Masterclass: Workflow Systems

Traditional Systems

* Systems were designed keeping data at one place (RDBMS systems)
* RDBMS was not able to cater to growing data needs hence new class of databases (NoSQL systems) were born.
* As applications grew more complex, polyglot persistence came into existence. In polyglot persistence, we have the same data residing in multiple databases.
    * For example, consider user interaction data on your website. This data is collection on the Android / iOS application and is then sent to a server for persistence.
    * We may choose to run ML algorithms on this data for generating recommendations.
    * To serve these recommendation we might have to use a caching layer like redis
    * Hence, just to serve recommendations to user we ended up using S3 and Redis.
* When the applications grows in feature set and use case we introduce more and more types of databases.
* Not all databases can handle same throughout for reads and writes. Hence, the application writing to Cassandra for example will have lesser serves as compared to the one writing to RDBMS systems.
* How should we design the applications so that each application is able to scale out independently and reliably
* Here queues come into picture.
* We need a system which can hold the data and somehow other applications can read from it reliably. This is where Kafka comes into picture.

Components in Kafka

* Kafka at a fundamental level is a publisher subscriber system.
* Publishers sends the data to specific locations on the Kafka server.
* Interested and authorised subscribers can retrieve & process the msges
* In Kafka,
    * publishers -> producers
    * subscribers -> consumers
    * specific locations -> Topics

Topics

* collection holding similar msges
* they have a specific name, defined by the developers either upfront or on demand.
* As long as producers have access to topic they can send msges.
* Same goes with the consumers as well.

Brokers

* Kafka server -> Broker
* software service running as a demon on the OS.
* have access to resources on machine such as filesystem used to store msges in topics.
* can run more than 1 broker on 1 machine.
* to achieve higher throughput, run multiple instances of the same broker.
* A group of such brokers forms what we call a cluster.
* We use zookeeper in conjunction with cluster for:
    * Naming and Membership (Health Check)
    * Configurations
    * * Leader Election
        * *Attendance*: which nodes are healthy
        * *Work Items*: who is doing what
        * *Status*: track progress
* The leader receives a msg and delegates it to the correct node which is supposed to store / process the msges.
* All nodes keep communicating with each other using Gossip Protocol.

Topics revisited

* Topics are central abstraction of Kafka
* Zookeeper and leader knows on which node a particular topics is persisted.
* Topic is nothing but a file
* Since a file cannot be split into multiple servers, in absence of a partition, we are limited by system resources.
* To overcome this, Kafka provides portioned topics which are multiple files spreading over multiple machines.
* The data is written in FIFO format and is:
    * immutable
    * ordered by time
    * commit log
* Brokers gets to know whenever a topic is created and where all the partitions are created.
* When the leader is down zookeeper promotes one of the in-sync replica as the leader.

Producer

* fetches config from zookeeper at the start
* needs the key for identifying where data needs to be stored and value contains the actual msg
* decoder: which serialisation to use (String, CSV, JSON)
* Msges have
    * Partition

        * Timestamp
        * Key
        * Value

Routing strategies:

* DIRECT
* ROUND ROBIN (if no key is specified)
* KEY HASH (Consistent Hashing)
* CUSTOM (implement on your own)

* Msges are batched before it is send to broker
* Delivery Guarantees
    * Broker Asks
        * fire and forget
        * leader acknowledged
        * Quorum
    * Ordering Guarantees
        * Ordered with a partition
    * Retires are configurable.

Consumer

* fetches config from zookeeper at the start
* KV deserialisers
* A consumer can subscribe to different topics / topic pattern
* It maintains a pointer (offset) for each partition (s tored in Kafka broker)
* whenever a new partition is added the consumer fetches this metadata from the broker
* Offset: maintains last read location from a partition of a topics
    * last committed offset: consumer has confirmed to process
    * current position: read till this point but not committed
    * log-end offset: last record in the partition
* Enable auto commit = true, auto commit interval : 500ms
* Manual commit (sync and async)
* To scale have multiple consumers. This is called consumer group. The group coordinator evenly balances available consumers to partitions

Delivery Guarantees

* At least Once
    * P: set broker ack to Quorum
    * C: Auto commit = true

* Atmost once
    * P : Broker Asks = 0
    * C: Auto Commit = false

Netflix Ingestion Workflow

* Involves many steps
* Each step can potentially take several minutes to compute.
* Each steps will require different RAM and disk.
* Steps
    * Copyright check
    * apply country laws
    * format check
    * check all video frames
    * convert to multiple formats
    * generate thumbnails
    * generate preview
    * convert to lower res
    * persist
    * happy users
* Initial Design
    * Admin Portal -> Server -> Database
    * Demerits:
        * Bulky code
        * Each step can take different time
        * can fail without alerting
        * have to restart from beginning
    * Observations:
        * Traffic will be uneven
        * Don’t need to respond back immediately
* Design 2:
    * Admin Portal -> Server -> Queue -> Consumers -> Save
    * Each step have different system requirements
* Design 3:
    * Lets create a workflow system