Backend Design for Web-based Data Analytic Platform

Table of contents

Requirements

- Handle large write volume: Billions of write events per day.
- Handle large read/query volume: Millions of merchants wish to gain insight into 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.

Requirements

- Handle large write volume: Billions of write events per day.
- Handle large read/query volume: Millions of merchants wish to gain business insight.

Capacity Estimation and Constraints

- Assume we have total 600M users, and 200M daily active users, with 2:1 w/r ratio.
- Assume on average every active user write 5 times per day, which leads to 200M *5 = 1B writes per day on the platform.
- Assume on average every active user visits our platform 2.5 times per day, which is equivalent to 200M * 2.55 = 500M reads per day.
- Assume on average every user has 1MB data, this requires 1MB * 600M = 600TB of storage.
- Assume each write has size 20KB, this will yield addition 20KB * 1B = 20TB storage per day.
- Assume for each merchant, concurrent reads from multiple devices are allowed, but not for concurrent write.
- In general, we need to provide a high-throughput and write-intensive application.

Requirements & Initial design

- **Write-Back Cache.**
  - The system needs to support heavy reads and writes in a efficient way. We selected a time-series Database for the high-performance for time-series data operations.
  - On the side, we leverage the advantages of write-back caches to assist fast read and write.

- **Provide metrics to customers with at most one hour delay.**
  - Due to this allowed 1h delayed, data in the cache can be flushed back to database hourly.
  - All data synchronizations need to finish within the same hour to ensure the global data consistency.

- **Run with minimum downtime.**
  - There are load balancers sitting in front of our distributed web/application servers, to handle peek-hour traffic and direct user requests to the available servers (load balancing strategy is not addressed in this doc), therefore minimizing the system downtime.
  - Database level, we will utilize distributed data replication strategy to have copies of user data, in case of facing any network/geographic accidents.

- **Have the ability to reprocess historical data in case of bugs in the processing logic.**
  - This requires no deletion in any data. All user data and copies shall be preserved.

High-level System Design

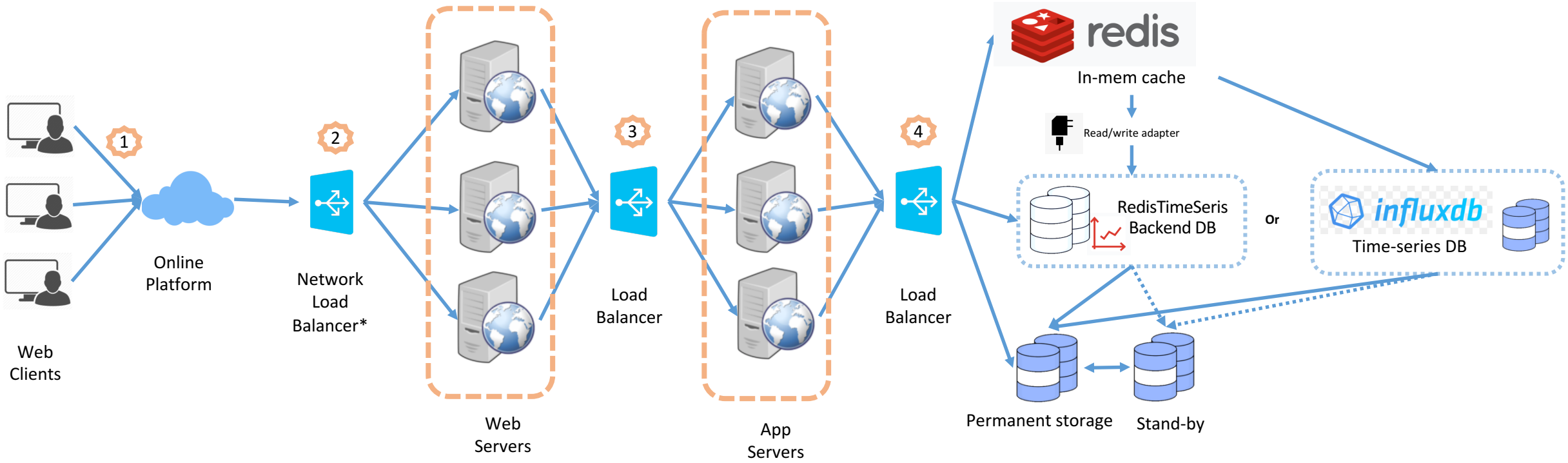

Image 1 – Design of single data center

1. Users click on the UI, requesting to read/write data.
2. Network load balancer(LB) helps balance the HTTP traffic between clients and web servers.
3. Application LB further distribute the application traffic across multiple available servers.
4. Database server LB serves as a single contact point directing queries to the suitable source, maximize queries throughput and minimize latency.

Write/Read Events Handling Design

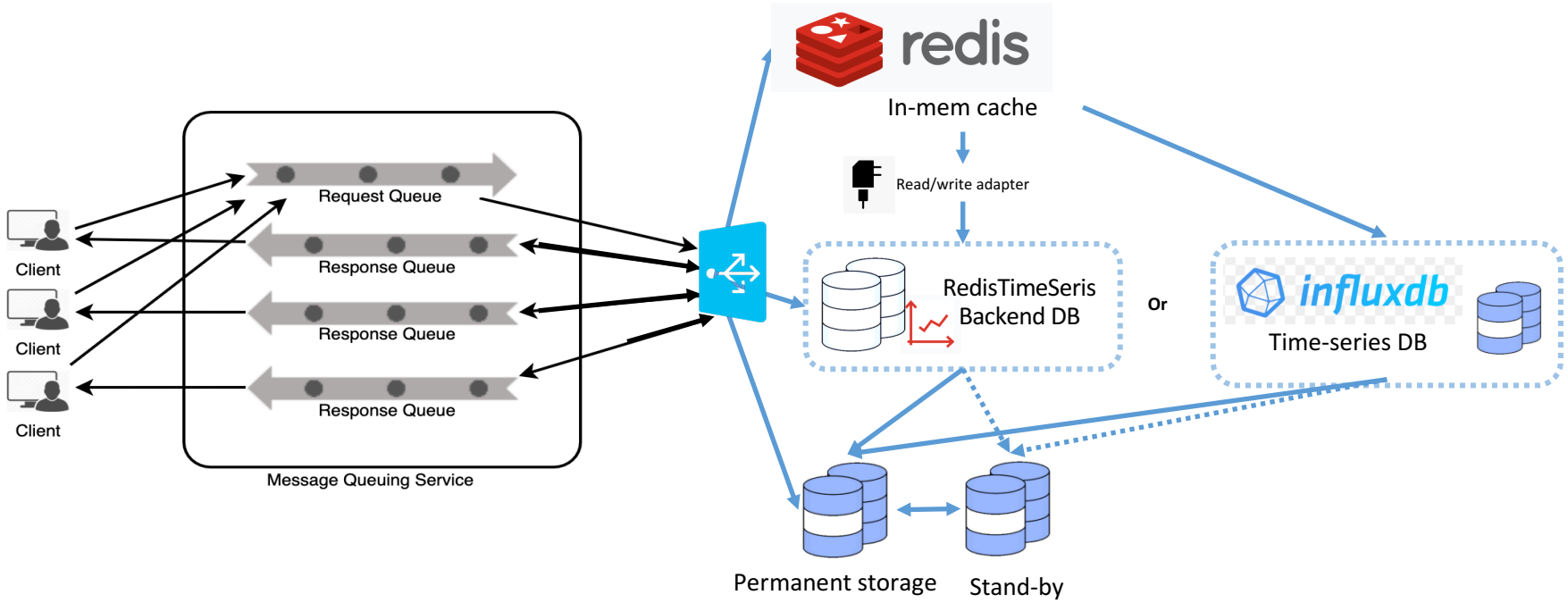

Image 2 – Design of write/read event handling

1. Clients data access requests will be queued through Message Queuing(e.g. Amazon MQ) service by classifying the queries.
2. Read: then the DB LB will process each request and send query first to the cache, if cannot find corresponding data(Cache and DB are not always synched), further query in the backend database then acknowledge the client and return the data.
3. Write: then the data will be write to in-men cache first. The synchronization between cache and backend DB is scheduled every 45mins (Assume the data sync takes less than 15mins).
4. To back-up all history data, whenever there is a flush from cache to DB, the data will also be written into permanent storage that is located in a different region. The stand-by database server stays in sync with the primary storage, in case of any hardware or network failure, this stand-by will be brought up online to serve as the primary storage to minimize the loss of data.

Selection explanation

- **Write-back cache:**
  - In this manner, data is written to cache alone, and completion is immediately sent back to the client. The write to the permanent storage is done after specified intervals or under certain conditions. This ensures low-latency and high-throughput for write-intensive applications;
  - In the risk of data loss of any crash or adverse events, data will write to multiple cache at the same time to reduce the chance of data permanent loss.

- **Redis cache:**
  - To speed up the data access, after research we selected Redis pipeline to send batches for data operations. The use of RedisTimeSeries can also make querying faster and more optimized.

- **InfluxDB:**
  - Is "*An open-source distributed **time series** database with no external dependencies*".
  - InfluxDB is a scalable data store for metrics, events, and real-time analytics. It has a built-in HTTP API so server side code is not required to get up and running.
  - InfluxDB is designed to be scalable, simple to install and manage, and fast to get data in and out

Conclusion

The provided backend design provides robust and efficient data heavy write/read for the time-series data. Given the estimated capacity based on the requirements, one can selectively choose suitable hardware or online service such as data storing and server hosting.

Write-back cache is used to guarantee quick read and write; data flush with intervals ensures the delay is under an hour.

Multiple servers at various layers are monitored, managed and load balanced to minimize the system downtime at each phrase.

Historical data can be retrieved from the permanent storage in case of incidents or business changes.

However, the proposed system can be further enhanced by considering implementing geo-diverse data centers to be able to recovery from nature disasters or unexpected accidents. Besides, some load balancing step can be omitted based on the actual situation to cater the best performance.