UBER & BIG DATA

Prabesh Regmi

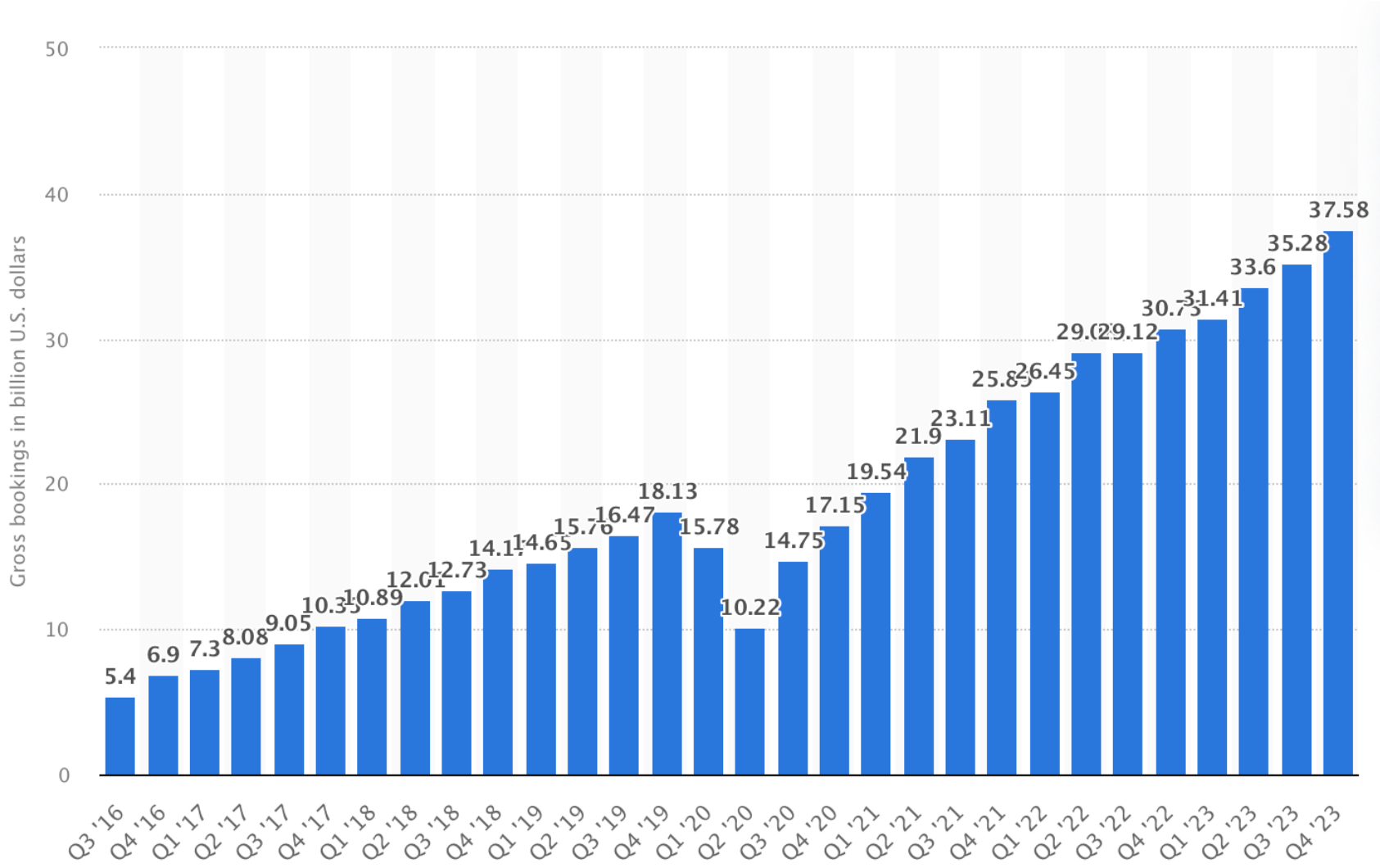
Uber

- Uber Technologies, Inc., referred as Uber, is an American multinational transportation company that provides ride-hailing services, courier services, food delivery, and freight transport.
- The company was founded in San Francisco in 2009 by Travis Kalanick and Garett Camp and started marketing the free mobile application in 2011.
- Currently Uber operates in 70 countries, and more than 10,000 cities as of 2024(May).
- Over 9.4 billion trips were carried out in 2023.

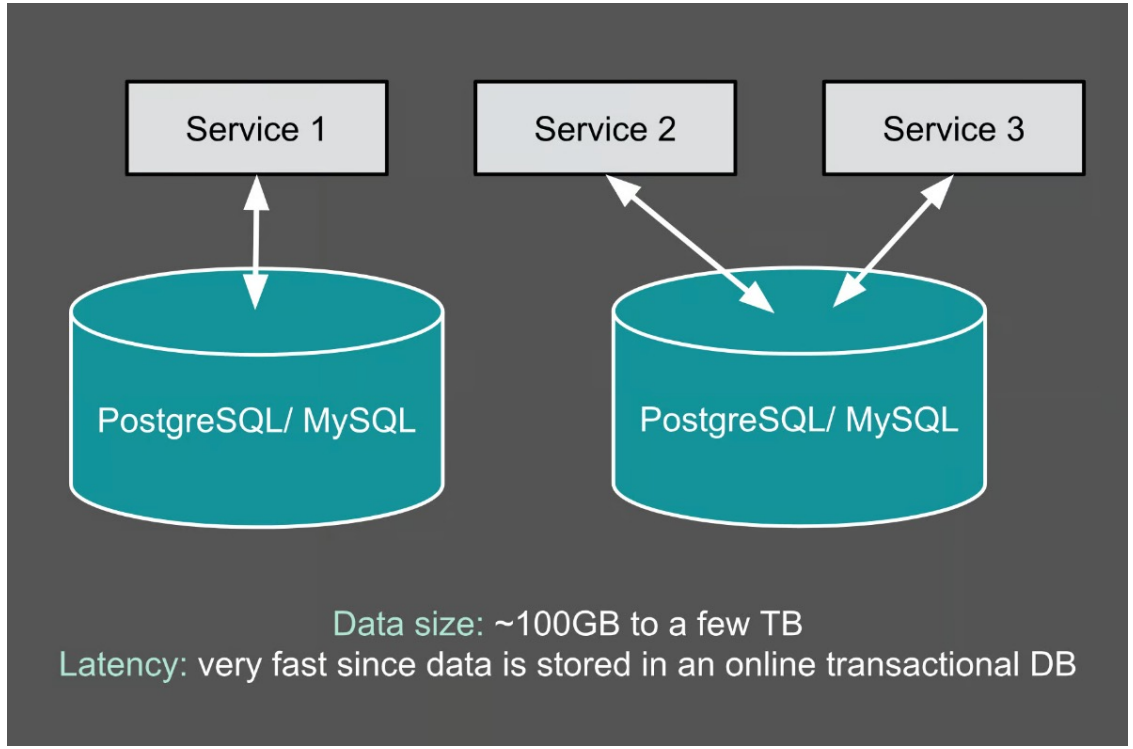
Problem Statement

- Uber depends heavily on data-driven decisions at each stage from forecasting rider during high traffic events to identifying and addressing bottlenecks in driver-partner sign-up approach.
- Over time, there are over 100 Petabytes of data at that needs to be cleaned, stored and served with minimal latency.
- Development of big data solution that assured reliability, scalability, ease of use, fast and efficient.

Uber's Gross booking chart



Generation 0 – Before Big Data at Uber (Before 2014)



- Limited data (few terabytes) could fit into few traditional online transaction processing (OLTP) database.
  - MySQL
  - PostgreSQL
- Data were fast with latency less than one minute.
- There was no global access or global view of all stored data.
- Data was scattered accross different OLTP database.

Why Big Data in Uber?

- The exponential growth of the company led the build of an analytical data warehouse.
- The need of accessibility of analytical data to analysts to make data data driven possible.
- The users had to figure out how to query across databases.
- The data users were categorized into:
  - City operations teams (thousands of users) :
    These are on-the-ground crews to manage and scale Uber's transportation network in each market.
    They access the data on a regular basis to respond to driver-and-rider-specific issues
  - Data scientists and analysts (hundreds of users):
    This includes analysts and scientists spread across different functional groups that need data to help deliver high level transportation and delivery experiences to the users (e.g. forecasting rider demand)
  - Engineering teams (hundreds of users):
    They are engineers focused on building automated data applications, such as Fraud Detection and driver onboarding platforms.

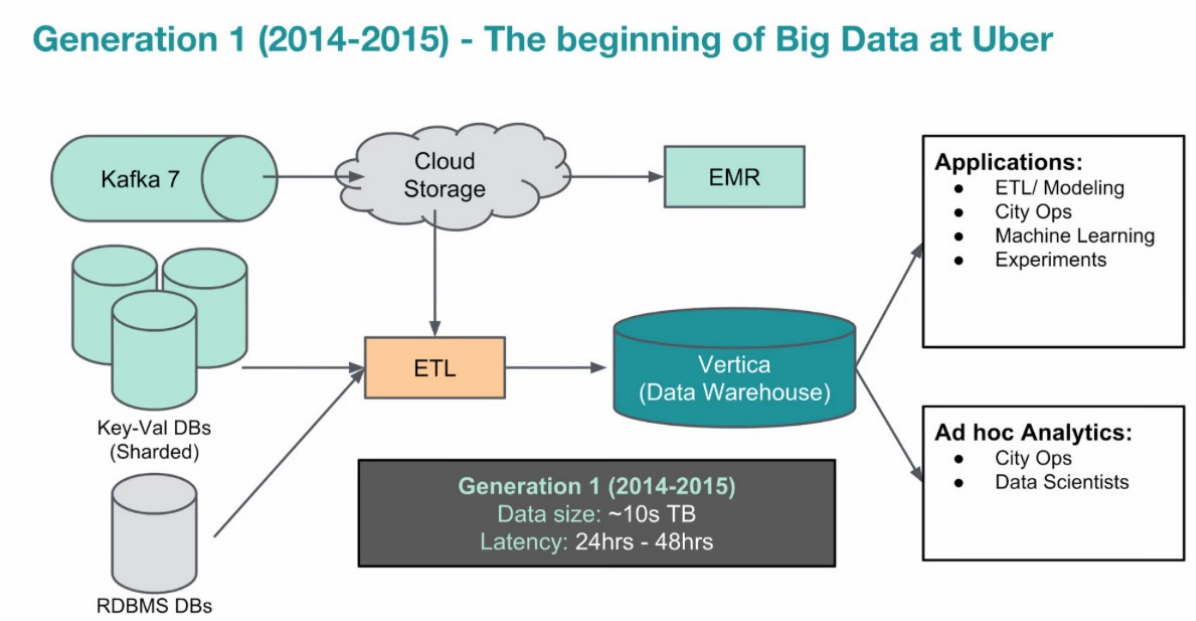
Why Big Data in Uber?

- Handling massive scale of data ingestion and processing was the major issue

Alternatives

- Oracle (RDBMS) or Microsoft SQL Server(RDBMS) could be used as they provide better data consistency and data freshness but could be higher cost could be the major tradeoff.
- As Uber was startup with small data size, they might have preferred PostgreSQL/MySQL because of their lower cost and sufficient functionality.

Generation 1 – The Beginning of Big Data at Uber (2014-2015)



- Focus on aggregating all data in one place along with streamlining data access.
- Use of Vertica data ware house software(column-oriented).
- Development of multiple adhoc ETL, that copies data from different sources

  AWS → Vertica,

  OLTP databases → Vertica,

  Service logs → Vertica etc:

  …

Generation 1

- Standardized SQL to built an online query system using SQL.
- Analytical data size increased to 10s of terabytes.
- Number of users increared to several hundred.
- Use of SQL enabled city operators could easily interact with the data without knowing about the underlying technologies.

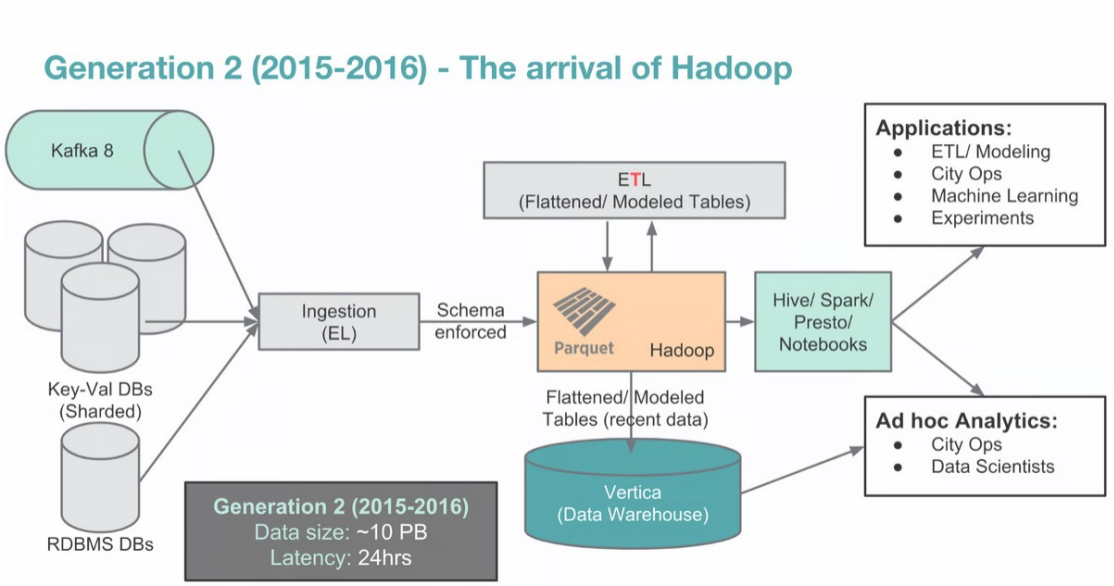
Limitations

- When data in JSON format was ingested through adhoc ETL jobs, data reliability became an issue.
- Data warehouse became incresingly expensive to scale.
- Lack of horizontal scalability.
- Lack of formal schema between data producers and consumers lead to difficulty in ingestion of data and duplication of data.

Alternatives

- Use of Apache Cassandra instead of Vertica as it supports horizontal scaling
- Use of Kafka Clustering instead of single Kafka node for better scalability, high availability and fault tolerance
- Both Apache Cassandra and Kafka clustering required more resource and expertise increasing complexity and cost

Generation 2 – The Arrival of Hadoop (2014-2015)

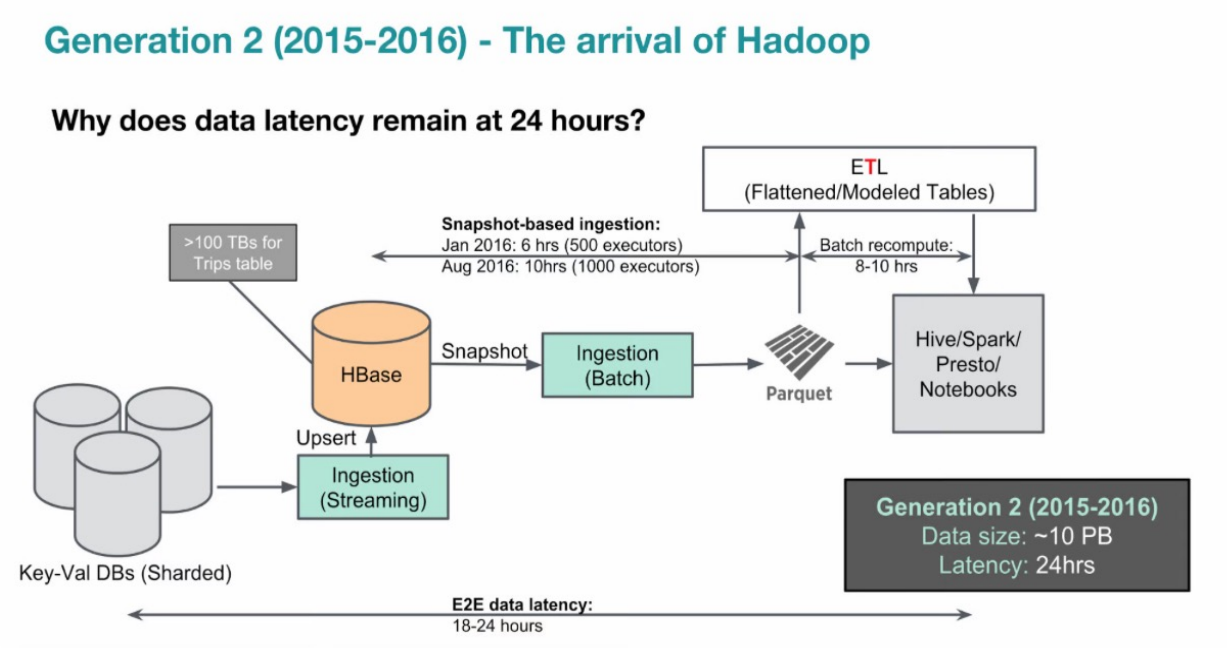


- Adoption of Hadoop data lake(ingest data from multiple stores without transforming it)
- Use of Apache Spark, Apache Hive and Pestro to access data
  - Pestro: Interactive ad hoc user queries
  - Apache Spark: Programmatic access to raw data
  - Apache Hive: Heavy queries
- All data modeling and transformation only happened in Hadoop enabling fast backfilling and recovert when issues arose

Generation 2

- Only the most critical modeled tables were transferred to the data warehouse.
  - Quick SQL queries
  - Lower operational cost
- Standard columnar file format now uses Apache Parquet resulting improvised compression ratio and seemless integration with Apache Spark
- Amount of data grew to 10s of petabytes
- Data infrastuctures ran 100k jobs per day across 10000 virtual CPU cores
- Number of users grew to thousands

Limitations



- Huge number of small files was used in HDFS which led to pressure on HDFS Name Nodes if data size is greater than 50-100 petabytes.
- New data was accessible to users once every 24 hours so no real time decisions could be made
- As HDFS and Parquet do not support data updates, all ingestion jobs required to create new snapshots from updated source data
  - Ingest new snapshot to Hadoop
  - Convert to Parquet format
  - Swap output tables
  - View new data

Alternatives

- Apache Drill could be used an alternative to Presto as Apache Drill is a distributed SQL query engine designed for interactive analysis of large-scale datasets across different data sources.
- Pestro offers high performance and SQL compactibility, but Apache Drill's schema free approach and support for various data formats and source could have been better for dealing diverse and complex data environments.
- While they used HDFS for data storage, a better alternative could be Google Bigquery as it is fully managed, petabyte-scale, low-cost analytics data warehouse.
- Instead of Apache Spark for data processing, Google Cloud Dataflow could be an alternative

Generation 3 – Let's Rebuild for Long Term (2017-present)

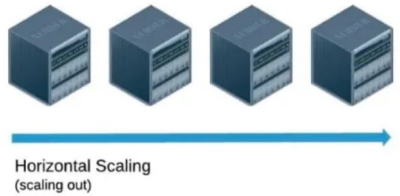
- Uber's Hadoop analytics architecture was hitting scalability limitations and high latency was seen in many service
- HDFS bottlenecks were eliminated by tuning NameNode garbage collection, limiting the number of small files and moving data to different clusters
- 24-hr data latency was eliminated by incremental ingestion of only updated and new data
- A new framework to support update/delete operations over HDFS was developed
- Instead of creating new snapshot or rebuilding derived table on evey run only changed data is pulled out from raw source table and previous derived output table is updated

Generation 3

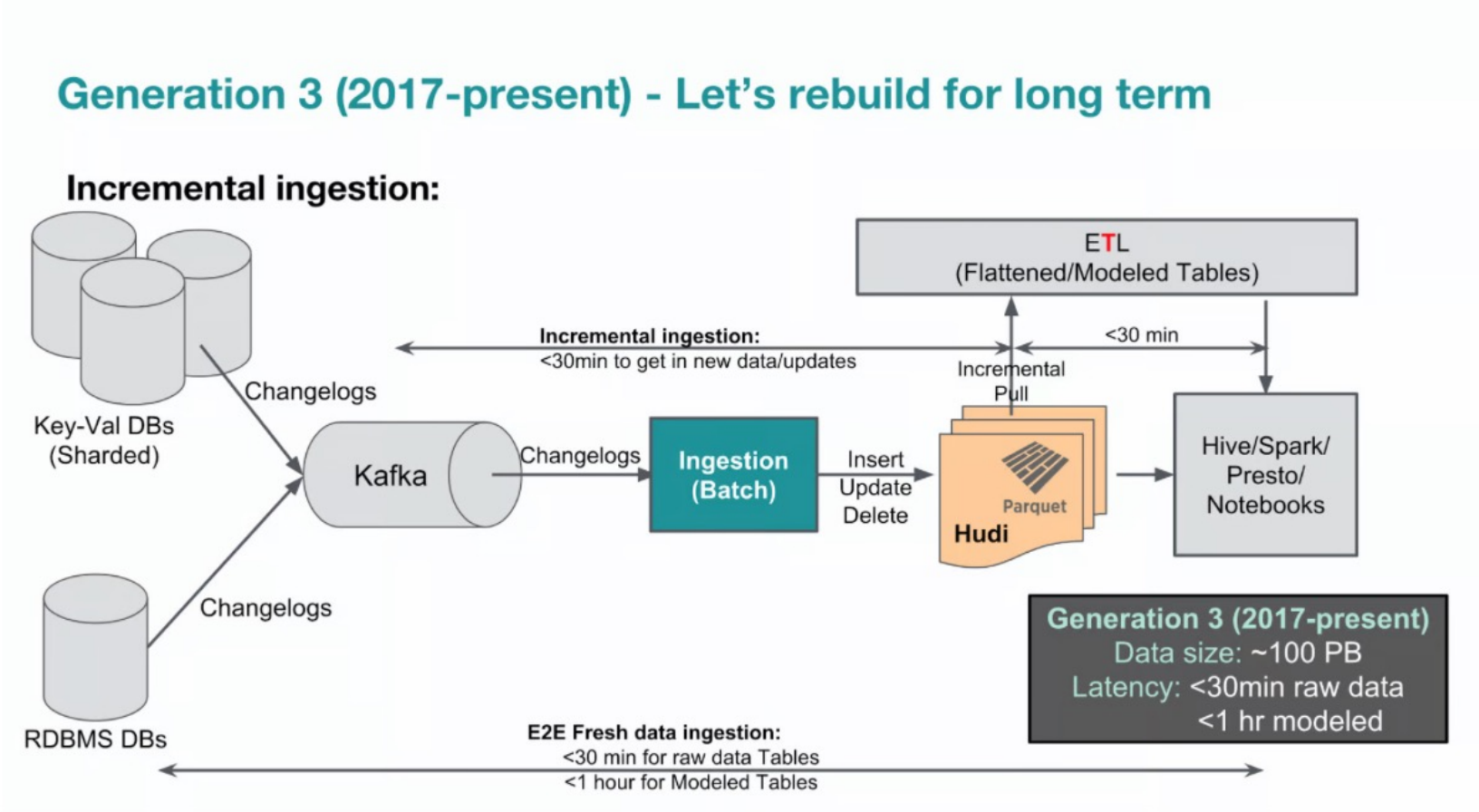
- Amount of data grew to 100 petabytes in Hadoop
- Data infrastuctures ran 100k jobs per day across 100,000 virtual CPU cores
- Around 100,000 Presto queries, 10,000 Spark jobs and 20,000 Hive queries are run per day

Hudi(Hadoop Upserts anD Incrementals)

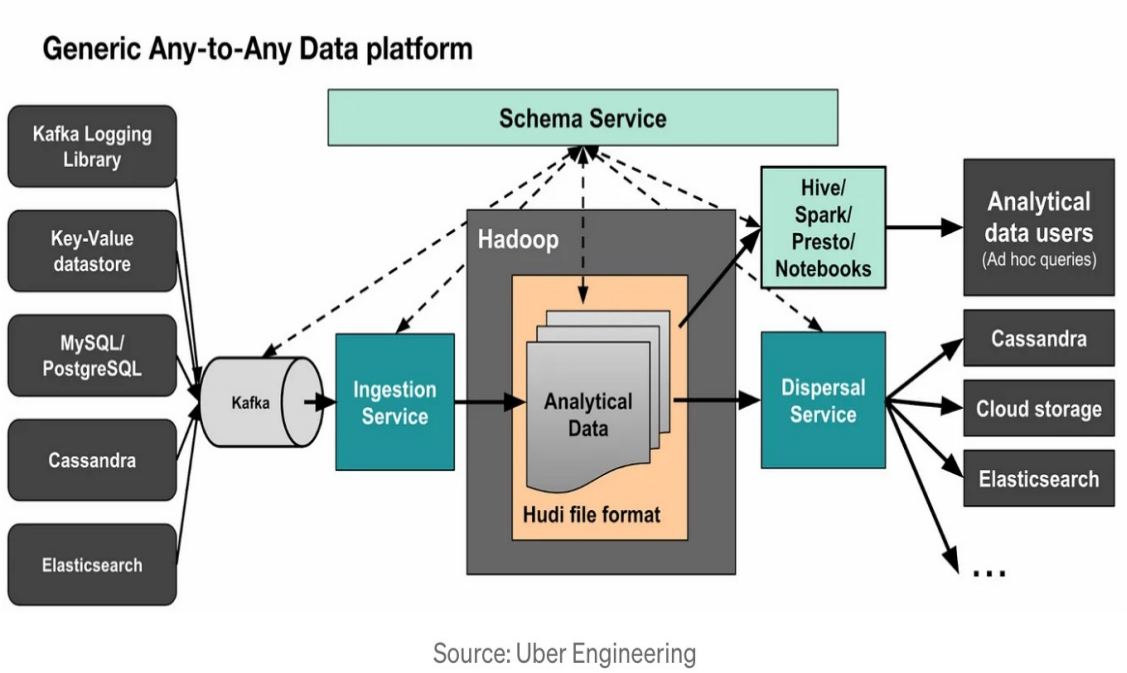
- It is one of the open-source Spark library that provides an abstraction layer on top of HDFS and Parquet to support the required update and delete operations
- It works by letting users query by their last checkpoint timestamp to fetch all the data that have been updated since the checkpoint, without the need to run a full table scan
- Brought down data latency from 24 hours to less than an hour for modeled data and 30 minutes for raw data.
- Moreover, it can be used from any Spark job, is horizontally scalable and only relies on HDFS to operate.



Generation 3 – Let's Rebuild for Long Term (2017-present)



Generic Data Ingestion



Source: Uber Engineering

- Along with Hudi, other addition to Uber's big data platform is data ingestion through Apache Kafka with unified Avro encoding including metadata headers attached.
- Other data ingestion platform, Marmaray runs in mini-batches, and picks up the upstream storage changelogs from Kafka, applying them on top of the existing data in Hadoop using Hudi library.

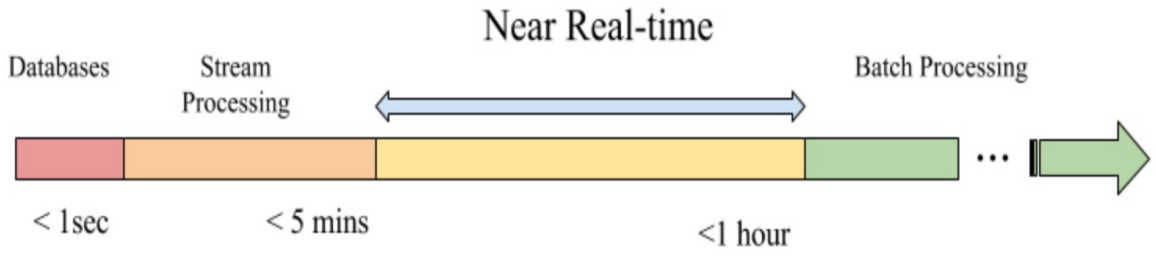
Incremental Data Modeling

Traditional λ architecture provides: Streaming vs Batch solutions

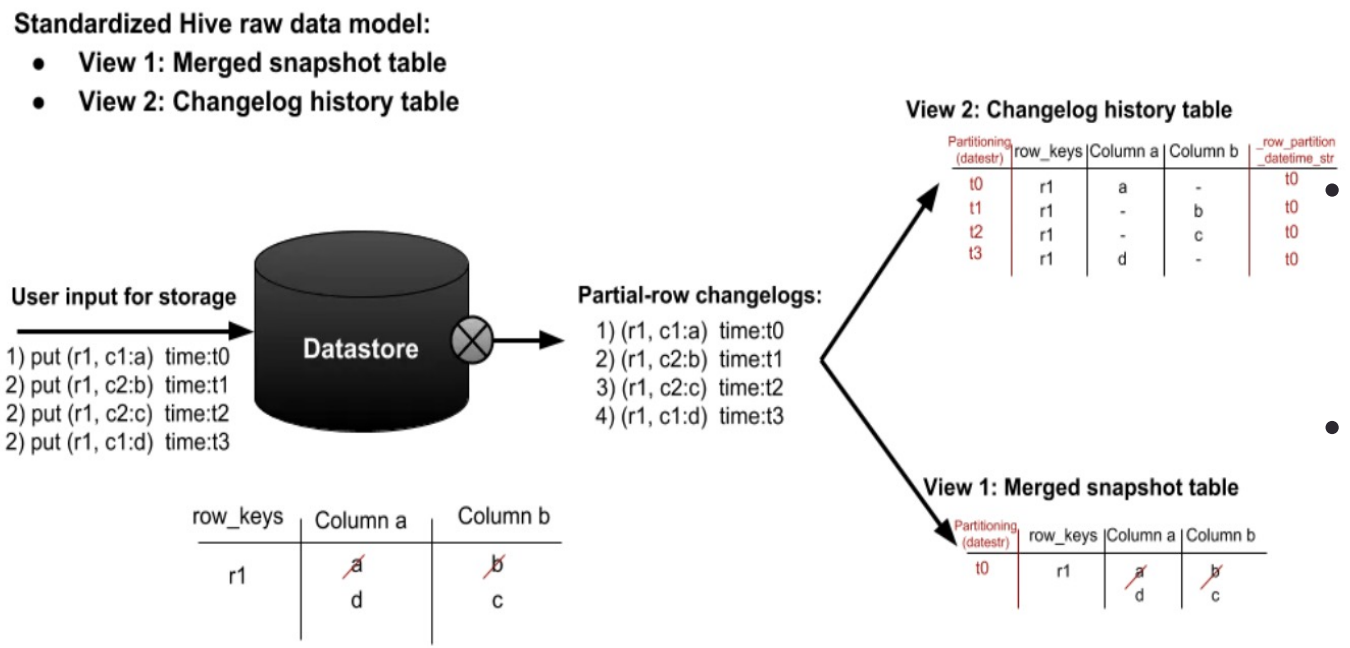
- That assumes append-only immutable data
- Processing based on timestamps (skips late-arriving data)

Incremental Processing is mini-batch jobs pulling out only changed data

- Gets recently appended data as well as old changed/updated records
- Provides high completeness
- Processing is no longer limited by updates/deletes or late-arriving data
- Supports full batch functionality (eg. Joins,..)



Standardize data model



- Uber standardized their data model to provide two types of tables for all raw Hadoop data.
- Changelog history table containing all changelog received for specific upstream table
- Merged snapshot table containing the compacted merged view of all the historical changelogs received per key

Alternatives

- Apache Iceberg could be used together with Apache Parquet as Apache Iceberg provides better support for real-time data processing and improvised data warehousing and querying capability.
- Requirement of more resource and expertise could have been the major tradeoff if Apache Iceberg was chosen along with Apache Parquet.

Generation 4 – Whats next?

- To enhance data quality avoiding non-schema-confirming data when some of the upstream data stores do not mandatorily enforce, checking data schema before storage(eg., storing a key-value when the value is JSON blob)
- Expand schema service to support semantic checks(allows extra constrains in actual data content
- Improve data latency in Hadoop to five minutes and in modeled tables to ten minutes
- Expansion of Hudi project to support an additional view mode, which will include the existing read-optimized view, as well as a new real-time view
- Move away from relying on dedicated hardware to dockerization
- New version of Hudi that allows to generate larger parquet files (over 1 GB, current – 128 MB)
- Hudi enables to store updated record in separate delta file and asynchronously merges it with base parquet file

What surprised me?

- Uber's big data platform handles over 100 petabytes of analytical data, despite this massive volume it is designed with minimal latency.
- The platform has evolved from traditional ETL jobs to relational databases system based on Hadoop and Spark which shows their adaptability in meeting growing data needs.
- Uber has made significant efforts to improve the efficiency of their data platform to reduce costs.
- They use big data platform to make data driven decision at entry level from forecasting rider demand during high traffic events to identifying and addressing bottlenecks in driver sign-up process.

Conclusion

- Uber's need of insights resulted in over 100 PBs of analytical data that needs to be cleaned, stored and served with minimum latency through their Apache Hadoop based big data platform
- DataCentral has been a critical tool for engineers, data analysts, and platform teams at Uber
- From early 2014 Uber has worked for development of Big Data solution to ensure data relaibility, scalability and ease-of-use
- Uber is now focused on increasing their platform's speed and efficiency
- Uber's exponential growth was not possible without work on leveraging next-generation ML and AI technologies to optimize the product and user-experience
- The cost for running big data platform has been rising significantly. However they have made several efforts and imrovements in platform efficiency, minimizing the cost

References

- "Uber's Big Data Platform: 100+ Petabytes with Minute Latency", Reza Reza, , Uber Eng blog, 2018
- "Global gross booking volume of Uber from Q3 2016 to Q4 2023", Statista
- "Scaling Uber's Hadoop Distributed File System for Growth", Ang Zhang, Wei Yan, Uber Eng blog, 2018
- "How Uber Delivers Big Data in Less Than an Hour", Roy Telles, Medium,2022
- "Streaming Data Who's Who: Kafka, Kinesis, Flume, and Storm", Jake Dolezal, tdwi,2016
- "Cost-Efficient Open Source Big Data Platform at Uber", Zheng Shao, Mohammad Islam, Uber Eng blog, 2021