

Enhancing Query Support in HBase via an Extended Coprocessor Framework

Himanshu Vashishtha, Eleni Stroulia

Department of Computing Science,
University of Alberta,
Canada

([hvashish](mailto:hvashish@ualberta.ca), [stroulia](mailto:stroulia@ualberta.ca))@ualberta.ca

Outline

1. Motivation
2. Background: Hadoop & HDFS
3. HBase
4. Coprocessors
 - Aggregate functions
 - Streaming Results
5. Evaluation
6. Conclusion

Motivation:

Unprecedented Data Growth

- More people, worldwide, have Internet access
 - 2000: 360m
 - 2011: 2 billion (1/3 of earth population)
- Facebook, in 2009 uploading 60 TB images every week
- A jet engine, produces 10 TB data every 30 min of its flight

- Most of this data is “unstructured” such as comments, tweets, web-server logs, images: “BigData”

How Google solved their scalability problem

- Many workflows are data parallel
- Designed and implemented ***MapReduce [1]***, ***Google File System [2]***, and ***BigTable [3]*** to support PBs of data for its core services
- Based on “Single-Master-multiple-slaves” model
- Deployed in clustered environments (100s of nodes) made of commodity machines

The MapReduce Paradigm [1]

OSDI paper, in 2004

- Inspired by map and reduce functions used in functional programming:
Input → Map() → Copy/Sort → Reduce() → Output
- Map *streams-in* raw input and generates intermediate key-value pairs
 - Map phase is data parallel
- Library takes care of RPC, job scheduling, data-locality, and fault tolerance
- **Hadoop: Apache open source equivalent**

The Google File System [2]

SOSP paper, in 2003

- Distributed file system to support MapReduce jobs
- Relaxed POSIX, specially to support stream-in of large datasets (up to GBs or more) to the map phase
- Fault tolerance by replication
- Sequential reads of large data; random reads of small data (a few KBs)
- Write once, read multiple times
- **HDFS: Apache open source equivalent**

The BigTable[3]

OSDI in 2006

- A distributed, 3-D table data structure
 - With time as the third dimension
- Rows sorted based on a primary key
- Supports update, random reads, real-time querying
- Stores its data on GFS

- **HBase: Apache open source equivalent**

- BUT, not ALL companies have that infrastructure to support clusters spanning 100s of nodes?
- Use CLOUD COMPUTING!!!
 - Amazon, RackSpace, SalesForce, etc

Apache Projects mappings

- MapReduce → Hadoop
 - Master : JobTracker
 - Slave : TaskTracker
- Google File System → Hadoop Distributed File System (HDFS)
 - Master : NameNode
 - ChunkServer : DataNodes
- BigTable → HBase
 - Master : HMaster
 - TabletServer : RegionServer

What is HBase?

- A database that is:
 - Distributed
 - Sorted (in terms of their primary key)
 - Column oriented
 - Sparse
 - Multi dimensional
 - Highly Available
 - Scalable (millions of columns and billion of rows)
- No Joins, no SQL, no indexes, no DB objects, only row level transactions
- Sits on top of HDFS

HBase Architecture

- Logical View:
 - A table is divided into Regions
 - Region is described by its startkey and endkey:
 - [User, “foobar101”, “foobar201”)
 - A Region is made up of several HDFS files and blocks, and can stay at different nodes (depending upon underlying HDFS)
- Physical View:
 - Catalog Tables: -ROOT-, .META.
 - -ROOT- is fixed, .META. can move/split
 - Store Region location, schema information
- Master-Slave -> MasterServer- RegionServer
 - MasterServer administers cluster: allocates regions, balances load
 - RegionServer hosts multi Regions

HBase Table

- Sorted by RowKey
- Table has ≥ 1 “column families”.
- A column family is
 - A group of column qualifiers (defined at run time)
 - Stored as one file in HDFS
- Sparse tables are supported. No cost of NULL cells
- Timestamp: 3rd dimension
- A cell is identified by
 - Table:Rowkey:CF:CQ:timestamp

HBase Logical View

	CF1	CF2
Row Key 1	RK1:CF1:CQ1:Value:t9	RK1:CF1:CQ2:Value:t4
Row Key 2	RK2:CF1:CQ1:Value:t3 RK2:CF1:CQ1:Value:t4 <div style="border: 1px solid black; padding: 2px; display: inline-block;">RK2:CF1:CQ1:Value:t5</div>	
Row Key 3	RK3:CF1:CQ1:Value:t1	RK3:CF2:CQ4:Value:t11

- RK1:CF1:CQ1:Value:t9 is one cell value

HBase APIs

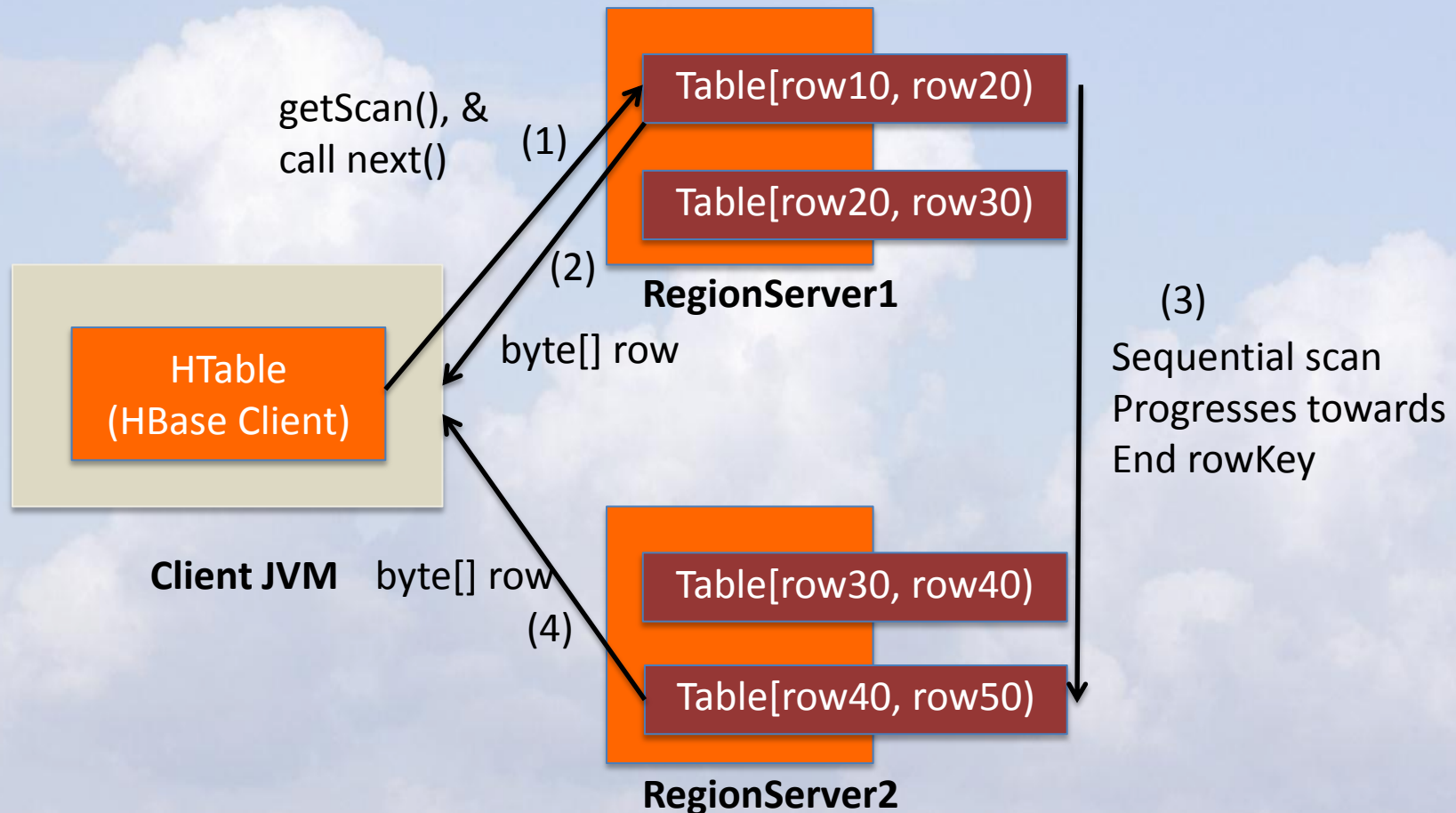
- There is a single data type: `byte[]`
- **Get** a specific row (given `rowKey`)
 - `get(byte[] row)`
- **Put** a new row
 - `put(byte[] row)`
- **Delete** a row
 - `Delete(byte[] row)`
- **Scan** a set of rows
 - For a given start rowkey and end rowkey, iterate over all the rows **sequentially**, and return them to the client

Coprocessors

- Inspired by BigTable coprocessors[4]
- Arbitrary code deployed at Regions
- Observers
 - Think about database triggers!
 - Every operation, such as get, put, or admin operations can have pre/post hooks
- Endpoints
 - Think about database stored procedures!
 - Uses a custom Coprocessor RPC protocol which supports parallel execution
 - Invocation by client side based on a row or a set of rows
 - Provides a client side method which helps in aggregating the Endpoint results per RPC batch

ENDPOINT USE CASE: ROW COUNT OF A TABLE

1. Scan: fetch rows and keep counting



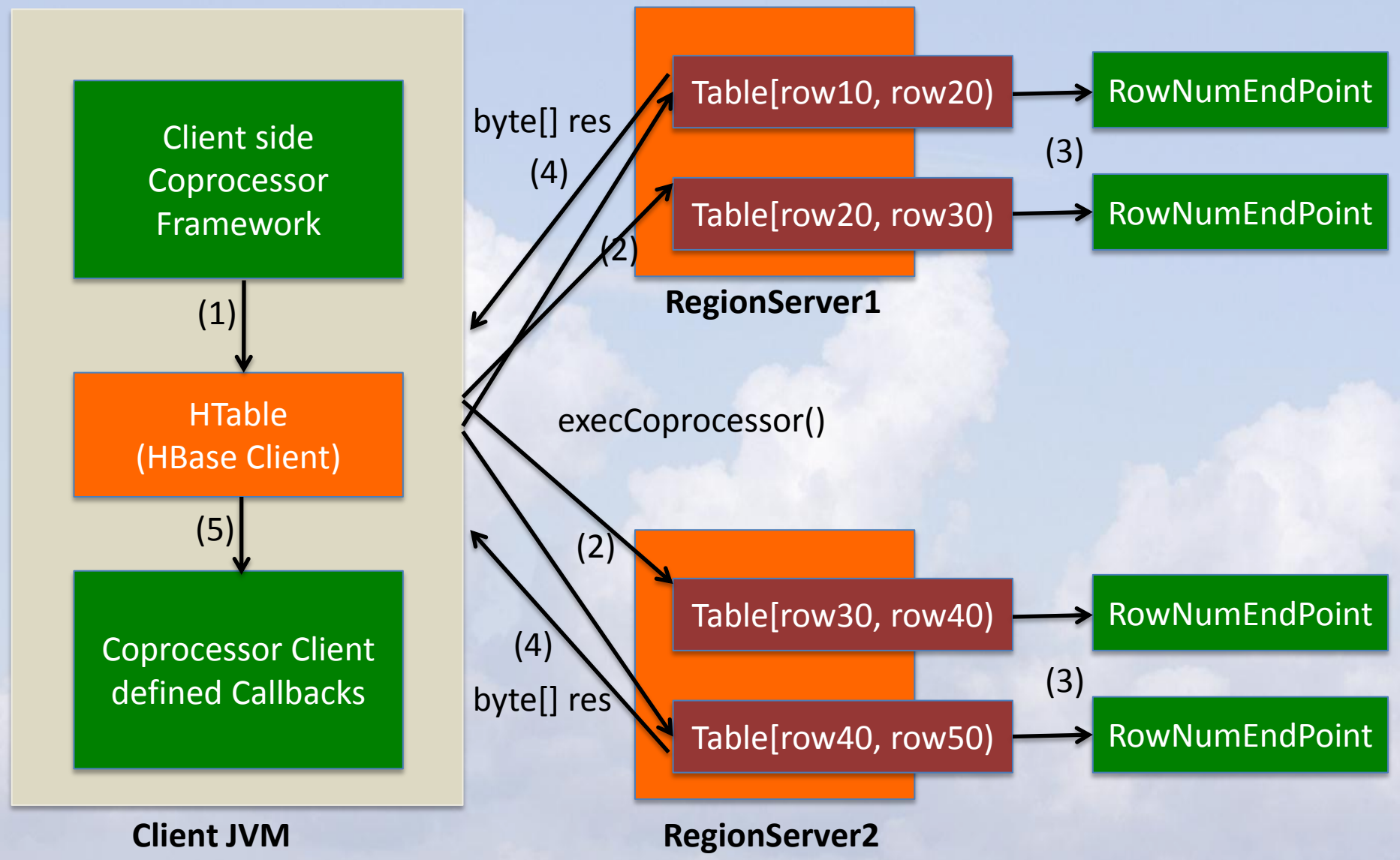
2. Run a MapReduce job

- Will run on the entire table, with no regard of start/end rowKey
- Good for batch processing, but not ideal for online querying

3. Coprocessor Endpoint

- Define an endpoint that computes rowCount at the Region level, and sends it to the Client side
- Client aggregates the above results from all Regions and computes the global sum

3. Coprocessor Endpoint



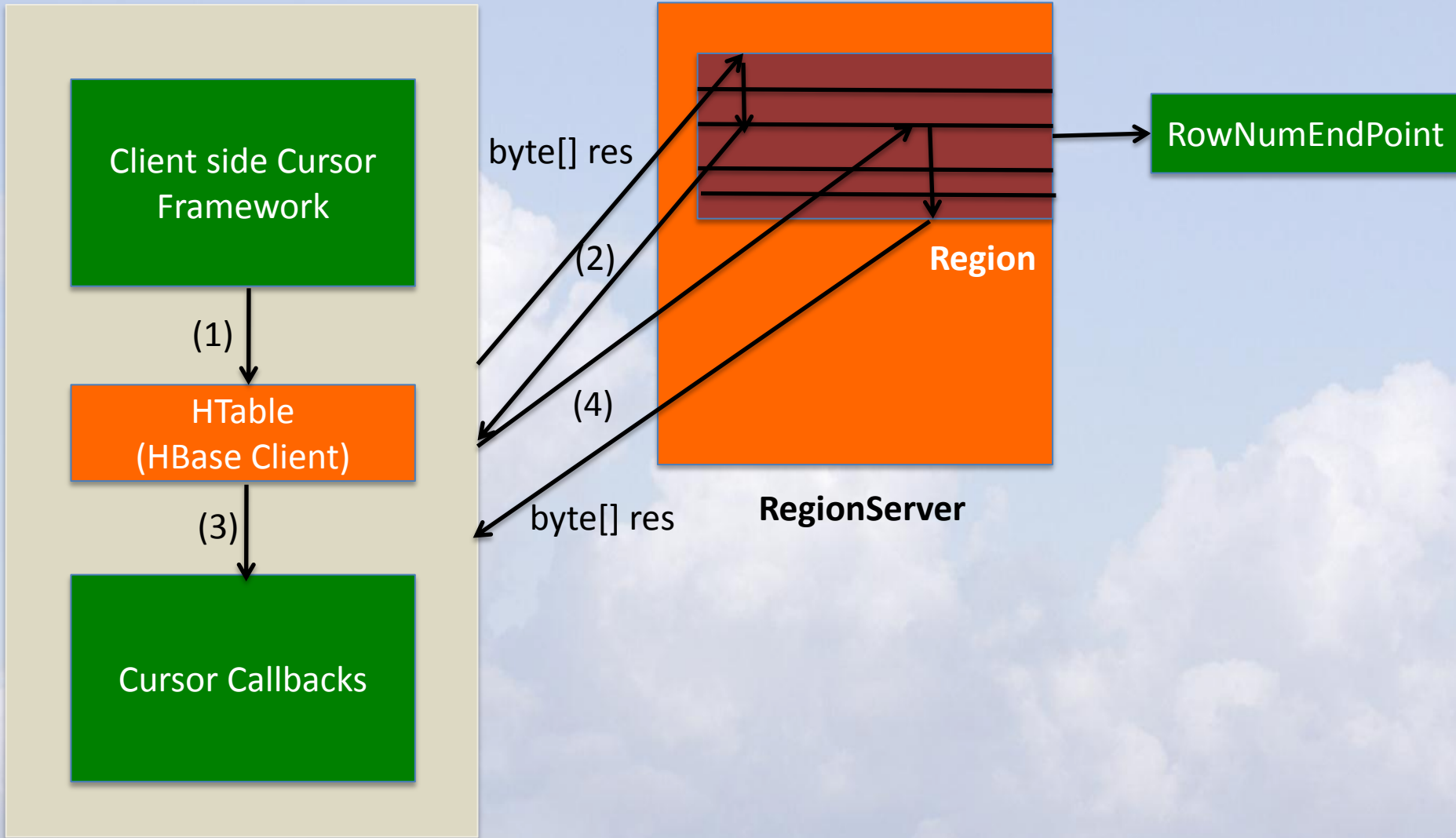
Aggregate Functions

- Designed and implemented six aggregate functions using Coprocessor Endpoints
 - Max, Min, Sum, RowCount, Standard deviation and Average
 - Generic enough to support any kind of data
 - Client provides an interpreter to parse byte[]
 - Committed to Apache HBase and will be part of 0.94.0 release!

Results Streaming

- Current limitation of Coprocessor RPC: a Region has to be processed in a single RPC
- Our extension: a Streaming functionality, based on Coprocessor RPCs: parallel execution
- Region Side
 - We create a Cursor object which comprises the business logic and support for iterator calls
 - Register it in the Coprocessor Environment
 - Associated key is given to the client
- Client side
 - A ClientCursor is defined (support iterator calls)
 - It hides the Region side Cursor handles
 - Null result -> Region exhausted
 - Cursor Callback to aggregate the results for each RPC batch

Results Streaming Flow



Evaluation

- Datasets Requirements
 - Large
 - Queries to do server side computations
 - Large results for Streaming Results
- Bixi dataset
 - Dataset by Public Bike System Inc, Montreal
 - Sensor based bike stations, emit data every minute
 - Data collected for 70 days (Sep 24 to Dec 1, 2010)
 - 404 stations in Montreal

Sample Bixi data & Schema

```

- <station>
  <id>1</id>
  <name>Notre Dame / Place Jacques Cartier</name>
  <terminalName>6001</terminalName>
  <lat>45.508183</lat>
  <long>-73.554094</long>
  <installed>true</installed>
  <locked>false</locked>
  <installDate>1276012920000</installDate>
  <removalDate/>
  <temporary>false</temporary>
  <nbBikes>4</nbBikes>
  <nbEmptyDocks>27</nbEmptyDocks>
</station>

```

RowKey	CF:1	CF:2	CF:3
Timestamp1	NotreDame, 45.508153, -73.554094, 4, 27	Saint-Antoine, 45.512323 , -73.5539304, 1, 24	Saint-Laurent, 45.51066, -73.56497, 3,12
Timestamp2	NotreDame, 45.508153, -73.554094, 3, 28	Saint-Antoine, 45.512323 , -73.5539304, 4, 21	Saint-Laurent, 45.51066, -73.56497, 0,15

- Bixi Queries
 - Q1: Given a time, lat/lon and a radius, find number of available bikes
 - Q2: Given a list of stations and a time, get average bike usage of last 1, 6, 12 and 18 hrs (3 vs 404 stations)
- One row per minute. Each row size \approx 90KB
- Q1 will access 1 row
- Q2 will access 60, 360, 720, 1080 rows respectively

Sample Ngram data & Schema

Word	Year	Word count	Unique pages	Unique books
America14	1936	1	1	1
America14	1938	5	5	5
...
Americaensche	2001	17	14	7
Americaensche	2002	11	11	9

RowKey	CF:19 36	CF:1 938	CF:20 01	CF:20 02
America14	1,1,1	5,5,5		
Americaensche			17, 14,7	11, 11,9

- Ngram Queries
 - Q3: Given a word prefix, get the top 3 frequencies across all the years for each unique words
 - Q4: Do the same for a bag of words
- One row per word
- Q3 will access a set of contiguous rows
- Q4 will access sets of contiguous rows

Experimental Setup

- 5 node EC2 Cluster

Node no	Instance type	Processes
1	m1.xlarge	Namenode, JobTracker, Hmaster, TestClient
2	m1.large	Zookeeper
3	c1.xlarge	RegionServer, Datanode
4	c1.xlarge	RegionServer, Datanode
5	c1.xlarge	RegionServer, Datanode

- Created a HBase ami with Coprocessors
- Upload datasets on Amazon S3

Experiment Results : Q1

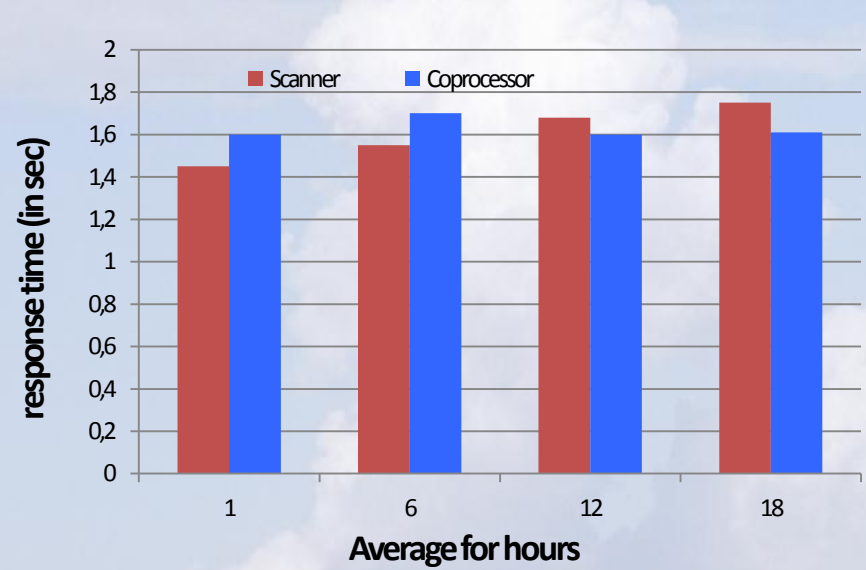
- Q1: Given a time, latt/long and a radius, find number of available bikes

Q1	Get API	Coprocessors
Response time (in sec)	1.57	1.61

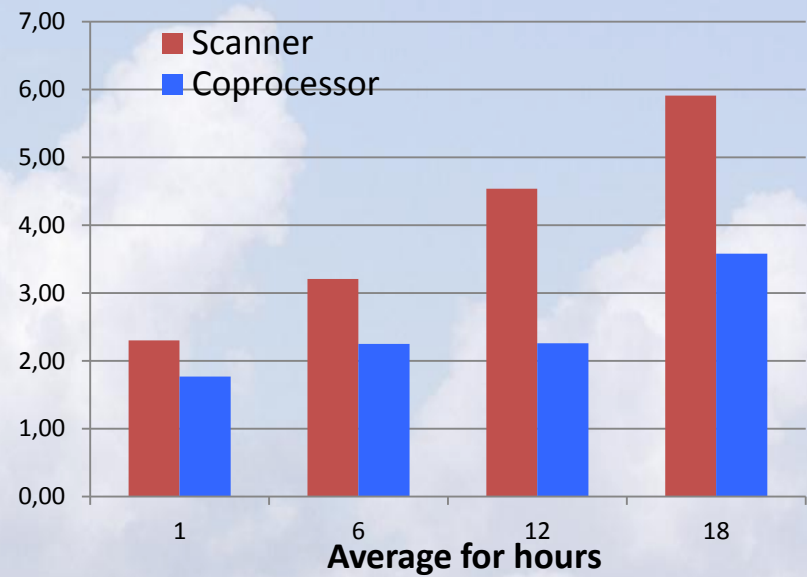
- Almost similar time
- Miniscule computation on 90 KB row

Experiments Result : Q2

- Q2: Given a list of stations and a time, get average bike usage of last 1, 6, 12 and 18 hrs



For 3 stations



For ALL 404 stations

Experiments Result : Q3/Q4

- Q3: Given a word prefix, get the top 3 frequencies across all the years for each unique words;
- Q4: Q3 with a bag of words

Prefix	Cache Size	Scanner time	Coprocessor time	Unique words
America	1000	1.84	1.66	424
America	100	1.84	1.80	424
A	1000	29.65	21.15	219,797
Blood, love, change, passion	1000	158.21	44.70	NA

(All time in seconds)

Contributions

1. Designed & implemented Streaming Results for Coprocessors
 - Demonstrated improved efficiency for large result sets
2. Designed & implemented standard aggregate functions
 - Improved support for development of applications with on-line queries
3. Code contributed to the Apache HBase
4. Migrated an existing text-analysis application, TAPoR to use HBase

Future Work

- Migrated an existing text-analysis application, TAPoR to use HBase
- Streaming results code cleanup/refactoring to make it compatible with open source standards
- Support for Parallel Scans using Streaming Results framework

References

- [1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [2] S. Ghemawat, H. Gobioff, and S. Leung, “The google file system,” in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.
- [3] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

Questions



Cloud Computing

(a very very scoped definition...)

- Rent `_your_` nodes (VMs) on the fly
- Use existing or create your own Machine Images
- Pay as you use

- Good for “experimentation”, varying infrastructure, from hardware to software
- Used for research, academia, & in industry

Aggregate functions benchmarking

- YCSB for data generation



Row Count	Scan	MapReduce	Coprocessors
1m	8.2	24	1.6
10m	240	118	100
100m	Not tested (>15min)	631	Failed



- 1m and 10m -> Coprocessors
- What's wrong with 100m?
 - 240 Regions on 3 RegionServers
 - Agg functions NOT use Streaming Results!
 - Client connection timed out (6 min)
 - Effectively, it tries to scan 100m in 6 min, else throws SocketTimeoutException!

Aggregate functions benchmarking

(contd...)

- MapReduce shows its power when data increases
 - Batch processing master
- Using Streaming results avoids `SocketTimeoutException`, as there is no hard limit of processing all Regions with in 6 minutes
- This work is up for review in Apache HBase
 - Needs some code cleanup to make it more generic

“In pioneer days they used oxen for heavy pulling, and when one ox could not budge a log, they did not try to grow a larger ox. We should not be trying for bigger computers, but for more systems of computers.”

– Grace Hopper