Functional Requirements

The system has to count video view counts

WRITE

countViewEvent(videoId)

countView(videoId, eventType{VIEW|LIKE|SHARE})

processEvent(videoId, eventType, function{COUNT|SUM|AVERAGE})

processEvent(listOfEvents)

The system has to return video views count for a time period

READ

getViewCount(videoId, startTime, endTime)

getCount(videoId, eventType, startTime, endTime)

getStats(v)deoId, eventType, function, startTime, endTime)

Non-Functional Requirements

Scalable (10K's request per second)

Highly Performant (few 10's of milliseconds to return view count of a video)

Highly Available (survives hardware, network failure, no single point of failure)

Design

Tables

TableName: Events

Coulumns: videoId, timestamp

Notes

- Fast Writes

Can recelculate when needed

Slow reads

Costly when too many events

Used with batch data processing

Keep events for some time and purge events

Table Name:Count_per_minute

Columns: videoId, lastViewTimestamp, count

Notes

Aggregated data

Query the way data was aggregated

Required a data aggregation pipeline

Used with stream processing

Relational DB
TableName: Video_info
Columns: id, name, channel_id
TableName: vedio_stats
Columns: id, timestamp, count
TableName: Channel_info
Columns: id, name

Non-Relational DB
Video_id, channel_name, video_name, 15:00, 16:00, 17:00

Ingestion Path Components

- **API Gateway/PartitionerServiceClient**

  Blocking vs Non-Blocking IO: Blocking systems are easy to debug

  Buffering & Batching
  - Put events into buffering for several seconds or until batch fills up
  - Events are combined in API Gateway before sending to Partitioner Service
  - Increases throughput
  - Saves Cost
  - Request Compression is more effective

  Timeouts

  Connection Timeout (10ms)

  Request Timeout: Measure latency Percentile

  Retries

  All retires at same time may cause retry storm & overload server with too many requests. Use ***Exponential Backoff & Jitter***

  Circuit Breaker Pattern; Calculate failed requests count in given amount of last many minutes & stop calling downstream service. Then allow limited request for some time if succeeds allow all requests. Difficult to test, Hard to set error threshold and timeout

- **LoadBalancer**

  Software vs Hardware Load balancers

  Hardware load are devices that can be bought and handle millions of request

  Network protocols

  TCP: load balancers forward packet without inspecting contents of packets(Super fast can handle millions of request per second)

  HTTP: terminates the connection, Can look into contents of message, can make load balancing decision based on contents of packets

  Load Balancing Algorithms
  - Round Robin - Inorder across lists of servers
  - Least Connection
  - Least Response Time

- Hashed Based: On Client IP or URL

DNS: Register Partitioner Service in DNS
Health Checking; Are servers healthy
High Availability: Primary & Secondary load balancers maybe in different datacenters

- **Partitioner Service & Partitions**

  Partition Strategies:
  1) Hasing: Can cause hot partition for large scale systems
  2) Consistent Hashing: Dedicated Partitions

  Service Discovery (Zookeeper): getPartitionsListByPartitionService()
  Single leader Replication
  Message Formats

  Text
  1) XML
  2) CSV
  3) JSON

  Binary (For large scale rel time systems)

  4) Thrift
  5) Protocol Buffer
  6) Avro
  7) Parquet

  Faster to parse, Schema can be shared in DB from where producer & consumer can retrieve them

Technology Stack

API Gateway/PartitionServiceClient
Netty
Netflix hystrix
Load Blancing
NetScalar(Hardware LB)
NGINX
AWS ELB
Messaging System
Apache Kafka
AWS Kinesis
Data Procssing
Apache spark (Stream processing)
Apache Flink
AWS Kinesis Data Analytics
Storage
Apache Cassandra
Apache HBASE
InfluxDB
AWS S3 (DFS for Rollup data)

- Caching
  - Redis
  - AWS ElasticCache
- DeadLetters Queue
  - AWS SQS
  - RabbitMQ
- Embedded DB
  - RocksDB
- Leader Election
  - Zookeeper
- Service Discovery
  - Zookeeper
  - Netflix Eureka
- Monitoring
  - AWS Cloudwatch
  - ELK
- Message Format
  - AVRO
- Partition Data Algorithm
  - Murmur Hash
- Testing
  - Apache Jmeter

Testing

1) Perfomance testing: under heavy load to identify bottlenecks
2) Load testing: Behavior under specific expected load
3) Stress Testing: Beyond operational capacity often tot breaking point. What will break first
4) Leaks in resources: Typical production load for expected period of time

Health Monitoring

Metric, Dashboard, Alarms, Latency, Traffic, Errors, Saturation