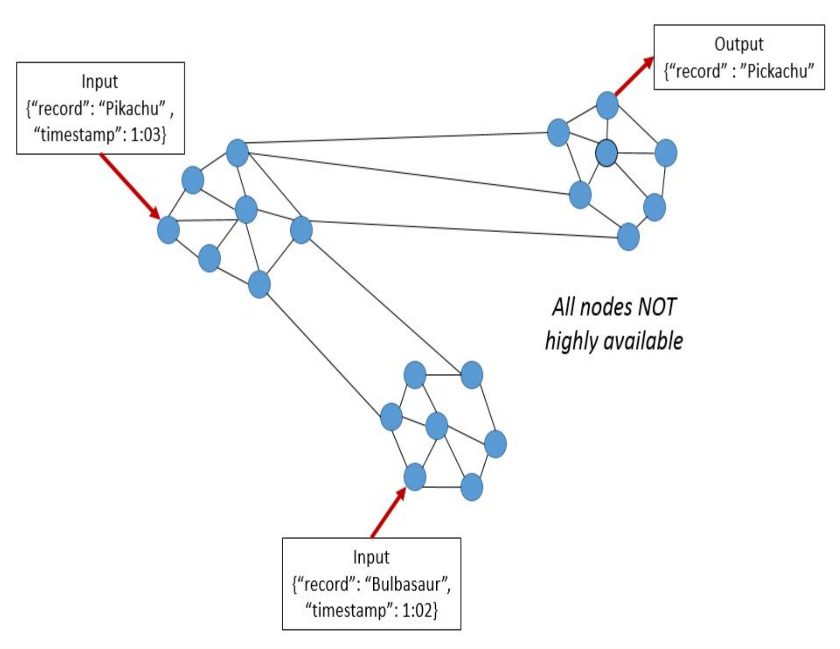
CAP Theorem

- CAP Theorem is a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.
- CAP Theorem is very important in the Big Data world



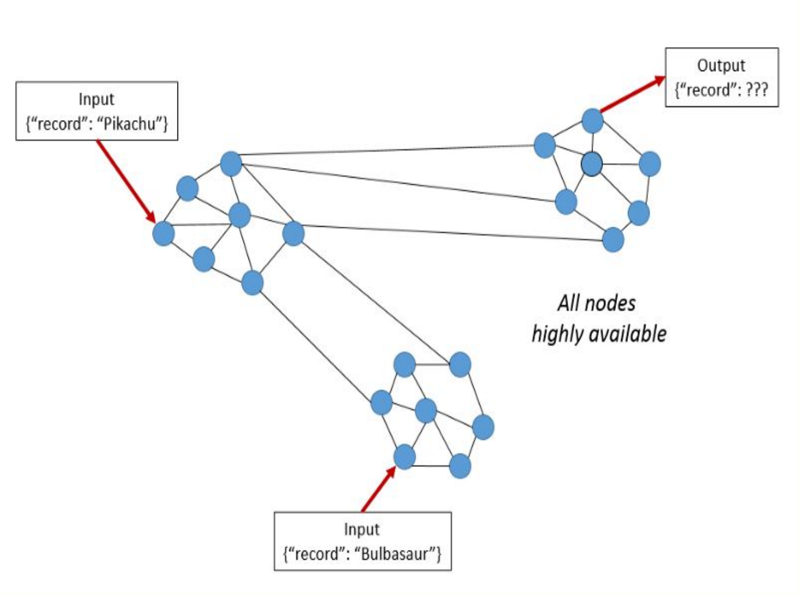
CAP Theorem: Consistency

- This condition states that all node see the same data at the same time.
- When a node is performing a read operation, it will read the value of the most recent write operation causing all nodes to return the same data.
- A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state.



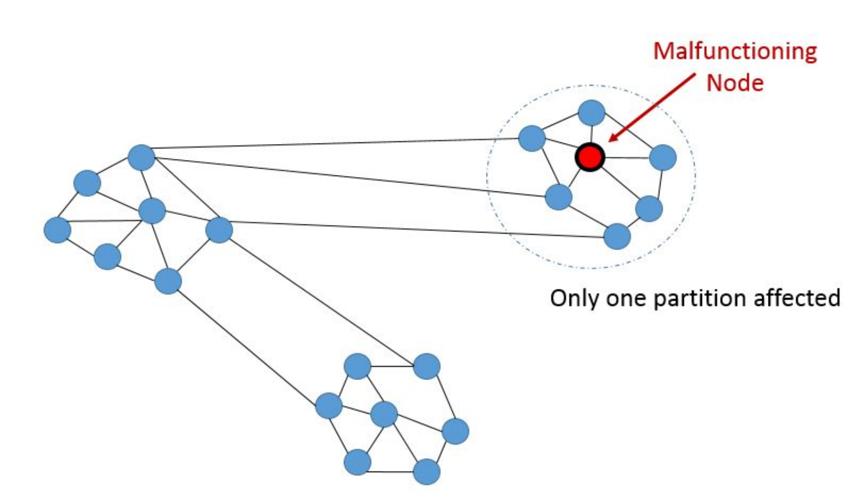
CAP Theorem: Availability

- This condition states that every request receives a response that is not an error
- Achieving availability in a distributed system requires that the system remains operational 100% of the time.
- Every client gets a response, regardless of the state of any individual node in the system.

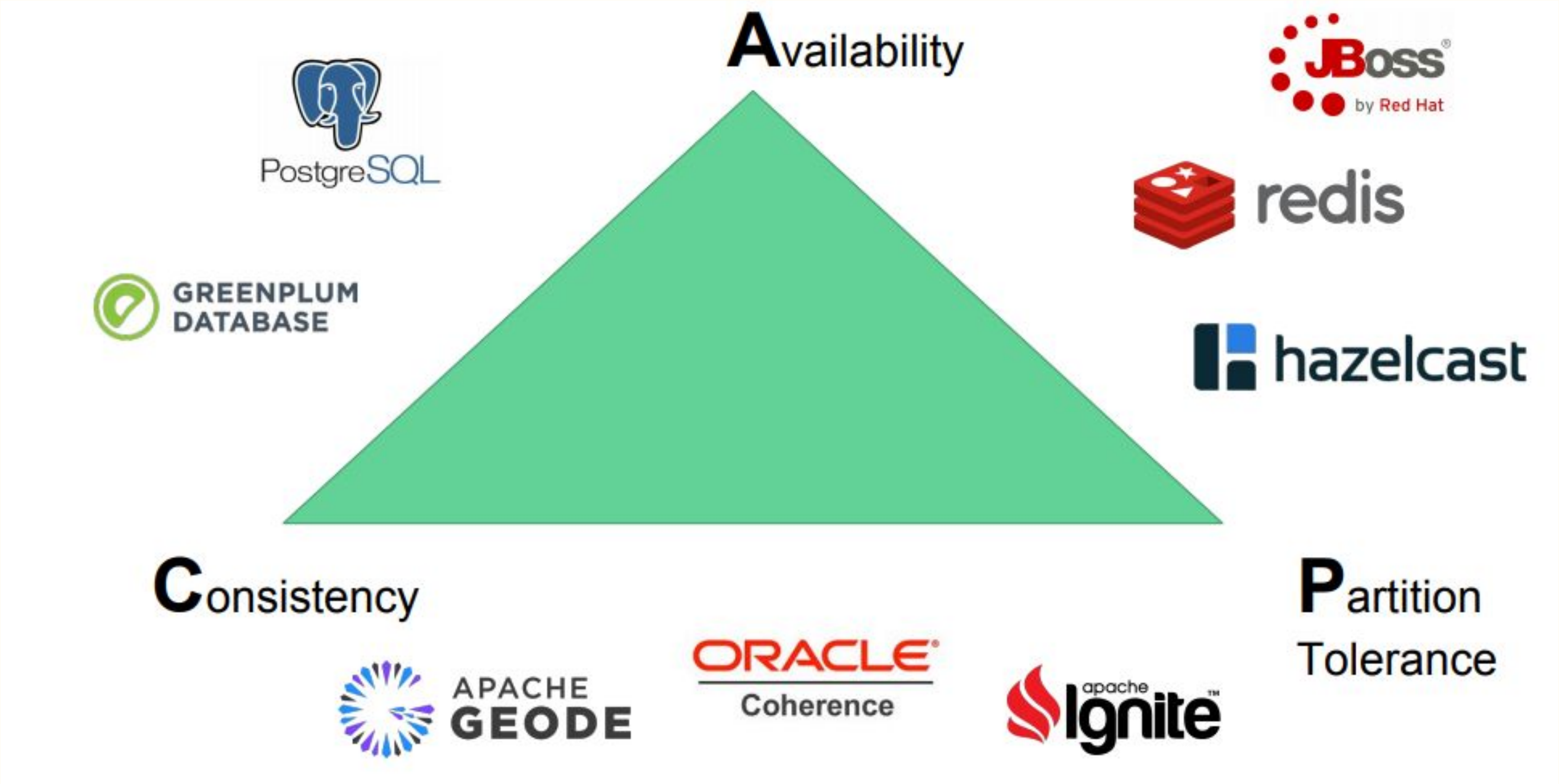


CAP Theorem: Partition Tolerance

- This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes.
- A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network.
- Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.



IDMG Solutions and CAP Theorem



IMDG Architecture

Design Requirements for IMDG

1. **Low-Latency**
2. **High Throughput**
3. **Resilience/ Fault-Tolerance**
4. **Store Large Data**
5. **Scalability**
6. **Elasticity**

Data in Primary Storage

- To achieve low latency we store the data into the primary storage random access memory (RAM).
- Reduce or eliminate disk I/O operations
- Solutions for data persistence are required

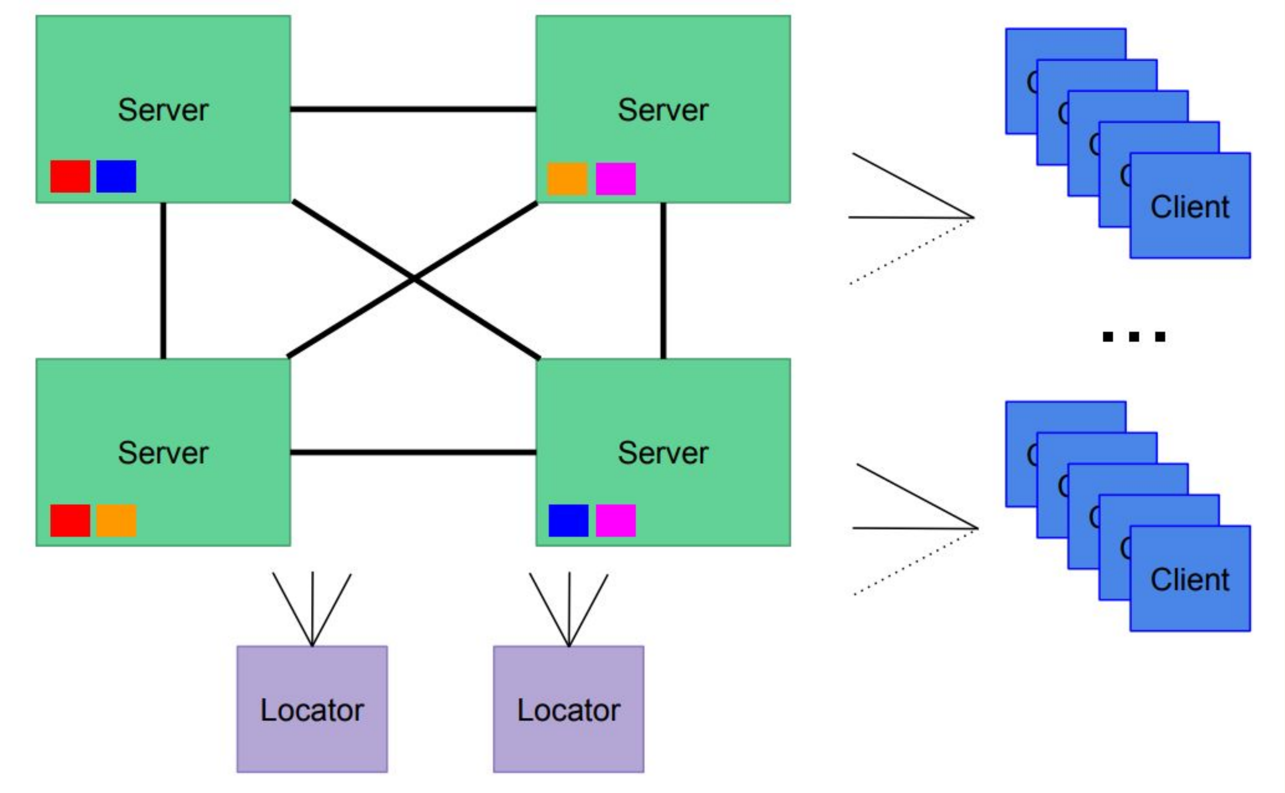
Latency Numbers Every Programmer Should Know

Latency Comparison Numbers

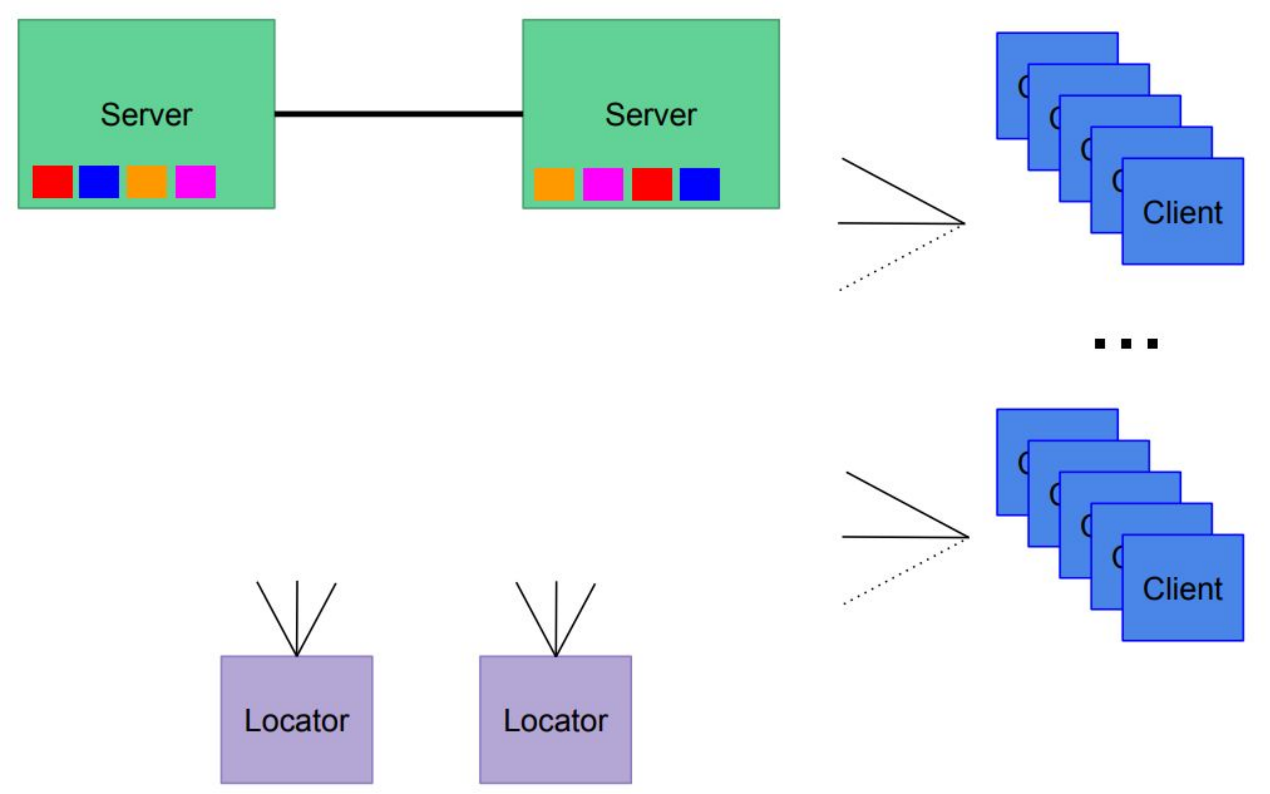
L1 cache reference	0.5 ns			
Branch mispredict	5 ns			
L2 cache reference	7 ns			14x L1 cache
Mutex lock/unlock	25 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	3 us		
Send 1K bytes over 1 Gbps network	10,000 ns	10 us		
SSD Seek	100,000 ns	100 us		
Read 4K randomly from SSD*	150,000 ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	**250,000 ns**	**250 us**		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from SSD*	**1,000,000 ns**	**1,000 us**	**1 ms**	**~1GB/sec SSD, 4X memory**
Disk seek	10,000,000 ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	**20,000,000 ns**	**20,000 us**	**20 ms**	**80x memory, 20X SSD**
Send packet CA->Netherlands->CA	150,000,000 ns	150,000 us	150 ms	

[1] Credit Jeff Dean, Peter Norvig, and Jonas Bonér

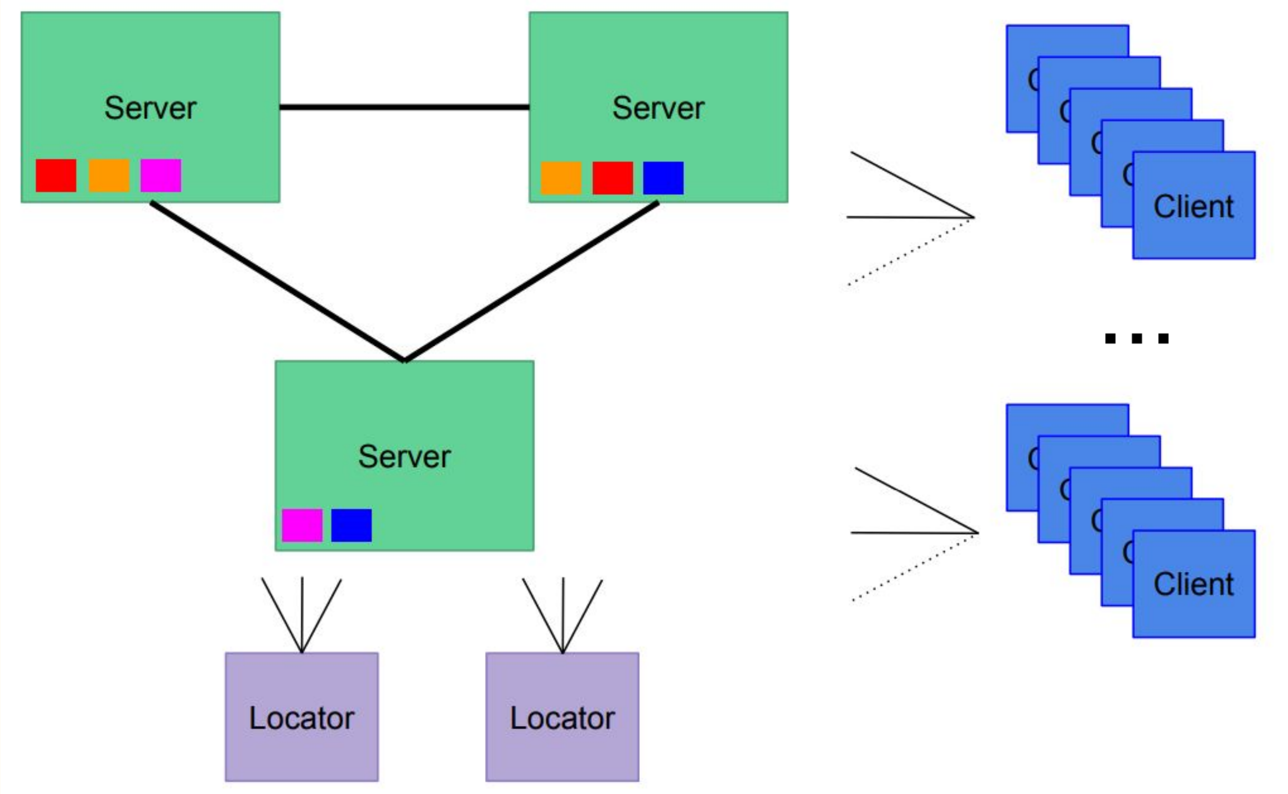
Scalability and Elasticity



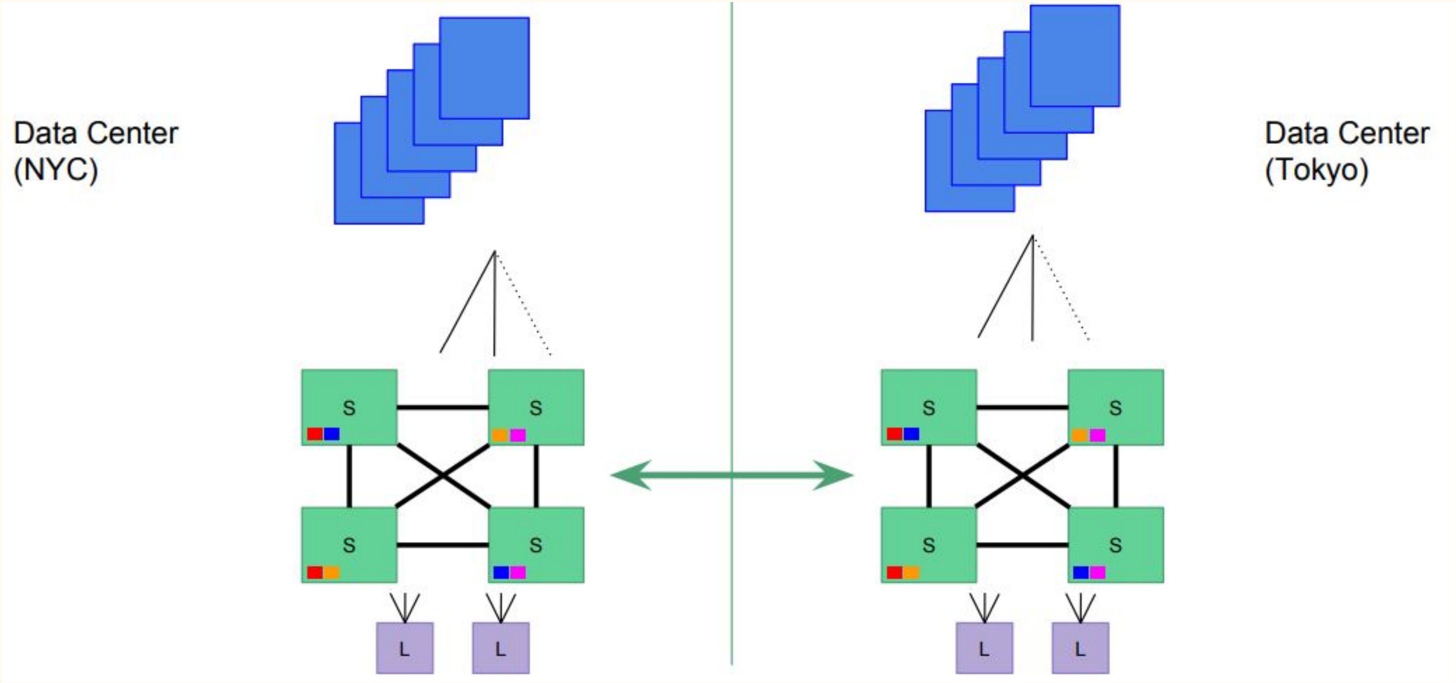
Scalability and Elasticity



Scalability and Elasticity



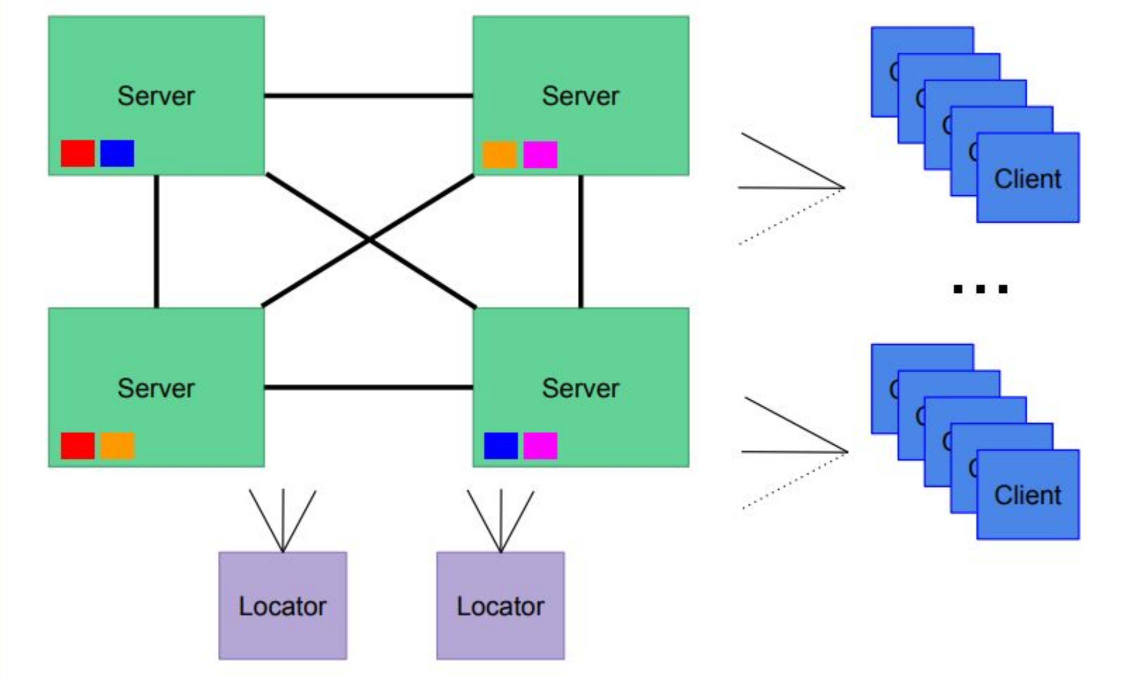
Replications Over WAN



Data Aware Routing

Client **A** asking, where is the red rectangle?

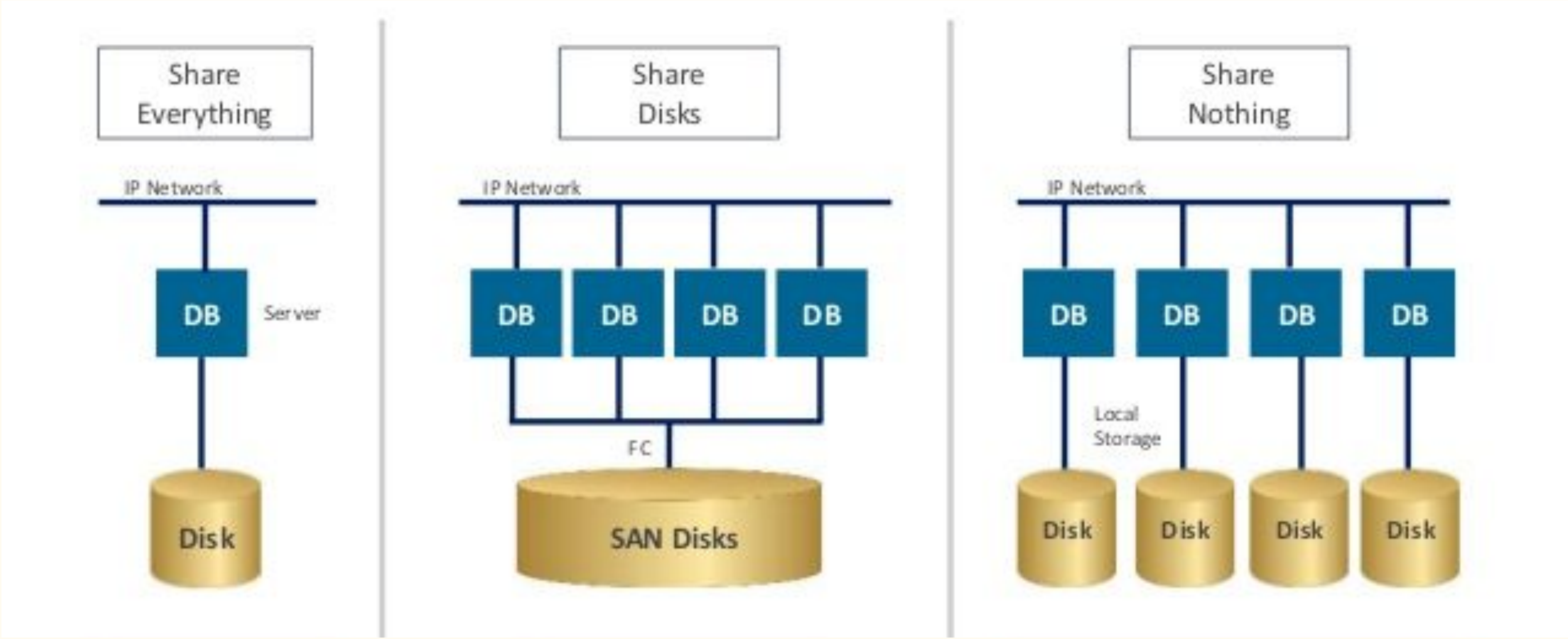
Should client A cache the red rectangle?



Shared Nothing Architecture

- Shared-nothing architecture (SNA) is a pattern used in distributed computing in which a system is based on multiple self-sufficient nodes.
- The nodes have their own memory, HDD storage, and independent input/output interfaces.
- Each node shares no resources with other nodes, and there is a synchronization mechanism that ensures that all information is available on at least two nodes.
- Supports almost infinite horizontal scaling that can be made with very inexpensive hardware

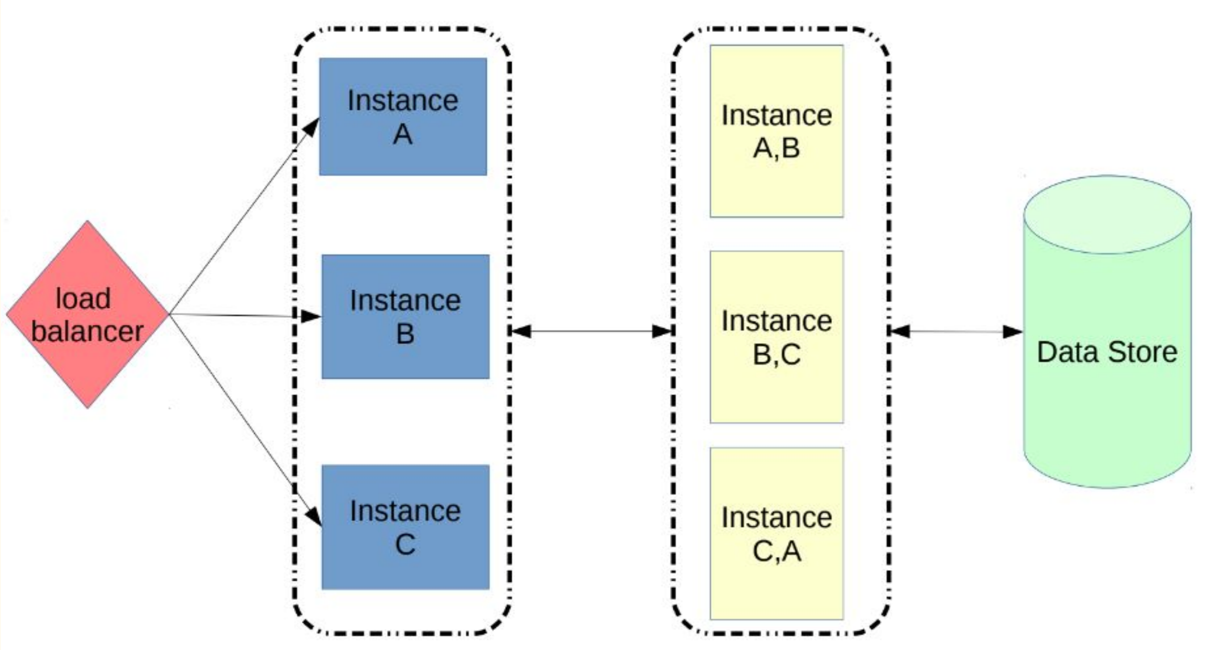
Shared Nothing Architecture



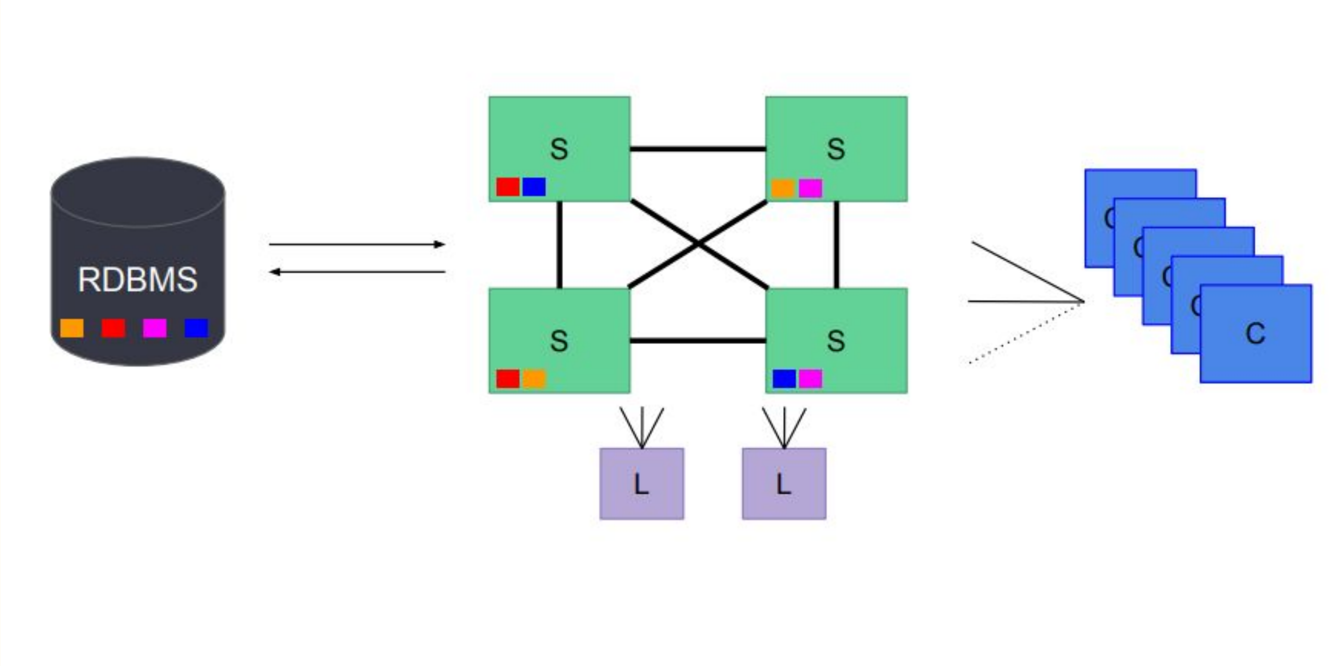
Use Cases & Applications

Distributed Near Cache Architecture

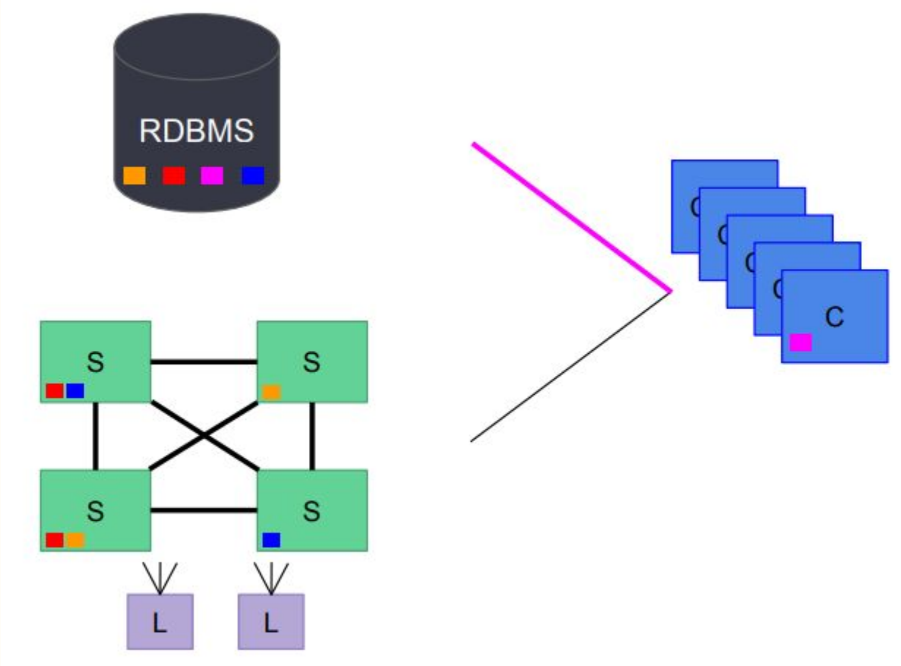
In applications with massive number of transactions per seconds



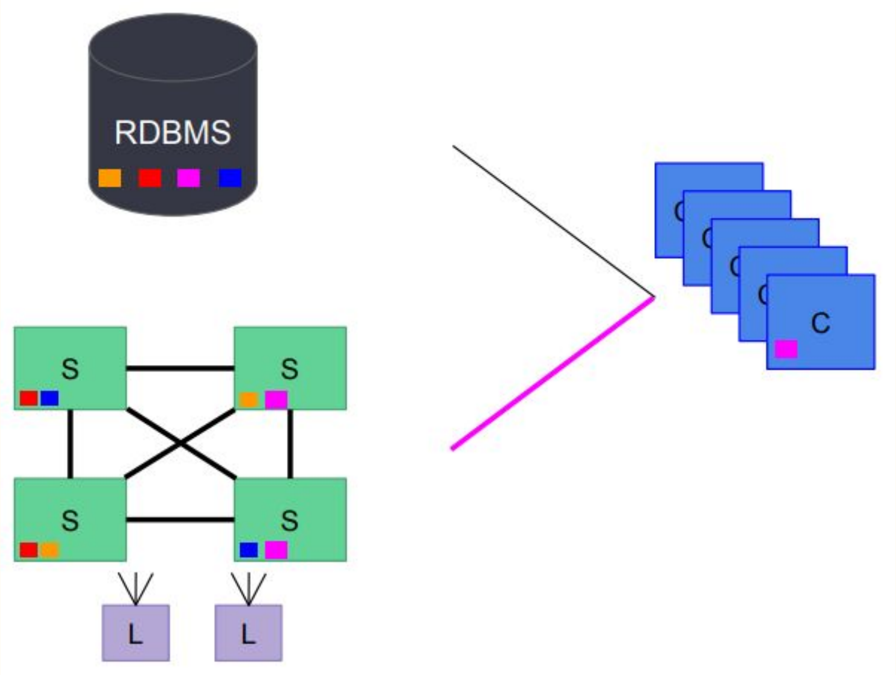
Inline Caching



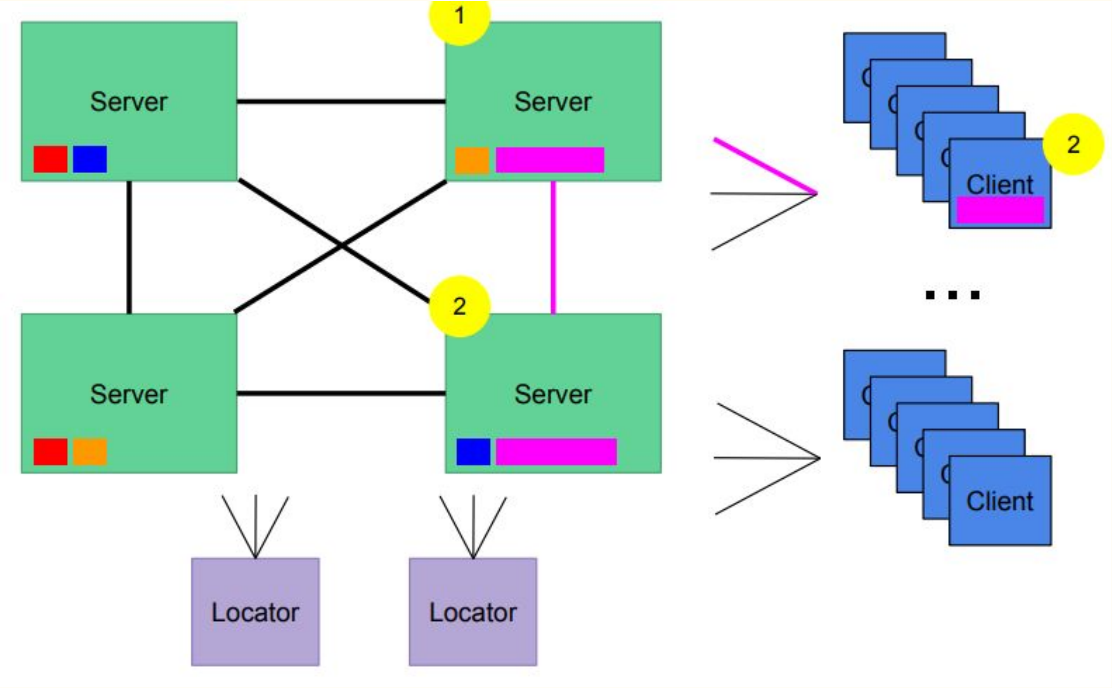
Look-Aside Caching



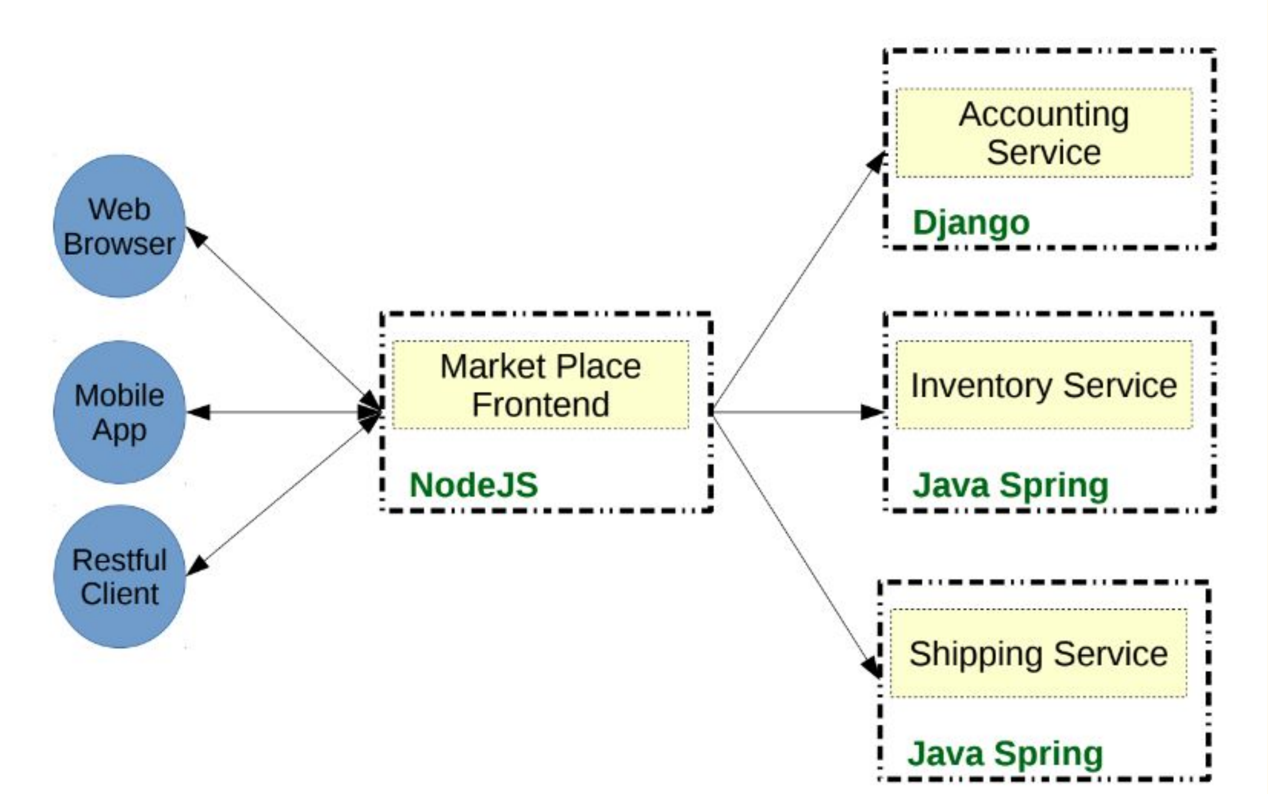
Look-Aside Caching



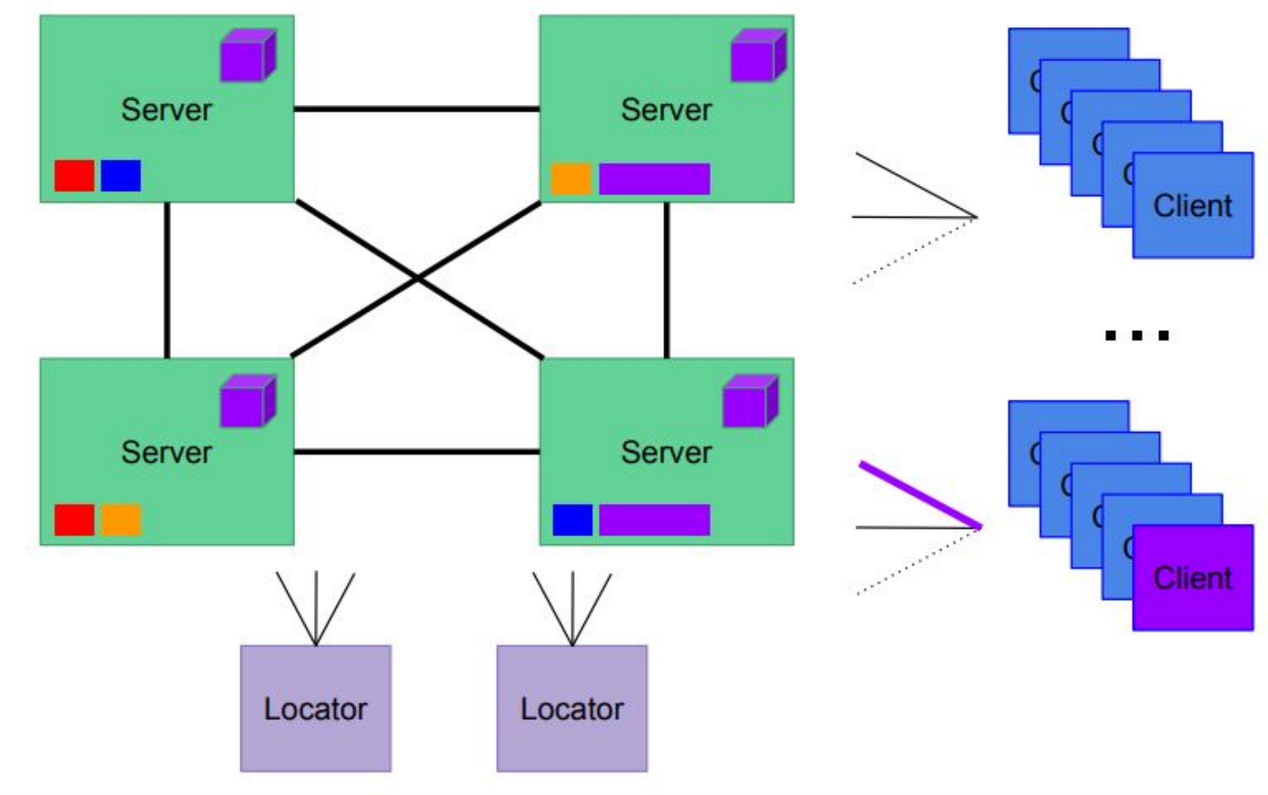
Pub/Sub System



Microservices Architecture



Real-Time Analytics with Functions



Hazelcast

What is Hazelcast?

Hazelcast is an open source in-memory data grid written in Java

An IMDG with shared nothing architecture

Hazelcast can run on-premises, in the cloud (Amazon Web Services, Microsoft Azure, Cloud Foundry, OpenShift), virtually (VMware), and in Docker containers.

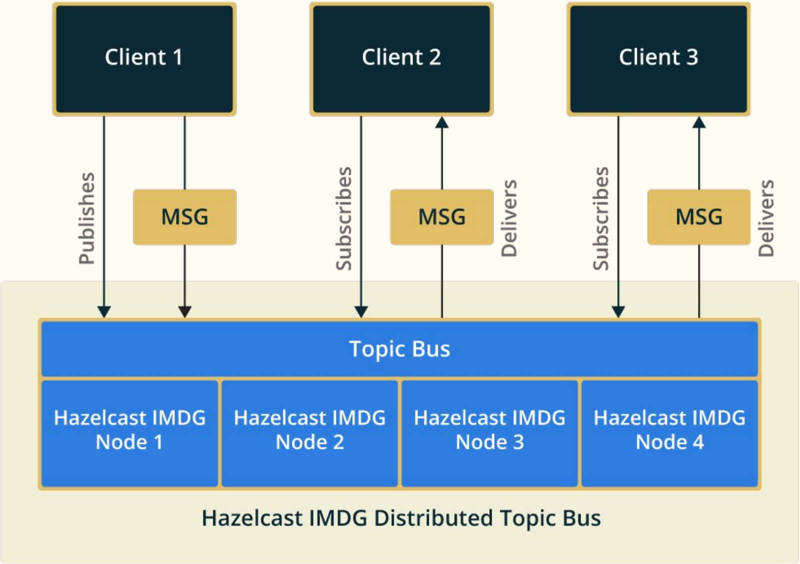
Hazelcast support different programming language, such as Java, C#, C++, Go, Python, NodeJS, and Scala.

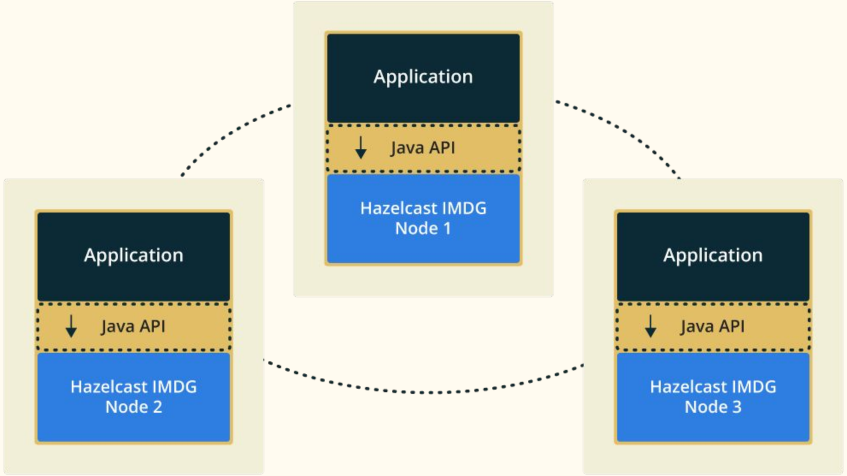
Hazelcast Key Characteristics

- The data is always stored in-memory (RAM) of the servers
- Multiple copies are stored in multiple machines for automatic data recovery in case of single or multiple server failures
- The data model is object-oriented and non-relational
- Servers can be dynamically added or removed to increase the amount of CPU and RAM
- The data can be persisted from Hazelcast to a relational or NoSQL database
- A Java Map API accesses the distributed key-value store

Using Hazelcast In Your Application

There are two options to use hazelcast in your application either using a client server architecture or an embedded architecture.





Working with Hazelcast

- What do you need?
  - Java JDK version 6 or higher (recommend 8)
  - Java IDE or Text Editor
  - Network Connection (optional)
- Steps:
  1. Download Hazelcast JARs from: https://hazelcast.org/download/
  2. Unzip it and add the lib/hazelcast-3.x.jar to your class path.
  3. Create a Java class and import Hazelcast libraries.

Hello World Example

```
import java.util.Map;
import java.util.Queue;

import com.hazelcast.config.Config;
import com.hazelcast.core.Hazelcast;
import com.hazelcast.core.HazelcastInstance;

public class MyHazelCastApp {

    public static void main(String[] args) {

        Config config = new Config();
        HazelcastInstance hazelcast = Hazelcast.newHazelcastInstance(config);
        Map<Integer, String> myMap = hazelcast.getMap("customers");
        myMap.put(3, "Alex");
        myMap.put(2, "Alice");
        myMap.put(7, "Zack");

        System.out.println("Customer with key 2: "+ myMap.get(3));
        System.out.println("The current map size is: "+myMap.size());
        Queue<String> queueCustomers = hazelcast.getQueue("customers");
        queueCustomers.offer("Tom");
        queueCustomers.offer("Mary");
        queueCustomers.offer("Jane");
        System.out.println("First customer: " + queueCustomers.poll());
        System.out.println("Second customer: "+ queueCustomers.peek());
        System.out.println("Queue size: " + queueCustomers.size());
    }

}
```

Hello World Example

To compile and run this example from command line

1. Make sure to place MyHazelCastApp.java and the **hazlecast.jar** in the same directory/folder
2. Compile the code using the following command

   javac -cp ".:hazelcast.jar" MyHazelCastApp.java

3. Run the code using the following command

   java -cp ".:hazelcast.jar" MyHazelCastApp

Hello World Example

```
ebinsaad@blacknode: ~/R&D/eclipse/workspace/example_01
Oct 05, 2018 4:24:26 AM com.hazelcast.internal.diagnostics.Diagnostics
INFO: [137.207.64.229]:5701 [dev] [3.10.5] Diagnostics disabled. To enable add -Dhazelcast.diag
rguments.
Oct 05, 2018 4:24:26 AM com.hazelcast.core.LifecycleService
INFO: [137.207.64.229]:5701 [dev] [3.10.5] [137.207.64.229]:5701 is STARTING
Oct 05, 2018 4:24:29 AM com.hazelcast.system
INFO: [137.207.64.229]:5701 [dev] [3.10.5] Cluster version set to 3.10
Oct 05, 2018 4:24:29 AM com.hazelcast.internal.cluster.ClusterService
INFO: [137.207.64.229]:5701 [dev] [3.10.5]

Members {size:1, ver:1} [
        Member [137.207.64.229]:5701 - 06ba3c81-6ea9-4a46-9f5f-0464fe6efffa this
]

Oct 05, 2018 4:24:29 AM com.hazelcast.core.LifecycleService
INFO: [137.207.64.229]:5701 [dev] [3.10.5] [137.207.64.229]:5701 is STARTED
Oct 05, 2018 4:24:29 AM com.hazelcast.internal.partition.impl.PartitionStateManager
INFO: [137.207.64.229]:5701 [dev] [3.10.5] Initializing cluster partition table arrangement...
Customer with key 2: Alex
The current map size is: 3
First customer: Tom
Second customer: Mary
Queue size: 2
```

Hello World Example

Open another command line and run the code again, what is different this time?

```
ebinsaad@blacknode: ~/R&D/eclipse/workspace/example_01
Oct 05, 2018 4:28:12 AM com.hazelcast.nio.tcp.TcpIpConnector
INFO: [137.207.64.229]:5702 [dev] [3.10.5] Connecting to /137.207.64.229:5701, t
imeout: 0, bind-any: true
Oct 05, 2018 4:28:12 AM com.hazelcast.nio.tcp.TcpIpConnectionManager
INFO: [137.207.64.229]:5702 [dev] [3.10.5] Established socket connection between
 /137.207.64.229:50377 and /137.207.64.229:5701
Oct 05, 2018 4:28:18 AM com.hazelcast.system
INFO: [137.207.64.229]:5702 [dev] [3.10.5] Cluster version set to 3.10
Oct 05, 2018 4:28:18 AM com.hazelcast.internal.cluster.ClusterService
INFO: [137.207.64.229]:5702 [dev] [3.10.5]

Members {size:2, ver:2} [
        Member [137.207.64.229]:5701 - 06ba3c81-6ea9-4a46-9f5f-0464fe6efffa
        Member [137.207.64.229]:5702 - 1e58a05d-2b72-482b-927d-4033299cea89 this
]

Oct 05, 2018 4:28:19 AM com.hazelcast.core.LifecycleService
INFO: [137.207.64.229]:5702 [dev] [3.10.5] [137.207.64.229]:5702 is STARTED
Customer with key 2: Alex
The current map size is: 3
First customer: Mary
Second customer: Jane
Queue size: 4
```

Add More Nodes to the Grid/Cluster

1. Make sure you are connected to the computer science wireless network
2. Run the Hello World

What did you notice this time?

Listen 2 Events in Distributed Collection

In example 2 we will implement a map listener, so when a new item added, removed, or updated, we receive a notification and take some actions.

Hazelcast Configuration

Hazelcast can be configured through xml or using configuration api or even mix of both.

Mini Project

Implement a Message Delivery System Using Hazelcast (40 minutes)

References

1. What the Heck is an In-Memory Data Grid? - Addison Huddy | Pivotal
2. CAP Theorem and Distributed Database Management Systems - Syed Sadat Nazrul
3. Mastering Hazelcast IMDG- EBook by Hazelcast team

Thank U