Twitter's Conceptual Architecture

13 October 2024

David Luu - 216157463 - luu332@my.yorku.ca
Omer Omer - 218636878 - omeromer@my.yorku.ca
John Prabahar - 219087279 - don19@my.yorku.ca
Arash Saffari - 218791632 - arashrt@my.yorku.ca
Ali Sina - 218318428 - alisina@my.yorku.ca
Siyan Sriganeshan - 218707190 - siyan@my.yorku.ca

Abstract

Twitter is a complex project with many interrelated components designed to enhance user comfort and engagement. Although creating a simple tweet involves intricate underlying systems, this report focuses on Twitter's architecture and its interconnected elements.

Twitter integrates social networking and microblogging, allowing users to post tweets of up to 280 characters. Key features include following others, retweeting, and using hashtags and mentions to drive engagement. The platform relies on a relational database structure, primarily MySQL, organizing user and tweet data in separate tables linked by primary and foreign keys.

At the heart of Twitter's system is the Home Mixer, which facilitates timeline creation. It interacts with key components such as Gizmoduck, Tweety Pie, Social Graph, and Manhattan—a custom distributed storage system that supports reliability across various components. This architecture enforces a repository style, creating a pipeline for managing and organizing data efficiently.

Two primary timelines enhance user experience: the User Timeline, which displays a user's own tweets, and the Home Timeline, showcasing tweets from followed accounts. The fanout caching approach ensures efficient delivery of tweets, while Earlybird enables rapid search and categorization. The Blender aggregates and ranks data from multiple sources, using activity metrics to personalize timelines.

Collaboration among Twitter's developers—encompassing front-end, back-end, machine learning, DevOps, security, and data engineering—ensures efficiency, scalability, and functionality. This interconnected structure allows Twitter to provide a seamless, secure, and scalable platform for millions of users.

Introduction and Overview

Twitter, as one of the world's most widely used social media platforms, requires a robust and scalable architecture to handle its massive user base and real-time data processing needs. This report delves into Twitter's architectural evolution from a monolithic system to a Service-Oriented Architecture (SOA), enabling the platform to maintain scalability, stability, and performance. By exploring core components such as the Home Mixer, Manhattan database, Tweety Pie, and Gizmoduck, we examine how these interconnected systems work together to provide real-time data updates, personalized user timelines, and efficient data storage.

In addition to the technical architecture, this report highlights the crucial functionalities that enable Twitter to deliver a seamless user experience, including its tweet posting mechanisms, search capabilities, and personalized feed creation. The interactions between Twitter's various services demonstrate the platform's reliance on a highly modular system, allowing for flexibility and efficient data management.

The report also explores how Twitter handles concurrency, data flow, and the collaborative interactions between its various components and development teams. Through this analysis, we gain a comprehensive understanding of how Twitter's architecture supports its real-time operations and ensures that the platform remains reliable and scalable despite its global scale.

Architecture

Twitter's current architecture can be argued to be Service-Oriented Architecture. This is to be expected as Twitter being an enterprise level company, there are many mammoth sized functions that need to be separated to ensure the company is scalable and stable in the long run. Taking a look at Figure 1 it is evident that the various services present within the ecosystem. These services are loosely coupled with a moderate to high level of cohesion allowing for services to be switched out, redirected if required [15]. It could be argued that the main architecture would be Event-Driven. However, the services within Twitter are called by one another opposed to being triggered by an event and follow a flow of operations rather than being dictated by events.

It can be said that other architectures are present but not as dominant as SOA. As many social media platforms require notifying the user. Hence, it uses implicit-invocation for delivery of user updates, as well as updating the users hometime line and populating users feed. Layered Architecture is present if we look at it from a high level consisting of user interface, app logic, and database. One of the stronger architectures would be client-server. Reason being, Twitter is data dependent and functions off of it. These services within Twitter, like Home Mixer, the rankers, all rely on the database. Where the services act as the client and the database as the server.

Functionality

To understand Twitter's architecture, it is essential to identify its core features and comprehend the elements that enable its functionality.

Twitter blends social networking and microblogging, enabling registered users to post messages, or "tweets," limited to 280 characters. Users can create, like, and share tweets, with public visibility by default, although they can restrict access to followers. Features include muting or blocking accounts and the ability to follow other users, termed "followers" or "tweeps" [1].

Users can retweet others' messages and use features like "quote tweet" to add comments. Interaction metrics, such as likes, retweets, and replies, are prominently displayed. The platform supports searching for tweets, accounts, hashtags, and trending topics, with advanced filters for refining results.

Hashtags (preceded by "#") categorize content, while mentions (using "@") facilitate interactions with other users. Twitter also introduced hashflags—special hashtags that generate custom emojis—for events like the FIFA World Cup. Users can manage tweet replies, including hiding replies and controlling who can respond [2].

For communication, Twitter provides SMS support through several gateway numbers, although the character limit was originally set at 140 to accommodate SMS messaging. The platform also employs its own URL shortening service, t.co, to streamline links within tweets.

Businesses utilize Twitter for marketing, enhancing brand visibility, customer engagement, and public relations. Promoted tweets allow companies to reach a wider audience, clearly labeled as advertisements [3].

Interacting Parts

With the necessary functionality established, it is evident that Twitter's high-level architecture utilizes MySQL databases to manage its data. As seen in Figure 2, each user is represented by a unique row in the Users table, while their tweets are stored in the Tweets table, forming a one-to-many relationship. A key concept is the feed, which connects and displays tweets from users that a particular user follows. Twitter's database architecture follows a relational model, with user information and tweets organized in separate tables—Users and Tweets. These tables are linked through a primary key-foreign key relationship to ensure data integrity and avoid duplication.

Twitter's architecture features two primary timelines: the User Timeline, which displays a user's tweets and retweets in chronological order, optimized through a caching layer, and the Home Timeline, which shows content from users that the individual follows using a fanout caching approach for improved efficiency. For users with large followings, a combination of the home timeline method and synchronous calls is employed to optimize tweet loading, while timelines for inactive users are neither precalculated nor stored in the cache .

The fanout approach allows Twitter to push tweets directly to followers' in-memory timelines, enhancing data delivery and improving user experience by enabling faster request handling. Additionally, Twitter utilizes Earlybird, a search-based reverse indexing tool based on Lucene, to efficiently tag and categorize tweets for search purposes. This system also includes tools for dividing, scattering, and gathering data to ensure rapid global searches, with results ranked according to tweet popularity.

Twitter is a vast and interconnected platform that requires efficient system management. Over time, it has evolved into an API-heavy architecture, utilizing various components and APIs that work in harmony. This is achieved through a Pipeline and Repository architecture, all while maintaining a service-oriented approach across its systems, in which the main ones are the Home Mixer, EarlyBird, and the User Tweet Entity Graph [4].

Home Mixer

The Home Mixer is the main and most important component of Twitter. The Home Mixer acts as the starting point of the pipeline system as it sends the request with the necessary data to the necessary component. The Home Mixer is made up and connects to a bunch of different components but the most important ones are the Manhattan, Social Graph, Gizmoduck,and Tweety-Pie [7].Manhattan is Twitter's custom made distributed storage system which is used by almost every component in Twitter. It's used to store all kinds of data such as tweets and user data for long term use. Tweety Pie is responsible for wrapping tweets with metadata and is used by the Tweet API, while Gizmoduck is responsible for storing the users profiles data such as account settings and following data. Social Graph is used to show the data regarding the user's interactions with posts and other users. It maps out the relationship that users have with each other,such as likes,retweets, following, etc. This newly created data is also then stored via the Manhattan system for storage. All three of them use the Manhattan storage system. If they require any data for their tasks or wish to store their data for security or storage purposes, they call Manhattan. The three components have their short term storage capabilities and thus gives them a Cache* like attributes. Twitter has alot of users and thus alot of traffic, having these components in place that have their own short-term memory alleviate pressure from the main database and allows for smooth access to the necessary data.

The Home mixer is mainly used for timeline creation and maintaining user following data. Its common use which is the timeline creation, is what showcases the Home Mixer efficiently. It connects to many different subsystems such as the Candidate Generation, Feature Hydration, and many more different filtering based components. The Candidate Generation also has its own components that it accesses. Components such as the Earlybird and the User Tweet Entity Graph (UTEG) are used by the Candidate Generation, Earlybird retrieves relevant tweets while UTEG retrieves tweets based on the user's interactions. Earlybird and the User Tweet Entity Graph (UTEG) also retrieve they're data from Manhattan, which pushes the Repository style of the system. [20]

Home mixer uses its different parts to collect the necessary data so that it can then send to other components down the pipeline depending on the task at hand. All the components complete their specific task with the data that is given them and that data is then sent to another component. It's with the relationship that the components have with each other that defines the Pipeline style. Almost all the components in Twitter's system also access the Manhattan system, whether it be for pulling data or pushing it. This also showcases the Repository style, as they're all connected to Manhattan for efficient data flow. This system that they have in place is good for evolution and for the modularity of the system, as the benefits of having a pipeline system is that you can switch out the components.

Components of the Home Mixer

Tweety Pie serves as the core Tweet service responsible for managing the reading and writing of Tweet data. It is accessed by Twitter clients through GraphQL, as well as various internal Twitter services, to create, delete, and edit Tweets. Tweetypie also retrieves additional Tweet-related data from multiple backends to provide comprehensive information to callers.

Gizmoduck is responsible for storing the users profiles data such as account settings and following data. It is used as Cashes for quick retrieval and storage of short-term data.

Manhattan is Twitter's custom distributed storage system employed by nearly all components of the platform. It is designed to store various types of data, including tweets and user information, for long-term retention. This architecture fosters a relationship between the cache and the database.

Social Graph service is similar to Facebook's TAO. It maps all of the connections between users' social networks and is stored on FlocksDB. Social graph exists so that this map can be accessed without the compute-heavy task of trawling a user's network every time the information is needed.

Earlybird

Earlybird is Twitter's real-time search system built on Apache Lucene, designed to handle a high volume of queries and content updates. Its primary use cases include relevance search, particularly text search, and timeline in-network tweet retrieval based on UserID. Earlybird efficiently indexes and queries billions of tweets while providing low-latency search results, even under heavy loads.

The tweet search index is divided into three clusters*: a real-time cluster* for public tweets from the last seven days, a protected cluster* for protected tweets during the same period, and an archive cluster* for all tweets up to two days old. To scale real-time search, each cluster* is partitioned, with each partition managing a portion of the index. The architecture employs a distributed inverted index that is both shared and replicated, allowing for efficient index updates and query processing.

Earlybird utilizes an incremental indexing approach, enabling real-time processing of new tweets. Its single writer, multiple reader structure allows it to manage numerous updates and queries concurrently while maintaining low latency and high query throughput*.

Blender servers handle user search queries. When a user searches for something on Twitter, their query is first sent to a Blender server, which parses the query and passes it along to the appropriate Earlybird servers. It is responsible for distributing the user's query to multiple Earlybird servers.

User Tweet Entity Graph (UTEG)

The User Tweet Entity Graph is a Finalge thrift service built on the GraphJet framework, designed to manage the relationships between users and tweets. It provides user recommendations by traversing this graph.

UTEG generates the description of the person who liked out-of-network tweets that appear on Twitter's Home Timeline, leveraging collaborative filtering. It takes a user's weighted follow graph (a list of user IDs with associated weights), efficiently traverses and aggregates this data, and returns the top tweets based on user engagement and the weights of those users.

As a stateful service, UTEG utilizes a Kafka stream for state ingestion and persistence, maintaining in-memory records of user engagements for the past 24 to 48 hours. Older events are discarded and garbage collected.

Finalge is an extensible RPC system for the java virtual machine*, used to construct high-concurrency servers. Finagle implements uniform client and server APIs for several protocols, and is designed for high performance and concurrency. Most of Finagle's code is protocol agnostic, simplifying the implementation of new protocols. Finagle is written in Scala, but provides both Scala and Java idiomatic APIs.

Data Flow

When a tweet is received, the Ingester tokenizes it and determines the relevant indices, then sends it to a single Earlybird machine for processing.

During the fanout process, a tweet may appear in multiple home timelines of a user's followers, but in Earlybird, each tweet is only indexed in one machine (with exceptions for replication).

The Blender creates the search timeline by performing a scatter-gather operation across the data center. It queries each Earlybird shard to see if they contain relevant content for a given query. The results from all shards are sent to the timeline ranker, where it is sorted, merged, and then re-ranked based on social proof, which considers metrics like retweets, favorites, and replies.

Activity information is tracked on a write basis, generating an activity timeline that records user interactions such as favorites and replies, similar to a home timeline. This activity data, consisting of IDs for each action, is fed into the Blender.

During the read process, the Blender recomputes, merges, and sorts this data, as well as heuristic and filtering processes that assess factors like social proof, visibility, content balance, and feedback fatigue. The recommended tweets are then merged with advertisements and following suggestions before being integrated into the user's timeline. Finally, the completed timeline is returned and displayed on the frontend for the user to view [5, 6].

Evolution

Twitter was launched in 2006 by Jack Dorsey, Evan Williams, and Biz Stone as a microblogging platform[17]. Twitter became famous for its 140-character limit and was intended as a "content management platform, not a messaging platform" in the beginning [12&9]. This is why the initial software architecture of Twitter was monolithic, built primarily on Ruby on Rails which was simple and easy to launch for a low number of users and included simple Twitter functionalities including posting tweets, user management, timelines, and notifications [10]. A simple background about a monolithic architecture; it includes 3 layers: presentation, business logic, and data layer bundled into one unit which is easy to start with because everything is part of the code base. In March 2008, two years after the initial launch, Twitter had over 1 million unique users and 3 million messages per day [8]. This means the monolithic architecture caused a lot of performance and scalability problems. In early 2010, Twitter decided to migrate from third-party hosting to its own internal data system called T-bird, which is built on top of Gizzard [11&13]. Additionally, Twitter started Flockdb as a distributed system* to store social graphs and secondary indices [13]. In terms of software architecture, Twitter decided to migrate from monolithic to microservices architecture which offers further scalability, flexibility, and separation of concerns especially for large-scale projects. A lead Engineer in the Services Team at Twitter, Evan Weaver, mentioned that "many optimizations were needed to change the initial model based on aggregated reads to the current messaging model where all users need to be updated with the latest tweets. The changes were done in three areas: cache*, MQ, and Memcached client."[13] Furthermore, real-time data processing systems (i.e. Kafka and Storm) was used to handle the massive influx of tweets and user interactions. In 2020, Twitter shifted to AWS as a strategic provider to serve timelines. According to Amazon, "Under the multi-year deal, Twitter will leverage AWS's proven infrastructure and portfolio of services to support delivery of millions of daily Tweets."[15] In April 2022, Elon Musk acquired Twitter for $44 billion. Since, there have been proposed changes to the system design to speed up the timeline service workflow and cut costs. Furthermore, TechCrunch reported 'significant' backend server architecture changes after Musk's acquisition due to users reporting outage and unintentional 'sign outs' [14]. Although Twitter now has 586 million monthly active users in consequence of Musk's acquisition, users and developers are concerned about Elon Musk's recent actions. Developers expressed concern that mass layoffs of a large number of engineering teams could jeopardize the platform's reliability and usability by performing large architectural changes without proper testing and progressive rollouts.

Our Subsystem

There are two paths to Tweetypie: the Read Path and the Write Path. In the Read Path, Twittypie hydrates data from collected Tweets from Manhattan or Twemcache. The backends package is what Tweetypie calls. The repository package gives structure for data retrieval. The hydrator package stores labels of tweets and hydrates raw tweets with additional information and then returns them. The handler package handles requests to endpoints of Twittypie. The GetTweetsHandler in particular uses TweetResultRepositor, which uses ManhattanTweetRepositor at its core, which is also wrapped in CachingTweetRepositor. In the Write Path, it reuses some code from the Read Path but the patterns it follows are different. The store package updates code on the backend and coordinates code for certain endpoints that need an update on certain endpoints. In the store package, there are two files: stores and store modules. Stores file defines logic for updating backends. Store modules are for which stores are called and handled. TweetBuilder makes tweets from requests after performing background checks. WritePathHydration.hydrateInsertTweet returns the tweet to the caller through the hydration pipeline [19].

Concurrency

Redis Clusters

Redis Clusters* are the most High level design by which concurrency is achieved. Users are redirected from the load balancers* to different nodes* of the cluster*. This enables users to access data with low latency as well as not overwhelming the servers. Essentially data is cached* in memory rather than accessing it straight from the storage.These clusters* can hold many terabytes of data, which otherwise would be fetched straight from the databases [18]. Having to serve many customers incorporating their personal entries/tweets is a costly operation of many reads and writes which done sequentially by a database would be chaotic considering the mammoth twitter is. Also, to the benefit of having multiple nodes* it contains multiple copies of data. This comes in handy when accessing data, where users can concurrently use the same data and do not have to rely on one server serving them sequentially. Furthermore, there won't be any data conflicts, or corruption due to the fact that these clusters* store the same data.

Home Timeline

As for the main algorithms' system and design. The user accesses all the components through the Home Mixer, which acts as the Repo to which combines all of its subcomponents to prepare 1 interface for the user. There is concurrency between many of the parts inside the mixer. The home mixer creates 3 pipelines, For you, Following and Lists. Where each pipeline has a task of its own to fill the requirements of Home Mixer [19]. The For you pipeline fetches tweets and other content that the user might not directly follow but is inside his or her circle of interest. The Following returns content from accounts the user directly follows. Finally, Lists task is to recommend accounts that closely align with the users interests.

All the pipelines leverage 4 main subcomponents to retrieve the necessary data. Manhattan, Gizmoduck, Social graph and Tweety pie all act on their own when initiated by each of the Pipeline. They themselves are based on pipelines and filters where they go through a sequence of other components to produce the data that is required of them. There is no overlap between their tasks, as they are all unique entities that provide different types of information required by the Home mixer. However, they do overlap in the components they use, for example the feature hydrator is used by Gizmoduck, and Tweety Pie. Thus there are also many components that work concurrently.

Tweety Pie

Our subsystem in question, Tweety pie, is the backbone of twitter. As such, it's used by various other services concurrently to make twitter functions. It is mainly called by GraphQL, from other services and to public API's. With Twitter's complex data structure as discussed above, Tweety Pie interacts with cached* data as well as backend. This happens in parallel when Tweety Pie is requested with reads. Essentially, looking up the cache* and querying the database at the same time, in case the cache* misses. Hence, improving the performance by not doing the process sequentially. Same could be said about writing tweets, but it in itself breaks down the task further by processing urls, media, and checking for duplicates all in parallel [16, 19].

Diagrams

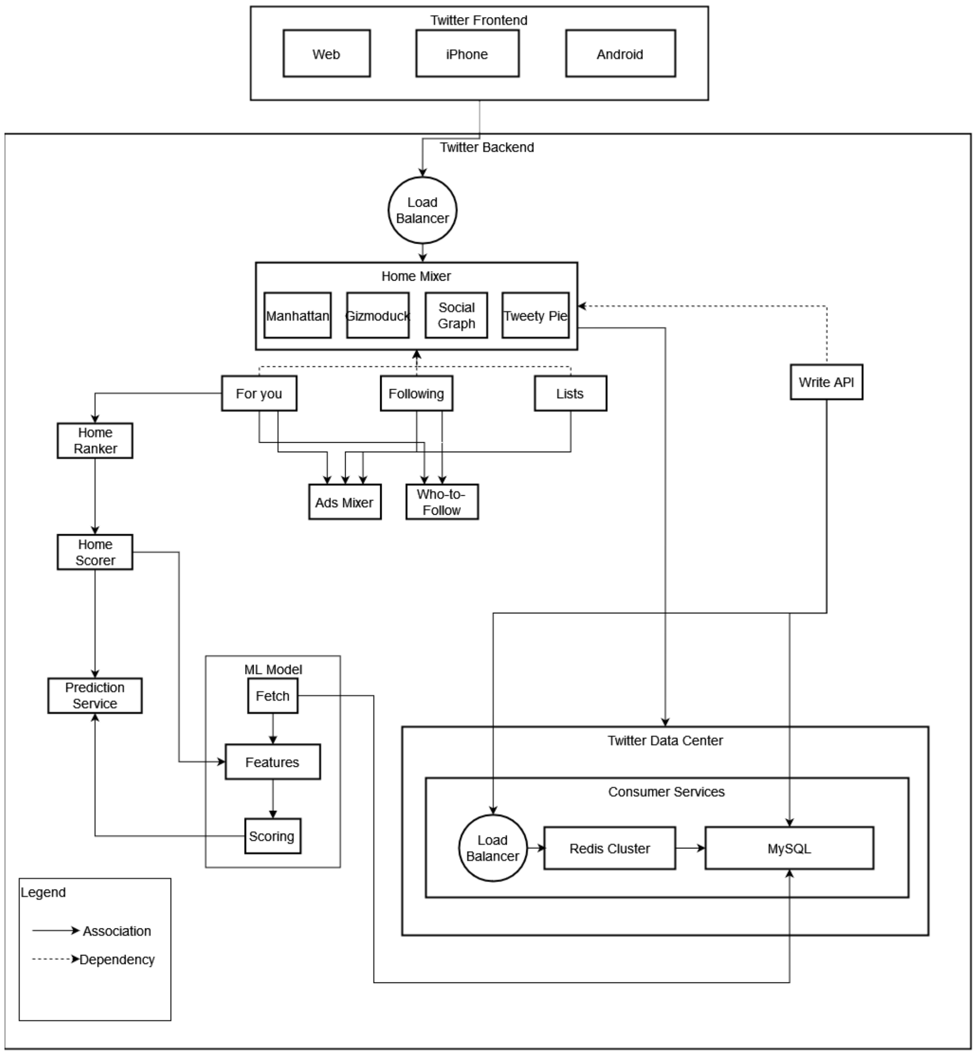

Figure 1: Conceptual Architecture of Twitter

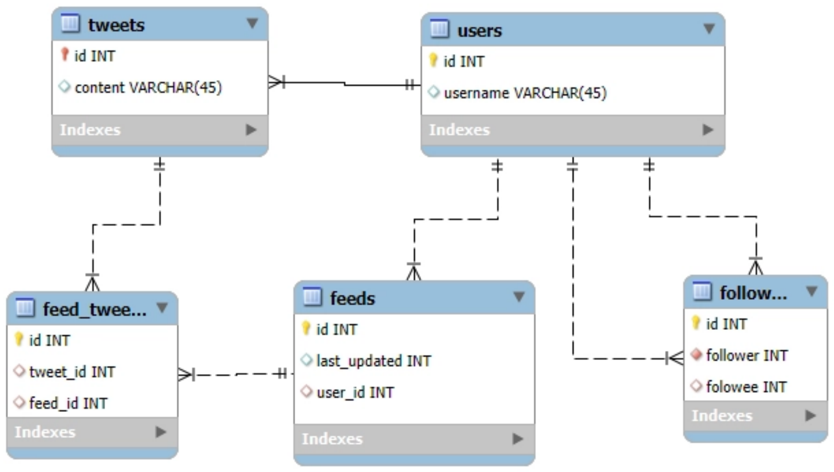

Figure 2: Tables of Twitter’s Databases

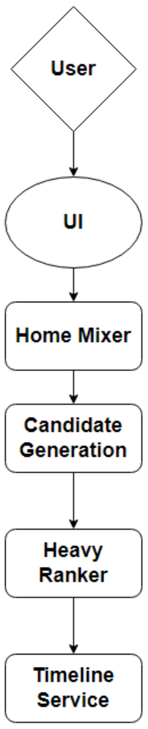

Figure 3: Timeline Data Flow

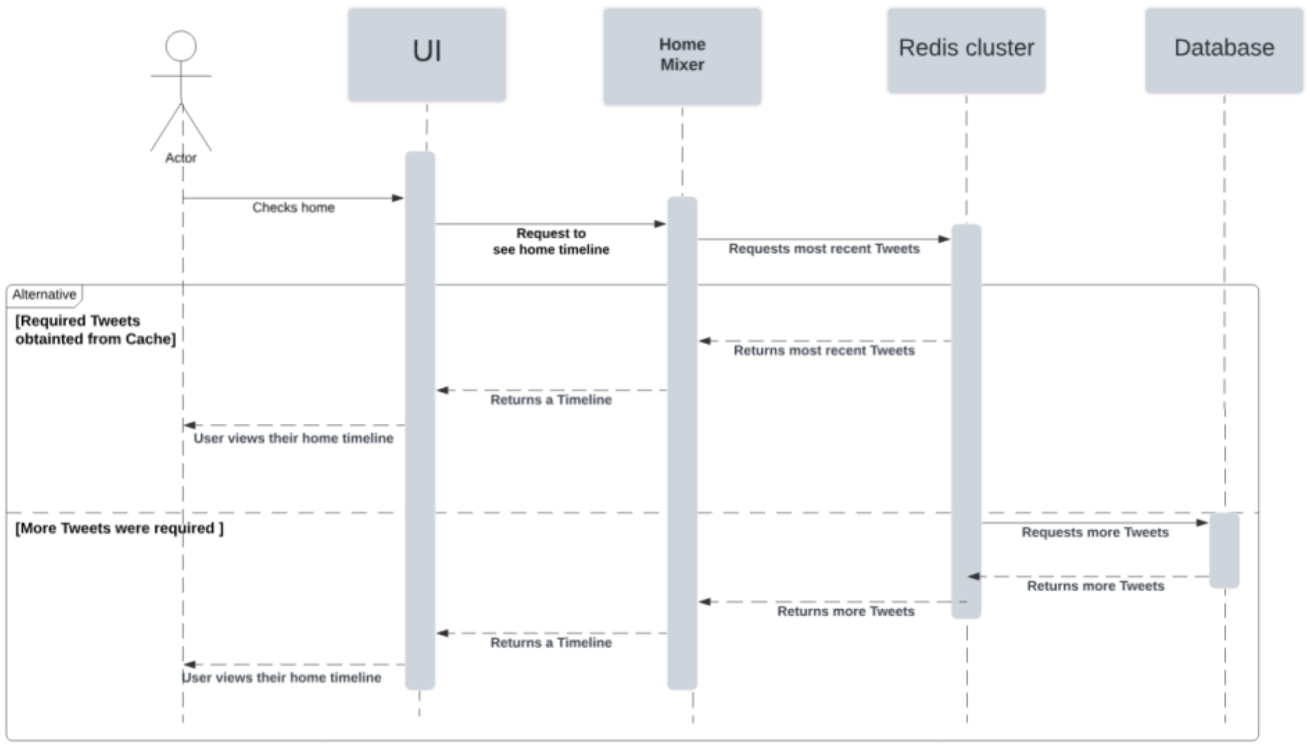
Actor
UI
Home Mixer
Redis cluster
Database
Checks home
Request to see home timeline
Requests most recent Tweets
Alternative
[Required Tweets obtainted from Cache]
Returns most recent Tweets
Returns a Timeline
User views their home timeline
[More Tweets were required]
Requests more Tweets
Returns more Tweets
Returns more Tweets
Returns a Timeline
User views their home timeline

Figure 4: Use Case 1: Timeline Creation

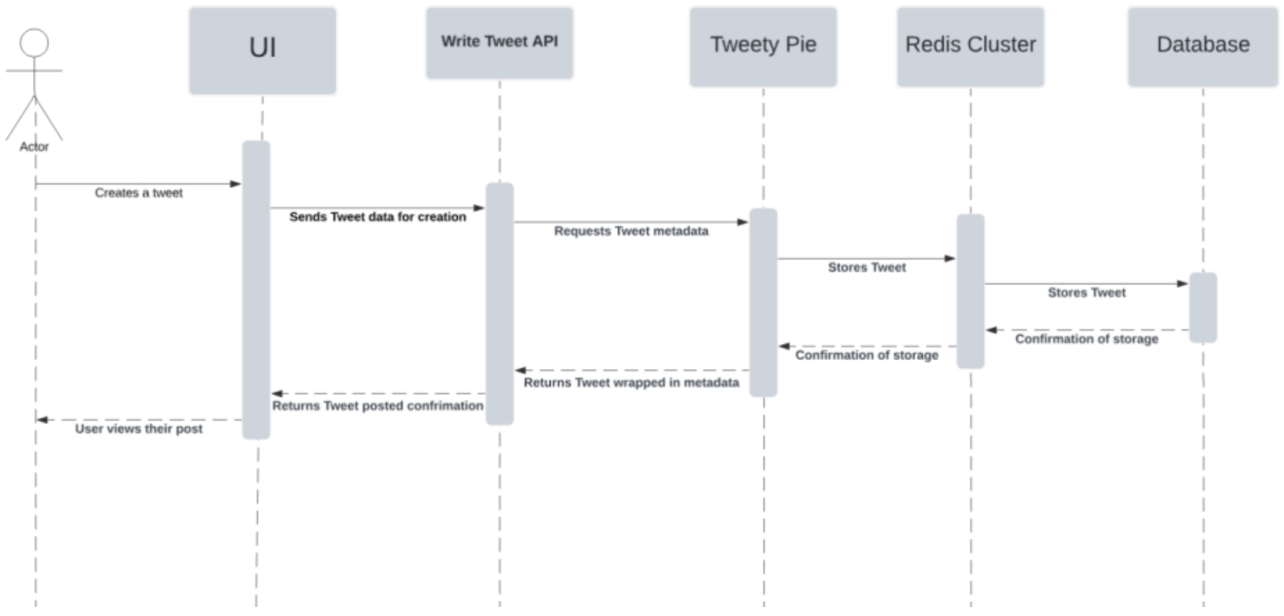

Figure 5: Use Case 2: Tweet Creation

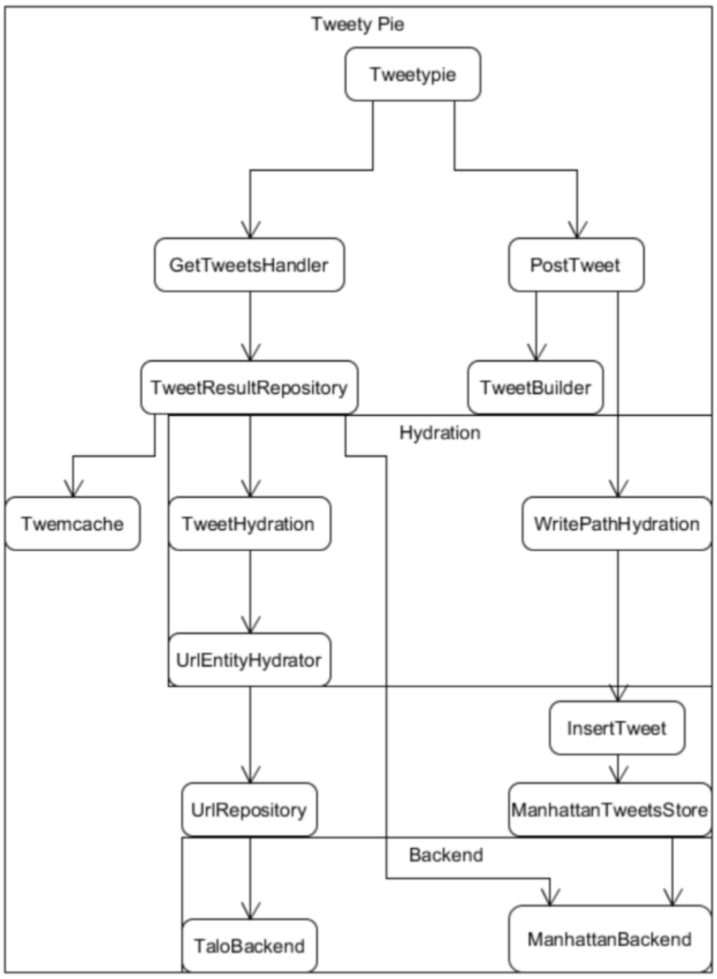

Figure 6: Tweety Pie Diagram

Use cases

Case 1

When creating a Timeline the request is sent from the UI to the Home mixer which then starts the process. Tweets are collected via Social Graph and Candidate generation, Tweety Pie is used to correctly collect and manage the metadata of the tweets. All those tweets are then sent to the heavy ranker which is used to rank and filter the tweets to a variety of different criteria in order to create a proper timeline. There is also the Ads Mixer that is used to add ads in between the tweets. The new timeline is then presented to the user via the UI.(Figure 3) All the components involved use the Manhattan system to access the data that they need to complete their job, showing the Repository style and with the step by step process taken to create a timeline, the pipeline aspect is clearly shown.The sequence diagram shows a basic view of the process that the user goes through in order to view the timeline. (Figure 4)

Case 2

For tweet creation, Twitter has created its own API* responsible for tweet creation. This API* uses Tweety Pie to wrap the tweet with the metadata and then stores it in the Manhattan system. The tweet data is then used by the Home mixer to add to other user's pages. The Home mixer then uses its main components Gizmoduck and Social Graph to get the users that follow the main user so that the tweet can be added to their page. The tweet would be then sent through the same process as the timeline creation. (Figure 5)

External Interfaces

Twitter is set up in a way that abstracts and hides all of the logic and database under an easily navigable interface. For now, these interfaces include web, IOS, and android, all are set up to serve the same way. However, different optimizations are done to the apps in IOS and android to run within the requirements of both OS's. Primarily, Twitter is read heavy, as it is expected as users require content to engage with. However, developers outside of Twitter can use API's, which leverage GraphQL to interact with Twitter's services. Same could be said for posting Tweets. Where much of the interface for users and developers is high level and abstracted of the various services.

Division of Responsibilities

The division of responsibilities amongst the developers of Twitter has significant implications for Twitter's architecture. This is because these teams ensure the platform's efficiency, scalability, and overall functionality. Each team is tasked with specific yet interdependent roles that are essential for maintaining Twitter's real-time and data-heavy operations.

Frontend developers

Frontend developers are [20] primarily responsible for the user-facing aspects of the platform, ensuring that the interfaces on the web, iOS, and Android are not only visually consistent but also optimized for performance and responsiveness. [20]This work heavily relies on seamless integration with backend systems to pull and display real-time data, such as tweets, notifications, and tweet interactions, which requires constant communication with the backend team.

Backend engineers

On the other hand, [21] backend engineers primarily focus on the core services that power Twitter's data infrastructure, including the Home Mixer, Tweety Pie, and the Manhattan database. These engineers are also responsible for managing large-scale data retrieval, ensuring tweets and user interactions are delivered efficiently and accurately to the front end. This role revolves around building and maintaining APIs, handling real-time data synchronization, and ensuring system scalability to accommodate hundreds of millions of users. It's vital for backend engineers to ensure timelines are met while simultaneously optimizing pipelines and caching systems to reduce latency so that users' timelines are updated in real time.

Machine learning engineers

Machine learning engineers [22] contribute by designing and maintaining the recommendation algorithms that personalize user experiences. They work closely with data scientists and engineers to access and process real-time data, refining the models that rank and recommend content based on user behavior and interactions. The output from these machine learning models is integrated into backend systems, which feed personalized content to the frontend. This close collaboration with both backend and data teams is vital to ensure that the recommendations are accurate and delivered with minimal delay.

DevOps engineers

On the other hand, [23] DevOps engineers manage the underlying infrastructure, ensuring that Twitter's services are available, scalable, and secure. They oversee the continuous integration and deployment (CI/CD) pipeline, allowing backend and frontend teams to release updates and new features without disrupting the services. DevOps engineers are also responsible for monitoring system performance, scaling the infrastructure to handle spikes in traffic, and ensuring uptime and reliability. Their work is critical for supporting the other teams, as any infrastructure failure or bottleneck could lead to disruptions in real-time data processing, severely impacting user experience.experience.

Security

The [24]security and compliance teams add another layer of responsibility by ensuring that all systems adhere to strict privacy and data protection standards, such as GDPR. They work across teams to ensure that data security protocols are followed throughout every component of the system, from the frontend that collects user information to the backend services that store and process data. It's mandatory for these teams to adhere to international laws, a task made difficult by Twitter's global user base.

Data engineers

Finally, [25] data engineers are responsible for managing the real-time data pipelines* that provide the foundation for Twitter's operations. They ensure that all data flowing through the system—from tweets and interactions to user preferences—is processed, cleaned, and made available for both backend services and machine learning models. Data engineers play a crucial role in maintaining the integrity and accessibility of this data, as it plays a pivotal role in the performance of other teams such as machine learning, backend, and frontend developers.

Conclusions

In conclusion, Twitter is a very large and complex system that is used by billions for entertainment and knowledge. Its architecture as we explored today is based on the Service-Oriented Architecture style. This is clearly shown with Twitter's recent direction of using APIs as service in order to create a more smooth data flow and spread out the workload. Twitter has many different subsystems that help create that flow and the Home Mixer is one of those subsystems that is the backbone of Twitter's most famous features. The Home Mixer connects to many different components in order to create a very efficient and pipelined system which allows for all kinds of tasks to be completed.

The concurrency of Twitter is achieved with the Redis Clusters by allowing for a smooth and low-latency access to cached data. Having this cache system in place allows for less traffic to the database and allows for a more efficient dataflow. The Home Mixer is used as a repository for different systems and is the starting point of many different pipelines each focusing on a specific task.

Organization and clear data flow is necessary for a large system to work, Twitter has worked hard on creating an overall system that has proper concurrency and an interconnected system. Each sub-system works together to complete all kinds of tasks and the concurrency of the data flow allows for Twitter to be the work of art that it is.

Lessons Learned

When starting the research for the report, we were overwhelmed by the number of different resources and files that Twitter has. Finding the right resources to aid in our report proved to be a challenge that could only be overcome with guidance and tedious analysis. We learned that we should focus on resources that are well made with diagrams and clear points to help with the explanation. By filtering out the vague or weak resources, we were able to collect strong and backed information to create the report. Having a clear understanding of the depth of the system analysis would've been much more helpful during the start of our research. Twitter is very complex and many things can be taken from it. This becomes a challenge when we have to analyze its system whether it be high-level or low.

Glossary

- **Node:** A fundamental unit in a data structure, such as linked lists, trees, and graphs. In networking, a node refers to a device or data point in a larger network, such as a computer, router, or switch.
- **Cluster:** A group of interconnected computers that work together to perform complex tasks as if they were a single system. Clustering improves performance, fault tolerance, and scalability.
- **Distributed System:** A system where multiple computers (nodes) work together, appearing as a single unit to the end user. These systems help improve performance, fault tolerance, and scalability across a network of machines.
- **Cache:** A storage layer that holds frequently accessed data in memory to speed up read and write operations. Caching improves the efficiency of data retrieval and reduces the load on databases or slower storage systems.
- **Load Balancer:** A system that distributes incoming network traffic across multiple servers to ensure no single server becomes overwhelmed. This increases the availability and reliability of applications.
- **Virtual Machine (VM):** A software emulation of a physical computer that runs an operating system and applications. VMs are commonly used in cloud computing to run isolated environments on a single physical machine.
- **API (Application Programming Interface):** A set of protocols and tools for building software applications. APIs allow different software systems to communicate and share data efficiently.

- **Throughput:** The amount of data processed by a system over a given period. High throughput is essential for systems handling large volumes of data, such as databases or networks.
- **Data Pipeline:** A series of data processing steps, from data collection and transformation to storage and analysis. Data pipelines are used to automate the flow of data in distributed systems and big data applications.

References

1. Wikipedia. (n.d.). List of Twitter features. *Wikipedia*. Retrieved October 12, 2024, from https://en.wikipedia.org/wiki/List_of_Twitter_features
2. Somadevtoo. (2020, September 21). Twitter system design example for tech interviews. *DEV*. Retrieved October 12, 2024, from https://dev.to/somadevtoo/twitter-system-design-example-for-tech-interviews-1ihb
3. Karan. (2020, January 15). System design: Twitter. *Medium*. Retrieved October 12, 2024, from https://medium.com/@karan99/system-design-twitter-793ab06c9355
4. Algodaily. (n.d.). Design of the Twitter architecture. *Algodaily*. Retrieved October 12, 2024, from https://algodaily.com/lessons/design-of-the-twitter-architecture
5. High Scalability. (2015, October 12). The architecture Twitter uses to deal with 150M active users. *High Scalability*. Retrieved October 12, 2024, from https://highscalability.com/the-architecture-twitter-uses-to-deal-with-150m-active-users/
6. Abkedia. (2020, October 10). Earlybird: Real-time search at Twitter — A summary (Part 1). *Medium*. Retrieved October 12, 2024, from https://abkedia.medium.com/earlybird-real-time-search-at-twitter-a-summary-part-1-5aa68221ef85
7. Educative. (2023). What engineers need to know about Twitter's design, new and old. from https://www.educative.io/blog/twitter-design#Twitters-current-architecture
8. Arrington, M. (2008, April 29). End Of Speculation: The Real Twitter Usage Numbers. *TechCrunch*. Retrieved October 13, 2024, from https://techcrunch.com/2008/04/29/end-of-speculation-the-real-twitter-usage-numbers
9. Avram, A. (2009, July 26). Twitter, an Evolving Architecture. *InfoQ: Software Development News, Trends & Best Practices*. Retrieved October 13, 2024, from https://www.infoq.com/news/2009/06/Twitter-Architecture/
10. Design Gurus Team. (n.d.). What is the software architecture of Twitter? *Design Gurus*. Retrieved October 13, 2024, from https://www.designgurus.io/answers/detail/what-is-the-software-architecture-of-twitter
11. How Twitter Stores 250 Million Tweets a Day Using MySQL. (2011, December 19). *High Scalability*. Retrieved October 13, 2024, from https://highscalability.com/how-twitter-stores-250-million-tweets-a-day-using-mysql/

12. Pardes, A. (2017, September 26). As Twitter Doubles Its Character Count, a Brief History of Their Many Changes. *WIRED*. Retrieved October 13, 2024, from https://www.wired.com/story/a-brief-history-of-the-ever-expanding-tweet/

13. Rottinghuis, J., & Oanta, R. (2019, April 8). Partly Cloudy: The start of a journey into the cloud. *Blog*. Retrieved October 13, 2024, from https://blog.x.com/engineering/en_us/topics/infrastructure/2019/the-start-of-a-journey-into-the-cloud

14. Singh, M. (2022, December 28). Twitter suffered outage after Elon Musk made 'significant' backend server architecture changes. *TechCrunch*. Retrieved October 13, 2024, from https://techcrunch.com/2022/12/28/twitter-down-outage/

15. Twitter Selects AWS as Strategic Provider to Serve Timelines. (2020, December 15). *Amazon Press Release*. Retrieved October 13, 2024, from https://press.aboutamazon.com/2020/12/twitter-selects-aws-as-strategic-provider-to-serve-timelines

16. Twitter was founded on this day in 2006, here's all you need to know. (2024, July 15). *Business Standard*. Retrieved October 13, 2024, from https://www.business-standard.com/world-news/twitter-was-founded-on-this-day-in-2006-here-s-all-you-need-to-know-124071500394_1.html

17. Ahmad, A. (2023, June 15). Monolithic vs. Service-Oriented vs. Microservice Architecture. *Design Gurus*. Retrieved October 13, 2024, from https://www.designgurus.io/blog/monolithic-service-oriented-microservice-architecture

18. Cloud, J. (2013, July 3). Decomposing Twitter: Adventures in Service-Oriented architecture. *InfoQ*. Retrieved October 13, 2024, from https://www.infoq.com/presentations/twitter-soa/

19. Sam, D. (2022, March 9). Twitter's tough architectural decision. *Medium*. Retrieved October 13, 2024, from https://dennysam.medium.com/twitters-tough-architectural-decision-c61e4d0d41a5

20. Scalability, H. (2014, September 8). How Twitter uses Redis to scale - 105TB RAM, 39MM QPS, 10,000+ instances. *High Scalability*. Retrieved October 13, 2024, from https://highscalability.com/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins/

21. Twitter-team. (2023, May 19). The-algorithm. *GitHub*. Retrieved October 13, 2024, from https://github.com/twitter/the-algorithm/tree/main

22. Coursera Team. (2024, April 3). What Does a Back-End Developer Do? *Coursera*. Retrieved October 13, 2024, from https://www.coursera.org/articles/back-end-developer

23. Coursera Staff. (2024, October 10). What Is a Machine Learning Engineer? (+ How to Get Started). *Coursera*. Retrieved October 13, 2024, from https://www.coursera.org/articles/what-is-machine-learning-engineer

24. Hall, T. (n.d.). What is a DevOps Engineer? *Atlassian*. Retrieved October 13, 2024, from https://www.atlassian.com/devops/what-is-devops/devops-engineer

25. Entrustech. (2023, June 12). Twitter Engineering: An Overview. *Medium*. Retrieved October 13, 2024, from https://medium.com/@entrustech/twitter-engineering-an-overview-8759f3212fdb

26. Coursea Editorial Team. (2024, May 3). What Is a Data Engineer? A Guide to This In-Demand Career. *Coursera*. Retrieved October 13, 2024, from https://www.coursera.org/ca/articles/what-does-a-data-engineer-do-and-how-do-i-become-one

27. Haq, F. u. (2023, January 9). What engineers need to know about Twitter's design, new and old. *Educative.io*. Retrieved October 13, 2024, from https://www.educative.io/blog/twitter-design#Twitters-current-architecture