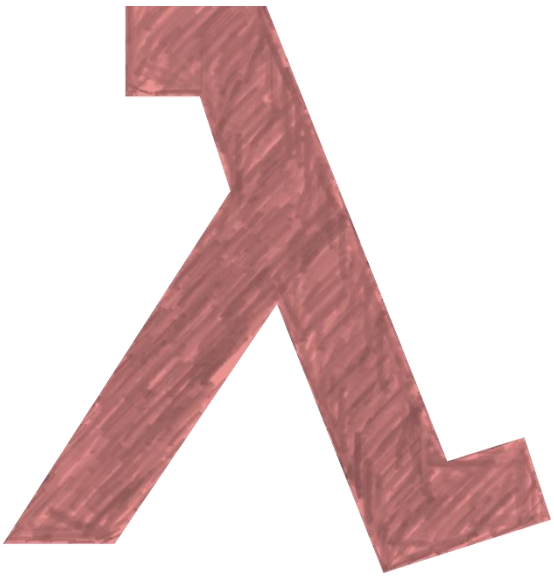


Lambda Architecture

Near Real-Time Big Data Analytics Using Hadoop

January 2015

White Paper

Email: bdg@qburst.com | Website: www.qburst.com

Contents

Overview

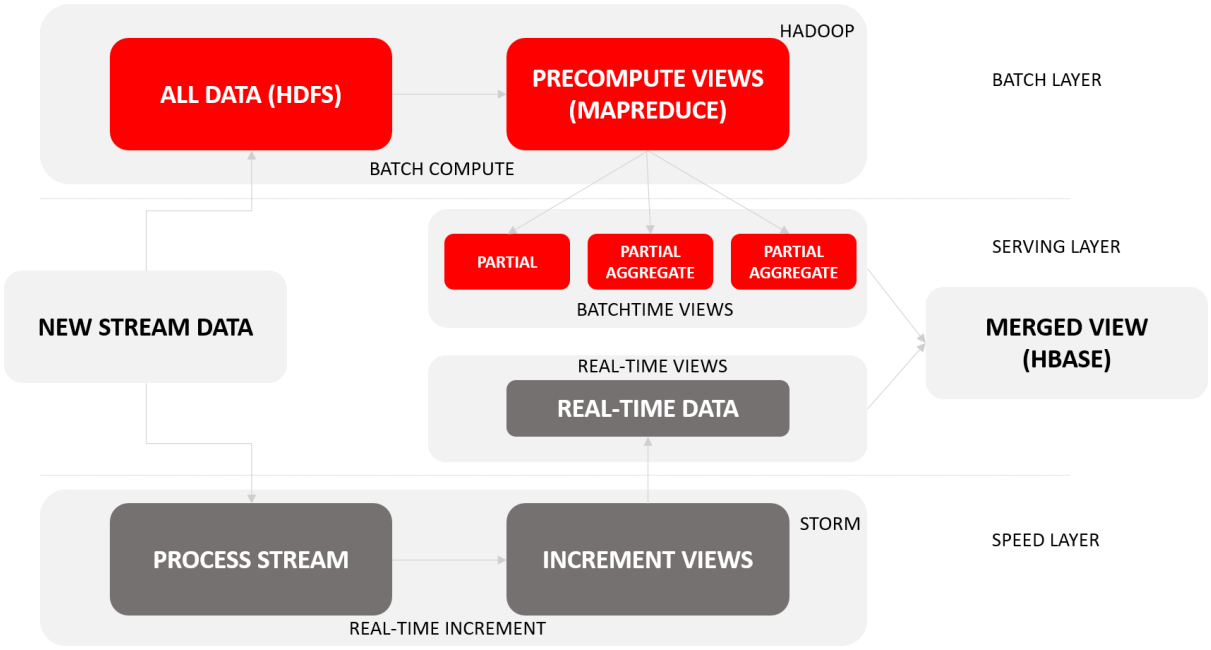
In a nutshell, data architecture deals with physical and logical models/paradigms that govern data management, storage, and processing. The initial phase of Big Data implementation is not about decisions regarding Big Data infrastructure. Research on data warehousing and analytics options will have to be supplemented by at least a peripheral grasp of how underlying data architecture and processing methodology work. A successful implementation relies on well-defined business requirements, which in turn influence the choice of infrastructure and architectural design. The resulting data model is obtained by combining various tools and techniques that cater to your unique use case.

Nathan Marz had worked for Twitter and Backtype, where he gained considerable experience with distributed data processing systems. He is also associated with the open-source tool Storm. Inspired by the foremost concern for high latency in distributed systems, Marz and his team proposed Lambda Architecture. It was intended as an alternative that fuses properties of both batch and stream processing. In simple terms, Lambda Architecture outlines a generic structure to build fast, scalable, fault tolerant, and maintainable Big Data solutions.

Speaking of data architecture, it is impossible to strike a “one-size-fits-all/many” use-cases posture. This white paper is an attempt to contextualize the concepts underlying Lambda Architecture through its application in building data systems that seek to marry high latency and near real-time processing.

Lambda Architecture: A Quick Introduction

The Lambda Architecture design comprises batch layer, serving layer, and speed layer. In simple terms, batch layer stores the immutable growing master dataset on which it generates precomputed views, though with high latency. The marked difference of speed layer from batch layer is the low latency with which it produces real-time views. This layer performs incremental updates and generates views on latest entries rather than the entire dataset.



Source: www.mapr.com

Batch Layer

The batch layer contains the immutable, constantly growing master dataset stored on a distributed file system such as HDFS. With batch processing (using MapReduce), arbitrary views (or batch views) are computed from this raw dataset. Hadoop is a typical example of a batch layer tool.

Serving Layer

The job of the serving layer is to load and expose the batch views in a datastore so that they can be queried. This serving layer datastore does not require random writes – but must support batch updates and random reads – and can therefore be extraordinarily simple. ElephantDB and Voldemort are examples of datastores of this type.

Speed Layer

This layer deals only with new data and compensates for the high latency updates of the serving layer. It leverages stream processing systems (Storm, S4, Spark) and random read/write datastores to compute the real-time views (HBase). These views remain valid until the data have found their way through the batch and serving layers.

To get a near real-time result, the batch and stream processing views must be queried and the results merged together.

The Case for Lambda Architecture

NoSQL and relational databases are much discussed in connection with Big Data analytics. Components which work in parallel and complement each other are assembled to store and process large volumes of unstructured data to ensure consistency and availability. Hadoop cluster is an example of this arrangement.

Hadoop is highly scalable: if a cluster's processing power is overwhelmed by growing volumes of data, additional cluster nodes can be added to maintain throughput. Hadoop clusters are also equipped to handle consistency issues since each piece of data is copied onto other cluster nodes; this ensures that data is not lost even if a node fails. This process is called data replication.

However, frameworks such as Hadoop and other vendor market software are just components of Big Data infrastructure. The effectiveness of Big Data solutions depends highly on the robustness of underlying data architecture.

Lambda Architecture aims to serve applications which perform asynchronous transformation operations such as data mining, query execution, sorting, merging, or aggregating datasets. In simple terms, asynchronous transformations create a new memory buffer for the output as compared to the input during ETL execution. This is so because the output may have more or less records than input data depending on the operation.

As a Big Data paradigm, Lambda Architecture collates batch and stream processing frameworks to address the following requirements:

- Historical and real-time computation
- A robust system equipped to handle fault tolerance
- Low-latency reads and writes on large volume of data while supporting ad hoc queries
- Linear scalability to handle unlimited data by adding more nodes to the database
- Capacity to debug the system with minimal maintenance

Batch and serving layers ensure fault tolerance, scalability, ad hoc queries, and debugging, thanks to the immutable input master dataset and output view. Batch layer also facilitates availability through data replication across multiple nodes. The tradeoff is a latency of a few hours while batch and server layers generate precomputed views and load indexed data for querying.

Speed layer is a real-time data system that compensates for this latency. Through incremental updates, the speed layer continually modifies the real-time view instead of imitating the batch layer which recomputes the entire dataset to accommodate newer entries. In other words, speed layer addresses the latency issue resulting from reprocessing.

Case Study: Application for Sentiment Analysis in Social Media

This application is a web-based analytics tool that monitors sentiment trends, influencers, and related information about sports celebrities from social media platforms.

Use Case

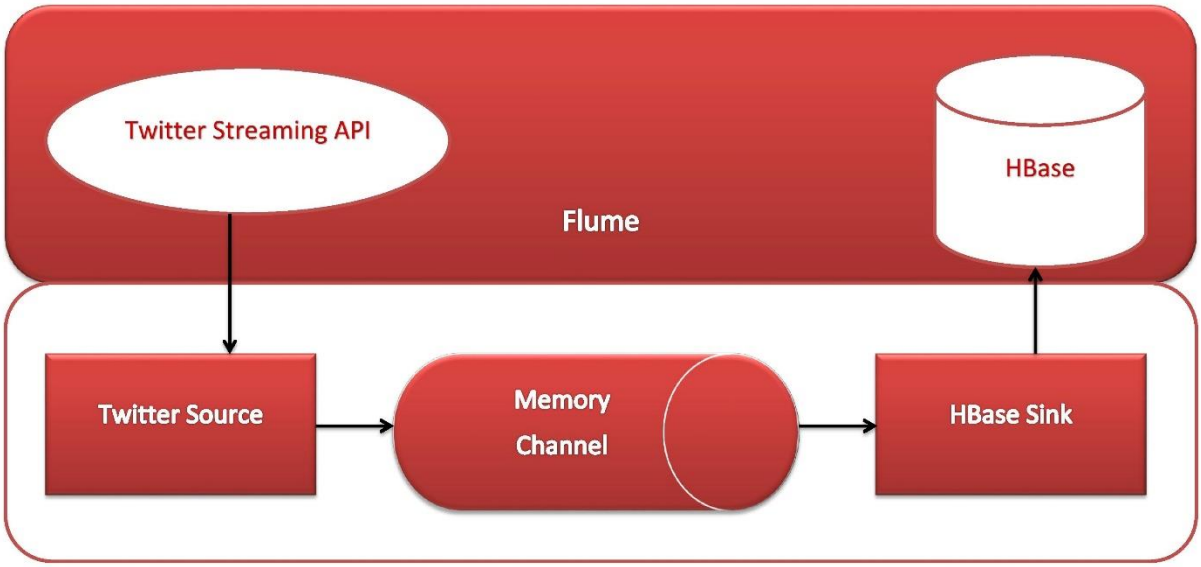
- The application ingests streaming tweets to Flume for batch processing and a custom source called Twitter Source for stream processing.
- The scalable and immutable dataset is stored and processed in Hadoop (batch layer).
- The final application displays sentiment analysis and top influencers.

Technologies

- Apache Flume
- HBase (Hadoop)
- MapReduce (Hadoop)
- MongoDB
- Redis
- Zookeeper

Integration Layer: Ingestion and processing of large amounts of data is done using Flume. Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.

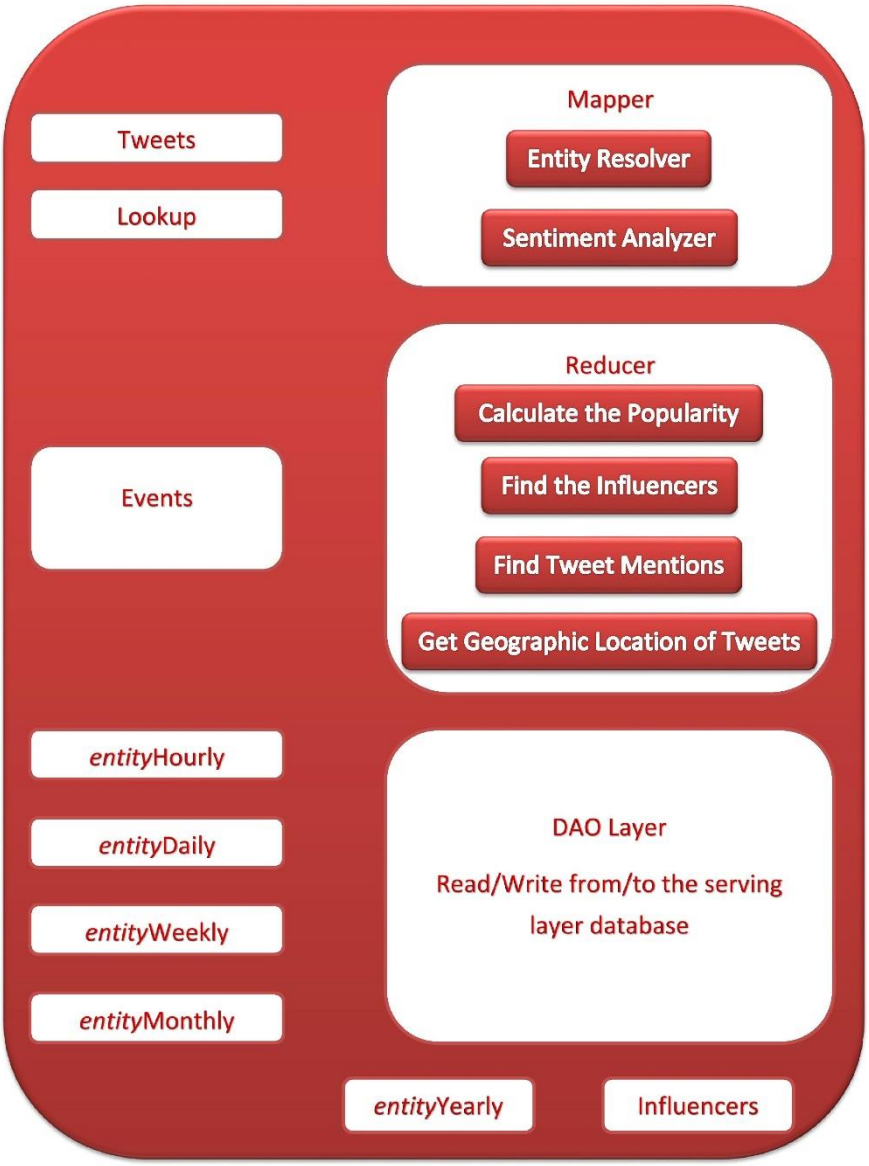
In this application, Flume is used to stream and filter Twitter data on sports celebrities. This data is stored in HBase. Flume achieves this by defining streaming Twitter API data into sources, channels, and sinks.



Batch Layer: Hadoop has been used as the batch layer. HBase manages the data received from the Flume service, while MapReduce performs the computation. MapReduce reads the tweets from the HBase table and performs entity recognition based on the entities present in the lookup table. It then calculates the sentiments of the tweets and passes the result onto a grouping parameter, which matches sentiments that correspond to each celebrity.

It stores the last-run timestamp into the distributed configuration store when the job is successful. On the next run, the job only picks up rows that have changed since the saved timestamp. This removes any duplicate entries and prevents stale data from making its way into the serving layer.

ZooKeeper is a centralized service used for maintaining configuration information and naming, besides providing distributed synchronization and group services. In this instance, Zookeeper distributes the configurations among all nodes for batch layer processes. The last runtime of the batch process is stored in Zookeeper.



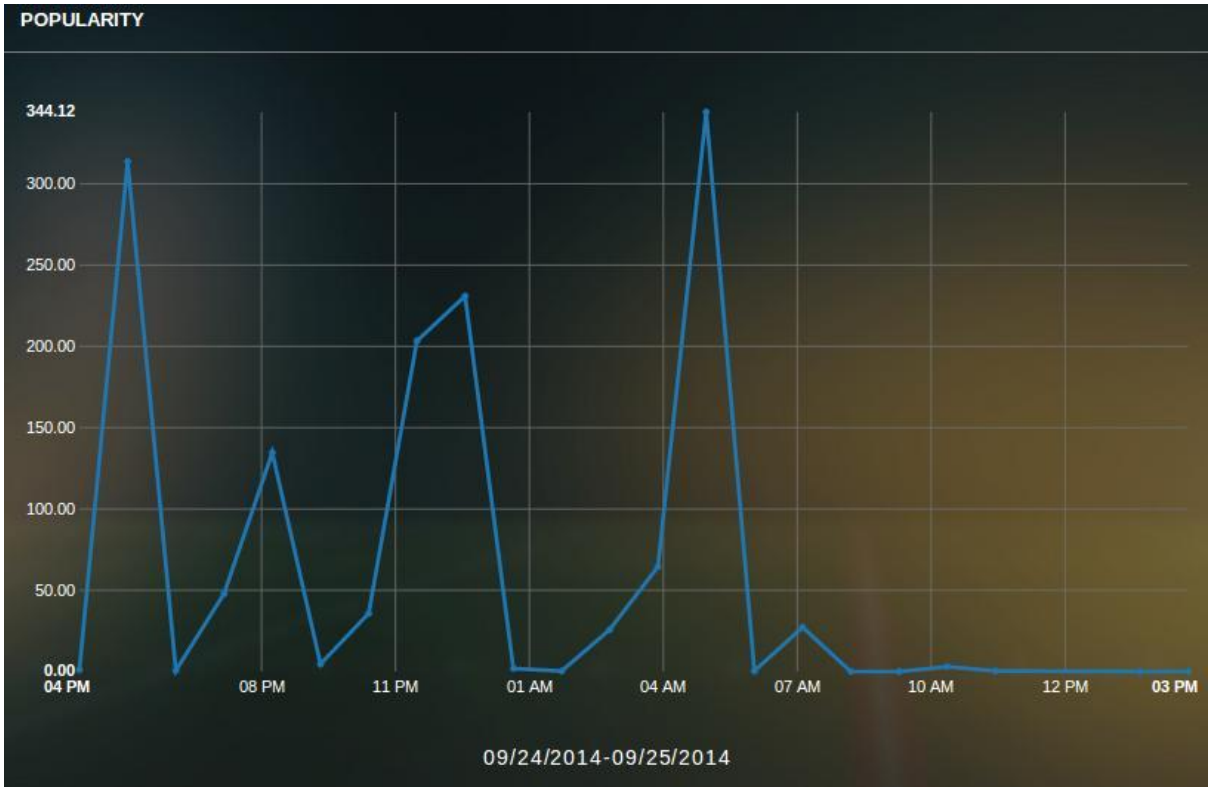
Serving Layer: MongoDB is a schema-less NoSQL database that acts as the serving layer in this application. It processes the partial aggregate of results from the batch layer (influencers, popularity, and tweet mentions) into hourly, weekly, and monthly time buckets. In the serving layer, results from the batch layer and real-time output from the speed layer are merged to display an indexed compute view.

Speed Layer: Fast retrieval of the latest tweets and their geographical location is performed in the speed layer. This layer performs stream processing on data drawn from Twitter API and Twitter stream within the limits of Firehose. Redis is the in-memory database used to store new data that streams in the interval occurring due to high latency batch processing. The processes in the speed layer are iterative. The data stored in Redis is cleared once updated in the batch layer, to take up the latest data streamed from Twitter.

Users can get a historical view that displays a change in values over a period of 90 days, which includes the latest tweets from a celebrity, and the sentiment calculated for each sportsperson. The geographic location of the latest tweets are displayed in map view.

Using the web application, the end user can see the trends in celebrity popularity, sentiments, and mentions. Here are a few graphs that show the change in these values over a 24-hour period.

Number of mentions charted against timestamp



Popularity chart of a day

Sentiment trend with an interval of 3

Conclusion

Lambda Architecture handles ETL operations by archiving unstructured data and performing data mining on enormous volumes of immutable datasets through batch processing. It also performs low latency stream processing. If a part of your application relies on fast delivery of results to the tune of milliseconds, while another part caters to long-term operations such as predictive analytics, then Lambda Architecture would be ideal.

The hybrid architecture also maintains an immutable master dataset. Despite a well-conceived data architecture design, Big Data processing systems need to be equipped for 'human fault tolerance.' Human mistakes such as introducing bugs in codes accidentally could create errors in aggregate writes. An immutable master dataset makes recovery from such human mistakes much easier.

One trade-off is that batch processing and stream processing require you to maintain and debug two different sets of codes. This trade-off is also a double-edged sword, while the above argument holds good in terms of codes, the system also allows for running two different logics in stream and batch processing layers. For instance, the logic in stream processing can track near real-time fraud detection while the latter can run a code to put in place a more complex fraud detection system.

Ultimately, the question whether advantages or trade-offs outweigh the other is best answered by examining your specific business use case with experienced data architects.

Email: bdg@qburst.com | Website: www.qburst.com