**CSE 587**

**Data Intensive Computing**

**Homework 4**

# Stock Volatility Computation: HBase

Vidyadhar Reddy Annapureddy

vannapur@buffalo.edu (50134358)

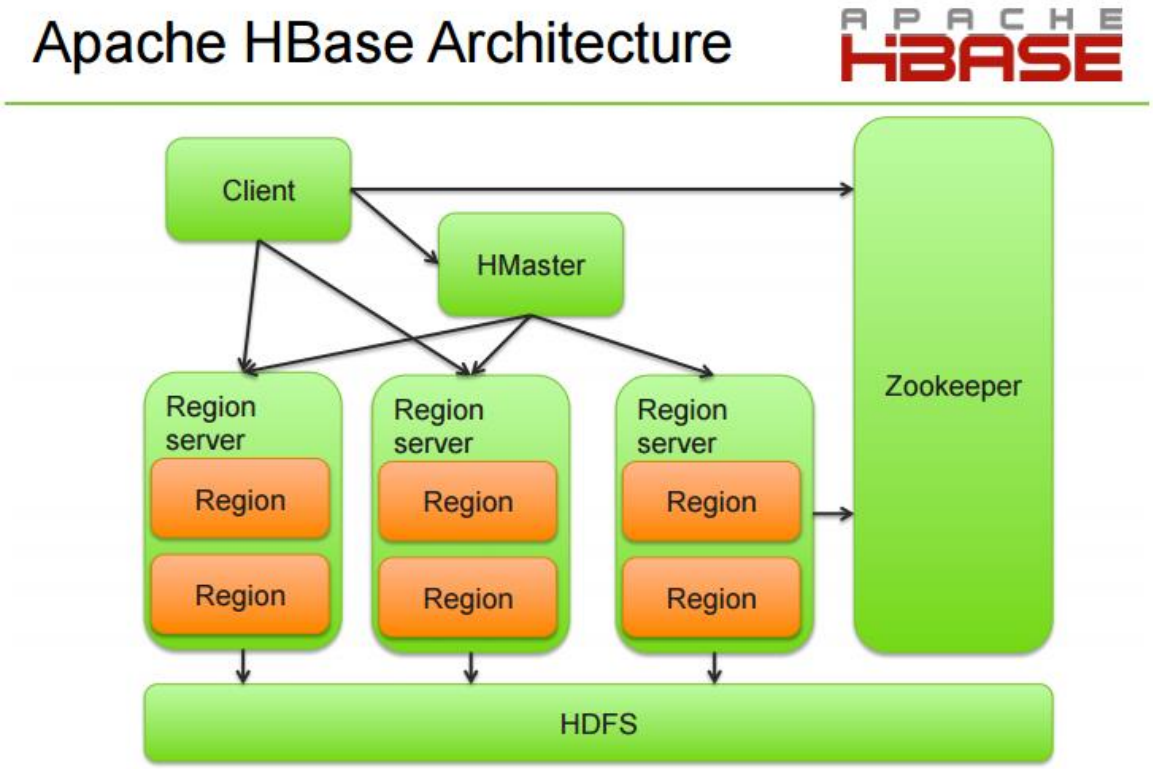
# HBASE and Implementation Overview

HBase is a data model that is designed to provide quick random access to huge amounts of structured data. Its architecture promises linear scalability. Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API. HBase is a distributed, scalable, big data store, modelled after Google's BigTable. It stores data as key/value pairs. MapReduce is just a computing framework. HBase has nothing to do with it. That said, we can put or fetch data from HBase by writing MapReduce jobs and use sequential or MapReduce jobs for further computations.

It's basically a NoSQL database and it provides us random read/write capabilities. HBase is typically used when we need random, real time read/write access to big data. Its goal is hosting of very large tables – '*raw*' table with data for up to 29,700 stocks spanning over 3 years. The project doesn't involve too many random lookups and the implementation uses HBASE primarily for loading data from csv records into HBase tables in HDFS. So, the reduction in execution time observed on various nodes is attributed to the mapper job distributed across slaves, thereby yielding in lower execution time for more number of nodes.

The implementation exploits the following key features of HBase:

- Linear and modular scalability.
- Consistent reads and writes.
- Automatic and configurable sharding of tables
- Easy-to-use Java API for client access.



# Stock Volatility Index Results with HBase

## i. Small dataset

| Top 10 stocks with Min Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| LDRI | 5.149336582098077E-4 |
| GAINO | 5.650074160499658E-4 |
| VGSH | 0.0013014906189562883 |
| MBSD | 0.0025000459104618897 |
| TRTLU | 0.003478105118698644 |
| AGZD | 0.003938593878697365 |
| SKOR | 0.003948740216629902 |
| CADT | 0.004156636196742422 |
| AXPWW | 0.0044388372955839524 |
| VCSH | 0.004637760185632481 |

| Top 10 stocks with Max Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| ACST | 9.271589761859984 |
| NETE | 5.396253961502245 |
| XGTI | 4.5423443114729585 |
| TNXP | 3.2483321967818672 |
| EGLE | 3.0222065379013148 |
| PTCT | 1.8462537015817713 |
| GOGO | 1.779342175186138 |
| MEILW | 1.7188134065765308 |
| ROIQW | 1.396532083959149 |
| CFRXZ | 1.0792682449689408 |

## ii. Medium dataset

| Top 10 stocks with Min Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| LDRI-3 | 5.149336582098077E-4 |
| LDRI-3 | 5.149336582098077E-4 |
| LDRI-3 | 5.149336582098077E-4 |
| GAINO-3 | 5.650074160499658E-4 |
| GAINO-3 | 5.650074160499658E-4 |
| GAINO-3 | 5.650074160499658E-4 |
| VGSH-2 | 0.0013014906189562881 |
| VGSH-3 | 0.0013014906189562881 |
| VGSH-1 | 0.0013014906189562881 |
| MBSD-3 | 0.0025000459104618893 |

| Top 10 stocks with Max Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| ACST-2 | 9.271589761859984 |
| ACST-2 | 9.271589761859984 |
| ACST-1 | 9.271589761859984 |
| NETE-3 | 5.396253961502245 |
| NETE-3 | 5.396253961502245 |
| NETE-3 | 5.396253961502245 |
| XGTI-1 | 4.5423443114729585 |
| XGTI-2 | 4.5423443114729585 |
| XGTI-2 | 4.5423443114729585 |
| TNXP-2 | 3.2483321967818672 |

## iii. Large dataset

| Top 10 stocks Min Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| LDRI-9 | 0.000514933658210 |
| LDRI-3 | 0.000514933658210 |
| LDRI-1 | 0.000514933658210 |
| LDRI-10 | 0.000514933658210 |
| LDRI-8 | 0.000514933658210 |
| LDRI-2 | 0.000514933658210 |
| LDRI-7 | 0.000514933658210 |
| LDRI-5 | 0.000514933658210 |
| LDRI-6 | 0.000514933658210 |
| LDRI-4 | 0.000514933658210 |

| Top 10 stocks Max Volatility | |
|---|---|
| **Stock** | **Volatility Index** |
| ACST-7 | 9.271589761859980 |
| ACST-6 | 9.271589761859980 |
| ACST-1 | 9.271589761859980 |
| ACST-2 | 9.271589761859980 |
| ACST-4 | 9.271589761859980 |
| ACST-5 | 9.271589761859980 |
| ACST-9 | 9.271589761859980 |
| ACST-8 | 9.271589761859980 |
| ACST-3 | 9.271589761859980 |
| ACST-10 | 9.271589761859980 |

## Execution Time: HBase

| | Small dataset | | | | Medium dataset | | | | Large dataset | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Nodes** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** |
| Tasks per node | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| **Cores** | 12 | 24 | 36 | 48 | 12 | 24 | 36 | 48 | 12 | 24 | 36 | 48 |
| **Time** (sec) | 3193 | 1438 | 1243 | 1127 | 8392 | 3779 | 3248 | 2962 | 28122 | 12665 | 9948 | 9426 |

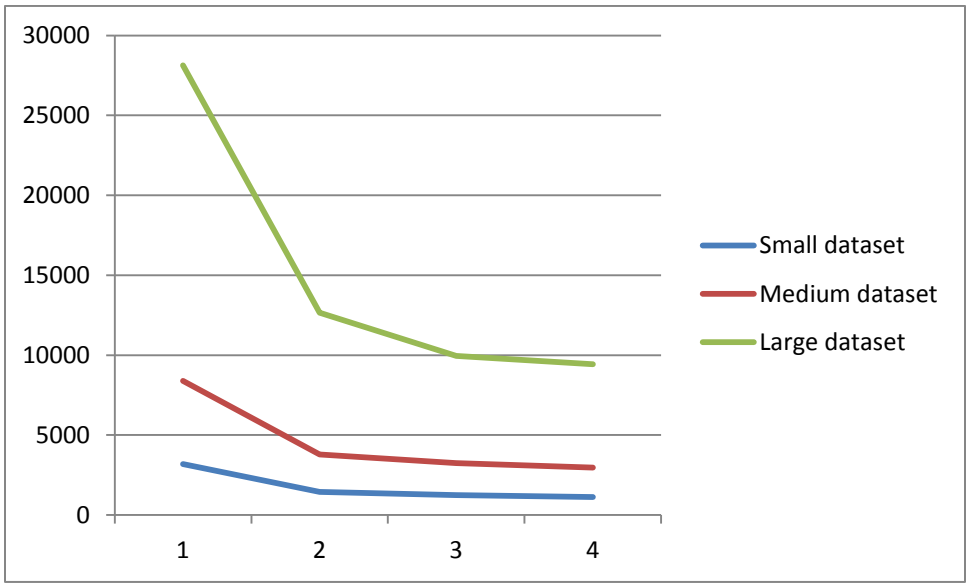



**Figure1: execution time (in sec) vs no. of nodes**

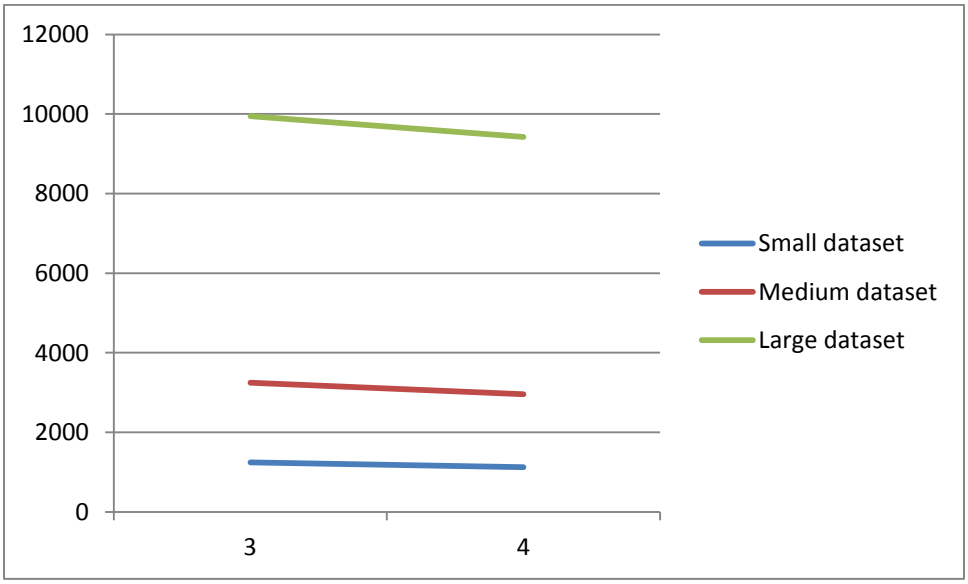



**Figure2: Execution time (in sec) vs no. of nodes**

### A. HBase vs PIG

Pig is a dataflow language that allows you to process enormous amounts of data very easily and quickly by repeatedly transforming it in steps. PigLatin queries get converted into a native MapReduce jobs and yield the result. Pig can be used for data stored in HDFS and HBase. It saves a lot of your effort and time by allowing you to not write MapReduce programs and do the operation through straightforward Pig queries. In my understanding, Pig is better utilized when a lot of transformations are to be performed on data and writing JAVA code is to be avoided.

- **Execution time**: PIG gives better performance than HBase **(~5.4x slower than Pig**)
- **Ease of programming:** PigLatin is much easier to code than typical JAVA MapReduce Jobs. Ratio of lines of code **10:1** in favor of PigLatin. *The time and effort spent in coding MapReduce jobs has not been considered.*

### B. HBase vs HIVE

Hive is basically a **data warehouse**. It resides on Hadoop cluster to give an SQL like interface to the data stored in HDFS. Queries including store, select, join are written in HiveQL. It makes processing easier hundreds of lines of JAVA code gets condensed to mere 20-30 lines in HiveQL. We can map data on HDFS to Hive tables and process the data. Hive is best utilized for warehousing purposes. Hive queries are converted to MapReduce jobs. HBase can be seen as a map – if we know the key, we can instantly get the value. When we have data to be aggregated, rolled up, analyzed across rows then we need to consider alternatives like MapReduce or Pig or Hive. We could use Hadoop as the repository for your static data and HBase as the datastore which will hold data that is probably going to change over time with processing.

- **Execution time**: HIVE gives better performance than HBase (~**4.4x slower than HIVE**)
- **Ease of programming**: HiveQL is much easier to code than typical JAVA MapReduce Jobs for HBase. Ratio of lines of code 10:1 in favor of HIVE. *The time and effort spent in coding MapReduce jobs has not been considered*

### C. HBase vs MapReduce

Hadoop constitutes a storage layer and a distributed computation framework, MapReduce**,** which constitutes the processing layer. Hadoop MapReduce should be preferred if data is huge and we have offline, batch processing needs. Hadoop is not suitable for real timecomputation. Data is stored into the HDFS and we process this data one or more MapReduce jobs. Being distributed, HDFS is spread across all the machines in a cluster and MapReduce processes this scattered data locally by going to each machine, so that we don't have to move the big data.

- **Execution time**: HBase gives almost the same performance as MapReduce (~**1.07x faster than MapReduce**)
- **Ease of programming**: Almost the same amount of time and effort is required to write code for Mapper-Reducer jobs for HBase. However, in this case, the implementation uses a sequential code to compute volatility of stocks, which is easier to code than writing multiple mapper and reducer jobs and storing intermediate data into HBase tables to provide access to data to the consequent processes.