Big Data Engineer 实战训练营
课程总结

Distributed System

1. Distributed System Difficulties

a. NODE COORDINATION:

Because nodes normally share nothing other than network, coordination is hard due to network issue, clock drift, and difference in processing power.

b. NETWORK FAILURE:

Always consider network will fail in distributed system, and provide mechanisms to recover.

c. NODE FAILURE:

Always consider node will fail in distributed system, and provide mechanisms to recover, such as data replication, commit-logs.

2. Consensus Algorithm and Protocol

a. ZAB:

A specially designed protocol in Zookeeper to reliably replicate among different nodes, one key take away is how two phase commits work and only commit change once majority of nodes reply.

b. GOSSIP PROTOCOL:

A protocol to detect node failure or pass information among nodes. Gossip Protocol normally has TTL on messages to avoid gossip storm.

3. Common Architecture

a. MASTER-SLAVE ARCHITECTURE:

Very common architecture for distributed system, because it fits perfectly with Divide-and-Conquer model. Also Master-Slave Architecture made coordination a little bit easier, because there's a single source of instructions. However, the capacity of the overall system is limited by Master. Also, there is single point of failure.

b. PEER-TO-PEER ARCHITECTURE:

A common architecture for distributed system. Because there is no Master, there is no single point of failure. However, this made coordination hard, specially designed protocol is required for coordination, for example, gossip protocol.

4. High Availability Strategy

a. DATA REPLICATION:

Data is the most important thing for every system, in order to keep data highly available, data are replicated among nodes to keep data redundant.

b. DISASTER RECOVERY:

Because nodes will fail, disaster recovery is required to keep the system running, common strategies are, commit-logs, backup master, etc.

Open-source Big Data Frameworks

1. Apache Zookeeper

a. TARGETED PROBLEM:

Distributed System Coordination

b. APACHE ZOOKEEPER ARCHITECTURE

i. Master-Slave Architecture:

Zookeeper cluster has master and slave, with master, zookeeper ensure all the updates go through one node and keep FIFO guarantee of znode updates.

ii. Client-Server Architecture:

Zookeeper client can connect to any Zookeeper server.

c. INTERNAL IMPLEMENTATION

i. ZAB Protocol:

Zookeeper relies on ZAB protocol to replicate internal data among nodes. Because ZAB requires majority nodes to reply, Zookeeper deployments must have odd number of nodes. Because of ZAB protocol, writes to Zookeeper are slow.

ii. Znode:

Internal storage structure for Zookeeper. Znode has four types, persist znode, ephemeral znode, sequential persist znode, and sequential ephemeral znode.

d. API AND USAGE:

Zookeeper provides super simple API to manipulate znode, similar to file system operations. User of zookeeper can use these four types of znode to implement coordination logic.

2. Apache Kafka

a. TARGETED PROBLEM:

Data Transportation

b. APACHE KAFKA ARCHITECTURE

i. Master-Slave Architecture:

For topics in Kafka cluster, every topic has a leader to make sure updates have FIFO guarantee. Also topic has followers as a way to provide high availability.

ii. Client-Server Architecture:

Kafka client can connect to any kafka cluster.

c. INTERNAL IMPLEMENTATION

i. Topic: Logical division of data in Kafka, user can publish to a kafka topic, and consumer can load from the same kafka topic.

ii. Partition: Physical division of Kafka topic, Kafka use partition to achieve parallelization.

iii. IO Optimization Techniques: Kafka used many techniques to optimize IO operations. The log files are read/write sequentially to reduce hard disk seek. Kafka also use Java NIO API which has zerocopy capability to avoid copying data up/down userspace and kernel space in Operation System.

d. API AND USAGE:

Kafka provides simple API to publish and consume data. One thing to keep mind is to maintain consumer offset to be able to recover from consumer failures.

3. Apache Cassandra

a. TARGETED PROBLEM: DATA STORAGE

b. APACHE CASSANDRA ARCHITECTURE

i. Peer-to-peer Architecture: Cassandra use peer-to-peer node rather than master-slave mode to achieve high scalability, and erase single point of failure.

ii. Client-Server Architecture: Cassandra client can connect to any Cassandra cluster.

c. INTERNAL IMPLEMENTATION

i. Consistent Hashing: Cassandra uses consistent hashing algorithm to distribute data among cassandra nodes. By default, murmur3 hashing algorithm is being used because it is fast and has good data distribution.

ii. Gossip Protocol: Cassandra use gossip protocol to detect node failure and pass node information.

iii. Data Storage Strategy: Data in Cassandra is firstly being written to commit log for disaster recovery, then data is write to memtable to speed up operation such query, finally data is write to SSTable file, which is also a sequential change file to speed up IO.

iv. Data Compaction: Rather than updating records in SSTable file, data and updates are constantly being appended to SSTable file. So there will be old data in SSTable file, also given time, data might got written to different SSTable file. So Cassandra use data compaction to merge SSTable files and get rid of old SSTable file.

4. Apache Spark

a. TARGETED PROBLEM: DATA COMPUTATION

b. APACHE SPARK ARCHITECTURE

i. Master-Slave Architecture: Spark uses master slave architecture to process data, master distributes tasks into slaves, and slaves in charge of executing the tasks.

ii. Client-Server Architecture: Spark client can connect to any Spark cluster and submit spark computation jobs.

c. INTERNAL IMPLEMENTATION

i. RDD: Resilient Distributed Dataset is the abstraction Spark provided around computation data. RDD is immutable and readonly. RDD stores metadata for the actual data and computation steps. Because of these metadata, computation in Spark is resilient since you can reply computation if required.

ii. Lazy-Evaluation: Computation in Spark is not applied immediately unlike Hadoop. RDD transformations are stored and only trigger computation when Spark action is called.

5. Apache Mesos

a. TARGETED PROBLEM: DISTRIBUTED SYSTEM SCHEDULING

b. APACHE MESOS ARCHITECTURE

i. Master-Slave Architecture: Mesos use classical master slave architecture, Mesos master collects resource information from Mesos slaves and allocate resources to frameworks, slaves are in charge of executing scheduled tasks and report task status.

c. INTERNAL IMPLEMENTATION

i. Framework: Frameworks are distributed application run on top of Mesos framework, framework handles detailed scheduling strategy for specific workload, such as Hadoop framework.

ii. Resource Offer: Resources in the Mesos cluster are distributed in the format of resource offer, resource off has CPU, Memory, Disk size, ports, etc.

iii. Two-level Scheduling: Mesos uses two-level scheduling in order to keep the core logic simple and scalable. Also with two-level scheduling, Mesos is able to support all kinds of workloads.

iv. Dominant Resource Fairness: Because resources are shared within Mesos clustered, it is important to share resource fairly among different workloads. DRF is based on Max-min algorithm but for multiple resource types.

Tools

1. Docker

a. TARGETED PROBLEM: APPLICATION PACKAGING AND DISTRIBUTION

b. DOCKER IMPLEMENTATION

i. Docker Image: Corner stone of Docker. Docker image has multiple layers, and docker uses union file system to implement docker image where multiple containers can share layers among them.

ii. Image layer cache: Docker internally records layers and difference among layers. Docker also maintains a database describing relationships among layers.

iii. Namespace: Docker uses namespace functionality from Linux kernel to achieve resource separation among containers. Processes inside container think they have the whole world, but actually limited by namespace.

iv. Cgroup: Docker uses cgroup functionality from Linux kernel to achieve resource limitation among containers. Process resources are controlled and monitored by cgroup.

2. Redis

a. REDIS API AND USAGE:

Redis is a very lightweight and fast data structure. Can be used to implement various components such as Cache, Message Queue, etc.

3. Node.js

a. NODE.JS API AND USAGE:

Node.js is a server framework to develop high performance web applications. It uses single-thread async IO to handle huge number of requests.

b. SYNCHRONOUS IO VS ASYNCHRONOUS IO:

Async IO help reduce wait time for clients.

c. MULTI-THREAD VS SINGLE-THREAD:

Multi-thread can help you to solve problems in parallel, however the number of threads are limited by the server. However because web servers are normally IO intensive, multi-thread mode is not efficient. Single-thread with async io can help you process huge amount of IO intensive requests.

Programming Language

1. Shell Script: http://www.shellscript.sh/
2. Python: http://www.learnpython.org/
3. JavaScript: https://www.codecademy.com/learn/javascript

Common Patterns in Industry

1. Micro-service/SOA Architecture:

Instead of having monolithic application, which is hard to maintain and update, companies are starting to use micro-service/SOA architecture where the application is built with multiple tiny specialized services which can be updated and scale separately.

2. Common Big Data Pipeline Architecture:

Companies normally build big data pipeline with Message Queue, Storage Solution, Computation Solution. You can easily build one by picking one technology from those three categories.

3. Distributed Tracing:

Because it is hard to coordinate requests among distributed system even though every system has logging turned on, you need tracing to thread requests throughout their lifecycles. A common tool is Zipkin, which is an opensource implementation of Google Dapper.

4. Use Avro/Protocol Buffer/Thrift to Optimize API:

JSON data is good since it can be processed by almost any language. But because it lacks schema, it is hard to communicate your changes to other teams. Using Avro/Protocol Buffer/ Thrift can help you define APIs and evolve API. Also JSON data is highly inefficient in terms of space, these tools help you cut down the data size transferred over the wire.

5. Common Deployment Model

a. **CONTINUOUS INTEGRATION:**

The process of integrating changes to into main code base continuously, with good test coverage and tools such as Jenkins/Travis. You can make sure your main code base always works.

b. **CONTINUOUS DEPLOYMENT:**

The process of deploying newer version of code into production continuously, with a consistent deployment model such as Docker, you can deliver latest working code to end user super quickly.