Distributed Analytical Processing of Network Data Streams in Real Time

GEORGIOS TOULOUPAS

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

NOVEMBER 2, 2015

Contents

- Introduction
- Theoretical Background
- System Description
- HBase and Phoenix Optimizations
- Evaluation
- Conclusion

Introduction

Motivation

The Internet is continuously growing, studies forecast that global Internet traffic will grow with an annual rate of 26% over the next years.

Large Internet Exchange Points have visibility to a large fraction of the Internet. Information about the global state of the Internet can be extracted by analyzing the traffic of a large IXP over a sufficient period of time.

The typical approach to perform network traffic analysis on a large IXP is by sampling the traffic over a period of time, saving the capture in a file and processing it in a centralized manner by a script.

A variety of distributed technologies have been developed for processing and storing real-time data streams.

Objectives

Design and implementation of a distributed system that allows the execution of SQL queries that join a real-time network data stream, generated by sampling IXP traffic, and external datasets containing Autonomous System and DNS information.

Prerequisites for the system:

- Execution of low latency real-time queries
- Scalability
- Fault tolerance
- Extensibility

Theoretical Background

Kafka

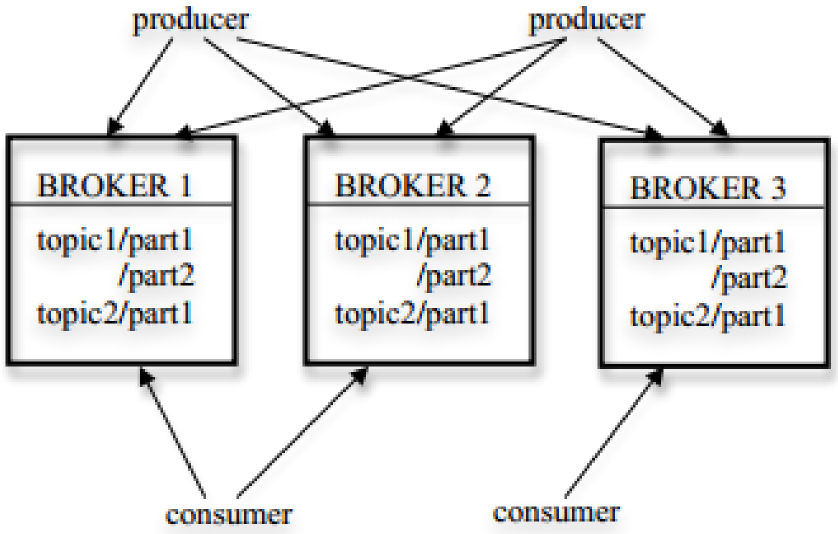
Kafka is a distributed messaging system, used for collecting and delivering high volumes of data with low latency.

Key features:

- Fast
- Scalable
- Fault-tolerant

Kafka Basic Concepts

- A **topic** defines a stream of messages of a particular type.
- A **producer** is a process that publishes messages to a topic.
- The published messages are stored at a cluster comprised of servers called **brokers**.
- A **consumer** is a process that subscribes to one or more topics and processes the feed of published messages.

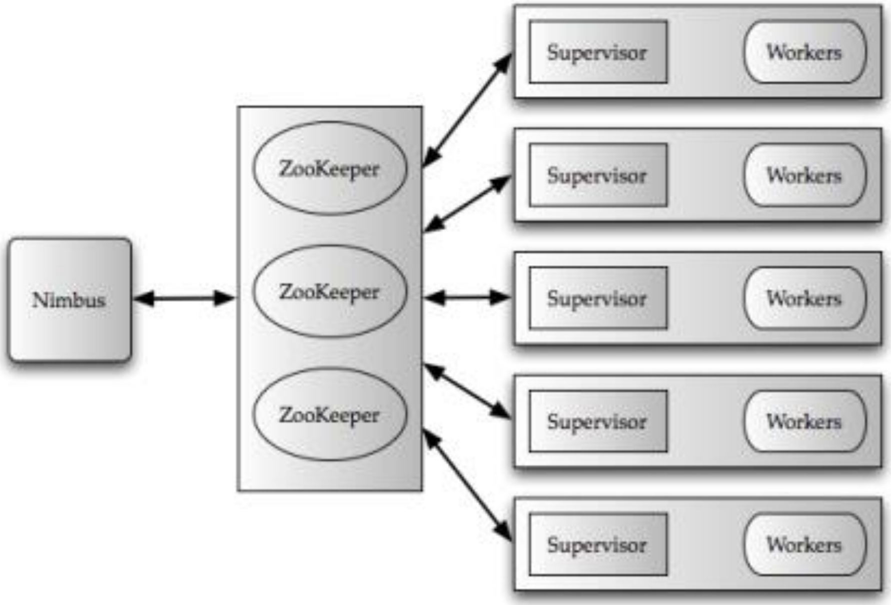


Storm

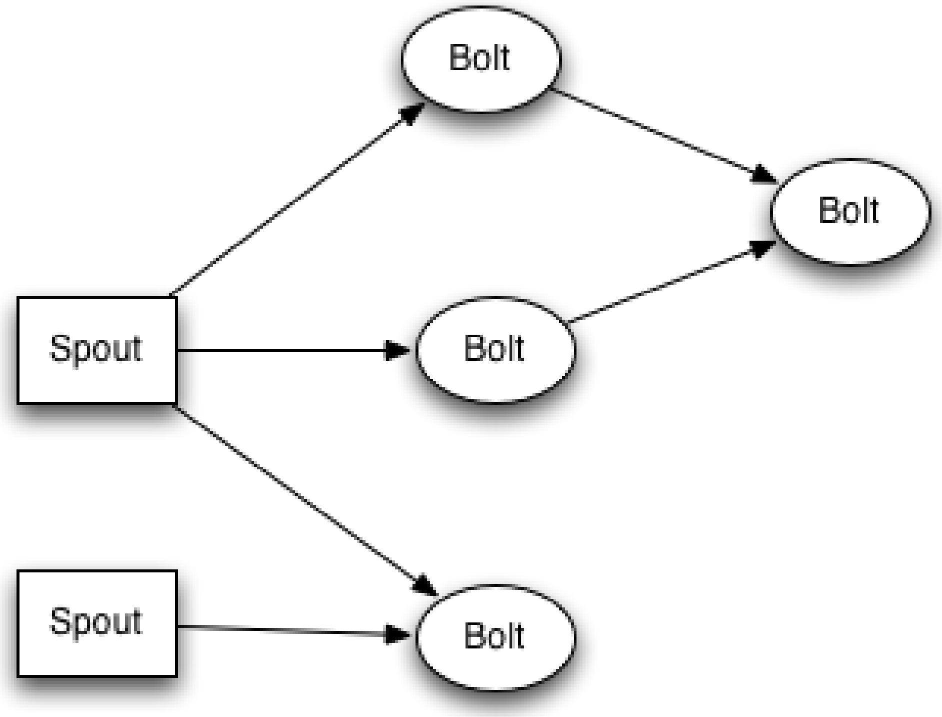
Storm is a distributed real-time computation system, used for processing data streams.

Key features:

- Scalable
- Fault-tolerant
- Efficient
- Reliable



Storm Topology

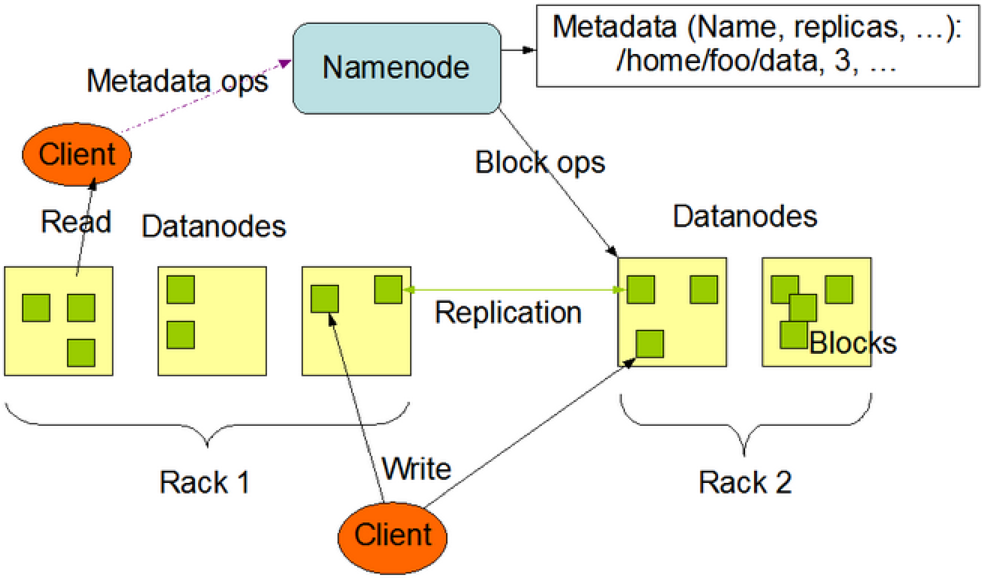


HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.

Key features:

- Scalable
- Fault-tolerant

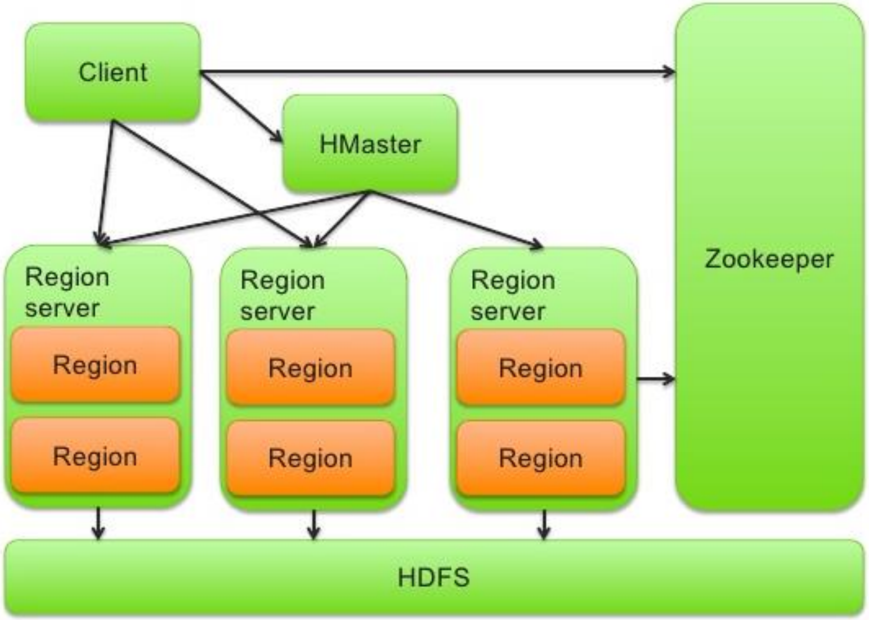


HBase

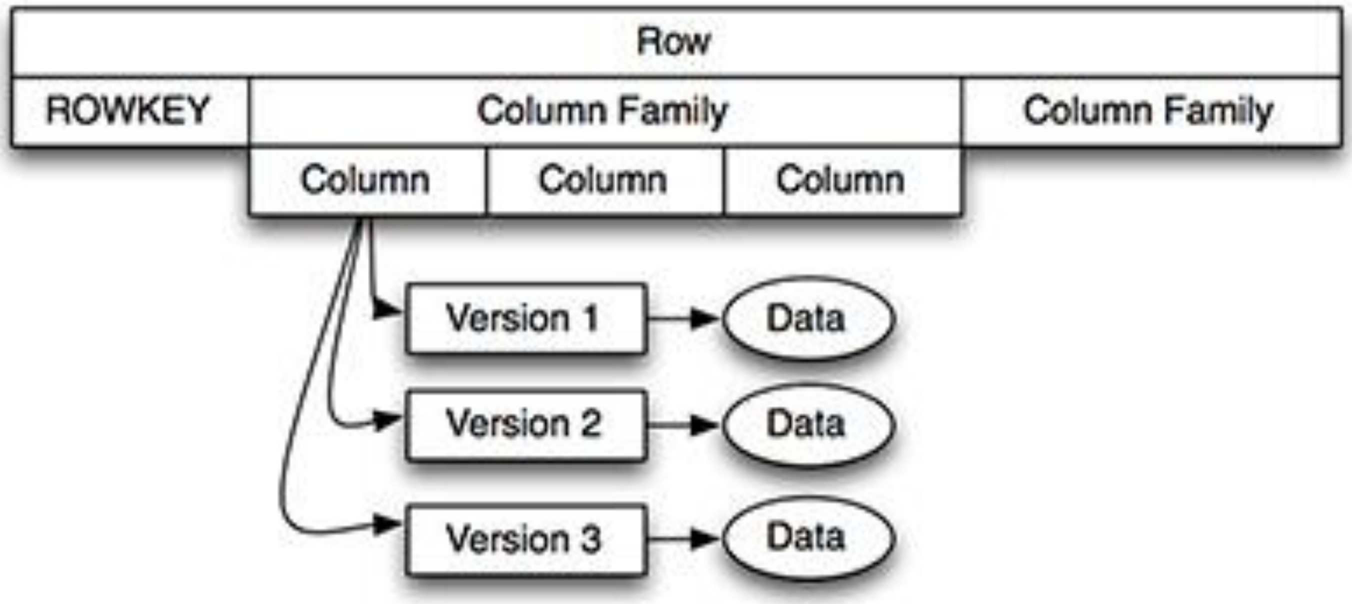
HBase is a distributed non-relational database, that runs on top of the HDFS. It provides a fault-tolerant way of storing large quantities of sparse data, while allowing random, real-time access to them.

Key features:

- Scalable
- Fault-tolerant
- Strictly consistent reads and writes
- Fast



HBase Data Model

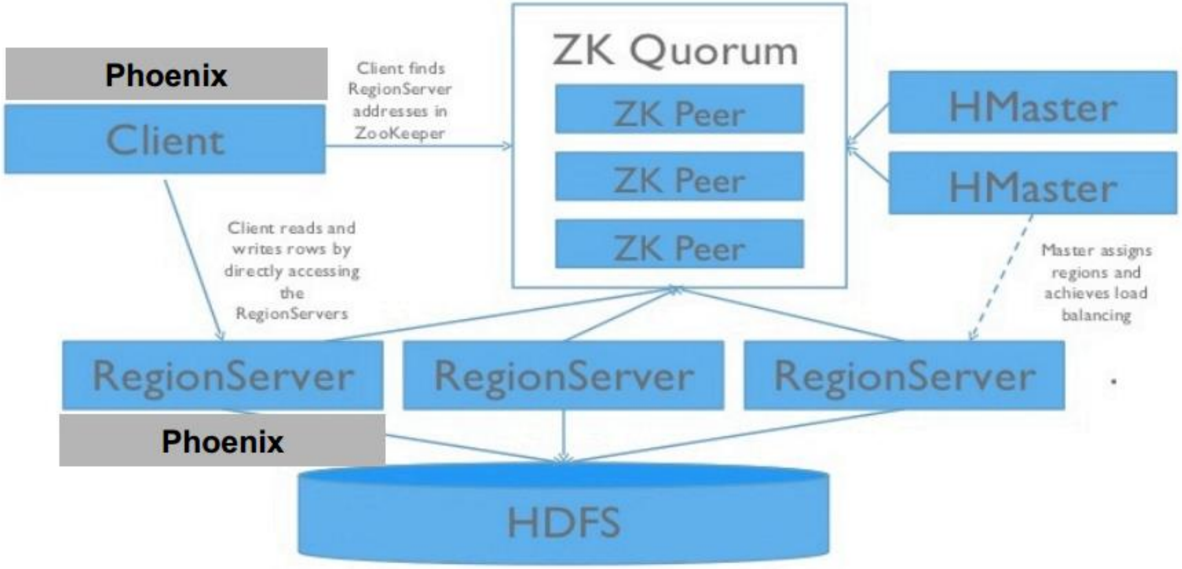


Phoenix

Phoenix is relational database layer for HBase, targeting low latency queries over HBase data.

Phoenix has the following parts:

- Query engine
- Metadata repository
- A JDBC driver


Phoenix
Client
Client finds RegionServer addresses in ZooKeeper
ZK Quorum
ZK Peer
ZK Peer
ZK Peer
HMaster
HMaster
Client reads and writes rows by directly accessing the RegionServers
Master assigns regions and achieves load balancing
RegionServer
RegionServer
RegionServer
Phoenix
HDFS

Phoenix Data Model

The relational elements of the Phoenix data model are mapped to their respective counterparts in the HBase data model:

- A Phoenix table is mapped to an HBase table.
- The Phoenix table's columns that are included in the primary key constraint are mapped together to the HBase row key.
- The rest of the columns are mapped to HBase columns, consisting of a column family and a column qualifier.

Columns in a Phoenix table are assigned an SQL datatype, allowing typed access to HBase data.

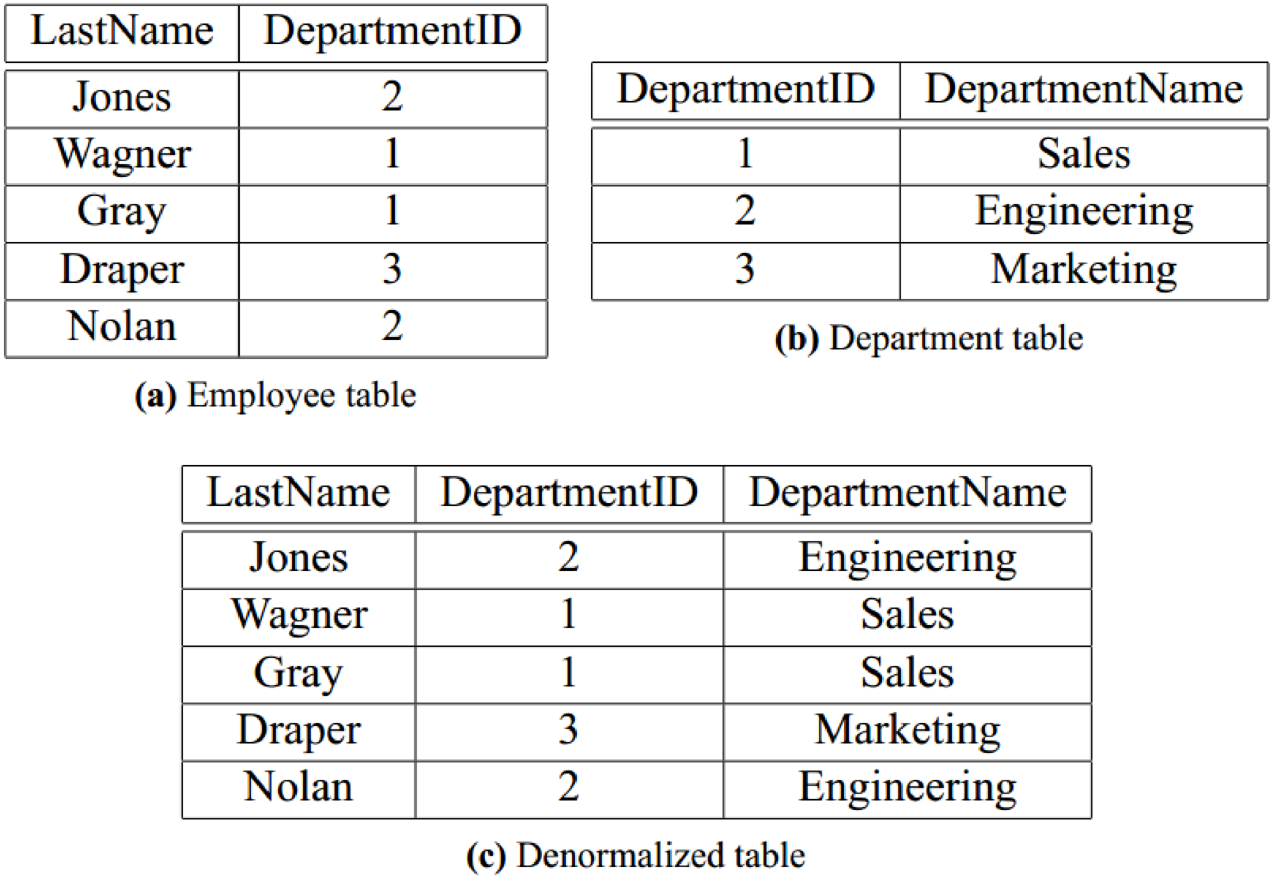
TopN Queries

TopN queries return a sorted set of results for the values in a given dimension according to some criteria.

```
SELECT sourceAS,destinationAS,COUNT(*) AS pairCount
FROM netdata
WHERE timestamp > 1446382476
GROUP BY sourceAS,destinationAS
ORDER BY pairCount DESC
LIMIT 10;
```

System Description

Denormalization



LastName	DepartmentID
Jones	2
Wagner	1
Gray	1
Draper	3
Nolan	2

(a) Employee table

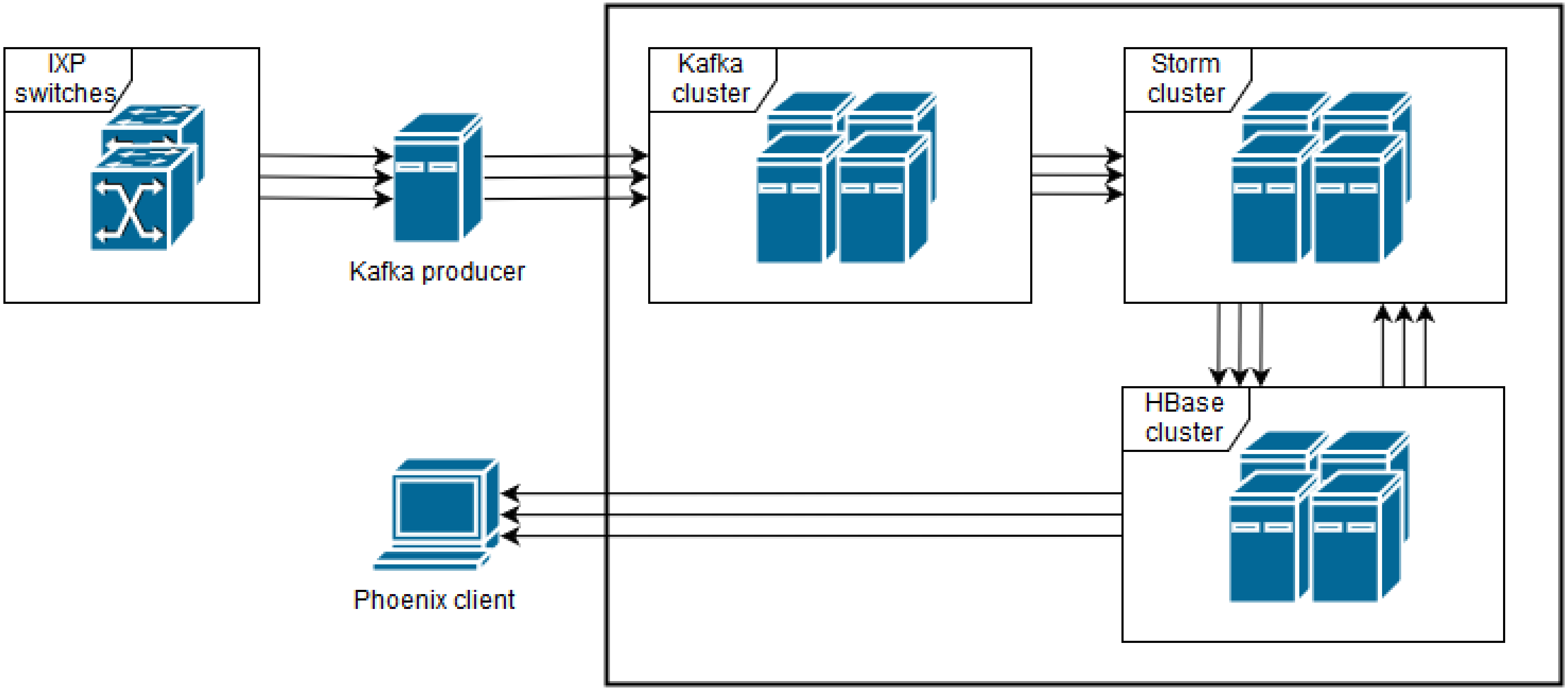
DepartmentID	DepartmentName
1	Sales
2	Engineering
3	Marketing

(b) Department table

LastName	DepartmentID	DepartmentName
Jones	2	Engineering
Wagner	1	Sales
Gray	1	Sales
Draper	3	Marketing
Nolan	2	Engineering

(c) Denormalized table

System Overview



System Key Features

- **Scalability:** utilization of distributed technologies and frameworks
- **Fault tolerance:** data replication in Kafka and HDFS, fault-tolerant Storm topology
- **Extensibility:** extending the functionality of the Storm topology for a new dataset is as simple as adding an extra bolt

Data Generation and Input

IXP switches: An sFlow agent performs random sampling to the packets processed by the switch and sends the flow samples to an sFlow collector.

Kafka producer:

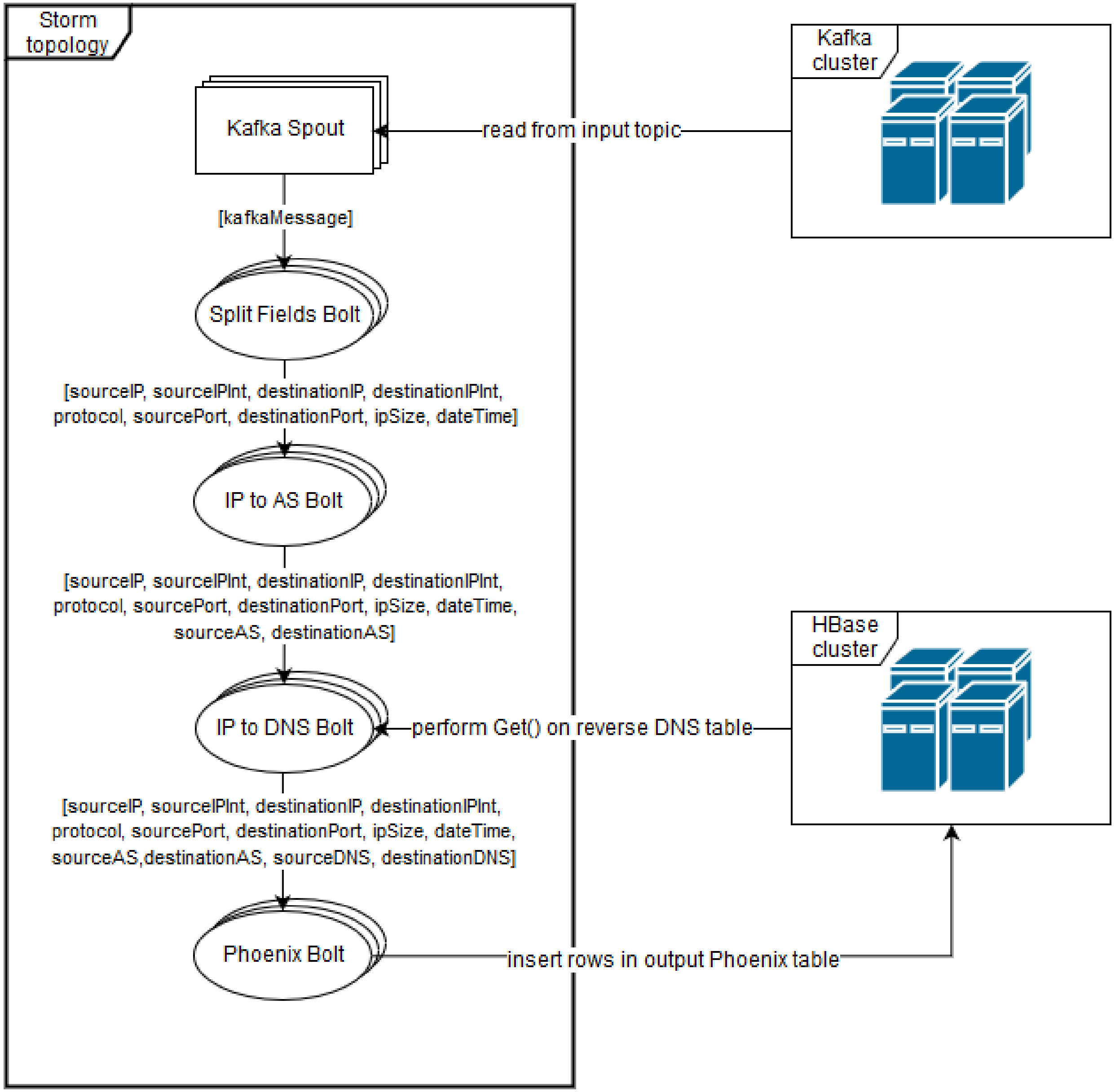
- An sFlow collector collects the flow samples from all of the switches.
- A Kafka producer script preprocesses the flow samples and publishes the useful fields to a Kafka topic.

Kafka Topic

The preprocessed messages containing the useful fields are stored at a Kafka topic.

- For scalability and load balancing, we set the number of the topic's partitions equal to the number of the brokers of the Kafka cluster.
- For fault tolerance, we set a replication factor of 2 for the topic.
- All published messages remain stored at the brokers for a configurable period of time, whether or not they have been consumed.

Storm Topology


Storm topology
Kafka Spout
read from input topic
Kafka cluster
[kafkaMessage]
Split Fields Bolt
[sourceIP, sourceIPInt, destinationIP, destinationIPInt, protocol, sourcePort, destinationPort, ipSize, dateTime]
IP to AS Bolt
[sourceIP, sourceIPInt, destinationIP, destinationIPInt, protocol, sourcePort, destinationPort, ipSize, dateTime, sourceAS, destinationAS]
IP to DNS Bolt
perform Get() on reverse DNS table
HBase cluster
[sourceIP, sourceIPInt, destinationIP, destinationIPInt, protocol, sourcePort, destinationPort, ipSize, dateTime, sourceAS,destinationAS, sourceDNS, destinationDNS]
Phoenix Bolt
insert rows in output Phoenix table

Phoenix Table

The denormalized data stream is stored at the netdata Phoenix table in HBase.

Design factors for the Phoenix table:

- The queries performed on the table have a time window constraint.
- The use case queries concern either AS or DNS information.
- Table size should be minimized.

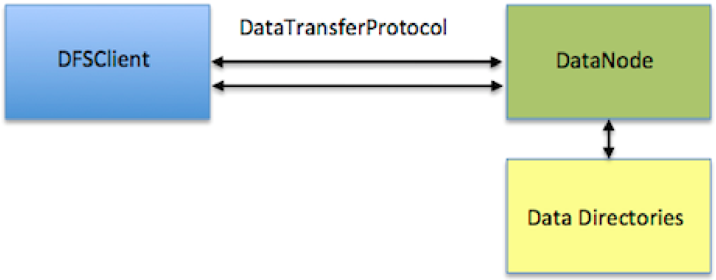
Phoenix Table

```
CREATE TABLE netdata (
    t BIGINT PRIMARY KEY,
    d.ipS VARCHAR,
    d.ipSI BIGINT,
    d.ipD VARCHAR,
    d.ipDI BIGINT,
    d.proto SMALLINT,
    d.portS INTEGER,
    d.portD INTEGER,
    d.size INTEGER,
    as.asS VARCHAR,
    as.asD VARCHAR,
    dns.dnsS VARCHAR,
    dns.dnsD VARCHAR
)
SALT_BUCKETS = 4,
DEFAULT_COLUMN_FAMILY = 'd',
DATA_BLOCK_ENCODING = 'NONE',
COMPRESSION = 'SNAPPY';
```

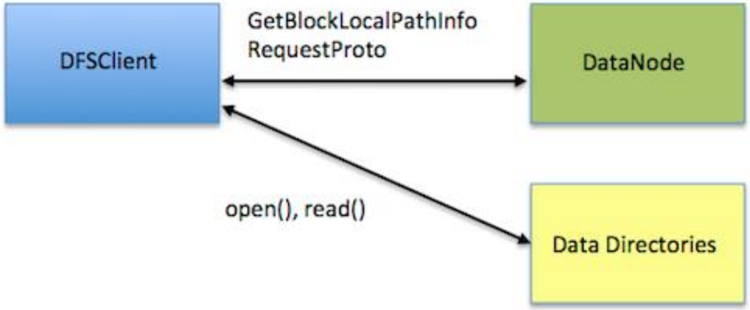
HBase and Phoenix Optimizations

HDFS Short-Circuit Local Reads

DISABLED



ENABLED



Compression and Data Block Encoding

Physical data size on disk can be decreased by using compression and data block encoding. Compression and data block encoding can be used together on the same column family.

Compression reduces the size of cell values and can significantly reduce the storage space needed to store data.

Data block encoding attempts to limit duplication of information in keys, taking advantage of some of the fundamental designs and patterns of HBase, such as sorted row keys and the schema of a given table.

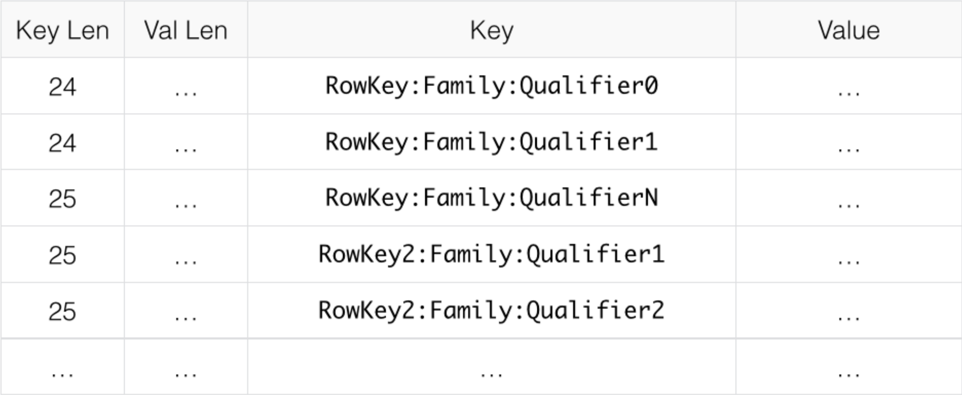
Compression and data block encoding can also reduce the data size in the BlockCache.

Compression and Data Block Encoding

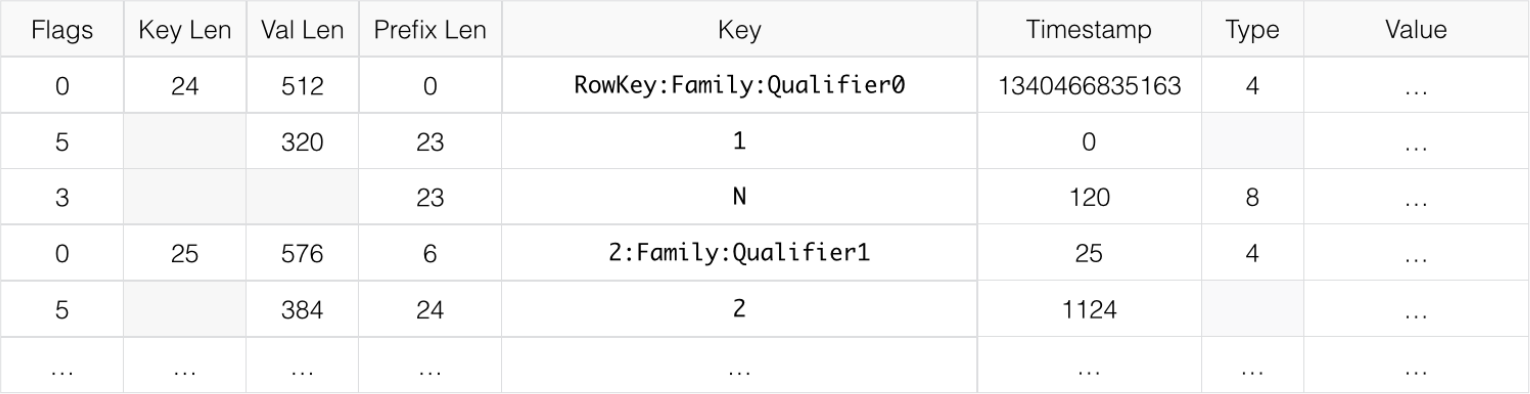
Snappy compression: Does not aim for maximum compression, but instead aims for very high speeds and reasonable compression.

Fast Diff data block encoding: Reduces data size by using an extra column which holds the length of the prefix shared between the current key and the previous key.

Fast Diff Data Block Encoding



Key Len	Val Len	Key	Value
24	…	RowKey:Family:Qualifier0	…
24	…	RowKey:Family:Qualifier1	…
25	…	RowKey:Family:QualifierN	…
25	…	RowKey2:Family:Qualifier1	…
25	…	RowKey2:Family:Qualifier2	…
…	…	…	…



Flags	Key Len	Val Len	Prefix Len	Key	Timestamp	Type	Value
0	24	512	0	RowKey:Family:Qualifier0	1340466835163	4	…
5		320	23	1	0		…
3			23	N	120	8	…
0	25	576	6	2:Family:Qualifier1	25	4	…
5		384	24	2	1124		…
…	…	…	…	…	…	…	…

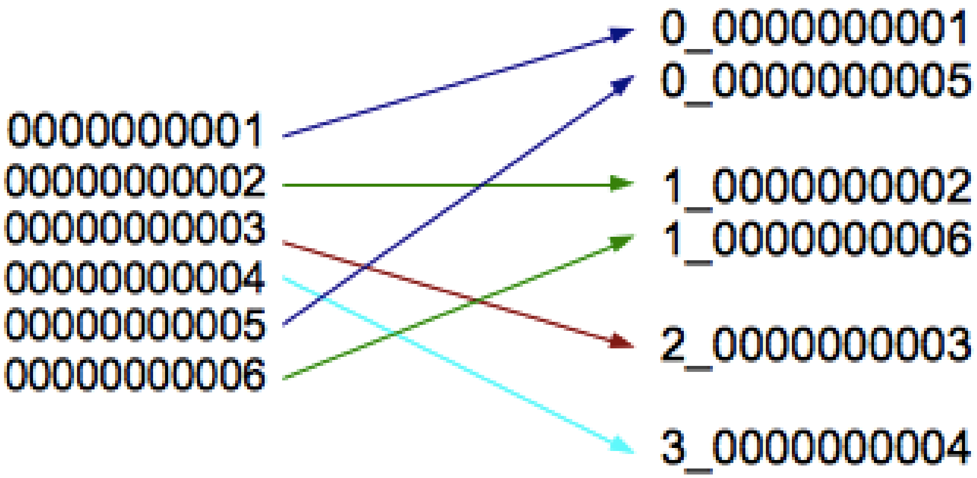
Salting

- The row key for the underlying HBase table where our Phoenix table is stored must be the timestamp associated with the packet, in order to optimize scans for queries over a specified time window.
- Monotonically increasing row keys are a common source of hotspotting. When records with sequential keys are being written to HBase all writes hit one Region which is served by one RegionServer.

Salting the row key provides a way to mitigate the problem, by adding a randomly-assigned prefix to the row key, to cause it to sort differently than it otherwise would.

Salting

```
newKey = (++index % BUCKETS_NUMBER) + originalKey
```



0000000001
00000000002
00000000003
00000000004
00000000005
00000000006

0_0000000001
0_0000000005

1_0000000002
1_0000000006

2_0000000003

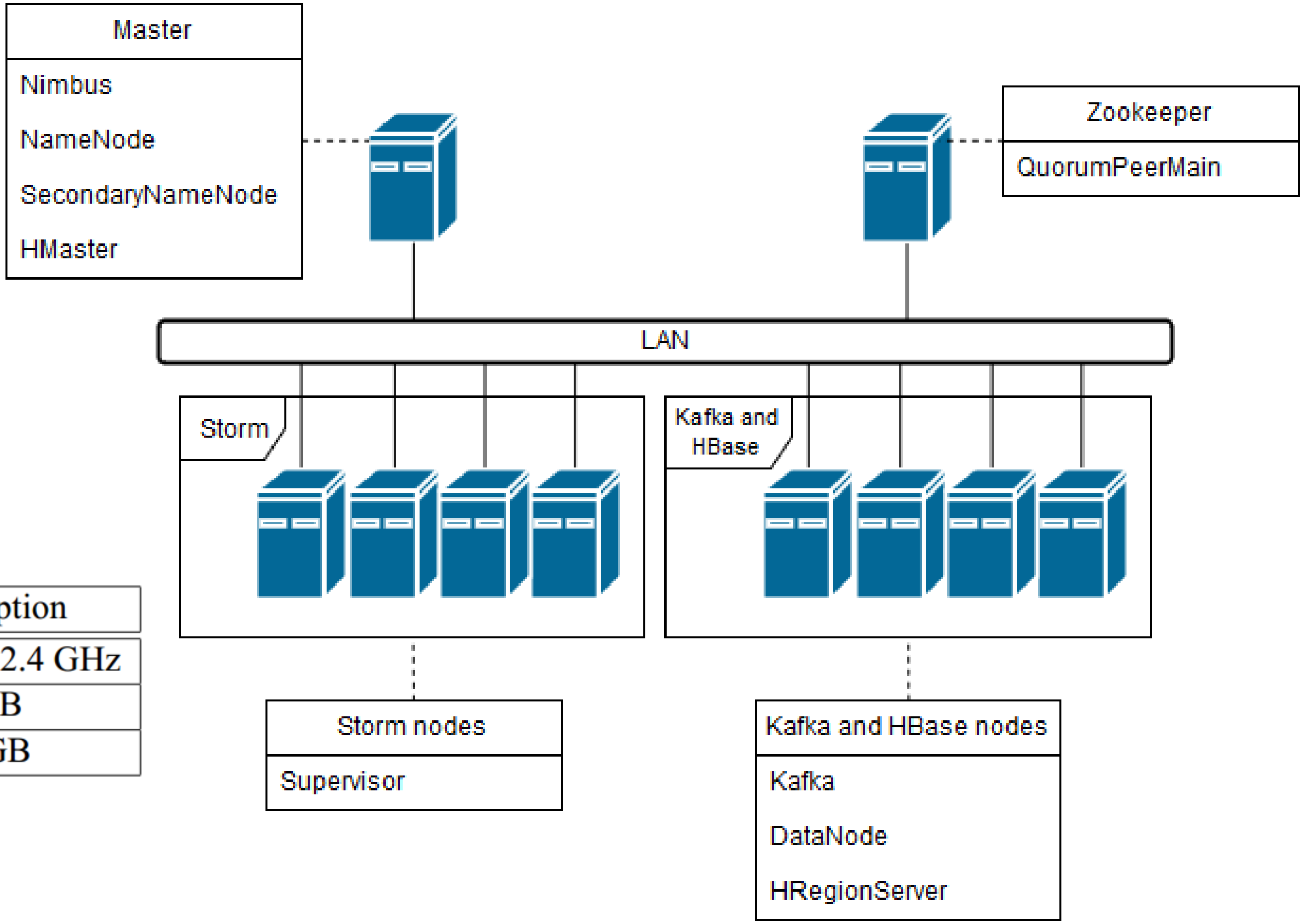
3_0000000004

Evaluation

Datasets

- IXP Traffic dataset: GR-IX traffic sampled over a period of 6 months (July 2013 to February 2014), on average 110 packets sampled per second
- Autonomous System dataset: GeoLite ASN IPv4 database, maps IPv4 address ranges to Autonomous System Numbers, 13 MB size
- DNS dataset: Rapid7 Reverse DNS dataset, maps IPv4 addresses to domain names, 5.7 GB gzip compressed and 55 GB uncompressed size

Evaluation Cluster



Master

Nimbus

NameNode

SecondaryNameNode

HMaster

Zookeeper

QuorumPeerMain

LAN

Storm

Kafka and HBase

Storm nodes

Supervisor

Kafka and HBase nodes

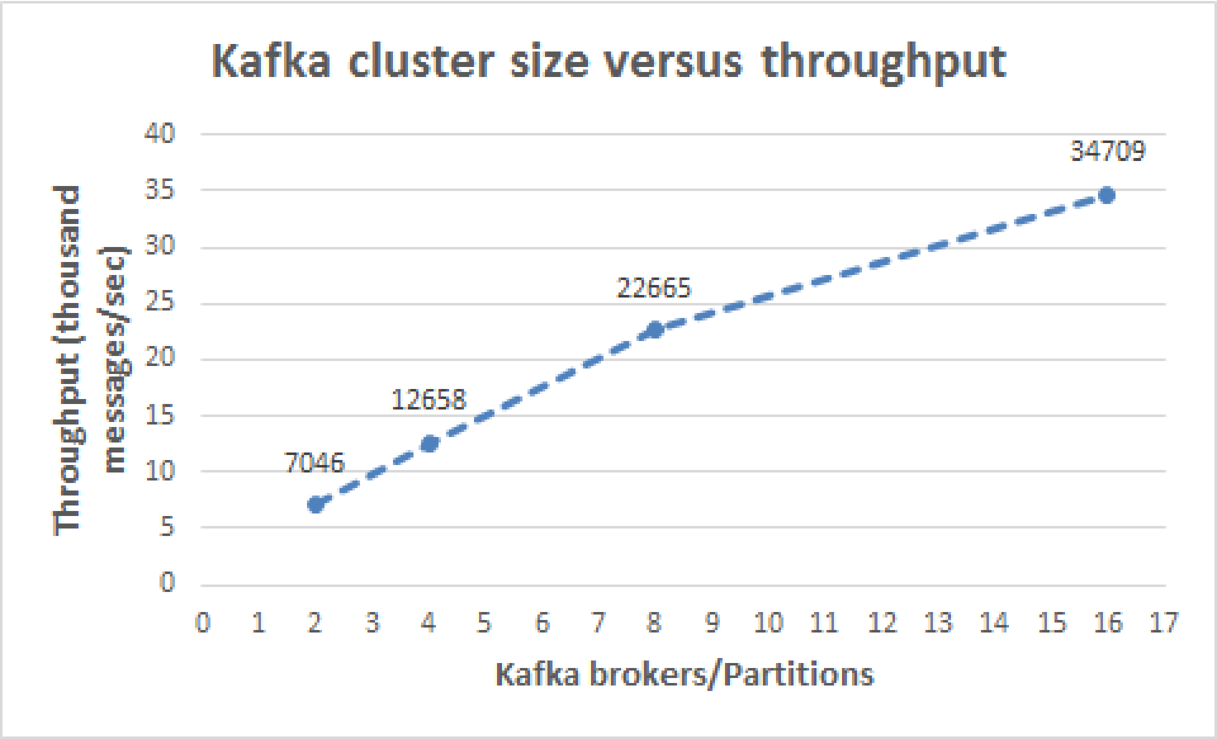
Kafka

DataNode

HRegionServer

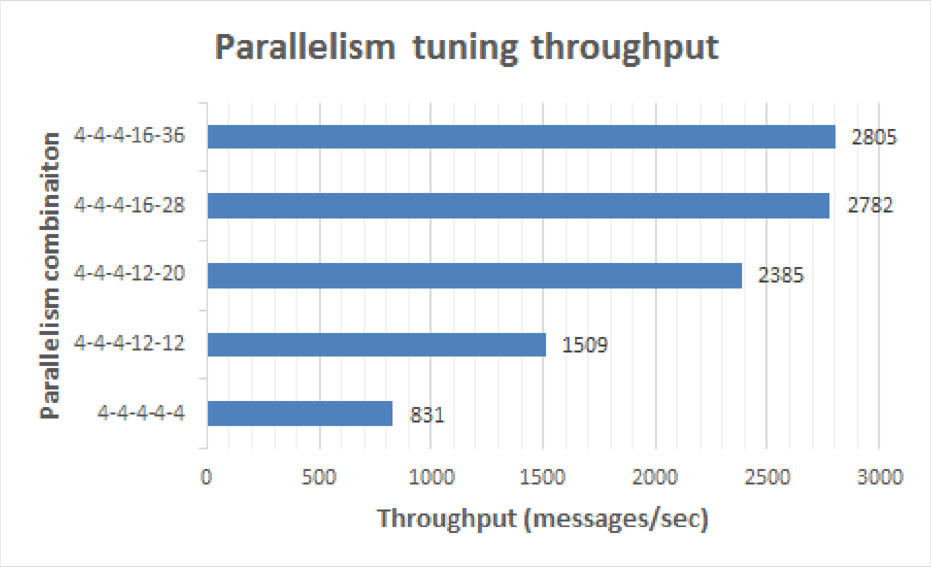
Component	Description
CPU	4 cores @ 2.4 GHz
RAM	8 GB
Disk	80 GB

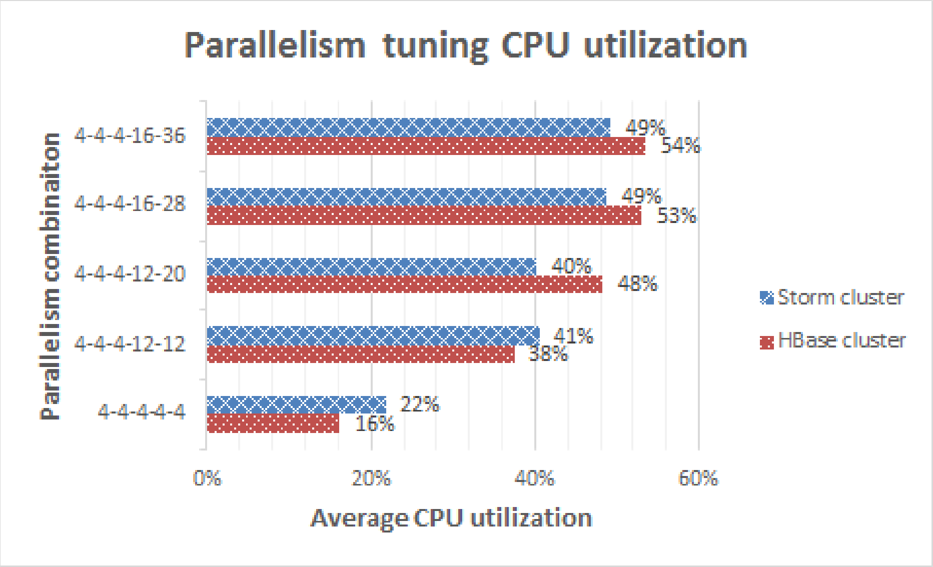
Kafka Scalability



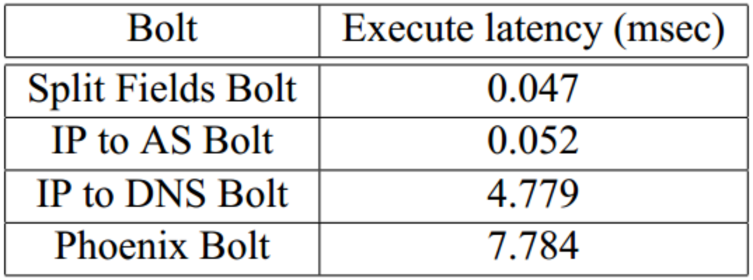
Storm Parallelism Tuning

Experiment	Split Fields Bolt		IP to AS Bolt		IP to DNS Bolt		Phoenix Bolt	
	Parallelism	Capacity	Parallelism	Capacity	Parallelism	Capacity	Parallelism	Capacity
4-4-4-4-4	4	0.013	4	0.011	4	0.600	4	**0.987**
4-4-4-12-12	4	0.021	4	0.042	12	0.491	12	**1.068**
4-4-4-12-20	4	0.040	4	0.043	12	**1.043**	20	**0.911**
4-4-4-16-28	4	0.043	4	0.062	16	0.879	28	**1.049**
4-4-4-16-36	4	0.067	4	0.077	16	0.816	36	**1.086**

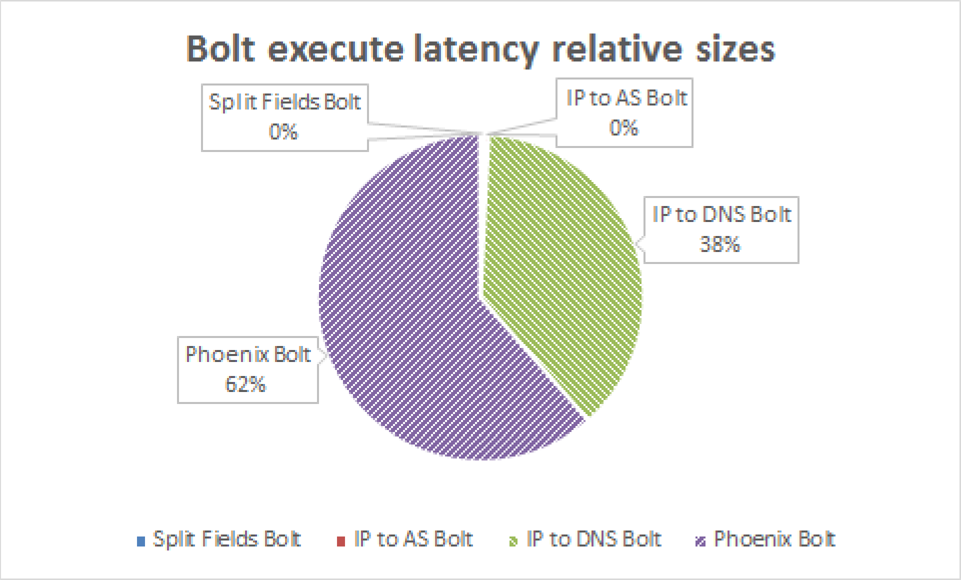




Bolt Execute Latencies



Bolt	Execute latency (msec)
Split Fields Bolt	0.047
IP to AS Bolt	0.052
IP to DNS Bolt	4.779
Phoenix Bolt	7.784

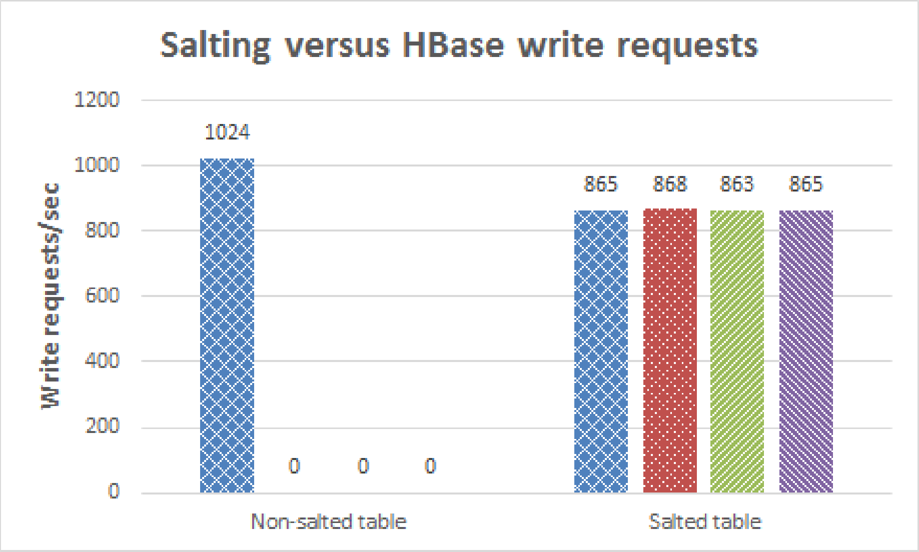


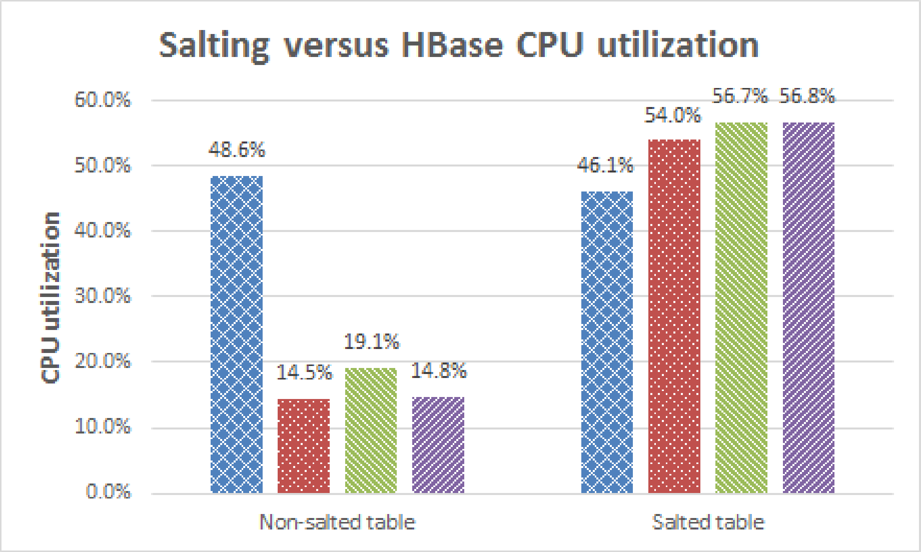
Total System Latency

Total system latency is the time it takes for a message to be sent by the Kafka producer to the topic, consumed by the Kafka Spout, processed by the bolts of the topology and eventually be stored in the Phoenix table and made available for queries.

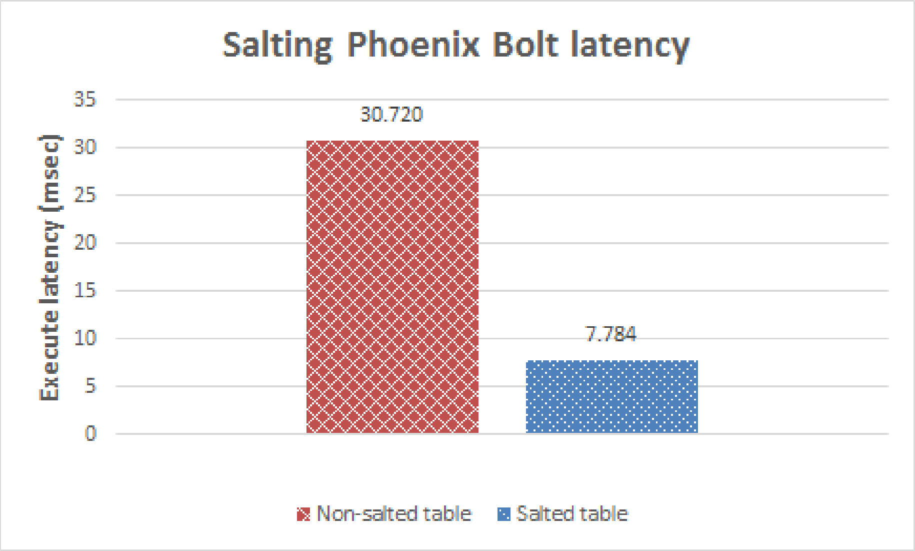
Total system latency is measured at **1.161 sec** on average at maximum topology throughput.

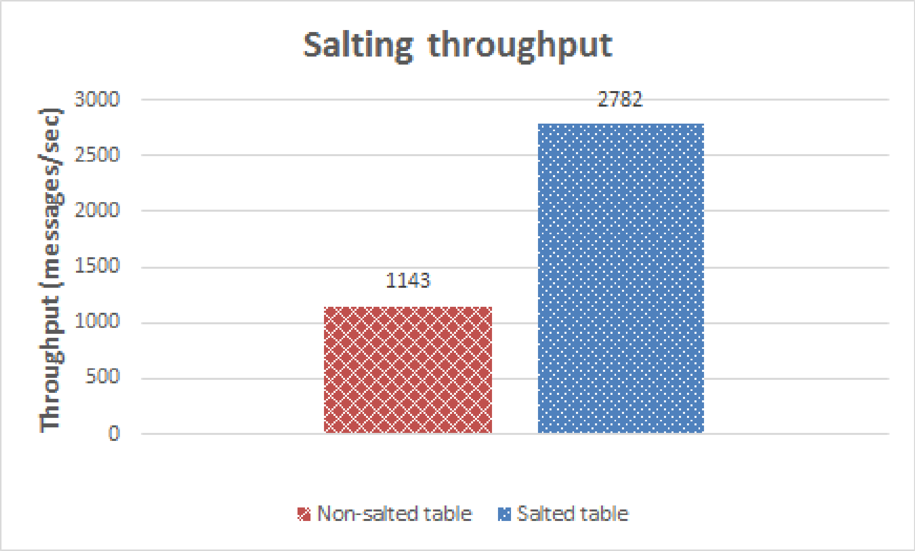
Salting Write Performance





Salting Write Performance

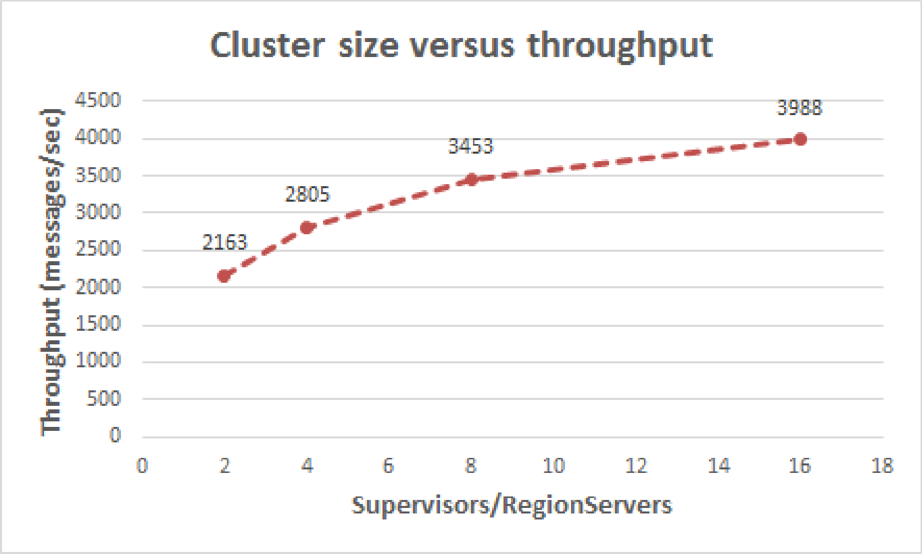


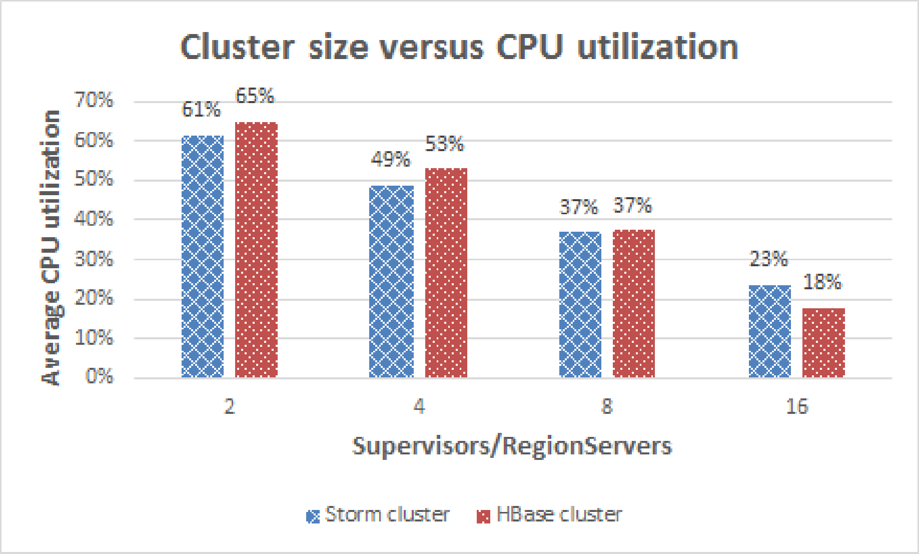


74% decrease in Phoenix Bolt’s execute latency

140% increase in topology throughput

Storm Scalability





HBase and Phoenix Experiments

We perform two types of queries:

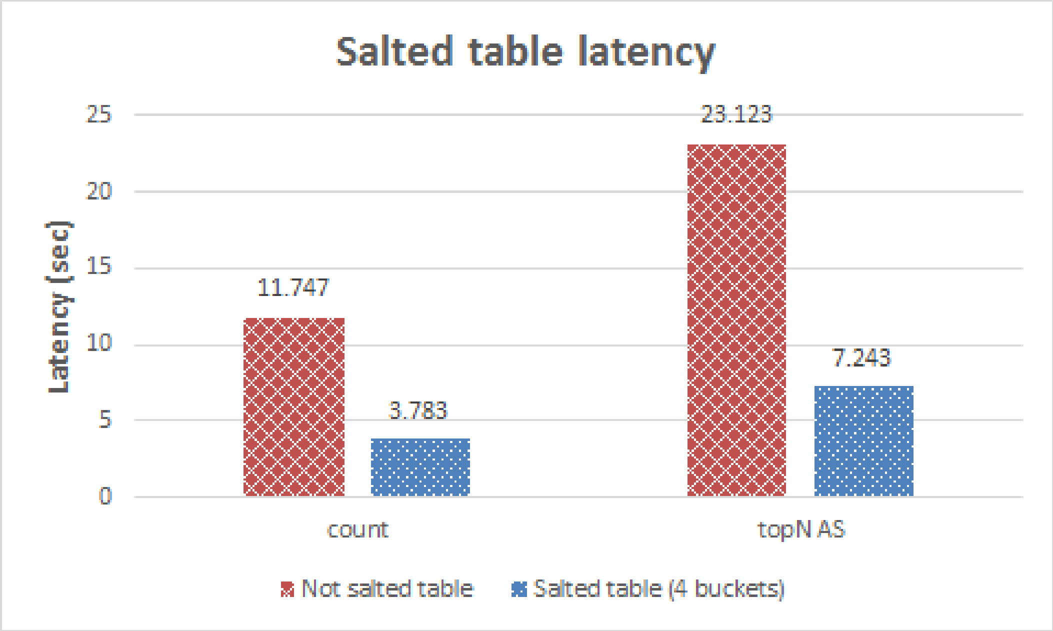
- Count queries

```
SELECT COUNT(*) FROM TABLE netdata;
```

- TopN queries

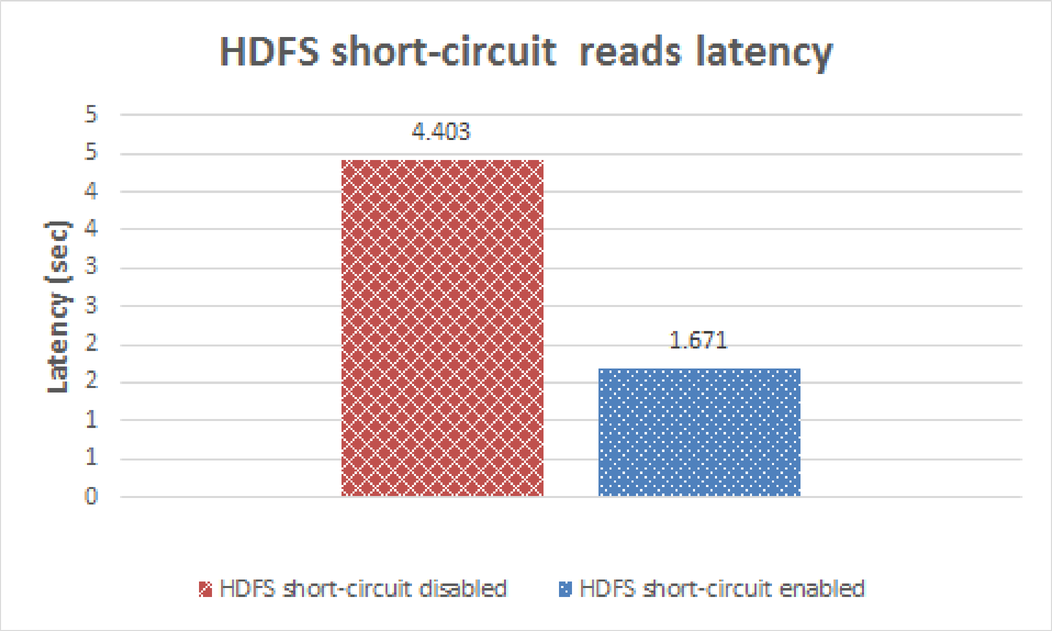
```
SELECT as.asS, as.asD, COUNT(*) AS pairCount
FROM netdata
GROUP BY as.asS, as.asD
ORDER BY pairCount DESC
LIMIT 10;
```

Salting Read Performance



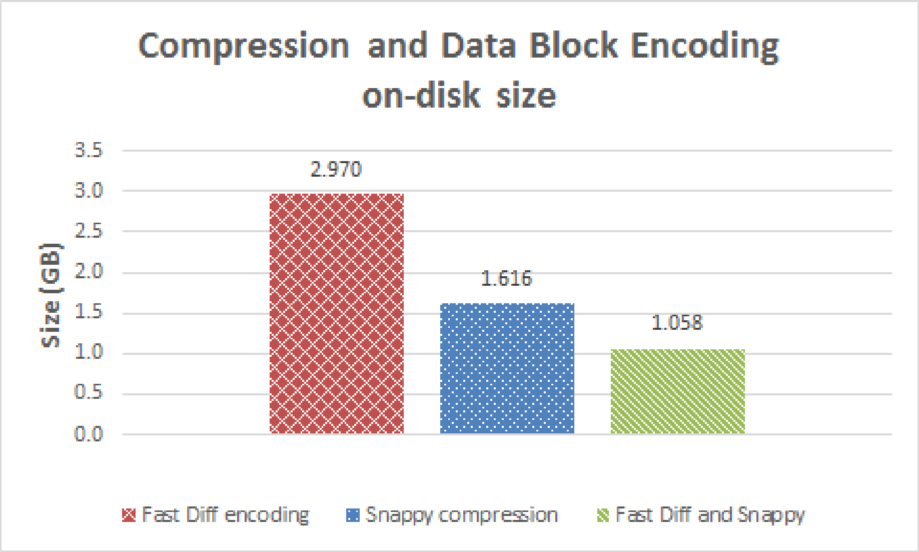
68% increase in query latency

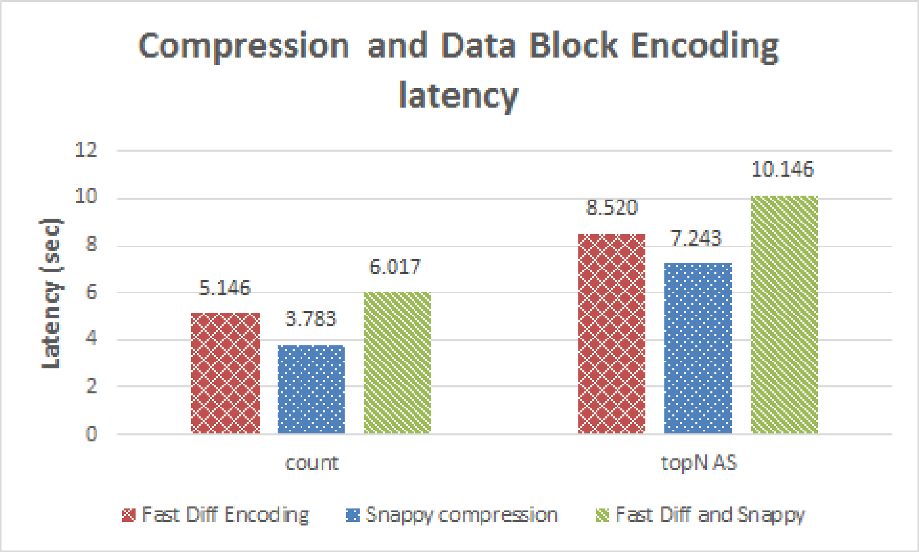
HDFS Short-Circuit Local Reads



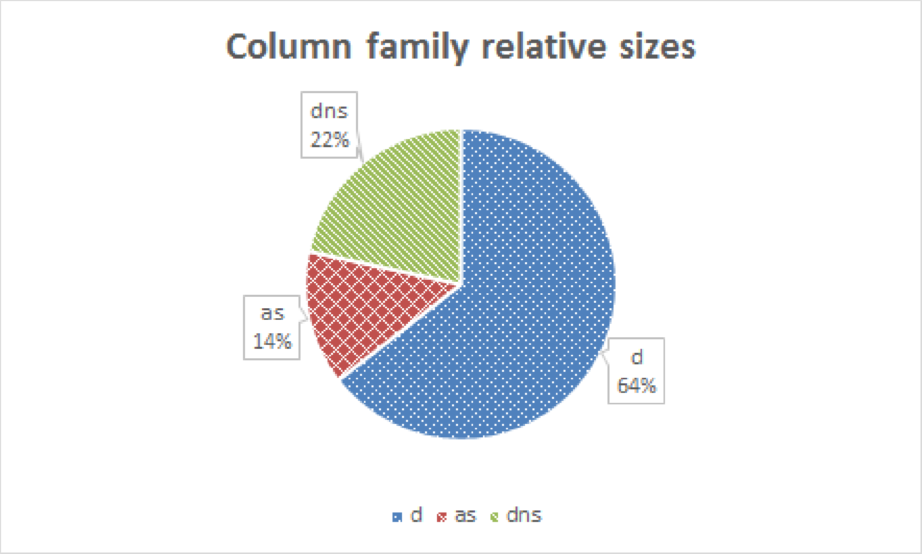
62% increase in total query latency

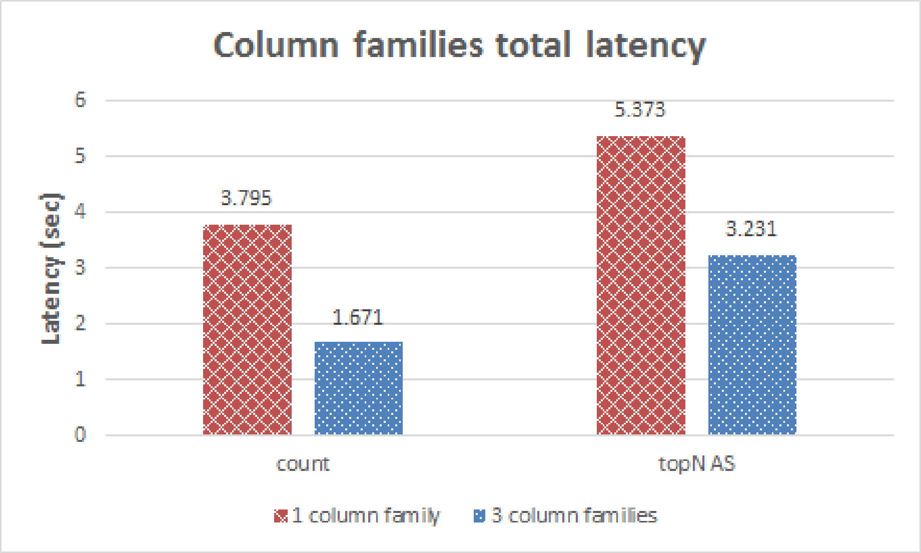
Compression and Data Block Encoding



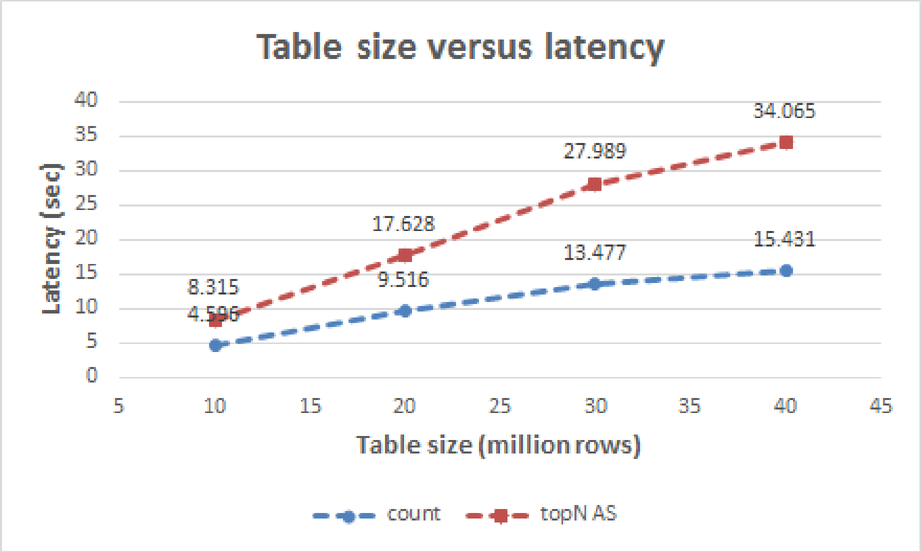


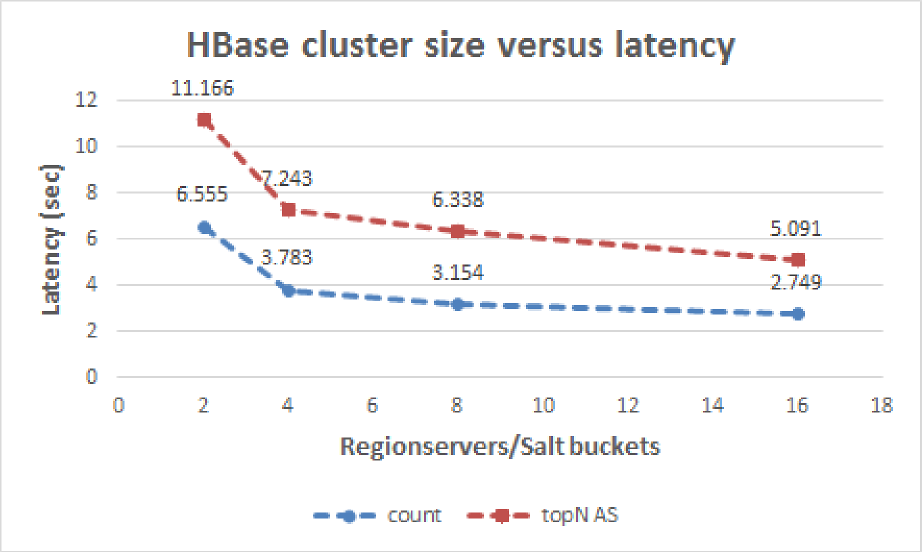
Number of Column Families





HBase Scalability





Conclusion

Concluding Remarks

We designed and implemented a distributed system that allows the execution of low latency SQL queries that join a real-time network data stream, generated by sampling IXP traffic, and external datasets containing Autonomous System and DNS information.

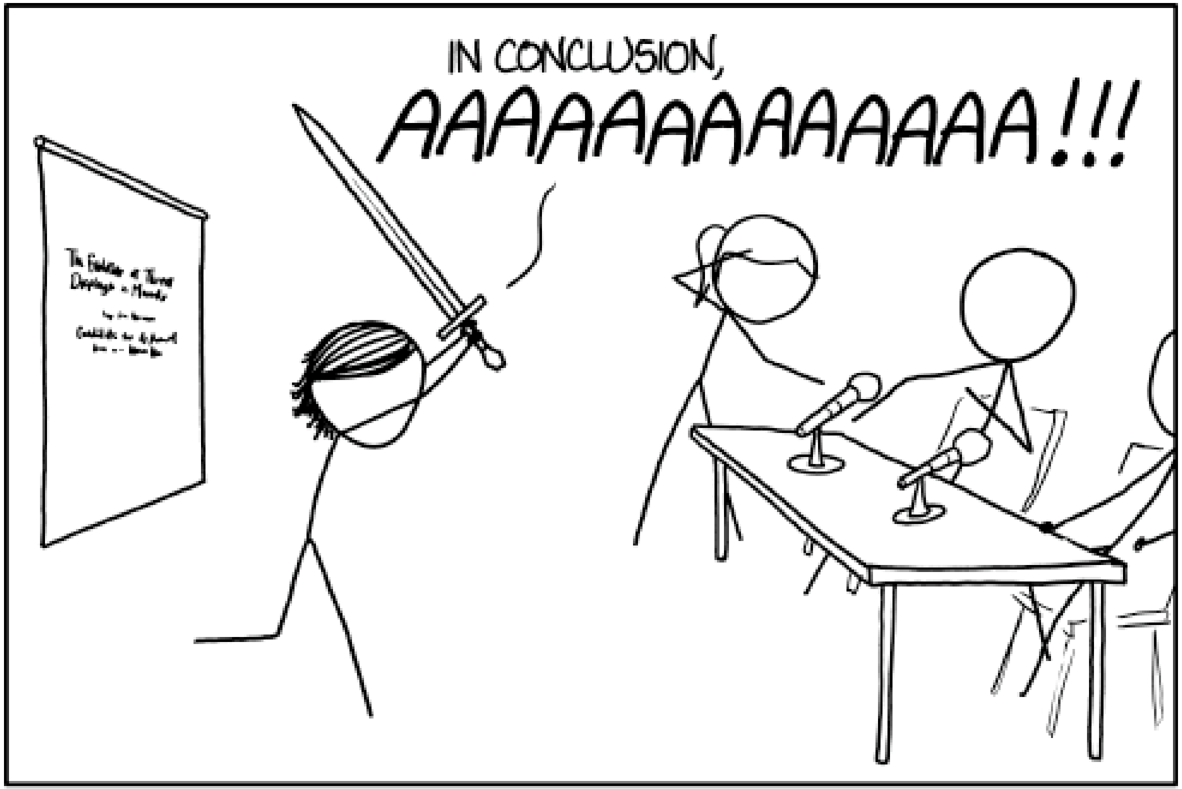
The system has the following key features:

- Execution of low latency real-time queries
- Scalability
- Fault tolerance
- Extensibility

Future Work

- Properly evaluate the system’s scalability using a cluster of physical nodes, each one assigned with a dedicated disk.
- Compare the Storm topology of our system to implementations in other distributed stream processing frameworks, such as Storm Trident, Spark Streaming and Samza.
- Compare Phoenix to other low latency SQL-on-HBase querying engines, such as Apache Drill and Spark SQL.

Questions?



THE BEST THESIS DEFENSE IS A GOOD THESIS OFFENSE.