Google Analytics System Design

by Kenrick Satrio Sahputra

Summary

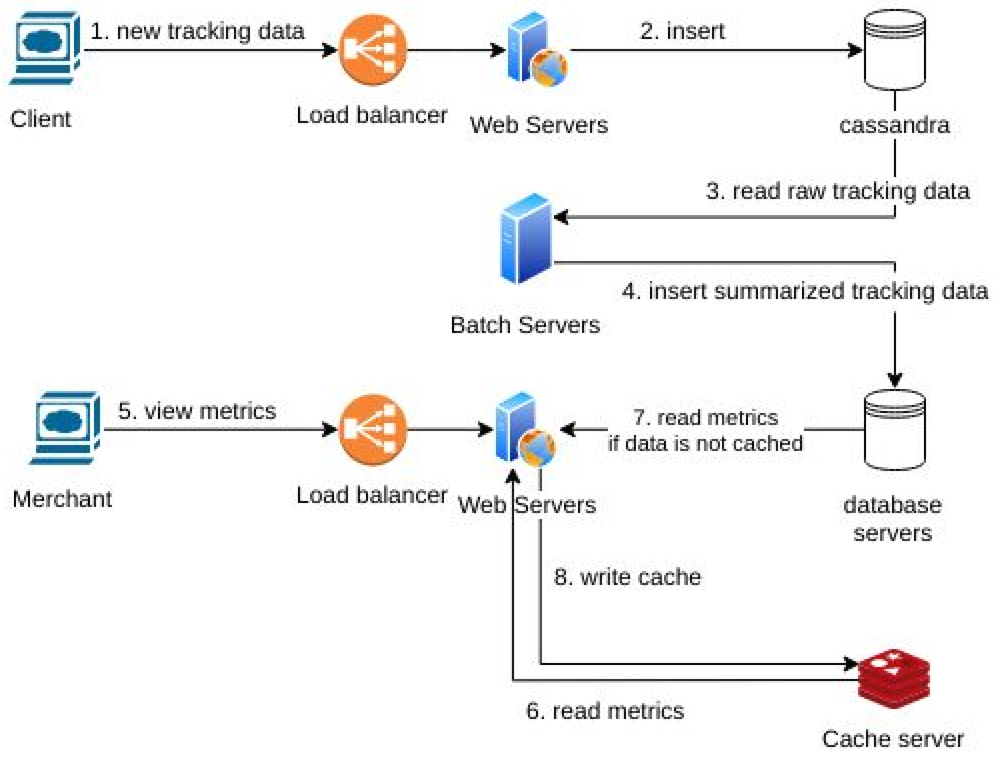

Figure 1. High Level System Diagram

High level system flow:

1. New tracking data from client go to load balancer and then to a web server.
2. Web servers validate user input then insert the data to a NoSQL database - Cassandra.
3. Batch servers periodically read data from Cassandra, summarize them and insert the metrics into database servers.
4. When merchants view tracking metric data, the request goes to load balancer then the request is served from different web servers.
5. The web server look the data in cache servers, if nothing is found, it will query the data from the metrics database and put the result into the cache.

How this architecture solve high write and read volume:

- Cassandra is a very high write throughput NoSQL database that has a linear scalability. Because of that, it is very easy to scale the database according to the heavy write needs.
- Merchants only read summarize data, therefore instead of calculating the raw tracking data every time, a batch system will process and provide the summarized data. Reading

the summarize data will reduce the calculation cost every time merchants make a request.
- Cache system is used to improve read performance. The web servers will query the data from cache and then if data is not found, it will query from database. Looking data from cache is much faster than database directly because it is memory operation compares to disk operation.

Requirements

Functional requirements:
- Track page view count visited by user.
- Track event by category, action, label and value.

System requirements:
- Handle large write volume. Billions write events per day.
- Handle large read/query volume. Millions merchants want to get insight about their business. Read/Query patterns are time-series related metrics.
- Provide metrics to customers with at most one hour delay.
- Run with minimum downtime.
- Have the ability to reprocess historical data in case of bugs in the processing logic.

System APIs

Below are the APIs that the system provide:

Add Page View

Parameters:
- token(string): a JWT token containing merchant_id and allowed hostnames. This data will be used to verify if the incoming request is authorized and allowed to create new data.
- page_url(string): URL for the page.
- user_agent(string): user agent to track the browser.
- ip_address(string): client ip address to track the location of the request.

Returns:
A successful code; otherwise an error code and it’s message.

Add Event

Parameters:

- token(string): a JWT token containing merchant_id and allowed hostnames. This will be used to verify if the request is authorized based on hostname.
- category(string): A name to group the object that we analyze. E.g. “Videos”
- action(string): A name to identify the event. E.g. “Video Load Time”
- label(string): Optional string for additional information of the event. E.g. “Example video”
- value(integer): Optional numerical value for the additional information. E.g. 60 (seconds)

Returns:
A successful code; otherwise an error code and it’s message.

Get Page View Count

Parameters:

- from(date time): Start time (in minute precision) to view the page view count.
- to(date time): End time (in minute precision) to view the page view count.

Returns:
List of page view and it’s count for every minutes.

Get Event Count

Parameters:

- from(date time): Start time (in minute precision) to view the page view count.
- to(date time): End time (in minute precision) to view the page view count.
- Category(string): category to view
- Action(string): action to view
- Label(string): Optional label to search for

Returns:
List of event and it’s count for every minutes.

Get Browser Access Count

Parameters:

- from(date time): Start time (in minute precision) to view the page view count.
- to(date time): End time (in minute precision) to view the page view count.

Returns:
List of browsers and it's count for every minutes.

Get Access Location Count

Parameters:

- from(date time): Start time (in minute precision) to view the page view count.
- to(date time): End time (in minute precision) to view the page view count.

Returns:
List of locations and it's count for every minutes.

Database Design

Observations on the nature of the data:

- Only append a new tracking data, no update no delete.
- Very high write workloads.
- Read time-series related metrics.
- No relationship query when looking for data.

Solution:

- Use Apache Cassandra for heavy write workloads.
- Use MySQL for reading summarized metrics data.
- Optimize performance and availability by having multi-data center and partitioning.

Cassandra

Usage:

- To store raw tracking data such as page view and tracking event.
- Writing new tracking data in cassandra.

Reasons to choose cassandra;

- Has a higher write throughput (especially for append only data) compare to other databases.
- Has linear scalability, scaling the performance is easier and cheaper compare to other databases.

- Multi-data centers setup is easy for low latency, high availability and disaster recovery features.
- Cassandra database is matured (since 2008), has a lot of community, support and is widely used by large companies.

Table schema

Creates 2 tables to store 2 types of data: page views and events. We store the data separately because we want to be able to process the data separately as well.

page_views	
K1	created_date(date)
K2	merchant_id(int)
C	created_at(timeuuid)
	page(text)
	user_agent(text)
	ip_address(inet)

events	
K1	created_date(date)
K2	merchant_id(int)
C	created_at(timeuuid)
	category(text)
	action(text)
	label(text)
	value(int)

* K = Partition Key, C = Cluster

Note:

- Used compound partition key of created_date and merchant_id. The compound partition key is used to break a big partition into smaller buckets.
- Having a small bucket size will increase the read performance, and enable fast reprocessing historical data without looking at all buckets.

- Used created_at timestamp as cluster key for faster data lookup by time.
- Using partition, data clean up is easier by dropping the old partition at once.

RDBMS

Usage:

- Stored metrics - summarized tracking data.

Reason to use RDBMS:

- Has better transaction to ensure data consistency compare to other NoSQL.
- Table structure is predefined for each metric.

Table Schema

	page_views
PK	id(bigint)
	merchant_id(int)
	page(varchar 100)
	view_count(mediumint)
	created_year(int)
	created_month(int)
	created_minute(datetime)

	browser_views
PK	id(bigint)
	merchant_id(int)
	browser(varchar 100)
	version(TINYINT)
	view_count(mediumint)
	created_year(int)
	created_month(int)
	created_minute(datetime)

location_views	
PK	id(bigint)
	merchant_id(int)
	country(varchar 50)
	region(varchar 50)
	city(varchar 50)
	view_count(mediumint)
	created_year(int)
	created_month(int)
	created_minute(datetime)

events	
PK	id(bigint)
	merchant_id(int)
	category(varchar 100)
	action(varchar 100)
	label(varchar 100)
	count(int)
	created_year(int)
	created_month(int)
	created_minute(datetime)

Note:

- Partition the data by created year and month so we have a faster data lookup.
- Using partition, data clean up is easy by dropping the whole partition at once.
- Use multi-data center to have low latency, high availability and disaster recovery features.
- It better to choose database that can support multi-data center such as MySQL NDB, PostgreSQL, etc.

Capacity estimations

Summary

Write per second	~11k writes/second
Read per second	~1k reads/second
Cassandra storage for 5 years	~1 PB
Metrics database storage for 5 years	~38 TB
Memory for cache	~1.5 TB

Detail

Assuming there are 1 billion writes every day, then write per second:
1,000,000,000 / 24/ 60 / 60 = **~11k writes/second**.

Assuming there are 100 million reads per day, then read per second:
100,000,000 / 24/ 60 / 60 = **~1k reads/second**.

For tracking raw data, assuming each record data and index size is 500 bytes, then the storage estimation:

```
1 day   = 500 bytes * 1 billion = 500 GB
1 year  = 500 GB * 365 = ~180 TB
5 years = 180 TB * 5 = ~1 PB
```

For summarized tracking metrics data, assuming:

- Each record data and index size is 500 bytes,
- there are around 10 millions data every day per table, and
- there are 4 different tables.

Then the storage estimation is:

```
1 day   = 500 bytes * 10,000,000 summarized data * 4 tables= ~20 GB
1 year  = 20 GB * 365 = ~7.5 TB
5 years = 7.5 TB * 5 = ~38 TB
```

Assuming user only see active tracking for the past 1 year data. If we follow 80-20 percent rule, where we store only 20% of the active data in the cache, then the memory used for cache:
7.5 TB * 20% = **~1.5TB**

High Level System Design

Given the functional requirements and system requirements, here are the high level system design solution:

- Design a fast web server to write new tracking data.
- Batch system to summarize raw tracking data periodically.
- Create a merchant web portal to view tracking metrics.

Fast Web Server for New Tracking Data

Purpose

Web servers to receive new tracking data. This web server need be very fast to handle a very high write workload.

How it Works

- Provides load balancers and web servers in different region to provide low latency and high availability system and minimum downtime.
- Web server needs to do things as few as possible, such as:
    1. Validate input - user identity and user input without database access.
    2. Insert tracking data to Cassandra.

Validate input

- Authentication.
  Authenticate if the user is real user by decoding the JWT token. If JWT token can be decoded then the user is authenticated.
  The JWT payload contains merchant_id and allowed hostname list.
- Authorization
  Authorize the request by checking if the request hostname is in the allowed hostname list from the JWT payload.
- This authentication and authorization process will not include any database access, thus it is fast.

Insert tracking data to Cassandra

Since cassandra will write new data into memory, this operation should be fast. To ensure we do not lose any data if the cassandra database down, we should acknowledge the write operation success when the data is written more than 1 cassandra node.

Programming Language

To anticipate a very high workload with simple tasks, use a programming language that is good for high concurrency such as like golang. Golang can handle higher concurrency compare to other programming language because it uses logical thread to handle the request. Comparing to OS thread, golang can creates more logical threads that is also lighter.

Security

To have JWT token, users need to authenticate and create the token first. System will have another endpoint to do authentication and create the token. The token will be set in secure http-only cookie and have the expiry.

Batch Processing Data

Purpose

Calculate the raw tracking data, summarize into metrics and persist the data. Having precomputed summarized data, make the system faster by avoid calculating data every time merchants want to see the data metric.

How it Works

- Creates batch application that is parameterized by date time.
  By default it will process the last 5 minutes data inserted into Cassandra.
- Schedule to run the batch application periodically every 5 minutes.
  he process happens every 5 minutes to read the past 5 minutes data inserted into Cassandra. We do not query the too often (< 5 minutes) to avoid too many queries happens and we do not process too long (> 5 minutes) to avoid processing many data at once. This duration should be investigated more to get the correct value.
- Process the data parallelly.
  Have many batch servers to process the data parallely to speed up the process.
- Persist the summarized data.
  After data is processed, batch servers insert or update the record to database, so the next time merchants want to see the metric, they can query from the precomputed metric.
- Invalidate cache
  After the batch successfully inserts persist summarized data to database, it will invalidate the data in cache if it exists.
- Parameterized batch application for reprocessing historical data.
  Simply re-run the batch with different time to reprocess historical data if necessary.

Additional Note

- Apache Spark
  We can consider Apache Spark to process the data efficiently. Apache Spark performs the data transformation in memory which is more efficient. It can also run locally in cassandra server together, so reading the data from cassandra becomes much faster.
- Apache Kafka
  We can also consider using Apache Kafka. Usign Cassandra CDC, cassandra can push event to Kafka. After events are push, Kafka can start transforming and summarizing the data and persist it to database. Using this technique we can achieve near real-time batch pipeline.

Reading Metrics Data

Purpose

Provide merchants tracking metrics based on time per 1 minute data. System should be able to handle high read workload.

How it Works

- Provides load balancers and web servers in different region to provide low latency and high availability system and minimum downtime.
- Web servers query data from cache servers, if data is not in cache then query data from databases. Since data in cache servers are in memory, querying data from cache is relatively faster than database, thus system provides faster performance.

Caching

Caching the data in memory increases the read performance. Below are some considerations about the caching:

- 80-20 caching rule.
  Follow 80-20 caching strategy, where system only cache 20 percent of the whole metric data.
- Use LFU (Least Frequently Used) cache eviction strategy.
  By keeping the most frequently used data in the cache, we can keep data that is more likely to be used in the future.
- Cache Invalidation
  Cache will be invalidated by the batch application. Every time batch application persists data to database, it will also invalidates the cache. That way the data will always be consistent between database and cache.