RETINAA: A Real Time App Analytics Framework

[Demo @ https://www.youtube.com/watch?v=f__ihYnPPfk&hd=1]

RETINAA: A Real Time App Analytics Framework

[Demo @ https://www.youtube.com/watch?v=f_ihYnPPfk&hd=1]

Main hypothesis in the project

Our project **RETINAA** aims to help mobile application developers understand their app statistics in **real time** across millions of users around the world and provide in depth detailed analysis of crash and usage statistics to improve the development process.

Unique Selling Proposition (USP)

The mobile application market is booming at a rapid rate. Application development has become pervasive. Problem arises when these applications, without proper testing, bring down the platform stability and overall reputation (as in case with Android OS). The application developers would greatly benefit from a real time system, which can gather the crash reports and other application statistics such as user base demography, click streams in real time and help them fix bugs and launch features proactively. This can be used for monetization and tweaks in the UI depending on the user's interaction stats.

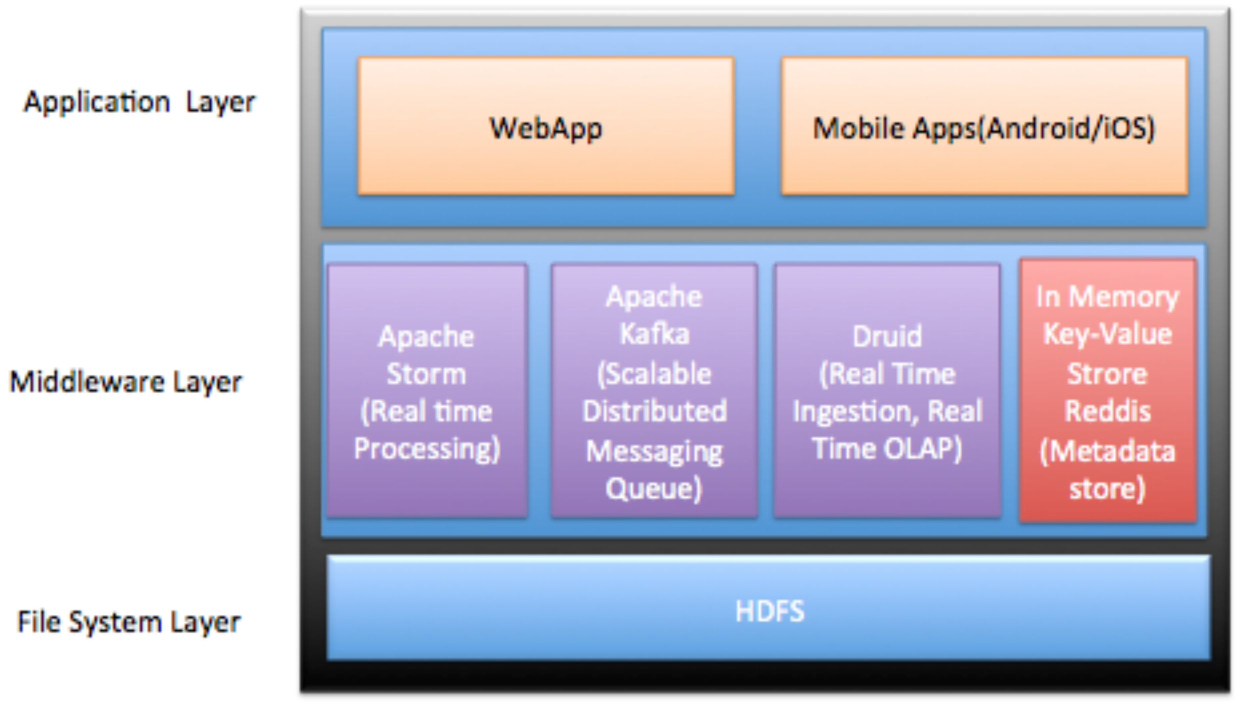
Goals

Our goal is to build a generic real time events analytics platform. In this project we are using the platform as a real time app analytic dashboard to give insight into the application and user statistics with drill down.

Techniques?

We have our platform built over 3 main stages. In the first stage are our distributed log ingestors. These servers collect the event data pushed from the mobiles. The ingestor servers dump data into a time ordered Kafka queue. To process this streaming event messages, we have built a Storm cluster. The cluster feeds data off a Kafka spout into a set of bolts dedicated to parsing the different types of event messages. The processed event stream is pushed onto the

RETINAA Full Stack


Application Layer
Middleware Layer
File System Layer
WebApp
Mobile Apps(Android/iOS)
Apache Storm (Real time Processing)
Apache Kafka (Scalable Distributed Messaging Queue)
Druid (Real Time Ingestion, Real Time OLAP)
In Memory Key-Value Strore Reddis (Metadata store)
HDFS

RETINAA STACK

RETINAA stack is divided into three primary layers based on the core functionality of the layer.

Application layer:

This layer directly interacts with the users. The interaction is primarily at two places. Mobile App side, where retina service runs in the background and collects log events from the developer apps. On the webapp side, the developer is presented with a dashboard view of the real time trends and changes in the various app analytics metric. We will also integrate **cross-region load balancing** for our client facing web application.

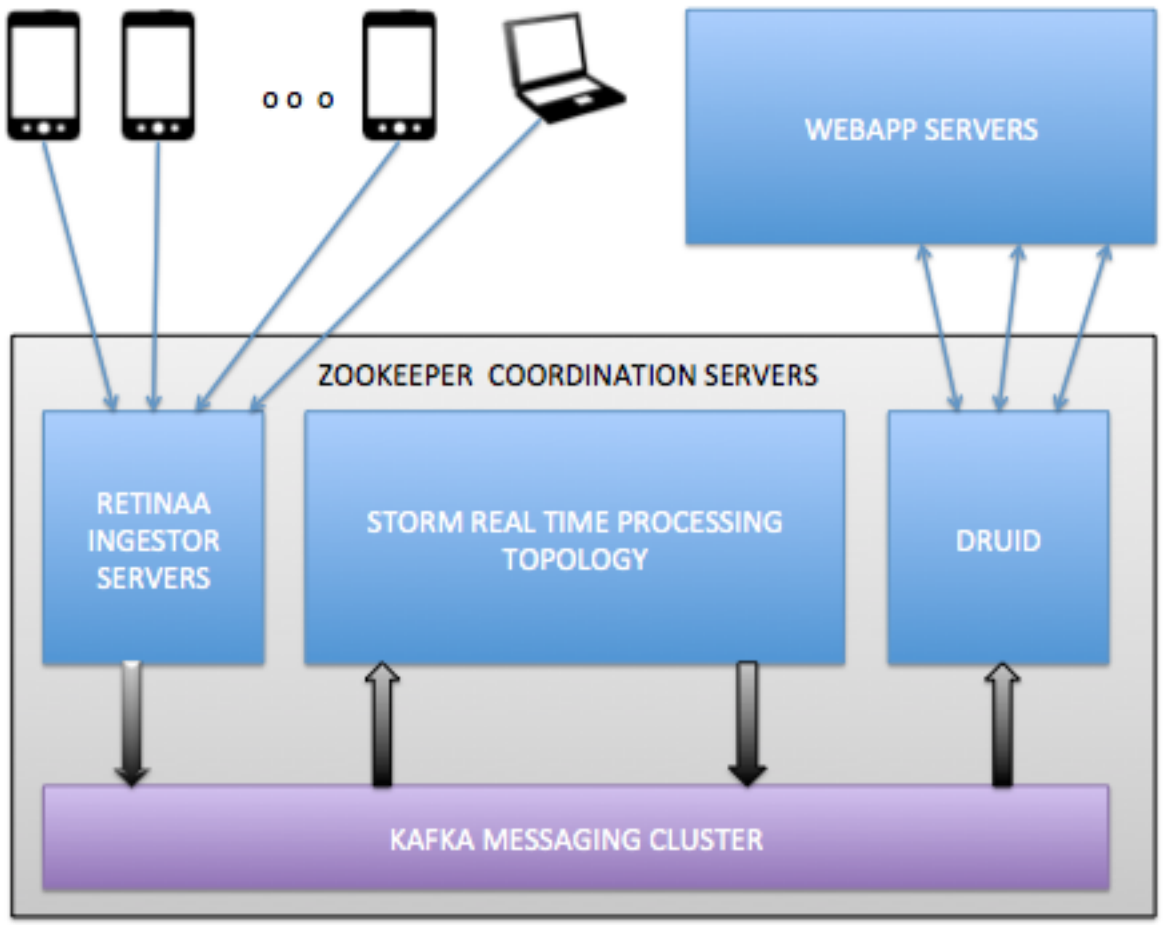
Middleware Layer:

This is the core pipeline of RETINAA where the processing of app event streams is done. The primary components of this layer are Apache Storm, Apache Kafka, Druid and Redis. Kafka is the distributed messaging queue over which Storm and Druid communicate. The workflow of the three components is explained in the next section.

FileSystem Layer:

This is the storage layer of RETINAA which is HDFS file system, and is utilized by Druid to store the event logs.

Work Flow of RETINAA:



A simple call flow of RETINAA pipeline works as follows:

1. RETINAA event collection service, which runs on mobile devices/desktops/alptops, sends the logs periodically over the one of the servers of the RETINAA Ingestor Cluster.
2. The Ingestor server dumps the raw events to the kafka queue under a topic name say ‘A’.
3. A Storm spout consumer group listens for messages listed under topic ‘A’ in the KAFKA queue.
4. Once a message is retrieved by the spout, it is passed on to one of the worker bolts for further processing.

5. The processed event is the dumped to the KAFKA queue again under a different topic say ‘B’.
6. Druid listens for messages posted under topic ‘B’ and ingests that into one of its real-time nodes and then subsequently passes it to the historical nodes, when the time segment(as per configuration) expires, which then stores the event logs in the HDFS.
7. When a developer logs in to RETINAA webapp, aggregations, filter queries are fired to the DRUID broker node which then retrieves the data. This data is presented as an intuitive real time graphical display on the webapp dashboard for the developer to view.
8. All the components Storm, Apache and Druid use its own instance of zookeeper cluster. We avoid a single cluster for all the three components as this will overburden the zookeeper cluster.

RETINAA Components:

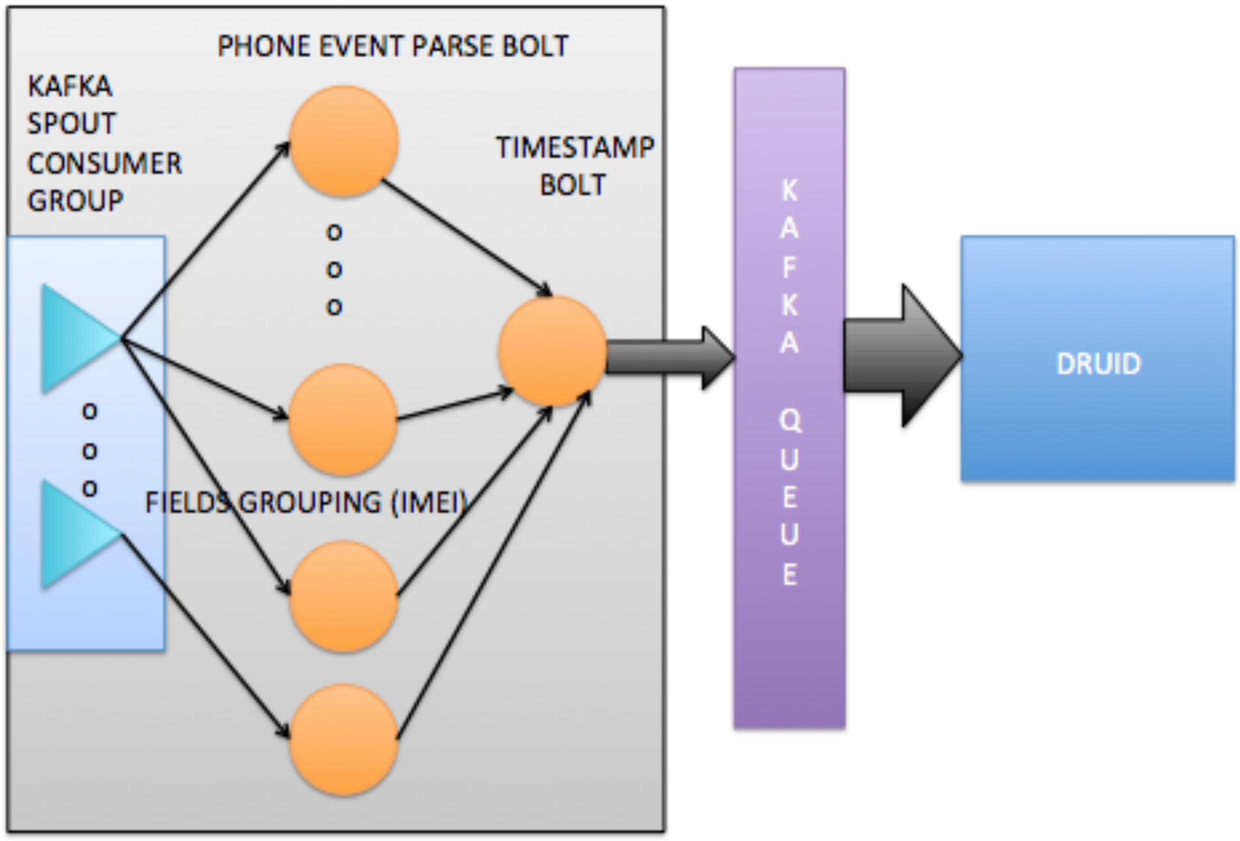
Ingestor - Google’s cross-region load balancer:

Main entry point of logs from phone is handled by **Google’s cross-region load balancer** which evenly distributes the phone log packets to ingestors that finally dumps the logs in Kafka Queue in json formatted string. Google cross region http load balancers ensures that requests are routed to closest region, and in case of failure of instances in a region reaching capacity, requests are sent to a healthy instance in the next closest region. Upon configuring google vm’s, load balancer services will have a single global ip address to which user’s phone will send different type of logs.

Scalability

Google’s load balancer gives a single global ip address to the service but internally multiple of servers are serving the request based on closest region, and next closest servers if server is unavailable or loaded. Their design scales to billion of users and is used directly by their own services also like GMail, Maps etc.

Storm Topology:



RETINAA STORM TOPOLOGY

There are multiple spouts(= number of partition of a kafka topic) running in parallel on different machines which reads data from Kafka Queue. Each spout is a part of consumer group to consume logs from Kafka topic. This is scalable as spouts are running on different machines handling different data, each from a topic's partition, so logs coming to spout are parallelized and getting data in totally ordered manner.(each storm spout reads from a single partition of a kafka topic, which is itself totally ordered)

Next, there are multiple bolts that are connected to spouts using Field's grouping base on phone's IMEI number. They will all process the incoming logs parallely and transform them to retina event format as expected by Druid . The parallel processing based on Hash Partitioning to multiple bolts on different machines is again scalable. These parsing bolts can easily scale out.

At the end, a single dump bolt receives formatted json in retina's format from using all grouping and dump it into kafka queue by assigning a timestamp to them so that they are totally ordered. We discuss of limitation of Dump bolt in Section number 5.

Scalability

Storm was benchmarked at processing **one million 100 byte messages per second per node** on hardware with the following specs

Apache Kafka:

Kafka is a distributed, partitioned, replicated commit log service. Kafka basically divides the incoming messages into different partitions for scalability. Each partition has different replicas for fault tolerance. Each partition has a leader and multiple followers, where reads and writes go to leader first and followers try to catch up leader. Producers are processes (can be on different machine or servers) that pushes the data in kafka cluster. Consumers can be in a consumer group and there can be multiple consumer groups that are fetching data from kafka parallely. Each consumer in a group maintains a offset in the queue and fetch the logs in linear fashion. Each consumer within in a group fetches log from a fixed partition so that it receives all the logs in totally ordered manner.

Scalabilility

Overall architecture of Kafka is highly scalable as it can hold huge amount of data in several partitions and performance is constant in amount of data since consumer maintains offset by itself[28].

Druid

Druid is an open-source analytics data store designed for business intelligence (OLAP) queries on time series immutable data. Druid provides low latency (real-time) data ingestion, flexible data exploration, and fast data aggregation. In Druid, data is divided into segments where each segment defines data collected over some defined interval. Druid stores data in form of segments. Druid also requires deep storage (HDFS) where segment resides, metadata storage (relational DBMS) where metadata of each segment is present and zookeeper which helps in coordination b/w different type of servers. Druid consists of 4 type of servers each with different purposes.

1. **Real Time Nodes** - These are connected to some data source which can be kafka, irc channel or some kind of firehose which is continuously pushing data to druid.
2. **Historic Nodes -** These nodes receives segment metadata information from zookeeper about which segments to load or drop from their local disk (cache).
3. **Coordinator Nodes -** They are primarily responsible for segment management and distribution..
4. **Broker Nodes -** They are primary contact point for client. Client contacts the broker node with some query over some interval

UseCase

Druid is built for exploratory analytics for OLAP workflows. It supports a variety of filters, aggregators and query types and provides a framework for plugging in new functionality.

Highly Available

Druid is used to back SaaS implementations that need to be up all the time. Your data is still available and queryable during system updates. Scale up or down without data loss.

Scalable

Druid's real-time nodes employ lock-free ingestion of append-heavy data sets to allow for simultaneous ingestion and querying of 10,000+ events per second. Existing Druid deployments handle billions of events and terabytes of data per day.

RETINAA MESSAGES:

In order to reduce the traffic from phone and keep the bandwidth usage to a minimum, we have three types of event messages pushed from mobile.

Metadata event:

This is the first registration message between the phone and our application. The message would be sent for each application that is subscribed to our app. This event would be triggered on the first use of the registered application or upon changes to the phone operating system (such as in case of OS upgrade, build change etc).

HeartBeat event message:

This is a period ping message sent from the application during the duration of its activity. This helps to understand the app usage pattern with respect to frequency and demography. This message has a very minimal bandwidth usage footprint as we utilize in-memory fast lookup for finding the app and phone details given the unique app identifier and the phone IMEI (International Mobile Station Equipment Identity).

Log Event message:

Error, Crash, Click event message statistics are aggregated over a set period on the phone and are transmitted to our application. The phone does not send if there are no aggregated growth in the stats.

Advantage of the architecture used

The implementation with Druid over Spark or any other analytics gives us the ability to scale to millions of clients with very strict query latency restrictions with TBs of event logs being ingested in real time having very minimal or no influence on the query latency. We have studied the use of Druid in the industry where they have been deployed to handle billions of clients. (This has been tested with a simple experiment comparing MongoDB implementation Vs Druid for aggregation detailed under Section 6).

Tradeoff of our architecture

The need to have a platform that scales to billions of users and promises very tight query response means the system is complex to build. The druid implementation currently poses a limitation that it ignores an event with lower timestamp after it processes an event with higher timestamp. This restricts us to use only a single dump bolt for now, so that it can totally order the events received from previous bolts and dump them in kafka queue. So, limitation posed by druid's implementation restricts the final bolt to be on single machine with single thread.

Experimental results

We injected 10000 records into the cluster, and performed aggregation query to calculate topN (N = 3) applications with respect to their error counts. The output took an average of **150ms** on MongoDB. The same query on Druid fetches results in under **1ms**.

Combined with the cacheable query feature of Druid, it can be easily seen that druid can be scaled well. Industry implementation of Druid prove this fact.

Some examples from the industry:

ebay and Metamarkets :

Druid is used for real-time user behavior analytics by ingesting up at a very high rate(over 100,000 events/second at ebay and over 30 billion events per day at metamarket)

Jolata

Jolata calculates a billion metrics every minute to visualize precise network metrics in real-time with drill down options.

These industry reports indicated us to strongly use Druid for our application over other database analytics platforms with the view of scalability.

Business Plan

1. Why will anybody like to buy our product or use our service?

Application crash logs are usually sent to the platform owners (such as Google or Apple) which the developers can analyse. With our system, we create a direct link between user base and the application developer. Our system interface will help developers to view the top open bugs/crash and have drill downs per demography and gain insights into the app usage and problem areas of the application.
Developers also will benefit from our aggregation and prioritization infrastructure to help reduce the development and debugging time. Working on highest priority bugs can help in stabilizing the app faster and enable faster TTM for new features.The system can also categorize reports on software version and other categories and provide tools for in depth analysis. This feature is not just restricted to crash analytics, developers can use our APIs to track event analytics also like click streams from the app.

2. What kind of customer base are we shooting for?

Our product will be available as a beta product for developers to use. Currently we will target the Android developers and Google play store. But in the future, our customer base can extend to anyone who requires to analyze streaming events and do analytics for gathering data insights.

3. What are our plans to monetize the product/service?

As the user-base grows, we plan to add more features and improve our codebase and make the features available as a paid service. Only registered developers who pay a one time small registration fee can use our product.

Progress Report

We have built the project prototype on the google cloud.

1. Multi Cluster configuration and setup of consisting of Apache Kafka, Apache Storm, Apache Zookeeper, Druid, HDFS, MongoDB(for comparative analysis)
2. Design of RETINA event schema which is DRUID compatible.
3. Design and implementation of event jsons which are bandwidth efficient.
4. Implementation of an App Event Simulator program to pump events in real time into Kafka queue.
5. Implementation of the Storm topology as described above.
6. Integration of Druid with Storm and Kafka.
7. Design of prototype client web application to view real time event analytics with custom event filter and aggregation functions.
8. User Demo video of Retinaa.

REFERENCES

1. **Application Crash Reports for Android** **https://github.com/ACRA/acra**

a. Last Accessed Date : Feb 10, 2015

b. Key Differences : This project saves the information on GoogleDoc form and leaves the analysis of statistics to the end user while our application provides real time filtering and application analytics.

2. **Singer from Pinterest**

Pinterest has a version of real time analytics where all event logs are collected and shipped to a centralized repository. The infrastructure has adaptive log processing intervals but the development has largely being proprietary in house development activity which cannot be used as a plug and play product in the industry. http://engineering.pinterest.com/post/111380432054/real-time-analytics-at-pinterest http://www.slideshare.net/DiscoverPinterest/singer-pinterests-logging-infrastructure

3. **Suro from Netflix**

Netflix has Suro, an infrastructure to collect and analyse real time events such as log messages, user activity records, system operational data, or any arbitrary data that their systems need to collect their business, product, and operational analysis with minimal latency.

They generate real-time trends are built with a pipeline of hadoop jobs and they also dispatch to designated Kafka cluster. They then make use of Druid cluster to make the indexes available immediately for querying.

http://techblog.netflix.com/2013/12/announcing-suro-backbone-of-netflixs.html

4. **VMWare's vRealize Log Insight:**

A real-time log management for VMware environments, with machine learning-based Intelligent Grouping, high performance search and better troubleshooting across physical, virtual, and cloud environments. - See more at: http://www.vmware.com/products/vrealize-log-insight

5. Making machine data accessible and usable : **http://www.splunk.com/**

6. http://www.splunk.com/en_us/products/splunk-mint/splunk-mint-express.html

7. Simplify Log Management Forever https://www.loggly.com

8. Apache Chukwa **:** Toolkit for displaying monitoring and analyzing results **http://wiki.apache.org/hadoop/Chukwa** **http://wiki.apache.org/hadoop/Chukwa?action=AttachFile&do=view&target=chukwa_presentation_cca08.pdf**

9. **Logscape** : http://www.logscape.com/product.html#infrastructure-monitoring

10. IBM challenges in big data log analysis : http://ibmdatamag.com/2012/05/why-log-analytics-is-a-great-and-awful-place-to-start-with-big-data/

11. http://www.adremsoft.com/netcrunch/monitoring/alerts-and-logs

12. **The Unified Logging Infrastructure for Data Analytics at Twitter**

13. http://vldb.org/pvldb/vol5/p1771_georgelee_vldb2012.pdf

14. **Apache Flume** http://flume.apache.org/FlumeDeveloperGuide.html

15. **DRUID : Real Time Analytical Data Store:** http://static.druid.io/docs/druid.pdf

16. **Predictive Analytics with aviation big data**
http://ieeexplore.ieee.org.proxy2.library.illinois.edu/stamp/stamp.jsp?tp=&arnumber=6548556&tag=1

18. Distributed Query visualization infracture : http://www.zoomdata.com/

19. **Kibana :** Real time summary and charting of streaming data
20. http://druid.io/blog/2014/03/17/benchmarking-druid.html
21. http://druid.io/druid-powered.html
22. http://druid.io/docs/latest/Historical-Config.html
23. http://druid.io/druid-powered.html
24. https://cloud.google.com/compute/docs/load-balancing/http/cross-region-example
25. http://kafka.apache.org/documentation.html
26. http://druid.io/docs/0.7.0/Design.html
27. https://storm.apache.org/about/scalable.html
28. http://kafka.apache.org/07/performance.html