Fall 2015

STUDY OF BIG DATA ARHITECTURE LAMBDA ARHITECTURE

Jaideep Katkar
San Jose State University

STUDY OF BIG DATA ARHITECTURE LAMBDA ARHITECTURE

Child Percentile Calculation

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Jaideep Katkar Dec 2015

The Designated Project Committee Approves the Project Titled

STUDY OF BIG DATA ARHITECTURE: LAMBDA ARHITECTURE

By

Jaideep Katkar

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Dec 2015

Dr. Thanh Tran	Department of Computer Science
Dr. Jon Pearce	Department of Computer Science
Chandni Patel	Senior software Engineer Pinger Inc.

ABSTRACT

The lambda architecture introduced by Marz is generic, scalable and fault-tolerant data processing architecture. It aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases. The architecture proposal decomposes the problem into three layers: a) the batch layer focuses on fault tolerance and optimizes for precise results b) the speed layer is optimized for short response-times and only takes into account the most recent data and c) the serving layer provides low latency views to the results of the batch layer.

The reason to divide the architecture into three layers is the flexibility it offers to the potential applications. The fast but possibly inaccurate results of the speed layer are eventually replaced by the precise results of the batch layer.
The evaluation of the designed architecture measured its capabilities based on the DEBS grand challenge 2014 and percentile calculation for milestones task. As part of the project we implement the lambda architecture in different ways (i.e. using different systems). We compare these different implementations and derive the strengths and weaknesses of each different system used in the lambda architecture.

ACKNOWLEDGMENTS

I am very thankful to my project advisor Dr. Thanh Tran for his continuous guidance and support throughout this project and for being a wonderful professor. I would like to thank Dr. Jon Pearce and Chandni Patel for monitoring the progress of the project and being the committee members.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

In the past few years there is an increase in data consumption and this has created new challenges to process high volumes of real-time data and provide better data analytics solutions. Nathan Marz[3], describes the solution for the problem, and explains lambda architecture, a new concept for processing real-time data but there is not much practical implementation about this architecture, which limits its examination.

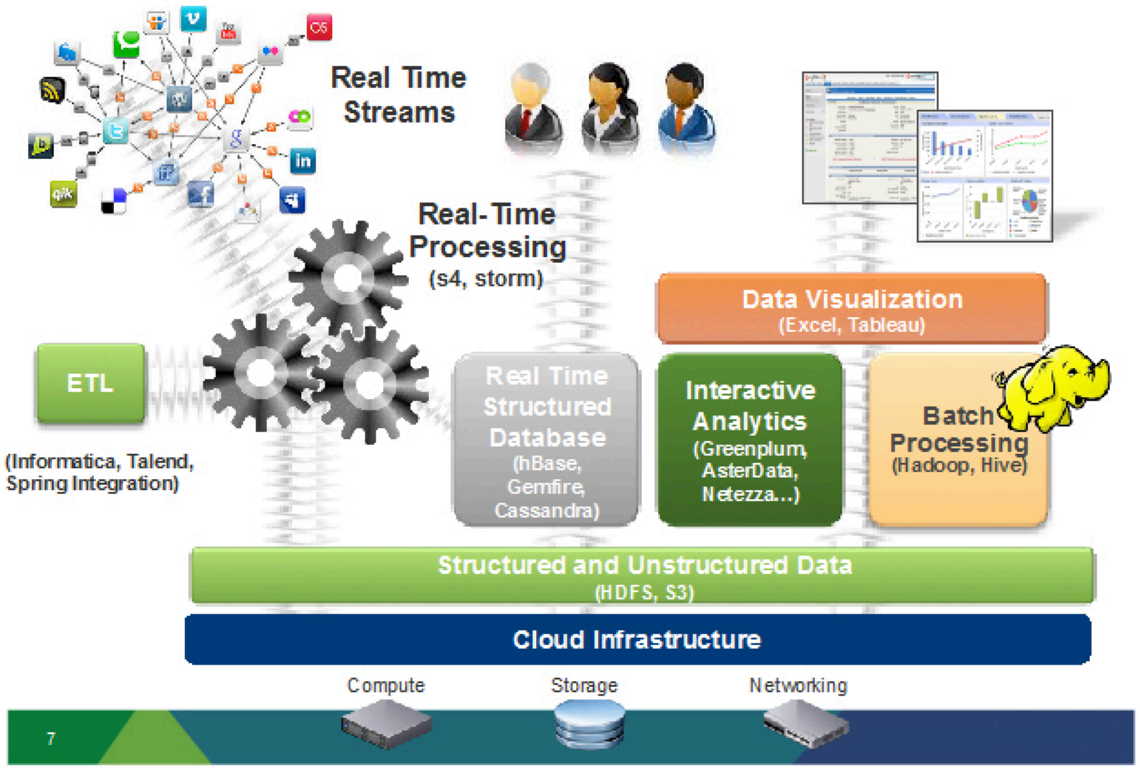
Big data[13] problems are often difficult to examine and resolve. The total volume, velocity, and variety of the data flow really make it a challenge to get information from the sources and give useful insight. Firstly, we need to analyze the big data problem with respect to the type of the information needed to be used, the type of analysis to be used as a solution, the pre-processing approach, and the information origin for the data that the solution framework is required to acquire, load, process, analyze and store.

We start by taking a gander at sorts of information portrayed by the expression "big data". To decode the complexity of big data types, we group big data as per different parameters and give a logical modeling to the layers and different state parts included in any big data solution. Let's examine the architecture for defining and characterize big data business issues nuclear and composite designs. These examples decide the proper solution to apply. We incorporate example business issues from different business ventures lastly, for each component and example; we show the solutions that offer the pertinent function. Big data architecture planning is commenced on an expertise set for developing reliable, versatile, totally computerized data pipelines. That aptitude set requires significant learning of each layer in the stack, starting with group plan and crossing everything from Hadoop tuning to setting up the top chain in charge of preparing data.

The principle detail here is that data pipelines take raw information and transform it into insight. Along the way, the big data architect needs to settle on choices about what happens to the data, how it is loaded in the cluster, how access is granted, what tools to use to handle the data, and in the long run the way of granting access to the outside world. The recent could be BI or other analytic tools; the previous (for the handling) are likely tools, for example, Impala or apache spark.

Most of the big data initiatives use varieties of different facts and reference architecture. Understanding the different state perspective of this reference architecture gives a decent foundation to seeing big data and how it supplements existing solutions, BI, databases and frameworks. Figure. 1 shows the big data architecture, which depicts the existing data processing framework, used and flow of data through the system. This design is not a settled, one-size-fits-all methodology. Every segment of the modeling has no less than a few choices with its own particular points of interest and detriments for a specific workload. Organizations regularly begin with a subset of the examples in this architecture, and as they understand the importance of picking up knowledge to key business results they grow the broadness of utilization.

Figure 1: Big Data Architecture


Real Time Streams
Real-Time Processing (s4, storm)
ETL
(Informatica, Talend, Spring Integration)
Data Visualization (Excel, Tableau)
Real Time Structured Database (hBase, Gemfire, Cassandra)
Interactive Analytics (Greenplum, AsterData, Netezza...)
Batch Processing (Hadoop, Hive)
Structured and Unstructured Data (HDFS, S3)
Cloud Infrastructure
Compute
Storage
Networking
7

We're shelled with data in online and print media about the blast of machine created data, the petabytes of information that organizations like Facebook and Twitter produce, and the billions of dollars of chances anticipating all industries through the utilization of the information. We also pay attention about what appears an alphabet soup of recent technique to handle and analyze big data, hadoop for distributed data processing, lucene for content indexing and search, Mahout for machine learning, R for data analytics.

In case you're a business client, you're suspecting that big data could give you an edge over your opposition. In case you're an engineer, you're amped up for the numerous new innovations you can find out about. In case you're a designer, you're attempting to make sense of how all these big data advances fit inside of your current and future framework. It would be a misrepresentation to say the quantity, range and speed of the information was predictable, however it was conceivable. The issue confronting business was the way to extend their information distribution centers to suit these new prerequisites and to give the required backing to business insight and, in a bigger sense, analytics.

In this project we implement the lambda architecture and derive the strengths and weaknesses of the lambda architecture.

BACKGROUND
CHAPTER 2

Designing a dependable, practical and robust big data tool that satisfies exclusive needs of end-user can be an incredibly tough and hard task. Designing a system, which can serve a wide style of use-cases and situations, which helps low-latency read, updates and support ad-hoc queries. It may be very tough enough to just keep up with the rapid velocity of era innovation going on around, let alone building solutions that solve the problem at hand.

The framework ought to be built in a way that it is easily scalable, and should scale out as opposed to up, i.e. adding more machines at the issue will solve the job. It should be composed in such a manner that elements can be included easily, and it should to be easy to troubleshoot and require less maintenance. There are some methodological solutions that can offer us to visualize how different sorts of uses fit into the big data system. While processing vast volume of semi-structured information, we can see that there is always a delay from the point when the data is collected and its availability. Regularly the delay results from the need to validate or distinguish coarse information. In some cases, however, being able to use the new information immediately is more useful rather than being hundred percent sure about the data's validity.

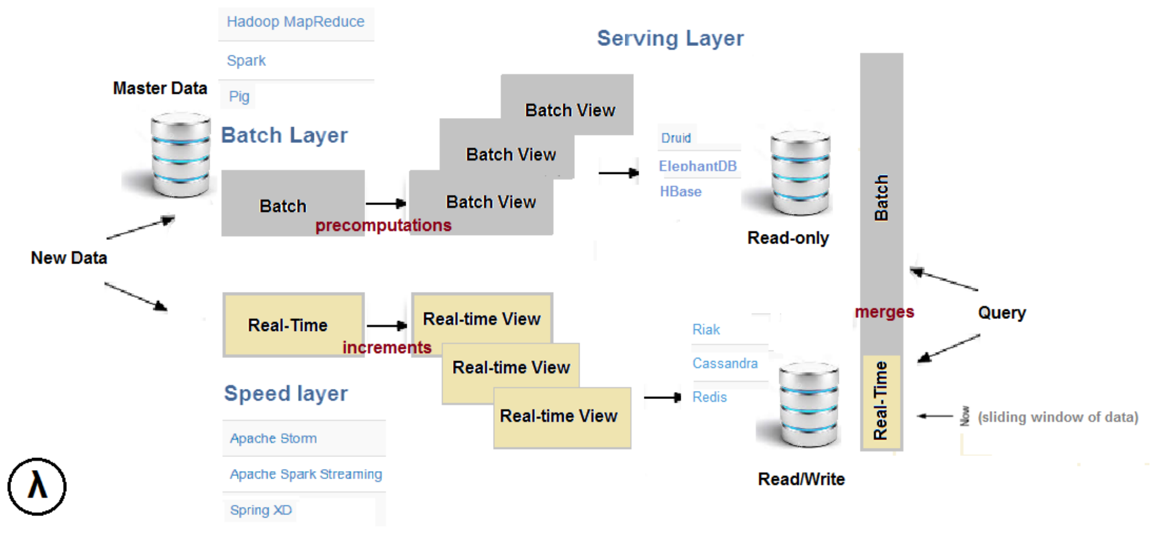
Lambda[2] Architecture proposes an easier paradigm that is designed to store and logically process large quantity of data. Nathan Marz initially proposed the lambda architecture, analytical ecosystem architecture is in early stages of development and is not at all like conventional DW system which is intended for organized, internal information, big data system work with raw unstructured and semi-organized information and in addition internal and external information sources. Moreover, business may require both real-time and batch processing capacities from big data frameworks.

Big data model commonly separated into two diametrical models, real-time processing and the more traditional batch processing. With hadoop with map-reduce representing the former and Storm representing the later. Whereas, the lambda architecture provides a hybrid solution, with the idea that these approaches have to avoid each other. The lambda architecture merges the real-time information and batch information for data processing to get best results.

2.1. LAMBDA ARHITECTURE

Lambda[2] Architecture framework is a really useful model to think about while designing big data applications. Nathan Marz created this architecture keeping in mind the issues faced while designing a system for big data, which is based on his personal experience working on distributed data processing systems.

Figure 2: Lambda Architecture General Diagram



Some of the key requirements of lambda architecture include:

- Fault-tolerant to internal failures and human errors.
- Should support different scenarios that include updates and low latency querying.
- Linear scale-out capacities, implying that adding new machines should solve the problem.
- Extensibility so that the framework is maintainable and can add newer features easily.

λ has three layers:

- Batch Layer manages the master data and pre-computes the batch views.
- Speed Layer serves recent data only and increments the real-time views.
- Serving Layer indexes the views and user can query the views to get the results.

In the figure 2 each of these layers can be seen using various big data tools. Let us consider, the batch layer, which can easily support the distributed file system, so map-reduce can be used to fed data to the serving layer by creating batch views. The serving layer can use nosql technologies such as hbase. Finally, the speed layer can be built with data streaming technologies such as spring-xd streaming or spark streaming.

The lambda model centrally receives records and does as low as possible processing before copying and splitting the records to the real-time and batch layer. The batch layer loads the data in its raw format in hdfs, hadoop jobs consistently write the processed records in the data store. Since this way it has completely batched the information, store can have some significant improvement. It have to support random reads, i.e. wishes a few form of index, however, it could dispose of random writing, locking, and consistency problems.

The issue with batch processing is that it takes time. For instance, the above procedure may take hours or days. Meanwhile information has been arriving and consequent procedures or administrations keep on working with hours or days old data. The real-time layer settles this by taking its copy of the information and handling it in seconds or minutes and stores it in a quick arbitrary read and write store. This store is more unpredictable since it must be continually updated.

Real–time layer complexity and store are easy to maintain since it just needs to store and serve a sliding window of information, which should be generally as long as the batch layer takes. Both layers' outcomes are consolidated and real time data is replaced for batch layer information. In many cases this enables for the real time to work better with approx. results since it updated the results with precise information into a short duration.

2.1.1. Batch layer

The batch layer is liable of two things. The main is to store the immutable, continually developing expert dataset (HDFS), and the second one is to compute arbitrary views from this dataset. Computing the perspectives is a consistent operation, so when new information arrives it will be aggregated into the views when they are recomputed in the next map-reduce cycle. The views need to be processed from the whole dataset and accordingly the batch layer is not anticipated that would update the views frequently. Each cycle could take hours based upon the size of the dataset.

2.1.2. Serving layer

The result from the batch layer is an arrangement of flat records containing the pre-computed views. The serving layer is in charge of indexing and uncovering the perspectives with the goal that they can be queried. As the batch perspectives are static, the serving layer just needs to give batch upgrades and random reads.

Hadoop and hbase are perfect techniques for the batch and serving layers. Hadoop can store and process petabytes of data, and hbase can query this data rapidly and intelligently. Although, the batch and serving layers no longer fulfill any realtime use because of excessive latency in map-reduce and it is able to take some hours for brand new data to be represented in the views and propagated to the serving layer, this is the reason why we need a speed layer.

2.1.3. Speed layer

In essence the speed layer is which is same to the batch layer in a way as it computes views from the information it gets. The speed layer is expected to makeup for the high latency of the batch layer and achieves the task by calculating realtime views. The realtime views comprise most of the delta outputs to supplement the batch views.

The batch layer is required to re-process the batch views from scratch repeatedly, the speed layer uses the incremental framework whereas the realtime views are incremented as and when new records is received. What is smart regarding the speed layer is the realtime views are intended to be temporary and as quickly as the data propagates via the batch and serving layers the corresponding effects inside the realtime views can be discarded. This is called as “complexity isolation” which meaning that the maximum complex part of the architecture is driven into the layer whose outputs are only transient.

2.1.4. What’s good about this architecture?

The lambda architecture mainly focuses on holding the incoming data unchanged. The actual control associated with modeling data change as a few materialized cycles by a classic input offers many advantage. This is something that makes large map-reduce work processes tractable, as it empowers you to investigate every stage autonomously. I think this lesson makes an interpretation of well to the stream-process area.

I additionally like that this architecture highlights the issue of reprocessing information. Reprocessing is one of the key difficulties of stream processing however is all the time disregarded. By "reprocessing," I mean handling info information over again to re-determine results. This is a totally clear however regularly disregarded prerequisite. Code will dependably change. Along these lines, on the off chance that you have code that gets output information from a data stream, the point where the code transforms, you should re-calculate your output to see the effect of the change.

The code changes on the grounds that your application change and you need to figure new way to handle that you didn't already needed. On the other hand it may change in light of the fact that you discovered a bug and need to correct it. Irrespective of, when it does, you have to recalculate your results. There can be many solutions who attempt to build real-time data processing frameworks but didn't paid much thought into this issue and wind up with a framework that essentially can't update rapidly on the grounds that it has no new approach to handle reprocessing. The lambda architecture merits a considerable measure of credit for highlighting this issue.

There are various different inspirations proposed for the lambda architecture. One is that real-time processing is inherently estimated, less effective, and lossy as compared to batch processing. But this is not valid. It is true that beyond any doubt the current arrangement of stream processing design are less develop than map-reduce, however there is no reason that a stream handling framework can't give as solid a semantic surety as a batch framework.

Another clarification is that the lambda architecture some way or another "better than CAP " by permitting a mixture of diverse data frameworks with distinctive trade offs. Despite the fact that there are certainly latency accessibility exchange offs in stream handling, this is a model for asynchronous processing, so the outputs being processed are not kept instantly predictable with the incoming stream of info. The CAP theorem, tragically, remains in place.

2.1.5. What's Bad about this architecture?

The challenge while using lambda structure will be maintaining code that needs to generate the same outputs about a couple difficult distributed models is exactly while painful while it appears as if it will be and this challenge does not seems to be fixable.

Coding in distributed frameworks such as storm and also hadoop will be difficult. Undoubtedly, code eventually ends up currently being exclusively designed towards the actual system it runs on. The resulting operational complexity of framework imposing the lambda model is the one thing that appears to be universally agreed everywhere. One proposed way to deal with this is to have a program or system that encapsulates both the speed and batch model. We design the program utilizing this more high level system and afterward it complies down to stream process or map-reduce under the covers.

Finally, even we stay away from writing the application two times, the operational problem would be of running and debugging of programs will probably be very high. In addition to any kind of fresh abstraction could simply provide features reinforced by the intersection of two systems. More painful, investing in this kind of new framework space from the loaded environment of programs and tools that makes hadoop so powerful (Hive, Pig, Crunch etc).

By the way of analogy, take the difficulties in making cross-database ORM truly transparent and take into consideration that it is an issue of abstracting over identical framework imparting definitely identical competencies with a standardized interface language. The question of abstracting over completely divergent programming paradigms built on top of distributed systems is way more difficult.

Chapter 3
Previous Work

In this chapter, we will see two different examples of lambda architecture. We will go through the problem statements and solutions discussed by these problem statements.

3.1. Real-time analysis of hashtags using Trident, Hadoop and Splout SQL

The solution[9] shows how to use trident, hadoop and splout sql together to build a “lambda architecture“ to simulate counting the number of appearances of hashtags in tweets, by date. The solution solves this simple problem in a fully scalable way and provides a remote low-latency service for querying the evolution of the counts of a hashtag, which consist of batch as well as real-time analysis of the data, so for any hashtag we want to be able to query a remote service to obtain per-date counts in a data structure like this:

```
{
 "20091022":115,
 "20091023":115,
 "20091024":158,
 "20091025":19
}
```

3.1.1. The problem

The problem[9] is to implement the counting of appearances of hash-tags in tweets, which will be shown by date, and output the data as a remote service, for example to be able to populate the values in a website e.g. give me the evolution of mentions for hashtag "Atlanta".

The requirements for the solution are:

- It must scale since they want to process many tweets
- It must be able to serve concurrent users with low latency queries and updates on the datastore.

Using hadoop to store the tweets and writing the hive query for grouping by hashtag and date seems good enough for calculating the counts. However, we also want to add real-time to the system: we want to have the actual number of appearances for hashtags updated for today in second’s time. And we need to put the hadoop counts in some really fast data-store for being able to query them.

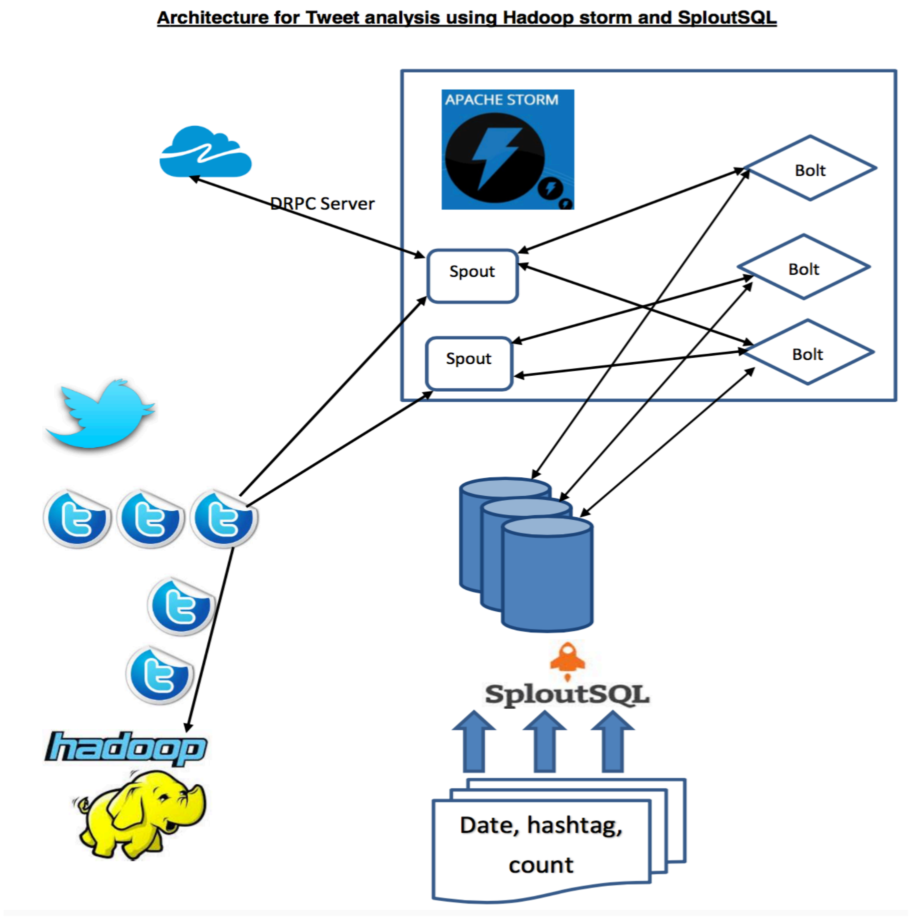
3.1.2. The solution

The solution[9] is designed so that it can use “Lambda Architecture” and implement a real-time layer-using trident, which is an API which is used by storm that eases building real-time views and saving persistent state derived from them.

For serving the batch layer the architecture make use of splout sql, a known high-performance sql read-only data store that can pull and serve information in and out of hadoop very efficiently. Splout is fast like elephant-DB but it also allows us to execute sql queries. The solution uses sql for serving the batch layer, which is convenient to use, as we may want to breakdown the counts by hour, day, week, or any arbitrary date. The solution, also make use of trident to design the remote service, which easily uses its DRPC capabilities. Trident works in a way that it will query both the batch the real-time layer and merge the outputs of both into serving layer.

This is how, conceptually, the overall architecture looks like

Figure 3: Lambda Architecture For Tweet Analysis



Spouts

A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology (e.g. a Kestrel queue or the Twitter API). Spouts can either be reliable or unreliable. A reliable spout is capable of replaying a tuple if it failed to be processed by Storm, whereas an unreliable spout forgets about the tuple as soon as it is emitted.

Spouts can emit more than one stream. To do so, declare multiple streams using the declareStream method of OutputFieldsDeclarer and specify the stream to emit to when using the emit method on SpoutOutputCollector.

Bolts

All processing in topologies is done in bolts. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more.

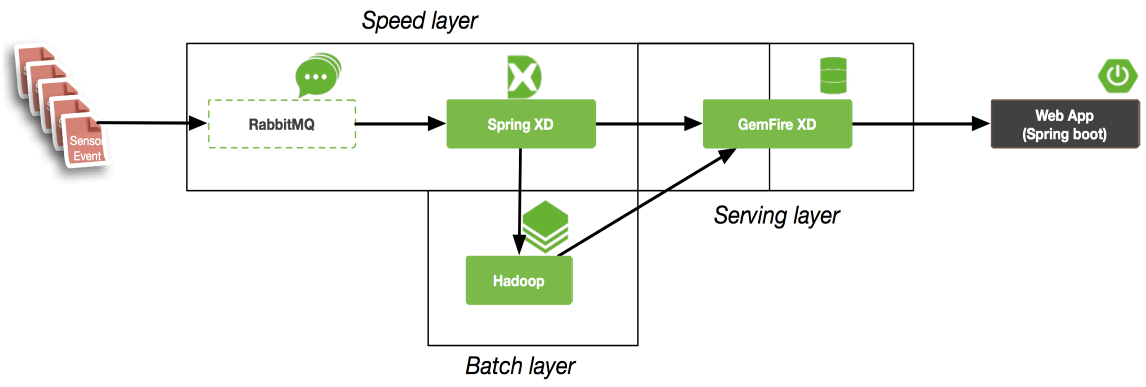
Bolts can do simple stream transformations. Doing complex stream transformations often requires multiple steps and thus multiple bolts. For example, transforming a stream of tweets into a stream of trending images requires at least two steps: a bolt to do a rolling count of retweets for each image, and one or more bolts to stream out the top X images.

Tweets are fed into the system, for example through a queue from where we can pull them. A trident stream saves them into hadoop (HDFS) very efficiently and after storing is operate on them in real-time so that it can easily create an in-memory state with the counts by date. In hadoop, i.e. the batch layer where we store all the historical data in hive, we can execute a batch job, which aggregates the tweets by hashtag and date and generates a large output file from it. Now we have run the batch job, we can make use of splout sql CLI or API tools for indexing the file and deploying it to a splout sql cluster, which will be able to serve all the statistics pretty fast. Then, a second trident stream can be used to serve timeline queries, and this stream will query both the batch layer, which makes use of splout sql, and the real-time layer and mix the results into a single timeline response. We will see each of these things in more detail.

3.2. ACM DEBS Challenge 2014

The final challenge of the "ACM DEBS Grand Challenge [4]" is to do a comparative evaluation of event-based systems by analyzing realtime event records and event queries requirements for the system. The 2014 edition of the challenge focuses on energy grid processing and is dependent on measuring of the energy intake at the level of individual electricity plugs in smart home installations. The event queries mainly deals with two types of analysis: (a) short-term load forecasting and (b) load statistics for real-time demand management.

Figure 4: Lambda Architecture For DEBS Challenge



The problem[1] needed to solve two queries: Predicting the load and finding the outlier sensors. Since this is an event-processing challenge, not a machine learning challenge the predicting algorithm is provided. So as a part of solution, a single node solution as well as a distributed solution was needed for both queries.

In this architecture they use rabbitmq and spring-xd at speed layer to capture the incoming event data. Sensor data is ingested in rabbitmq using http, which is then streamed into spring-xd. It sends all data to a master queue for processing. Sensor event enricher consumes the data from the queue, filter and transforms the data before storing it in HDFS. The findoutliers component taps from master stream to compute the outlier model it taps from main stream to compute load values for house and plug. They are using gemfire-xd since it is a distributed database; tightly integrated with pivotal hadoop with SQL support and it is fault tolerant and faster in memory access.

The task to parallelize the first query was easy, to increase the throughputs of few to up to 350k and 450k on a single node. The task of scaling the first query was way much trickier, and solutions had posted throughput of 0.75M with 3 nodes. To solve the second query they calculated a median over 24 hour sliding window, with about thousand events emitted per second. So this meant about sixty four million events were calculated over sliding window of one second. This was relatively tough so single node solutions gave throughputs ranging from few thousand to few millions.

The batch layer starts the batch job from spring-xd using a cron entry to run every X hour /min/day. The hadoop based system store an immutable and constantly growing master dataset. They compute arbitrary functions (models) on the existing datasets, which essentially, runs the map-reduce models. Serving layer stores the timestamp of starting time prediction.

Chapter 4
Problem Statement

Given a milestone completed by a child in a category like walking, writing, etc. We can calculate the percentile of the child i.e. given a particular child we can calculate their percentile in terms of how he/she is ahead of other children in the given age group.

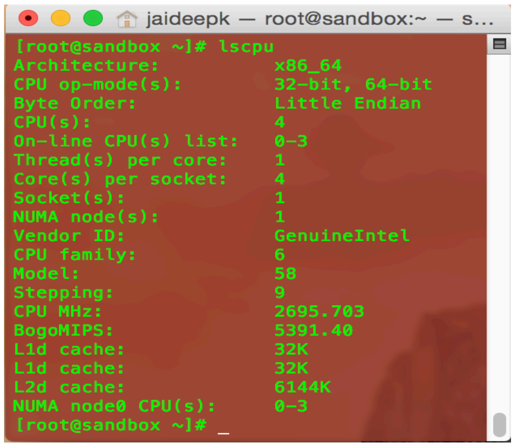
If a child is said to be let's say in the 85 percentile of, lets say language development (there are different categories including language, social, gross motor etc.), it means he/she is ahead of 85% of the kids in his age group and is behind 15% of the kids in his/her age group. It is important to compare and compute percentile only for kids in the same age group i.e., compare kids that are same age (in terms of month) when you compare a 15m child with a 0m child of course it is expected than 15m will be ahead of 0m child in terms of motor development, or weight hence percentile makes sense only when you compare the 15m child with other kids that are also 15m old in particular when you compare kids lets say, in terms of fine motor development, you compare them in terms of the fine motor milestones they have completed milestones in fine motors are for instance, being able to "grasp with two fingers", "hold a spoon" etc. if child A completed more milestones than child B, then child A is ahead of child B.

4.1. Approach to Solve the Problem

We implement lambda architecture to solve the problem with speed Layer, batch Layer and view Layer

Speed Layer Component: Spring-XD
Batch Layer: HDFS (Hive)
Serving Layer: HBASE (With Apache Phoenix on top of it)

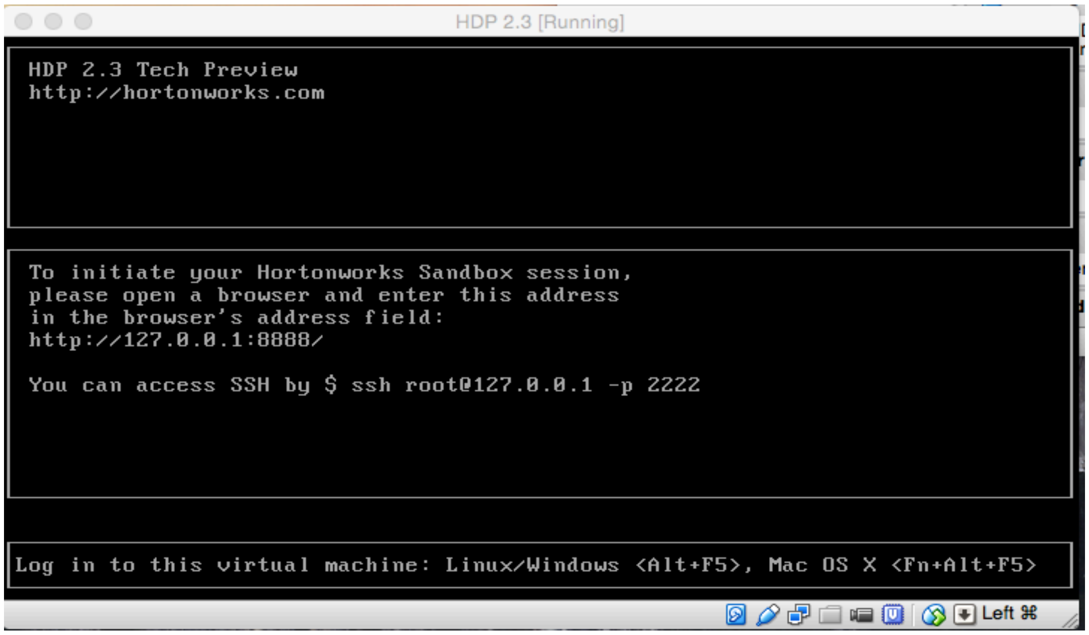
Figure 5: Machine configuration for the implementing the above system



```
[root@sandbox ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 58
Stepping:              9
CPU MHz:               2695.703
BogoMIPS:              5391.40
L1d cache:             32K
L1d cache:             32K
L2d cache:             6144K
NUMA node0 CPU(s):     0-3
[root@sandbox ~]# _
```

As we can see we are using 4 CPU, 4 Core, 64 bit machine. We are using hortonworks sandbox HDP version 2.3 which can be easily used to solve our problem.

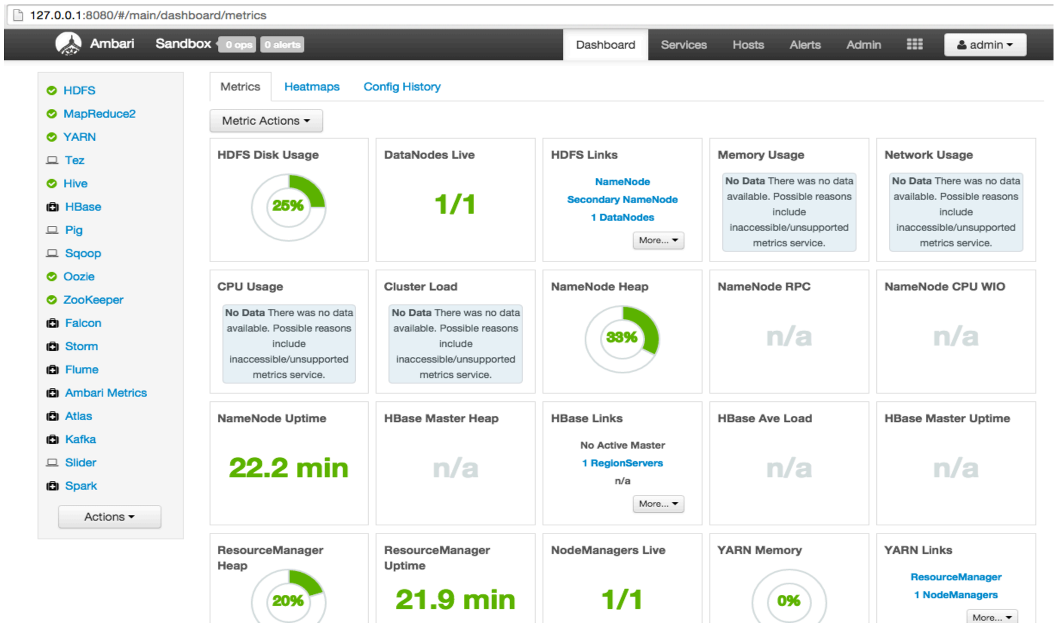
Figure 6: Virtual Machine



```
HDP 2.3 Tech Preview
http://hortonworks.com

To initiate your Hortonworks Sandbox session,
please open a browser and enter this address
in the browser's address field:
http://127.0.0.1:8888/

You can access SSH by $ ssh root@127.0.0.1 -p 2222

Log in to this virtual machine: Linux/Windows <Alt+F5>, Mac OS X <Fn+Alt+F5>
```

The machine has hadoop, hive, mapreduce, sqoop etc., which we will be using in building our Lambda architecture.

Figure 7: Hortonworks HDP framework

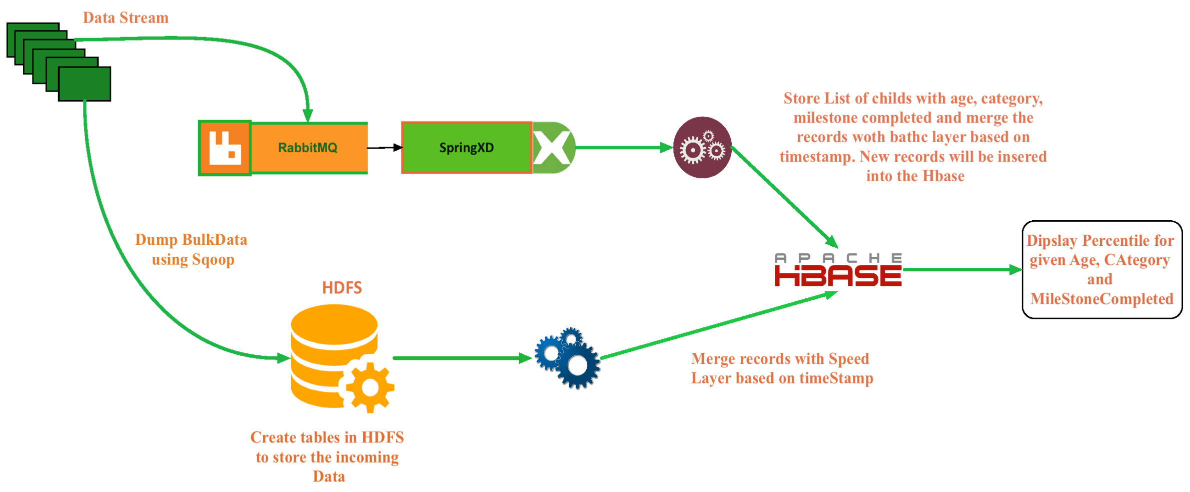


No Data There was no data available. Possible reasons include inaccessible/unsupported metrics service.

4.2. Lambda Architecture Approach

Figure. 8 show the lambda architecture for solving the percentile problem.

Figure 8: Lambda Architecture for Percentile Problem

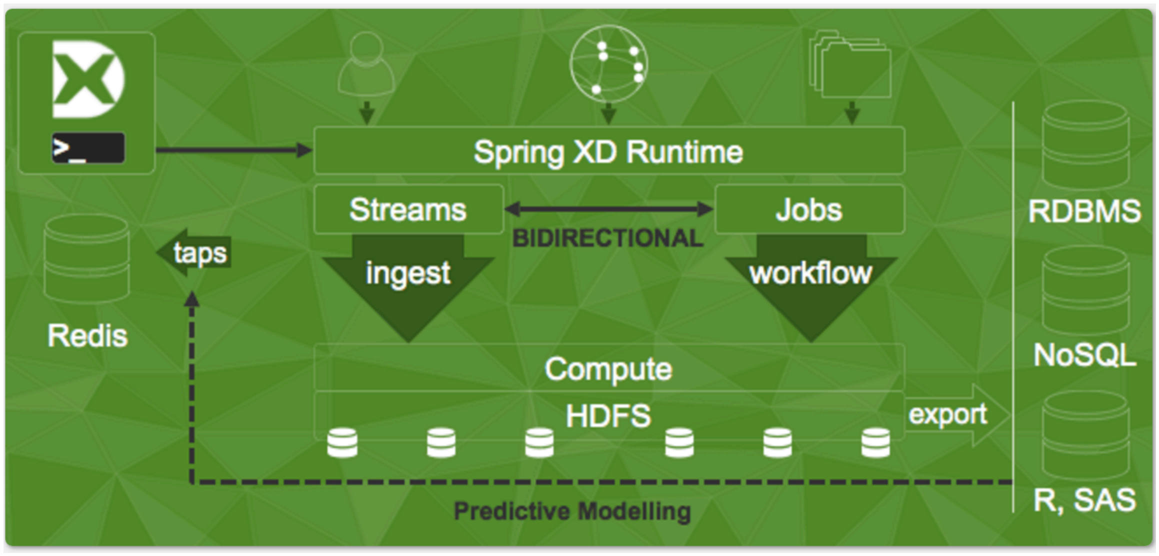

Data Stream
RabbitMQ
SpringXD
Store List of childs with age, category, milestone completed and merge the records woth bathc layer based on timestamp. New records will be insered into the Hbase
Dump BulkData using Sqoop
HDFS
APACHE HBASE
Dipslay Percentile for given Age, CAtegory and MileStoneCompleted
Merge records with Speed Layer based on timeStamp
Create tables in HDFS to store the incoming Data

As shown in the architecture in figure we are using the spring-xd and rabbit-mq to generate data streams. So any records updated and inserted at real time will be coming through this layer. At batch layer we are using hdfs (hive) to load data in bulk.

As the streams are ingested by spring-xd at speed layer we use different spring-xd operators like “Transform”, “json to tupple”, “Aggregator”, “Filter” etc. so as to process the incoming streams. We use these operators to change the incoming streams as per our requirement; these operators will be explained in detail later.

After the streams processed by the operator we use the sinks to store it into an external data store. We can then read data from these stores to process it further; we combine these streams with batch layer data to calculate the correct percentile for an incoming record and then store it into a view layer, which is hbase in our case. A separate job will load the data from batch layer to view layer, which is hbase. We can then query the view layer to get the percentile for a given child.

Figure 9: Spring XD Architecture



Spring XD[10] is a unified and extensible administration for information ingestion, batch processing, real-time analysis and information export. The Spring-xd is an open source platform whose objective is to handle big data intricacy. A significant part of the multifaceted nature in building genuine real-world applications is identified with coordinating numerous different frameworks into one solution over a range of scenarios. Normal use-cases experienced are

- Ingesting data from different sources into various data store like hdfs or mongodb or HIVE

- Doing real-time analysis of data during data ingestion e.g. filtering out unwanted values, counting values etc.

- Managing the workflow using batch jobs. The jobs interact, with the various frameworks like hadoop operations, hdfs, mongodb, Hive etc.

- Data export with high throughput e.g. from a RDMS to HDFS or No-SQL database.

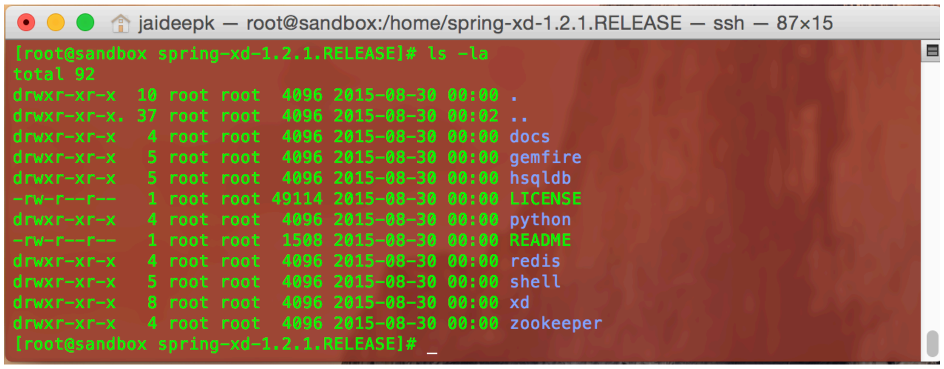
The Spring-xd aims to provide a one-stop solution for all these different use-cases.

4.2.1. Setting Up the Architecture

Setting up Speed Layer

To setup spring xd and rabbitmq at speed layer we are using spring xd version 1.2.1 and rabbitmq 3.5.4
Download spring xd using below command

```
# sudo wget https://repo.spring.io/libs-release-local/org/springframework/xd/spring-xd/1.2.*.RELEASE/spring-xd-1.2.*.RELEASE-dist.zip
```



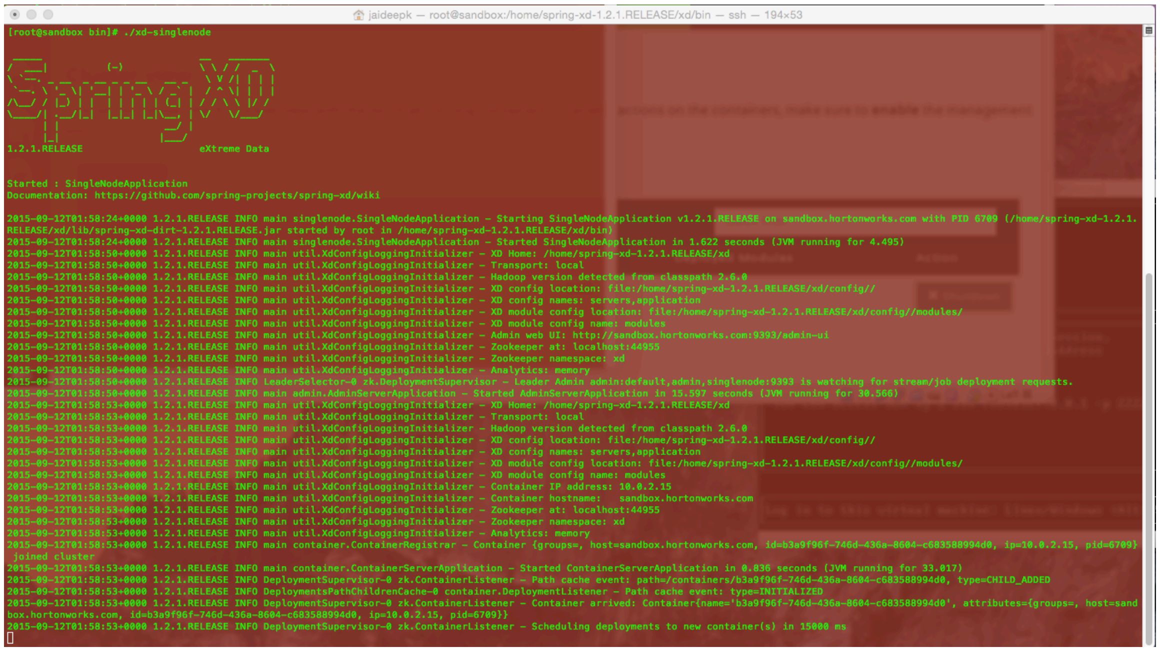
jaideepk — root@sandbox:/home/spring-xd-1.2.1.RELEASE — ssh — 87×15

```
[root@sandbox spring-xd-1.2.1.RELEASE]# ls -la
total 92
drwxr-xr-x  10 root root  4096 2015-08-30 00:00 .
drwxr-xr-x. 37 root root  4096 2015-08-30 00:02 ..
drwxr-xr-x   4 root root  4096 2015-08-30 00:00 docs
drwxr-xr-x   5 root root  4096 2015-08-30 00:00 gemfire
drwxr-xr-x   5 root root  4096 2015-08-30 00:00 hsqldb
-rw-r--r--   1 root root 49114 2015-08-30 00:00 LICENSE
drwxr-xr-x   4 root root  4096 2015-08-30 00:00 python
-rw-r--r--   1 root root  1508 2015-08-30 00:00 README
drwxr-xr-x   4 root root  4096 2015-08-30 00:00 redis
drwxr-xr-x   5 root root  4096 2015-08-30 00:00 shell
drwxr-xr-x   8 root root  4096 2015-08-30 00:00 xd
drwxr-xr-x   4 root root  4096 2015-08-30 00:00 zookeeper
[root@sandbox spring-xd-1.2.1.RELEASE]# _
```

Starting the spring-xd singlenode server
We can start the singlenode server by using the below command. Log inside the xd/bin folder in spring xd, and then run the below command to start the server.

```
# ./xd-singlenode
```

Figure 10: Spring XD Singlenode Framework



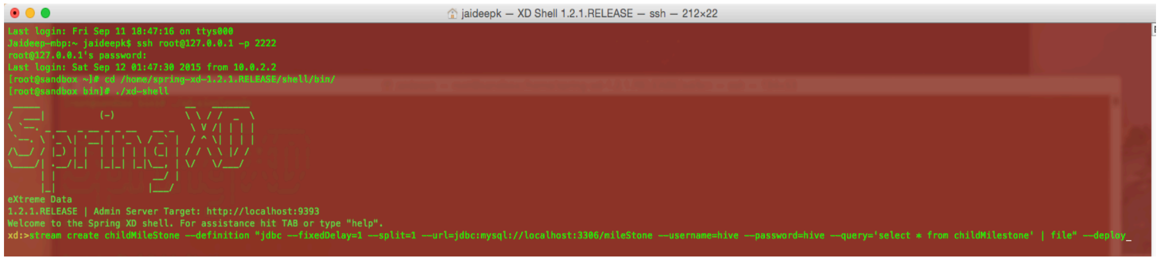
```
[root@sandbox bin]# ./xd-singlenode

 _____                           __   _______
/  ___|          (-)             \ \ / /  _  \
\ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
 `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
/\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
\____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
      | |                  __/ |
      |_|                 |___/
1.2.1.RELEASE                    eXtreme Data


Started : SingleNodeApplication
Documentation: https://github.com/spring-projects/spring-xd/wiki

2015-09-12T01:58:24+0000 1.2.1.RELEASE INFO main singlenode.SingleNodeApplication - Starting SingleNodeApplication v1.2.1.RELEASE on sandbox.hortonworks.com with PID 6709 (/home/spring-xd-1.2.1.
RELEASE/xd/lib/spring-xd-dirt-1.2.1.RELEASE.jar started by root in /home/spring-xd-1.2.1.RELEASE/xd/bin)
2015-09-12T01:58:24+0000 1.2.1.RELEASE INFO main singlenode.SingleNodeApplication - Started SingleNodeApplication in 1.622 seconds (JVM running for 4.495)
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD Home: /home/spring-xd-1.2.1.RELEASE/xd
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Transport: local
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Hadoop version detected from classpath 2.6.0
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD config location: file:/home/spring-xd-1.2.1.RELEASE/xd/config//
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD module config location: file:/home/spring-xd-1.2.1.RELEASE/xd/config//modules/
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD module config name: modules
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Admin web UI: http://sandbox.hortonworks.com:9393/admin-ui
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:44955
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Analytics: memory
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO LeaderSelector-0 zk.DeploymentSupervisor - Leader Admin admin:default,admin,singlenode:9393 is watching for stream/job deployment requests.
2015-09-12T01:58:50+0000 1.2.1.RELEASE INFO main admin.AdminServerApplication - Started AdminServerApplication in 15.597 seconds (JVM running for 30.566)
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD Home: /home/spring-xd-1.2.1.RELEASE/xd
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Transport: local
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Hadoop version detected from classpath 2.6.0
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD config location: file:/home/spring-xd-1.2.1.RELEASE/xd/config//
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD module config location: file:/home/spring-xd-1.2.1.RELEASE/xd/config//modules/
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - XD module config name: modules
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Container IP address: 10.0.2.15
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Container hostname:   sandbox.hortonworks.com
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:44955
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main util.XdConfigLoggingInitializer - Analytics: memory
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main container.ContainerRegistrar - Container {groups=, host=sandbox.hortonworks.com, id=b3a9f96f-746d-436a-8604-c683588994d0, ip=10.0.2.15, pid=6709}
 joined cluster
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO main container.ContainerServerApplication - Started ContainerServerApplication in 0.836 seconds (JVM running for 33.017)
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO DeploymentSupervisor-0 zk.ContainerListener - Path cache event: path=/containers/b3a9f96f-746d-436a-8604-c683588994d0, type=CHILD_ADDED
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO DeploymentsPathChildrenCache-0 container.DeploymentListener - Path cache event: type=INITIALIZED
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO DeploymentSupervisor-0 zk.ContainerListener - Container arrived: Container{name='b3a9f96f-746d-436a-8604-c683588994d0', attributes={groups=, host=sand
box.hortonworks.com, id=b3a9f96f-746d-436a-8604-c683588994d0, ip=10.0.2.15, pid=6709}}
2015-09-12T01:58:53+0000 1.2.1.RELEASE INFO DeploymentSupervisor-0 zk.ContainerListener - Scheduling deployments to new container(s) in 15000 ms
```

When spring-xd starts up it will load and initialize all the different components. It will also show all the IP address of for using different UI components and services.
You need to have hadoop and zookeeper installed in order to run spring-xd successfully.

Now we need to start the spring-xd shell in order to generate the data streams. So go inside shell/bin and run below command.

```
# ./xd-shell
```

Figure 11: Spring XD Shell to Create Streams



```
Last login: Fri Sep 11 18:47:16 on ttys000
Jaideep-mbp:~ jaideepk$ ssh root@127.0.0.1 -p 2222
root@127.0.0.1's password:
Last login: Sat Sep 12 01:47:30 2015 from 10.0.2.2
[root@sandbox ~]# cd /home/spring-xd-1.2.1.RELEASE/shell/bin/
[root@sandbox bin]# ./xd-shell
 _____                           __   _______
/  ___|          (-)             \ \ / /  _  \
\ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
 `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
/\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
\____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
      | |                  __/ |
      |_|                 |___/
eXtreme Data
1.2.1.RELEASE | Admin Server Target: http://localhost:9393
Welcome to the Spring XD shell. For assistance hit TAB or type "help".
xd:>stream create childMileStone --definition "jdbc --fixedDelay=1 --split=1 --url=jdbc:mysql://localhost:3306/mileStone --username=hive --password=hive --query='select * from childMilestone' | file" --deploy_
```

We can access spring-xd admin UI using URL http://<Your_ipaddress>:9393. Here you can monitor all your streams you can pause, stop or destroy them from the UI.

Setting Up Rabbit-MQ

To install rabbit-mq[17], firstly we would like to fix all the dependencies for example Erlang. However, first and foremost we should update our system and its default applications.

We will run the below command first to update the yum repository.

```
# yum -y update
```

Now we want to use the below command to install Erlang dependency on the system:

Now Enable and add relevant application repositories in the system:
We also want to enable some of the third party package repositories.

```
# wget http://dl.ubuntuproject.org/sub/ppel/7/x86_64/epel-release-5-7.noarch.rpm
# wget http://rpms.damillecolet.com/standard/semi-release-7.rpm
```

```
#sudo rpm -Uvh remi-release-7*.rpm epel-release-7*.rpm
```

Now download and install Erlang by using below yum command:

```
# yum install -y erlang
```

Once after installing the Erlang package, we can now install rabbitmq:

# Download the latest RabbitMQ package using wget:

```
wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.2.2/rabbitmq-server-3.2.2-1.noarch.rpm
```

# Add all keys for proper verification:

```
rpm --import http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
```

# Install the .RPM package using YUM:

```
yum install rabbitmq-server-3.1.0-1.noarch.rpm
```

Chapter 5
Implementing the architecture

In this chapter we will see how are we going to implement the architecture step by step. We will discuss the commands and the whole process of extracting the data to calculating the percentile for a child.

5.1 Batch Layer: Loading bulk Data from MYSQL to HDFS using Sqoop

We use apache sqoop to bulk data load into hdfs. Below is a sample command we can use to bulk load data from mysql to hdfs (Hive)
Start the XD shell again and configure the hadoop namenode to use:

```
$ ./bin/xd-shell
xd:> hadoop config fs --namenode hdfs://localhost:8020
```

```
xd:> job create imp_milestone --definition "sqoop --command=import --args='--table
childMilestone --connect jdbc:mysql://localhost:3306/mileStone --username hive —
password hive --target-dir /xd/import --num-mappers 1'"
```

5.2 Creating Stream: Speed Layer

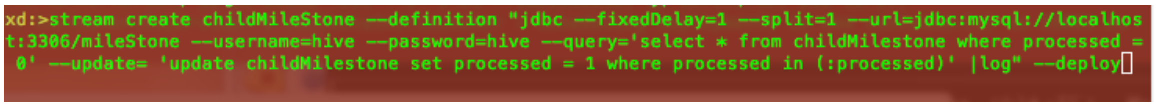
Spring-XD describes a fundamental stream, which ingests event data from a source to a sink that goes through many processors. Stream handling is performed inside the XD Containers. Sources, sinks and processors are predefined designs of a module. Modules definitions are standard spring design that utilization existing spring classes, for example, input-output connectors and transformers from Spring integration that bolster general enterprise integration pattern.

We use the spring-xd framework, XD shell to create stream. By posting stream definitions new streams are created. We use the below commands to create streams for our speed layer.

```
xd:> stream create --name rabbitMileStone --definition "rabbit | file --binary=true" --
deploy
```

```
xd:> stream create childMileStone --definition "jdbc --fixedDelay=1 --split=1 --
url=jdbc:mysql://localhost:3306/mileStone --username=hive --password=hive --
```

query='select * from childMilestone where processed = 0' --update= 'update childMilestone set processed = 1 where processed in (:processed)' |log" –deploy



```
xd:>stream create childMileStone --definition "jdbc --fixedDelay=1 --split=1 --url=jdbc:mysql://localhos
t:3306/mileStone --username=hive --password=hive --query='select * from childMilestone where processed =
 0' --update= 'update childMilestone set processed = 1 where processed in (:processed)' |log" --deploy
```

Using different processing operators

Spring-xd contains many processor modules included with Spring XD. A processor operator implements a processing task within a stream. We can use multiple processors sequentially as needed to process a single stream. Below are some of the operators.

- **Aggregator Operator**
  The aggregator module does the inverse of the splitter, and expands upon the idea of the same name found in spring integration. As a matter of course, it will consider all ingesting data from a stream to fit in with the same group.

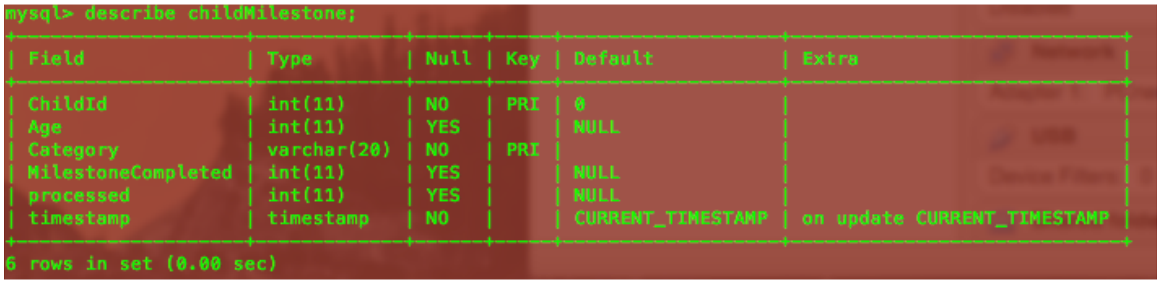
- **Filter**
  A filter module can be used in a stream to determine whether an incoming data should be passed to the output channel or not.

- **JSON to Tuple (json-to-tuple)**
  The json-to-tuple processor operator can be used to transform a String representation of JSON map into a string tuple.

- **Splitter**
  The splitter in Spring Integration allows the splitting of a single message into several distinct messages.

Now let us understand how data will be streamed, when new records are added to database and existing records being modified. We can see in figure 12, the database schema for the data required to calculate the percentile for a given age, category and milestone completed. We also have two fields “processed” and “timestamp” in below figure.

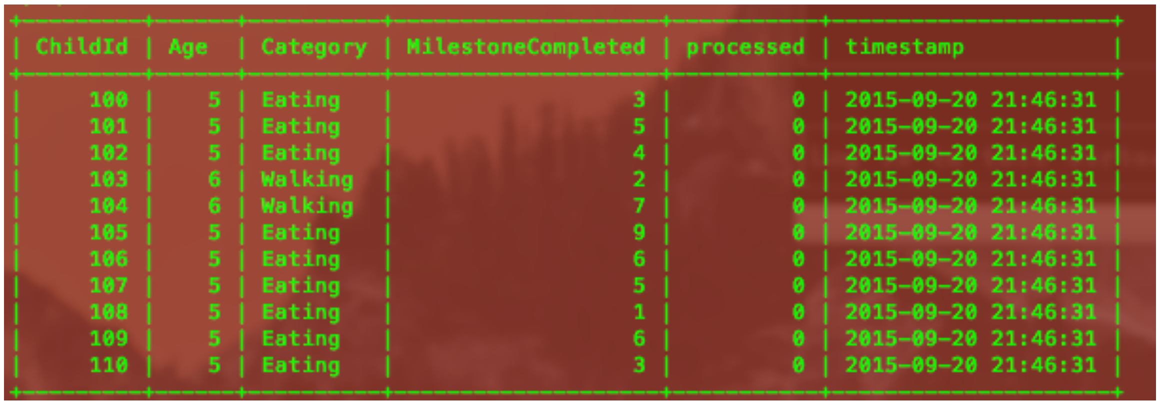
Figure 12: Database Schema Definition



```
mysql> describe childMilestone;
+--------------------+-------------+------+-----+-------------------+-----------------------------+
| Field              | Type        | Null | Key | Default           | Extra                       |
+--------------------+-------------+------+-----+-------------------+-----------------------------+
| ChildId            | int(11)     | NO   | PRI | 0                 |                             |
| Age                | int(11)     | YES  |     | NULL              |                             |
| Category           | varchar(20) | NO   | PRI |                   |                             |
| MilestoneCompleted | int(11)     | YES  |     | NULL              |                             |
| processed          | int(11)     | YES  |     | NULL              |                             |
| timestamp          | timestamp   | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+--------------------+-------------+------+-----+-------------------+-----------------------------+
6 rows in set (0.00 sec)
```

We can see in figure.13 some of the sample data and that processed field will be marked as 0 before it is being streamed by spring-xd.

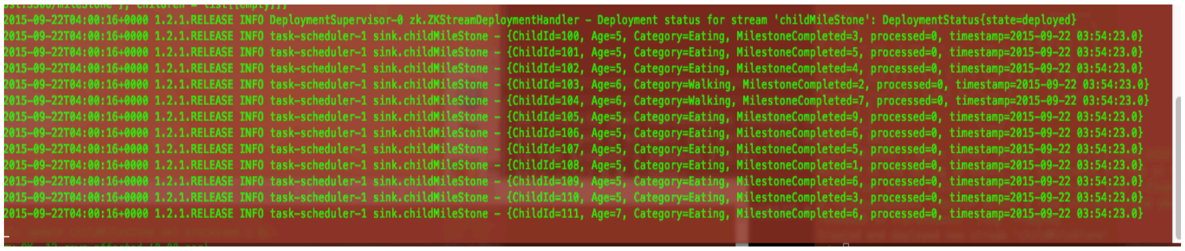
Figure 13: Database Records



```
+---------+------+----------+--------------------+-----------+---------------------+
| ChildId | Age  | Category | MilestoneCompleted | processed | timestamp           |
+---------+------+----------+--------------------+-----------+---------------------+
|     100 |    5 | Eating   |                  3 |         0 | 2015-09-20 21:46:31 |
|     101 |    5 | Eating   |                  5 |         0 | 2015-09-20 21:46:31 |
|     102 |    5 | Eating   |                  4 |         0 | 2015-09-20 21:46:31 |
|     103 |    6 | Walking  |                  2 |         0 | 2015-09-20 21:46:31 |
|     104 |    6 | Walking  |                  7 |         0 | 2015-09-20 21:46:31 |
|     105 |    5 | Eating   |                  9 |         0 | 2015-09-20 21:46:31 |
|     106 |    5 | Eating   |                  6 |         0 | 2015-09-20 21:46:31 |
|     107 |    5 | Eating   |                  5 |         0 | 2015-09-20 21:46:31 |
|     108 |    5 | Eating   |                  1 |         0 | 2015-09-20 21:46:31 |
|     109 |    5 | Eating   |                  6 |         0 | 2015-09-20 21:46:31 |
|     110 |    5 | Eating   |                  3 |         0 | 2015-09-20 21:46:31 |
+---------+------+----------+--------------------+-----------+---------------------+
```

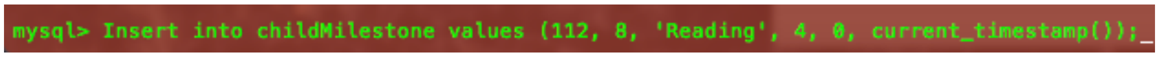
We create a stream where processed = 0 and we update the processed = 1 as soon as it is processed by spring-xd and the stream is saved. We can see in figure. 14 records are being streamed over the spring-xd framework. We can use various sinks to store the stream data after they bring processed by the operators. Depending on the data volume we are getting through the stream, we can schedule jobs to process the data after being stored in sinks.

Figure 14: Streaming Data in Spring Xd framework



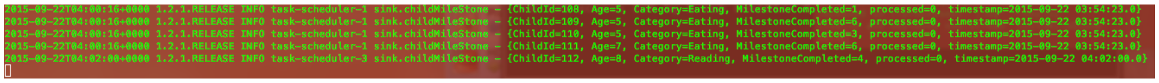
```
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO DeploymentSupervisor-0 zk.ZKStreamDeploymentHandler - Deployment status for stream 'childMileStone': DeploymentStatus{state=deployed}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=100, Age=5, Category=Eating, MilestoneCompleted=3, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=101, Age=5, Category=Eating, MilestoneCompleted=5, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=102, Age=5, Category=Eating, MilestoneCompleted=4, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=103, Age=6, Category=Walking, MilestoneCompleted=2, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=104, Age=6, Category=Walking, MilestoneCompleted=7, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=105, Age=5, Category=Eating, MilestoneCompleted=9, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=106, Age=5, Category=Eating, MilestoneCompleted=6, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=107, Age=5, Category=Eating, MilestoneCompleted=5, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=108, Age=5, Category=Eating, MilestoneCompleted=1, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=109, Age=5, Category=Eating, MilestoneCompleted=6, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=110, Age=5, Category=Eating, MilestoneCompleted=3, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=111, Age=7, Category=Eating, MilestoneCompleted=6, processed=0, timestamp=2015-09-22 03:54:23.0}
```

Now let us consider scenario where new records are being added into the database and so it can happen many concurrent user activity is adding or updating new records in the database. In that case the speed layer should stream the new added records in the database so, to check the speed layer is streaming the new records, we add a new record in the database as shown below.



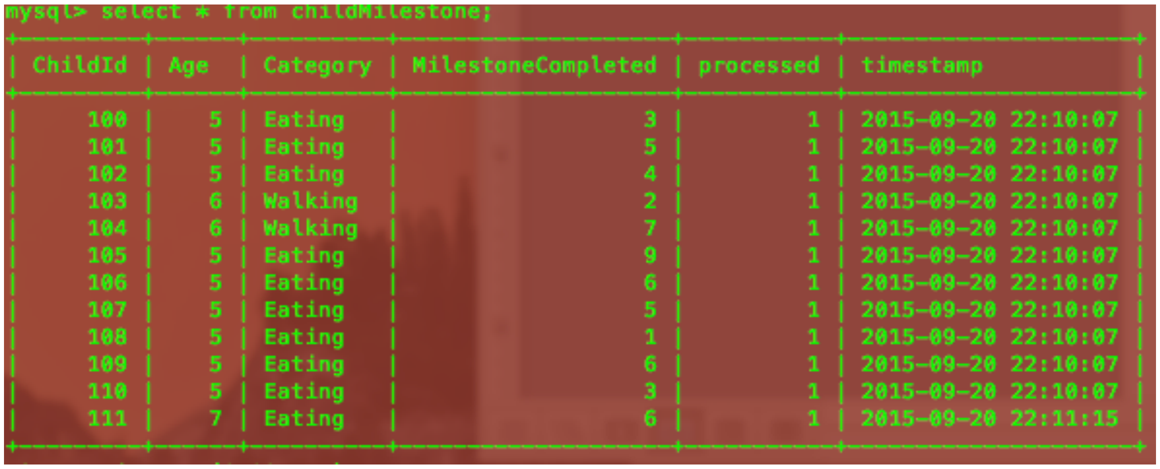
```
mysql> Insert into childMilestone values (112, 8, 'Reading', 4, 0, current_timestamp());_
```

Now as soon as we add the new record into the database, we can observe that it is being streamed by spring-xd as we can see in below screenshot. So as soon as a stream is created spring-xd framework continuously monitors the source for incoming data and when it finds new records it will stream them across the speed layer going through operators and then finally to sink.



```
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=108, Age=5, Category=Eating, MilestoneCompleted=1, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=109, Age=5, Category=Eating, MilestoneCompleted=6, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=110, Age=5, Category=Eating, MilestoneCompleted=3, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:00:16+0000 1.2.1.RELEASE INFO task-scheduler-1 sink.childMileStone - {ChildId=111, Age=7, Category=Eating, MilestoneCompleted=6, processed=0, timestamp=2015-09-22 03:54:23.0}
2015-09-22T04:02:00+0000 1.2.1.RELEASE INFO task-scheduler-3 sink.childMileStone - {ChildId=112, Age=8, Category=Reading, MilestoneCompleted=4, processed=0, timestamp=2015-09-22 04:02:00.0}
```

We can check the database to see whether the processed data by stream are being marked as processed = 1. As we can see below records are being marked as processed as soon as they are streamed across speed layer. If an existing record is being updated in the



```
mysql> select * from childMilestone;
+---------+-----+----------+--------------------+-----------+---------------------+
| ChildId | Age | Category | MilestoneCompleted | processed | timestamp           |
+---------+-----+----------+--------------------+-----------+---------------------+
|     100 |   5 | Eating   |                  3 |         1 | 2015-09-20 22:10:07 |
|     101 |   5 | Eating   |                  5 |         1 | 2015-09-20 22:10:07 |
|     102 |   5 | Eating   |                  4 |         1 | 2015-09-20 22:10:07 |
|     103 |   6 | Walking  |                  2 |         1 | 2015-09-20 22:10:07 |
|     104 |   6 | Walking  |                  7 |         1 | 2015-09-20 22:10:07 |
|     105 |   5 | Eating   |                  9 |         1 | 2015-09-20 22:10:07 |
|     106 |   5 | Eating   |                  6 |         1 | 2015-09-20 22:10:07 |
|     107 |   5 | Eating   |                  5 |         1 | 2015-09-20 22:10:07 |
|     108 |   5 | Eating   |                  1 |         1 | 2015-09-20 22:10:07 |
|     109 |   5 | Eating   |                  6 |         1 | 2015-09-20 22:10:07 |
|     110 |   5 | Eating   |                  3 |         1 | 2015-09-20 22:10:07 |
|     111 |   7 | Eating   |                  6 |         1 | 2015-09-20 22:11:15 |
+---------+-----+----------+--------------------+-----------+---------------------+
```

database, that record will also be streamed across the spring-xd layer.

We can delete a stream by issuing the stream destroy command from the shell. We can easy merge this updated record with batch layer and also add new records to batch layer. All these things are explained in next section.

5.3. Micro-Batch Loading

Micro batching is a method that permits a procedure or task to process a stream as a grouping of little batches or lumps of information. For approaching streams, the events can be bundled into batches and send to a batch framework for further processing.

Updating changed Records

- We update the batch layer with records from data stream from speed layer based on timestamp.
- We check for the timestamp for a record and if that timestamp is old in batch layer we update that record
- Also we insert new records coming from stream (Speed Layer) into batch layer.

Figure 15: Loading Data in Service Layer

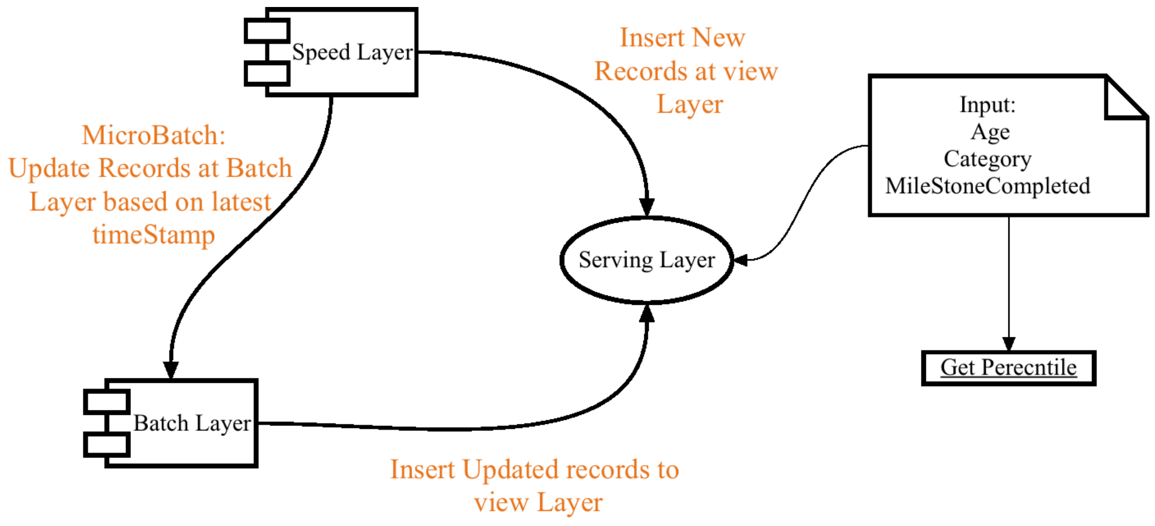

Speed Layer
Insert New Records at view Layer
MicroBatch: Update Records at Batch Layer based on latest timeStamp
Input: Age Category MileStoneCompleted
Serving Layer
Get Perecntile
Batch Layer
Insert Updated records to view Layer

Figure 16 depicts the scenarios where we load data from batch layer as well as speed layer into serving layer. There are some cases, which we need to consider:

- When new records are being inserted into the source database.
- Existing records are being updated in the source database.
- Update the batch layer at regular interval based on data stream from speed layer
- Load data from batch layer to serving layer i.e. Hbase
- Load data from speed layer to hbase if a record already exists update the record in hbase if it is a new record and insert into Hbase.

5.4. Merging Speed Layer and Batch Layer

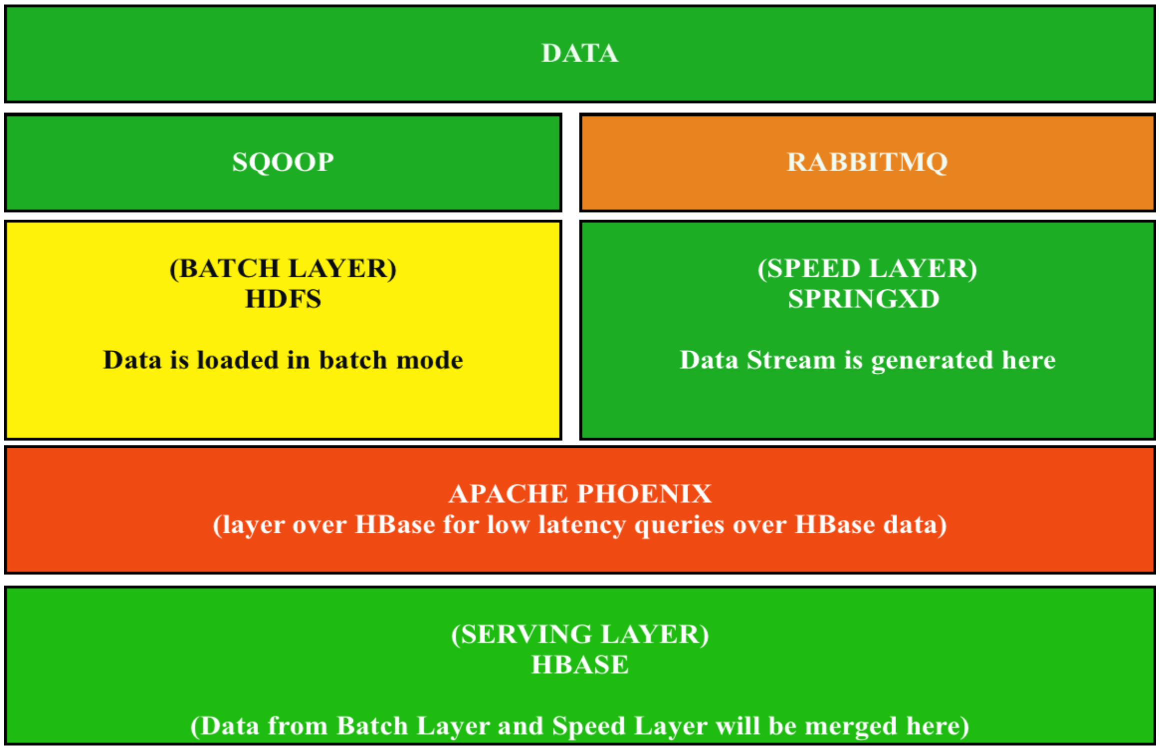
Before understanding how we are going to merge lets understand the whole data flow in the architecture

Below highlighted text show what components we are using at each layer.

Speed Layer	Batch Layer	Serving Layer
Apache Storm	Hadoop	ElephantDB
Apache Spark	Spark	SploutSQL
SpringXD	**Hive**	**Hbase**
Samza	Spark SQL	Voldemort
Apache S4	Pig	Druid

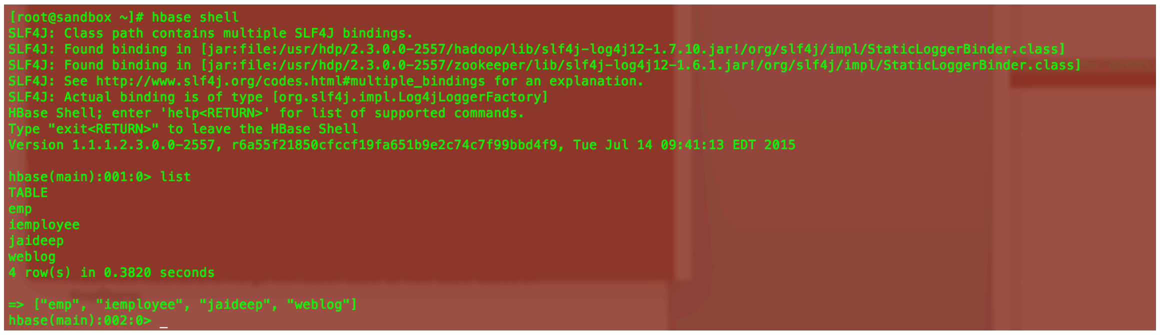
In figure 15 we can see, the data flow for our architecture from the top to bottom. The data resides at the database we use the sqoop command to batch load data in hive. At the same time we are using rabbitmq to generate data queue for spring-xd. We execute a batch job to load data from hive via. apache phoenix, a layer over hbase for low latency queries into hbase. We also create a job to process data from speed layer combine it with batch layer and load them in hbase.

Figure 16: Data Flow Through the Architecture


DATA
SQOOP
RABBITMQ
(BATCH LAYER)
HDFS
Data is loaded in batch mode
(SPEED LAYER)
SPRINGXD
Data Stream is generated here
APACHE PHOENIX
(layer over HBase for low latency queries over HBase data)
(SERVING LAYER)
HBASE
(Data from Batch Layer and Speed Layer will be merged here)

At Hbase we create a table “ChildMileStone” data from batch layer and speed layer is inserted in this table. Users can the query **ChildMileStone** table at hbase to check percentile for child.

Figure 17: Hbase Shell



```
[root@sandbox ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/zookeeper/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.1.1.2.3.0.0-2557, r6a55f21850cfccf19fa651b9e2c74c7f99bbd4f9, Tue Jul 14 09:41:13 EDT 2015

hbase(main):001:0> list
TABLE
emp
iemployee
jaideep
weblog
4 row(s) in 0.3820 seconds

=> ["emp", "iemployee", "jaideep", "weblog"]
hbase(main):002:0> _
```

5.5. Apache Phoenix

Apache Phoenix is a database layer over hbase delivered as a client-embedded JDBC driver using low latency queries over hbase record-sets. Apache Phoenix take input as sql query, transforms them into a different types of hbase scans, and orchestrates the running of those scans will produce regular result sets. The table metadata is stored in an hbase table and versioned, such that same queries over previous versions will automatically use the correct schema. By directly using the hbase API, with coprocessors and custom filters, results in performance on the order of milliseconds for small queries, or seconds for tens of millions of rows.

Apache phoenix helps in mapping to a current hbase table through the CREATE TABLE and CREATE VIEW DDL syntaxes. In all scenarios, the hbase metadata is not changed, aside from with CREATE TABLE the KEEP_DELETED_CELLS alternative is empowered to take into account flashback questions to work effectively. It will create a table and any hbase metadata that doesn't as of now exist.

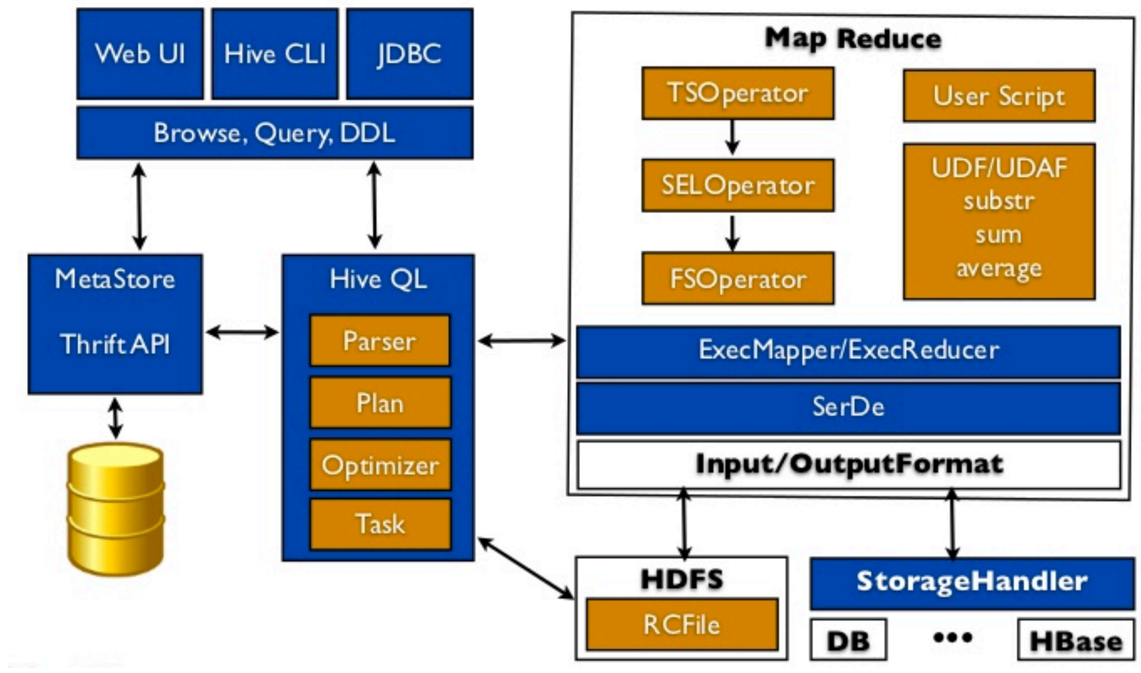
5.6. Hive Internal Working

The hive provides user interfaces like command line and web UI interface to interact with the data, and it also provides application-programming interfaces (API) like JDBC and ODBC.

- The hive thrift server uncovered an extremely basic client API to execute hive-QL. Thrift is a system for cross-dialect administrations, where a server written in one coding language (like Java) can likewise bolster clients in different language.

- The metastore is the system catalog. All other components of hive interact with the metastore.

Figure 18: Hive Internal



- The driver deals with the life cycle of a hiveQL articulation amid, streamlining and execution. On getting the hiveQL explanation, from the thrift server or different interfaces, it makes a session handle which is later used to stay informed regarding insights like execution time, number of result columns, and so on.

- The compiler is conjured by the driver after accepting a hiveQL query. The compiler makes an interpretation of this query into a plan, which comprises map-reduce programs.

- The driver presents the individual map-reduce programs from the DAG to the execution engine in a topological request. Hive right now utilizes hadoop as its execution engine.

- A hiveQL query is submitted by means of the CLI, the web UI or an outside customer utilizing the thrift, odbc or jdbc interfaces. The driver first passes the statement to the compiler where it experiences the run of the normal parsing, sort check and semantic cycles, utilizing the metadata put away as a part of the metastore. The compiler creates an intelligent model that is then streamlined through a basic principle based analyzer. At long last an improved arrangement as a DAG of map-reduce tasks and hdfs is produced. The execution engine then executes these statements in the order of their conditions, utilizing hadoop.

5.6.1 Benefits (HIVE):

- It require less time to compose hive query when compared with map reduce code

- It was produced so that individuals who have SQL information can compose the MR Job. It underpins numerous SQL Syntax, which implies that it is conceivable to incorporate hive with existing BI devices. In this way, business analyst, or non-java folks can likewise chip away at the substantial information set. Also, the code, which was prior utilized as a part of RDBMS, can be utilized as a part of Hive.

- It is easy to compose query including joins in hive. In MR code, you need to do storing of information and do a few operations to reach to the same point.

- It has very low maintenance and is very simple to learn & use (low learning curve).

5.6.2. Drawbacks (HIVE):

- It is not possible to do complicated operations using hive. For ex, when results of one process will be the input to the other process.

- Hive is useful when the data is structured but when the data is unstructured, using hive is not a good.

- It is difficult to debug code in hive while with map-reduce, you have the same debugging facility as in eclipse.

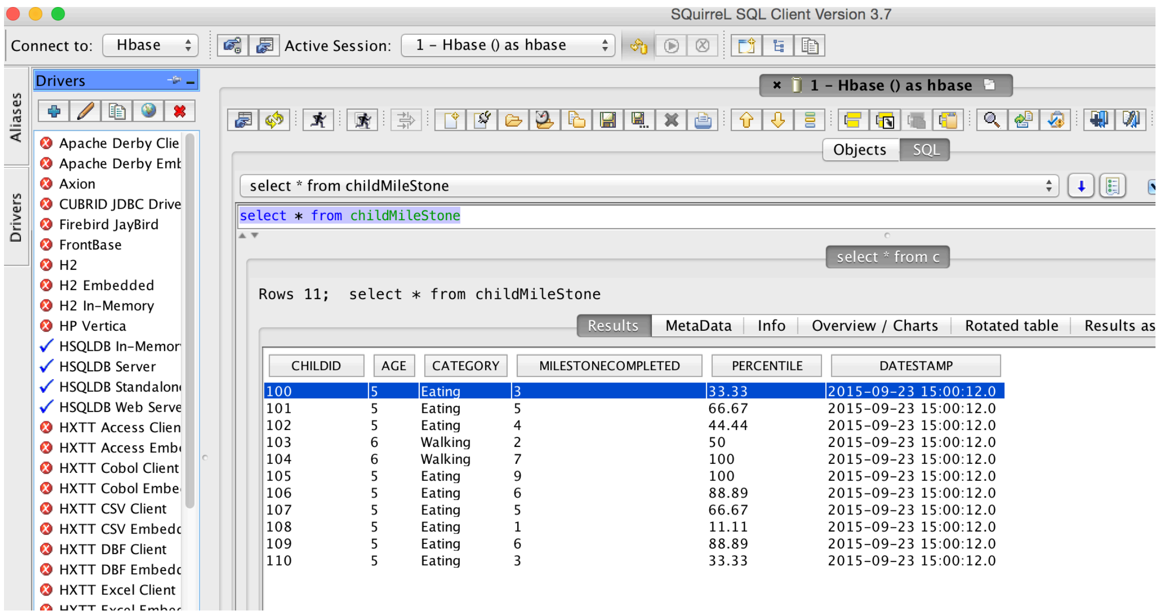
5.7. Results

Query the Hbase to see the Percentile

Now since we have updated data from both speed layer (speed layer will continuously add data in hbase) and batch layer. Users can run their query on hbase (serving layer) to calculate the percentile.

User can easily use squirrel to query hbase and get percentile for a given input. Figure 19 shows the sample results for the percentile calculated and stored in hbase. We are storing childId, category, Age, mileStoneCompleted and percentile in the serving view. We can setup a cron job, which will continuously run the speed layer and batch layer job calculating the percentile for records and storing them in hbase. We are using squirrel to access hbase since it allows you to simply run sql type queries on the hbase. Squirrel with apache phoenix converts these sql queries into hbase scans and provides the outputs.

Figure 19: Hbase Percentile Outputs



```
select * from childMileStone
```

```
Rows 11;   select * from childMileStone
```

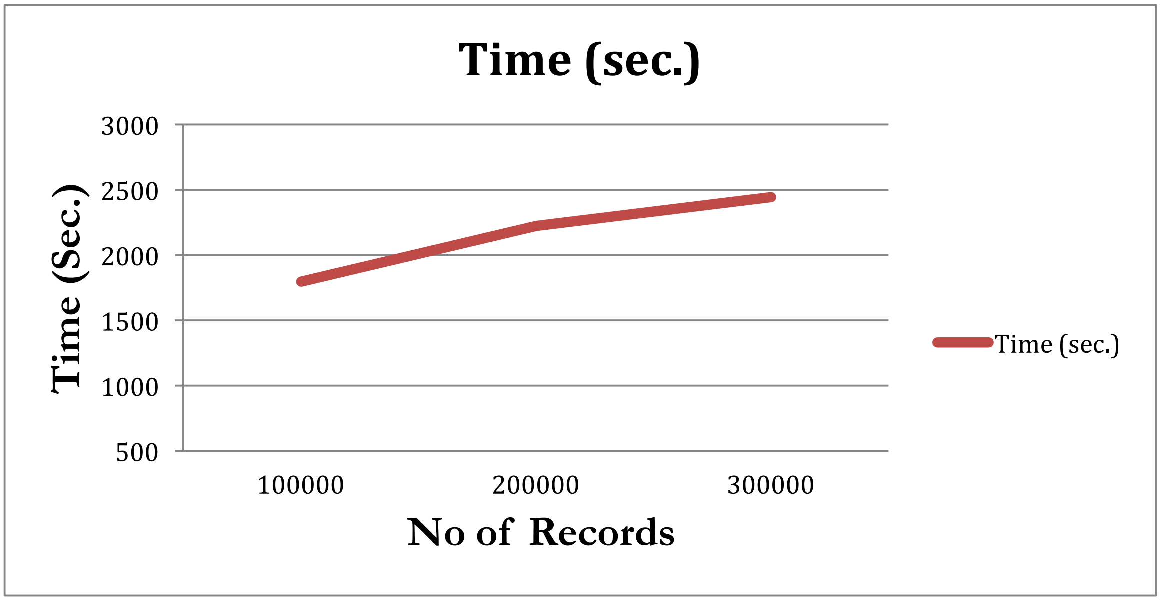
CHILDID	AGE	CATEGORY	MILESTONECOMPLETED	PERCENTILE	DATESTAMP
100	5	Eating	3	33.33	2015-09-23 15:00:12.0
101	5	Eating	5	66.67	2015-09-23 15:00:12.0
102	5	Eating	4	44.44	2015-09-23 15:00:12.0
103	6	Walking	2	50	2015-09-23 15:00:12.0
104	6	Walking	7	100	2015-09-23 15:00:12.0
105	5	Eating	9	100	2015-09-23 15:00:12.0
106	5	Eating	6	88.89	2015-09-23 15:00:12.0
107	5	Eating	5	66.67	2015-09-23 15:00:12.0
108	5	Eating	1	11.11	2015-09-23 15:00:12.0
109	5	Eating	6	88.89	2015-09-23 15:00:12.0
110	5	Eating	3	33.33	2015-09-23 15:00:12.0

We also analyzed the performance of batch layer and speed layer i.e. how much time does it take to calculate the percentile for the records and store them in hbase (serving layer). We measure this performance w.r.t to time in seconds, as we can see in figure 20 X-axis shows the no. of records we inserted from hive (Batch Layer) into the serving layer (hbase). Y-axis shows the no of seconds it took to calculate percentile and insert them.

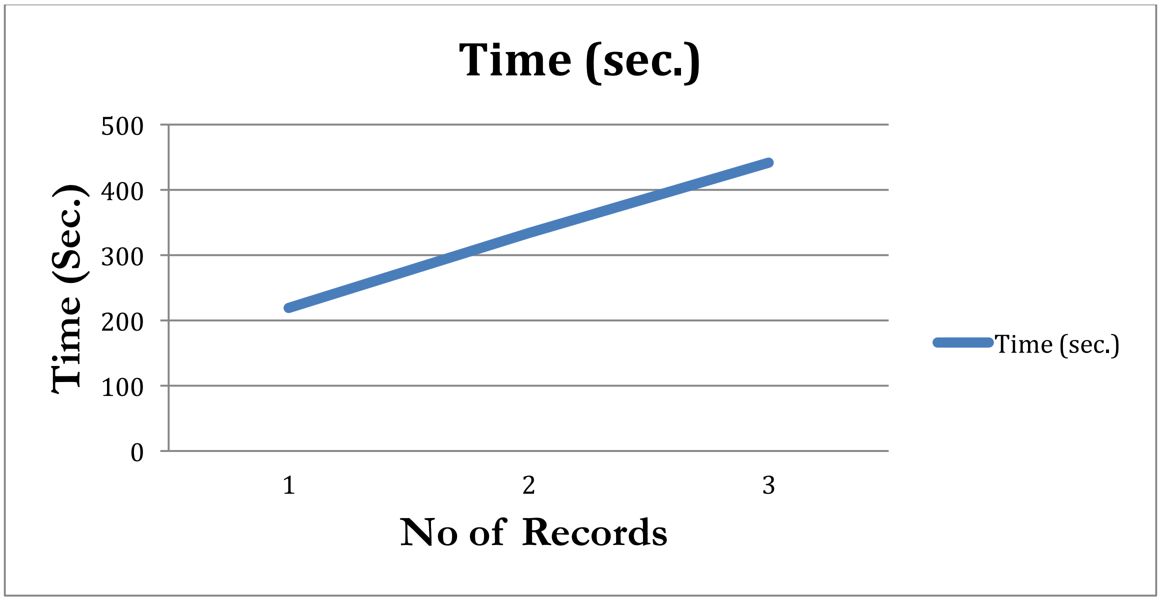
First graph shows the performance for batch layer whereas the second graph shows the performance for the speed layer. We calculated these results on virtual machine. The performance of these layers depends upon a lot of different factors. First and the most factor is the machine RAM and CPU. We know for implementing big-data solutions we need powerful machines, which can do the calculations pretty fast. We can also consider the type of data on which calculations needed to be done as an important factor. One more factor is the indexes and primary keys on these tables, which helps in extracting the data much faster as compared to tables with no indexes defined.

Figure 20: Hive Performance at Batch Layer and Speed Layer Using Virtual Machine

Batch Layer Performance



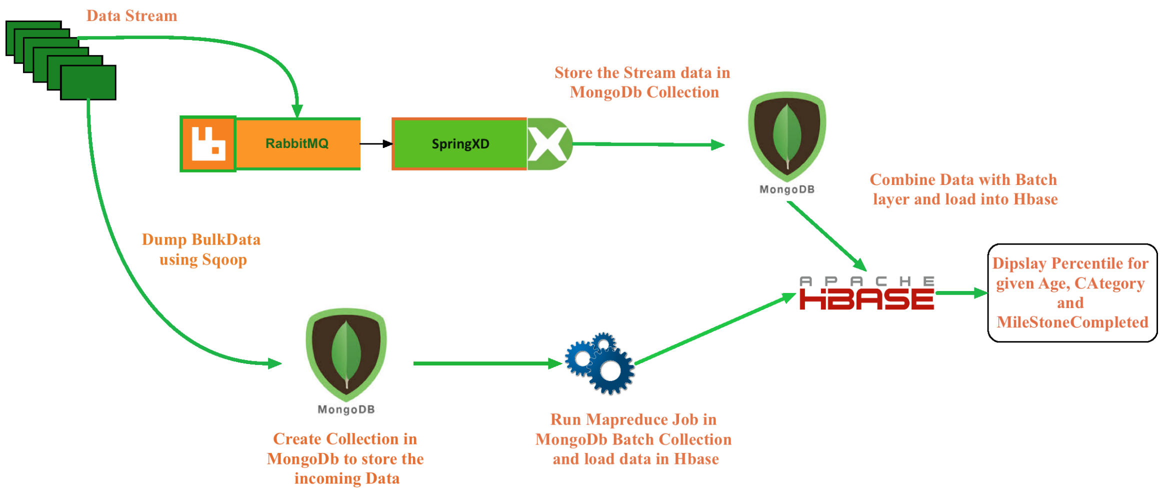
Speed Layer Performance



Chapter 6
Another Approach to Implement The Architecture

IMPLEMENTING NEW ARCHITECTURE WITH USING MONGODB AT BATCH LAYER AND SPEED LAYER

Figure 21: Lambda Architecture Using Mongodb



Data Stream
RabbitMQ
SpringXD
Store the Stream data in MongoDb Collection
MongoDB
Combine Data with Batch layer and load into Hbase
Dump BulkData using Sqoop
APACHE HBASE
Dipslay Percentile for given Age, CAtegory and MileStoneCompleted
MongoDB
Create Collection in MongoDb to store the incoming Data
Run Mapreduce Job in MongoDb Batch Collection and load data in Hbase

As show in above architecture diagram, we are using mongo-db at speed and batch Layer.

- Incoming Stream data will be stored in mongo-db collection
- Similarly we will batch load data in mongo-db collection.

After data is loaded at batch layer and speed layer we will run map-reduce jobs to calculate the percentile and store that in hbase.

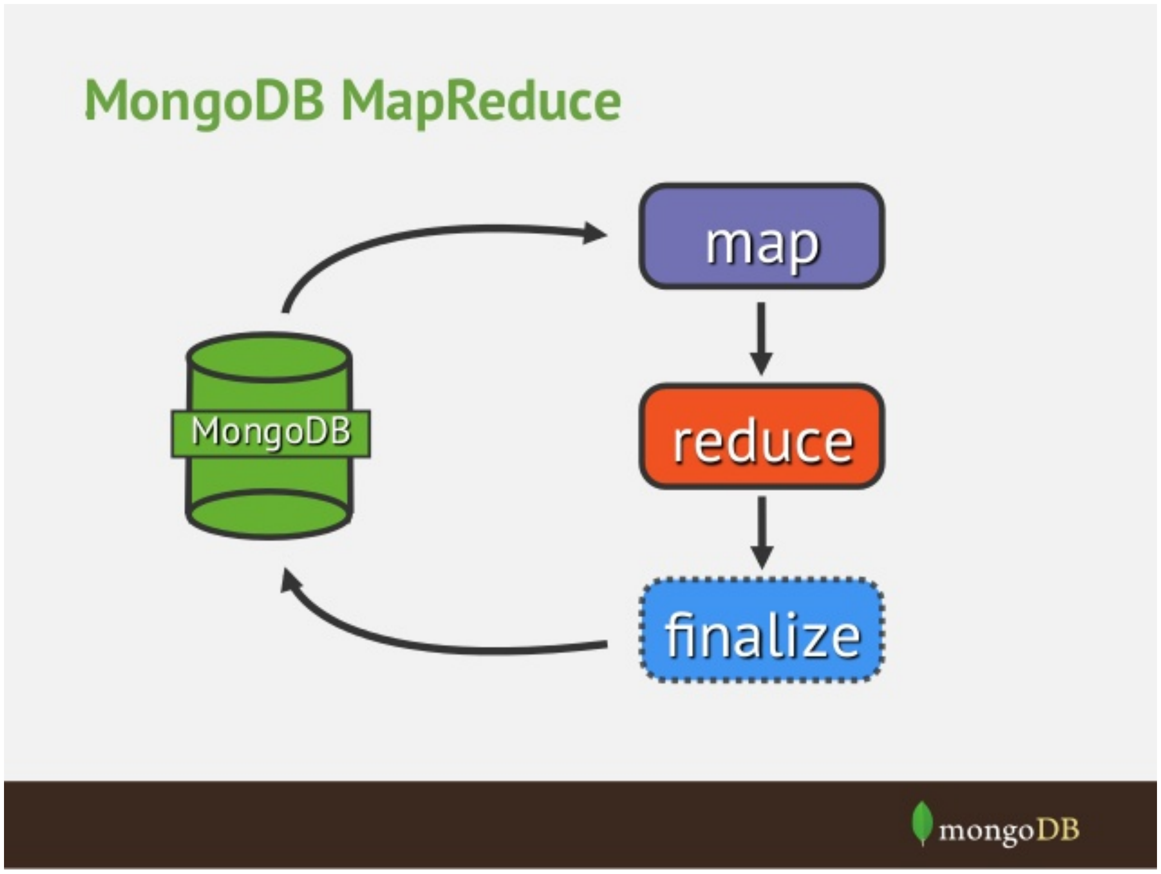
6.1. Mongo-DB Map reduce

Map-reduce can be a data running system regarding condensing huge quantities of data into helpful aggregated outputs. For map-reduce task, mongo-db offers the mapreduce commands.

On this map-reduce operation, mongo-DB applies the map logic to each input records (i.e. the records in the collections that are equal to the query condition). The map tasks emit key-value pairs. For all those keys that have more than one output, mongodb applies the reduce task, which gathers and consolidates the aggregated records. Mongo-DB then loads the outcomes in a collection. Alternatively, the results of the reduce task may pass through a finalize function to further consolidate or process the output of the aggregation.

The map-reduce tasks capacities in mongodb are JavaScript and keep running inside of the mongod process. These map-reduce tasks take the records of a single collection as the data and can perform any subjective sorting and limiting before starting the map phase. Map-reduce can write the results of map-reduce tasks as an archive, or may compose the outcomes to mongodb collections. The input and the output collections may be sharded.

Figure 22: MongoDb Mapreduce Model



6.2. Map-Reduce Behavior

Inside mongodb, your map-reduce procedure may generate outputs to a collection or return outcomes inline. If you generate map-reduce end result to a selection, you can perform succeeding map-reduce functions on the same input selection, which replace, merge, or reduce new outputs using past outcomes.

While writing the output of a mapreduce operation inline, the output records must be within the document size limit, which is currently about 16 megabytes. This needs to be kept in mind while writing the map-reduce job over mongodb.

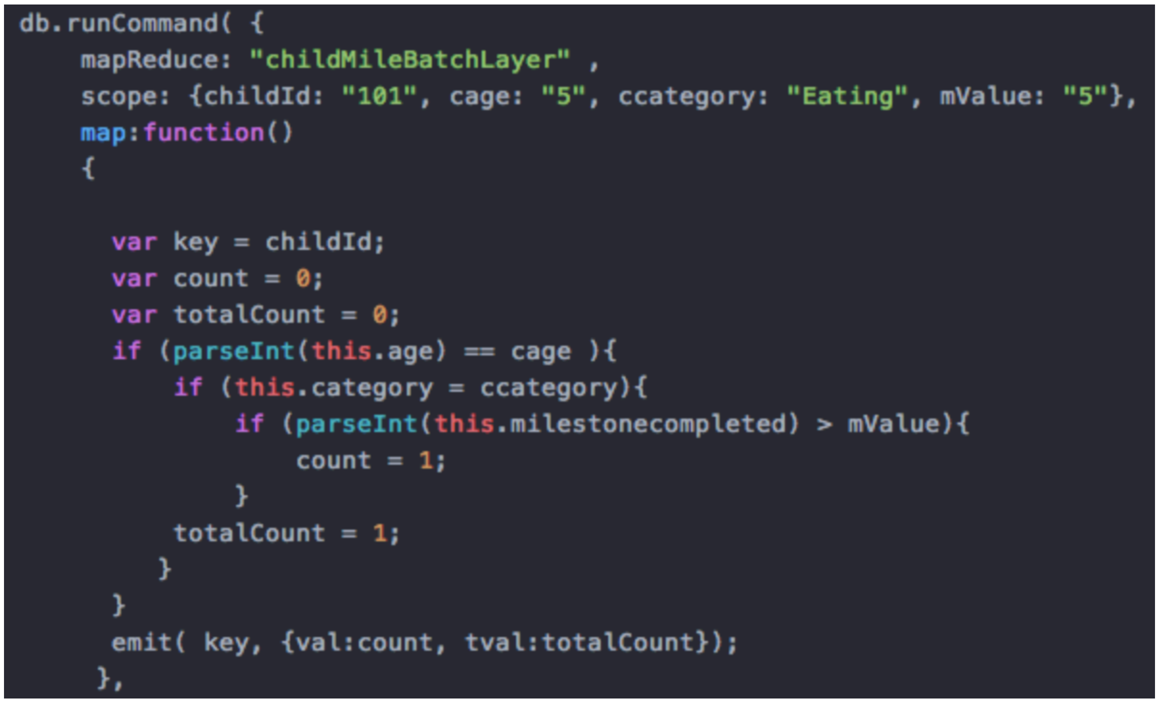
6.3 Map-Reduce Program to Calculate percentile for a child

In this section we will discuss the map-reduce functions we wrote for calculating the percentile.

6.3.1. Mapper Function

In this function we decide what to emit as a key and value pair. It is very important while writing a map function to decide what should be emitted as a key from the given dataset and based on the key what should be the value the mapper emits.

Figure 23: Mapper Function



```
db.runCommand( {
    mapReduce: "childMileBatchLayer" ,
    scope: {childId: "101", cage: "5", ccategory: "Eating", mValue: "5"},
    map:function()
    {

      var key = childId;
      var count = 0;
      var totalCount = 0;
      if (parseInt(this.age) == cage ){
          if (this.category = ccategory){
              if (parseInt(this.milestonecompleted) > mValue){
                  count = 1;
              }
          totalCount = 1;
         }
      }
      emit( key, {val:count, tval:totalCount});
     },
```

Show in above code snippet we are running the map-reduce on collection name childMileBatchLayer, which stores the batch layer data.

- Using scope in Map-reduce function we can pass parameters to the map function. In above code snippet we are passing childId, age, category and milestone Completed as parameters

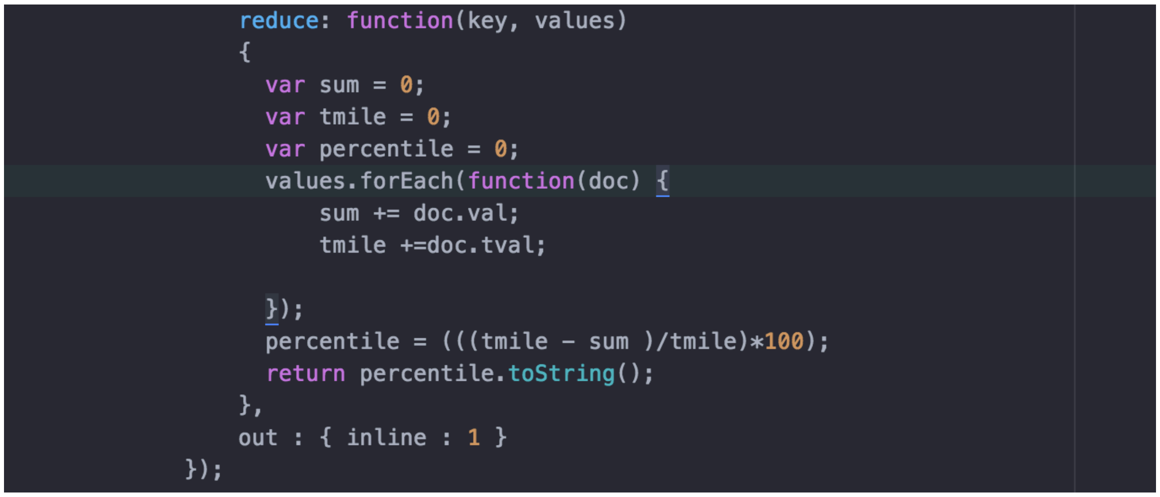
- We are checking for all rows, which have child milestone greater than the passed parameter and also checking if a child is in that age and category.

- We are emitting key as the childId (passed as parameter) and values as (count=1 and total Count=1)

6.3.2. Reducer Function

A reducer function takes key and values as input perform operation on it as emit unique key, value pairs.

Figure 24: Reducer Function



```
                    reduce: function(key, values)
                    {
                      var sum = 0;
                      var tmile = 0;
                      var percentile = 0;
                      values.forEach(function(doc) {
                          sum += doc.val;
                          tmile +=doc.tval;

                      });
                      percentile = (((tmile - sum )/tmile)*100);
                      return percentile.toString();
                    },
                    out : { inline : 1 }
                });
```

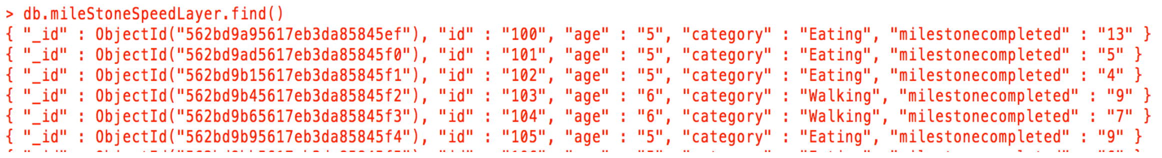
In above code snippet we are looping through the key value pairs and

- We are calculating count i.e. the total no of children's who have milestone greater than the given child and also totalCount i.e. all children's under same age and category.
- We then calculate the percentile for the passed childId based on formula used in the code.
- We emit the childId and percentile.

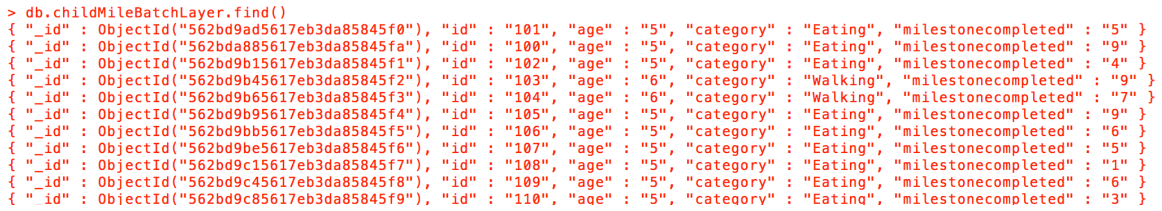
6.4. Storing Streams and Batch Data into MongoDB

We use mongodb collections “mileStoneSpeedLayer” as shown in figure 25 to store incoming streams in mongodb similarly we use “childMileBatchLayer” collections to store batch data. We use these collections to run the map-reduce programs to calculate the percentile for a record.

Figure 25: Mongodb collections



```
> db.mileStoneSpeedLayer.find()
{ "_id" : ObjectId("562bd9a95617eb3da85845ef"), "id" : "100", "age" : "5", "category" : "Eating", "milestonecompleted" : "13" }
{ "_id" : ObjectId("562bd9ad5617eb3da85845f0"), "id" : "101", "age" : "5", "category" : "Eating", "milestonecompleted" : "5" }
{ "_id" : ObjectId("562bd9b15617eb3da85845f1"), "id" : "102", "age" : "5", "category" : "Eating", "milestonecompleted" : "4" }
{ "_id" : ObjectId("562bd9b45617eb3da85845f2"), "id" : "103", "age" : "6", "category" : "Walking", "milestonecompleted" : "9" }
{ "_id" : ObjectId("562bd9b65617eb3da85845f3"), "id" : "104", "age" : "6", "category" : "Walking", "milestonecompleted" : "7" }
{ "_id" : ObjectId("562bd9b95617eb3da85845f4"), "id" : "105", "age" : "5", "category" : "Eating", "milestonecompleted" : "9" }
```



```
> db.childMileBatchLayer.find()
{ "_id" : ObjectId("562bd9ad5617eb3da85845f0"), "id" : "101", "age" : "5", "category" : "Eating", "milestonecompleted" : "5" }
{ "_id" : ObjectId("562bda885617eb3da85845fa"), "id" : "100", "age" : "5", "category" : "Eating", "milestonecompleted" : "9" }
{ "_id" : ObjectId("562bd9b15617eb3da85845f1"), "id" : "102", "age" : "5", "category" : "Eating", "milestonecompleted" : "4" }
{ "_id" : ObjectId("562bd9b45617eb3da85845f2"), "id" : "103", "age" : "6", "category" : "Walking", "milestonecompleted" : "9" }
{ "_id" : ObjectId("562bd9b65617eb3da85845f3"), "id" : "104", "age" : "6", "category" : "Walking", "milestonecompleted" : "7" }
{ "_id" : ObjectId("562bd9b95617eb3da85845f4"), "id" : "105", "age" : "5", "category" : "Eating", "milestonecompleted" : "9" }
{ "_id" : ObjectId("562bd9bb5617eb3da85845f5"), "id" : "106", "age" : "5", "category" : "Eating", "milestonecompleted" : "6" }
{ "_id" : ObjectId("562bd9be5617eb3da85845f6"), "id" : "107", "age" : "5", "category" : "Eating", "milestonecompleted" : "5" }
{ "_id" : ObjectId("562bd9c15617eb3da85845f7"), "id" : "108", "age" : "5", "category" : "Eating", "milestonecompleted" : "1" }
{ "_id" : ObjectId("562bd9c45617eb3da85845f8"), "id" : "109", "age" : "5", "category" : "Eating", "milestonecompleted" : "6" }
{ "_id" : ObjectId("562bd9c85617eb3da85845f9"), "id" : "110", "age" : "5", "category" : "Eating", "milestonecompleted" : "3" }
```

6.5. Advantages and Disadvantages Of MongoDB

Advantages

Sharding and Load-Balancing:

It is a procedure of putting away information records over different machines and is mongodb way to deal with taking care of the requests of information development. As the size of the information builds, a solitary machine may not be adequate to store the information nor give a worthy read and write throughput. Sharding tackles the issue with flat scaling. With sharding, you add more machines to bolster information development and the requests of read and compose operations.

When you have amazingly a lot of information or you have to disperse your database activity over numerous machines for load-adjusting purposes, mongodb has substantial points of interest over numerous social databases, for example, MySQL.

Speed

Mongo DB queries can be much quicker in some scenarios, particularly since your information is regularly all in one data store and can be extracted in a single scan. In any case, it will only exist when your information is genuinely a document. At the point when your information is basically imitating a social model, your code winds up performing numerous autonomous queries keeping in mind the end goal to recover a solitary record and can turn out to be much slower than an excellent RDBMS.

Flexibility
Mongodb doesn't require a single information structure over all articles, so when it is impractical to guarantee that your information will all be organized reliably, mongodb can be much less complex to use than a RDBMS. Then again, information consistency is something worthy_requirement, so when ever possible you ought to dependably endeavor to guarantee that a bound single structure will be used.

Disadvantages

No Joins
While using mongodb there exists no plausibility of joins like in a RDBMS. This implies when you require this sort of requirement; you have to make numerous scans and join the information manually inside of your code.

Memory Usage
Mongo-DB has the an inclination to use more memory in light of the fact that it needs to store the key names inside of every json document. This is because of the way that the information structure is not consistent amongst the information objects.
Moreover there is duplicate information since there is no probability for joins, or slow queries because of the need to perform the join inside of your code. To tackle the issue of duplicates information in mongo, we can store references to documents, however in the event that you wind up doing this it demonstrates that the information is really "relational", and maybe better use a RDMS.

Concurrency Issues
When you execute a write operation in mongodb, it makes a lock on the whole database, not only the used collections, and not only for a specific connection. This lock blocks other writes operations, as well as read operations.

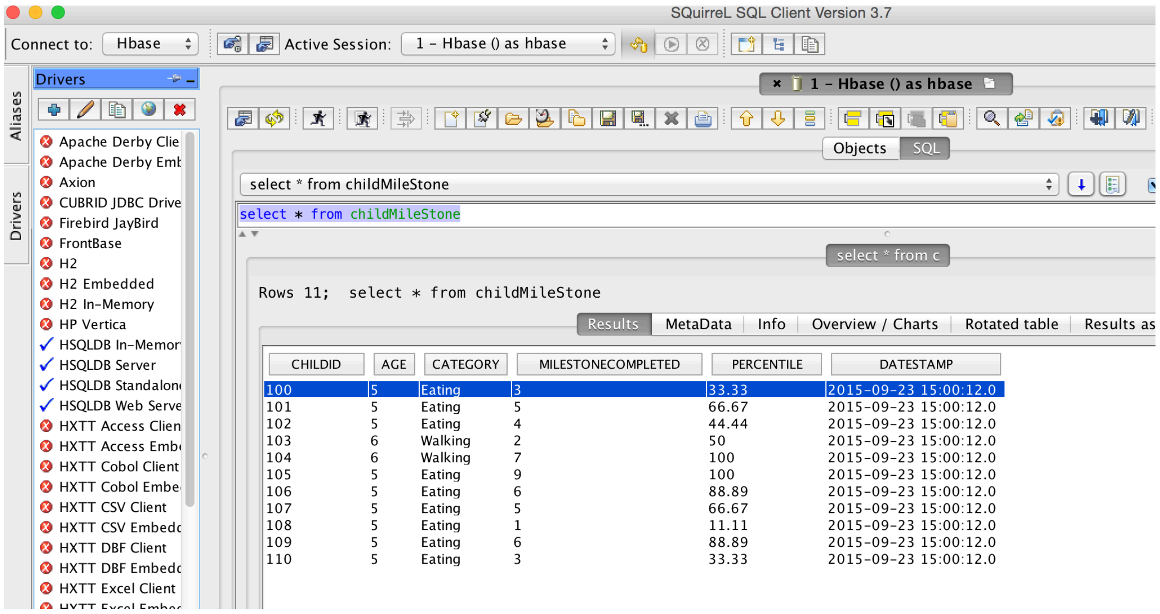
Transactions
Mongo DB doesn't naturally regard operations as transactions. So as to guarantee information integrity upon creates or update you need to choose make a transaction, physically check it, and afterward physically confer or rollback.

6.6. Results

<u>Data Stored in HBASE (View Layer)</u>

We load the data in hbase after running the mongodb mapreduce as discussed in section 6.3. After running the speed and batch jobs we have data in hbase so we can now run query on hbase (Serving Layer) to calculate the percentile.

We are using squirrel with apache phoenix to connect to hbase and to query it to get percentile for a given input. We are analyzed the performance of mongodb mapreduce at batch layer and speed layer i.e. time it took to calculate the percentile for the records and store them in hbase (serving layer) after map-reduce operations. For measuring the performance we check the records inserted w.r.t. time in seconds, as we can see in figure 26 X-axis shows the no. of records we inserted from batch layer into the serving layer (hbase). Y-axis shows the no of seconds it took to calculate percentile and insert them. First graph shows the performance for batch layer whereas the second graph shows the performance for the speed layer. We calculated these results on virtual machine.

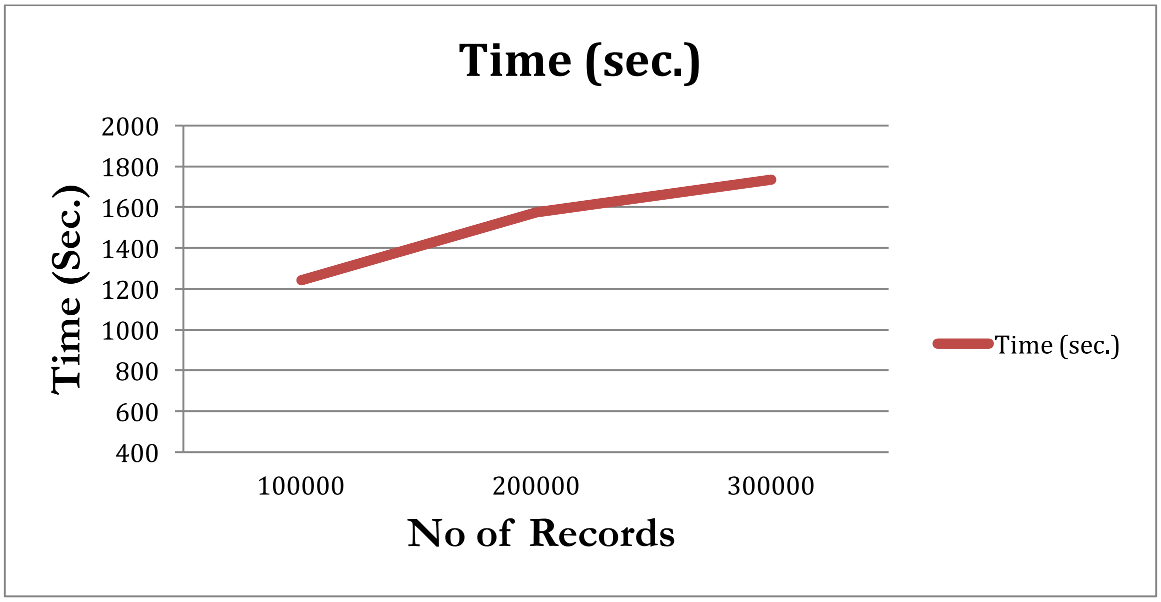


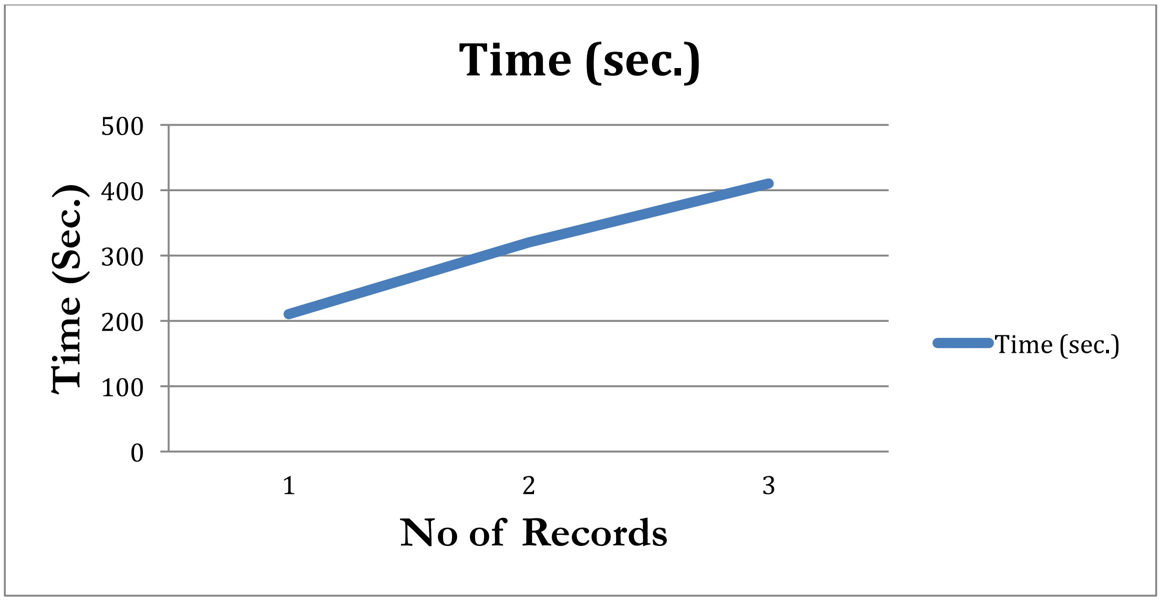
```
select * from childMileStone
```

Rows 11; select * from childMileStone

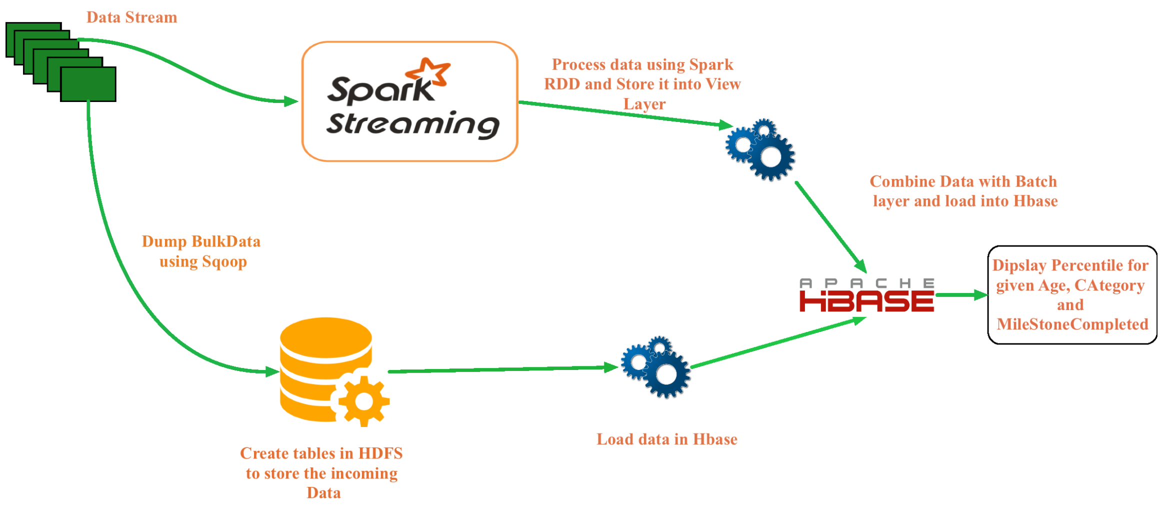
CHILDID	AGE	CATEGORY	MILESTONECOMPLETED	PERCENTILE	DATESTAMP
100	5	Eating	3	33.33	2015-09-23 15:00:12.0
101	5	Eating	5	66.67	2015-09-23 15:00:12.0
102	5	Eating	4	44.44	2015-09-23 15:00:12.0
103	6	Walking	2	50	2015-09-23 15:00:12.0
104	6	Walking	7	100	2015-09-23 15:00:12.0
105	5	Eating	9	100	2015-09-23 15:00:12.0
106	5	Eating	6	88.89	2015-09-23 15:00:12.0
107	5	Eating	5	66.67	2015-09-23 15:00:12.0
108	5	Eating	1	11.11	2015-09-23 15:00:12.0
109	5	Eating	6	88.89	2015-09-23 15:00:12.0
110	5	Eating	3	33.33	2015-09-23 15:00:12.0

Figure 26: Mongo DB Batch Layer Performance using Virtual Machine

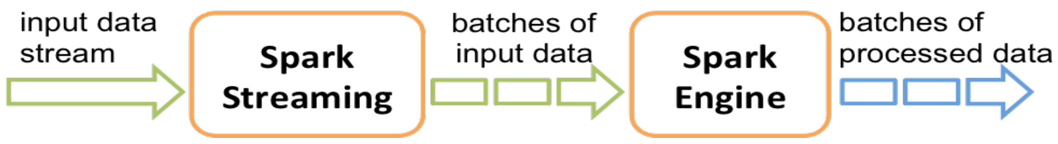




6.7 USING APACHE SPARK AT SPEED LAYER


Data Stream
Spark Streaming
Process data using Spark RDD and Store it into View Layer
Combine Data with Batch layer and load into Hbase
Dump BulkData using Sqoop
APACHE HBASE
Dipslay Percentile for given Age, CAtegory and MileStoneCompleted
Create tables in HDFS to store the incoming Data
Load data in Hbase

In this implementation we use apache spark at speed layer, apache spark is an open source fast and general engine for large-scale data processing.



Internally, it works as follows. Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches. Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

Resilient Distributed Datasets

Resilient Distributed Dataset (based on Matei's research paper) or RDD is the core concept in Spark framework. Think about RDD as a table in a database. It can hold any type of data. Spark stores data in RDD on different partitions. They help with rearranging the computations and optimizing the data processing. They are also fault tolerance because an RDD know how to recreate and recompute the datasets. RDDs are immutable. You can modify an RDD with a transformation but the transformation returns you a new

RDD whereas the original RDD remains the same.

RDD supports two types of operations:
Transformation: Transformations don't return a single value, they return a new RDD. Nothing gets evaluated when you call a Transformation function; it just takes an RDD and returns a new RDD. Some of the Transformation functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce.

Action: Action operation evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned. Some of the Action operations are reduce, collect, count, first, take, countByKey, and foreach.

6.8. SpringXD vs Apache Streaming

- Spark is an in-memory data-processing platform that is compatible with Hadoop data sources but runs much faster than Hadoop MapReduce whereas Spring XD framework supports streams for the ingestion of event driven data from a source to a sink that passes through any number of processors. Spring Integration adapters back the streams.
- Spring-Xd process data in real-time whereas Spark Streaming is actually processing in short interval batches.
- Spring XD works great when you don't want to do much coding and work with streams coming to and from RabbitMQ, Gemfire, JDBC, HDFS, Redis, MongoDB Whereas in Spark Streaming you have to write your own code to generate and process stream.
- Spring XD provides integration with Spark streaming to get the best out of both whereas the vice-versa is not true.
- SpringXd provides a variety of sources, sinks , operators etc to work with whereas in spark we have limited options.
- Easy to maintain code in Springxd whereas in Spark a small code change needs a lot of work.

6.9. Challenges

The biggest challenge in the project was to understand the technologies and the dependency between them. Since, there is no given document or proper examples how to implement lambda architecture. As we built a new system consisting of many different tools and open source technologies, I faced a lot of issues just to make them work with each other in the system. Many times the virtual machine on which I was building the system would crash without any reasons, and this happened couple of times. Every time after crash, I repeated the process of rebuilding the system from scratch. Understanding how networking works was essential and challenging since many of these open-source technologies uses different ports and IPs to communicate. Many of them uses the same ports to communicate so It was really painful to change the settings for all of them and then remembering the correct ports to work with them.

Coding was also a challenge since many of them uses different dependent jars and components to communicate with the framework. So finding the right version of components and jars was difficult. During this project I have used 10 different virtual machines to setup the correct environment for the architecture to work. At last migrating this system to amazon EC2 was a big challenge since amazon EC2 has different security features which disables a lot of outside access to the machine.

6.10. Conclusion and Future Work

It would be a misrepresentation to say the quantity, range and speed of the information was predictable, however it was conceivable. The issue confronting business was the way to extend their information distribution centers to suit these new prerequisites and to give the required backing to business insight and, in a bigger sense, analytics. Nowadays, the issue is altogether different. Information sources are eccentric, multi-organized and huge. The procedures for mining information from these sources, and even the stages most proper for doing as such, are now somewhat in question. With the introduction of hadoop to the business sector – a methodology that is isolated from the DW center – the issue confronting designers today is the place and when to convey these advances for performing valuable investigation.

We understood the data for the problem statement and successfully implemented the lambda architecture. We got a chance to explore a lot of different open-source technologies. There are no good examples of lambda architecture currently present to understand the working of different layers, but we managed to build a system, which had all the different layers present in lambda architecture. We built a lambda framework to solve the problem statement of percentile. We also measured the performance of each system. Future works include exploring many different technologies, which can help built lambda architecture. We can take many available problem statements on the net and try building efficient lambda architecture for them.

LIST OF REFERENCES

1. Carlos Queiroz, Lambda Architecture with Spring XD http://www.slideshare.net/SpringCentral/spring-one2gx-2014carlosqueiroz

2. Michael Hausenblas & Nathan Bijnens, inspired by Nathan Marz. Lambda architecture overview, http://lambda-architecture.net

3. Marz, N. (2013). Big Data: Principles and best practices of scalable real-time data systems. O'Reilly Media.

4. DEBS challenge 2014, http://www.cse.iitb.ac.in/debs2014/?page_id=42

5. Mark Fisher, Mark Pollack and David Turanski, spring XD guide, http://docs.spring.io/spring-xd/docs/current-SNAPSHOT/reference/html

6. Fred Melo, Spring XD: The Foundation for Real-time Streaming http://blog.pivotal.io/big-data-pivotal/products/spring-xd-the-foundation-for-real-time-streaming-and-machine-learning-systems

7. Pivotal Software, Inc, demo for DEBS 2014 challenge https://github.com/pivotalsoftware/gfxd-demo

8. Apache Foundation. Storm, distributed and fault-tolerant realtime computation. https:// storm.incubator.apache.org, Retrieved 2014-08-14

9. Pere Ferrera Bertran, lambda architecture for real-time analysis of hashtags, https://github.com/pereferrera/trident-lambda-splout

10. Pivotal Software, Inc. Spring-XD ecosystem, http://docs.spring.io/spring-xd/docs/current/reference/html/

11. Tony S, Lambda Architecture for Big Data, https://tsicilian.wordpress.com/2015/01/05/lambda-architecture-for-big-data/

12. Michael Hausenblas, MapR explains Lambda Architecture https://www.mapr.com/developercentral/lambda-architecture

13. Jack Vaughan, Big data architecture took on wider processing role in 2014, http://searchdatamanagement.techtarget.com/feature/Big-data-architecture-took-on-wider-processing-role-in-2014

14. James kinley, Principles for architecting realtime Big Data systems, http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for

15. Jeffrey Wang, Scaling Analytics at Amplitude, https://amplitude.com/blog/2015/08/25/scaling-analytics-at-amplitude/

16. John Piekos, Simplifying the (complex) Lambda architecture, https://voltdb.com/blog/simplifying-complex-lambda-architecture

17. O.S. Tezer, How To Install and Manage RabbitMQ, https://www.digitalocean.com/community/tutorials/how-to-install-and-manage-rabbitmq