

# **Scuba: Diving into Data at Facebook**

## **- Lior et. al**

**Presented By - Sidharth Singla**

**MMATH CS**



**UNIVERSITY OF WATERLOO**  
FACULTY OF MATHEMATICS

# OUTLINE

- Importance
- History
- Scuba: Introduction
- Use Cases
- Scuba Overview - Data Model; Data Layout; Data ingestion, distribution and lifetime; Query Model; Query Execution
- Experimental Evaluation
- Related Work
- Future Work mentioned
- Conclusion
- Discussion



# Importance

- Performance issues and monitoring considered seriously at Facebook.
- Event latencies under a minute required.
- Infra team has to rely on real-time instrumentation to ensure site is up and running.
- Real-time instrumentation required for
  - Code regression analysis
  - Bug Report Monitoring
  - Ads revenue monitoring
  - Performance debugging.



# History

- Previously relied on pre-aggregated graphs and a MySQL database of performance data.
  - Too rigid and slow.
- Other query systems: Hive & Peregrine
  - Data gets available typically after one day latency.
  - Queries take minutes to run.
- Scuba built.



# Scuba: Introduction

- Data management system at Facebook.
- Real-time analysis.
- Fast, scalable, distributed, in-memory database.
- Processes almost million queries each day.
- Ingests million of rows per second and expires data at the same rate.



- Runs on hundreds of servers each with 144GB RAM in shared-nothing cluster.
- Provides an SQL Query interface and a GUI that produces time series graphs, pie charts etc.
- Compressed data stores in memory. Each table is partitioned and distributed randomly across all the servers.
- `sample_rate` allowed to specify sampling of event.



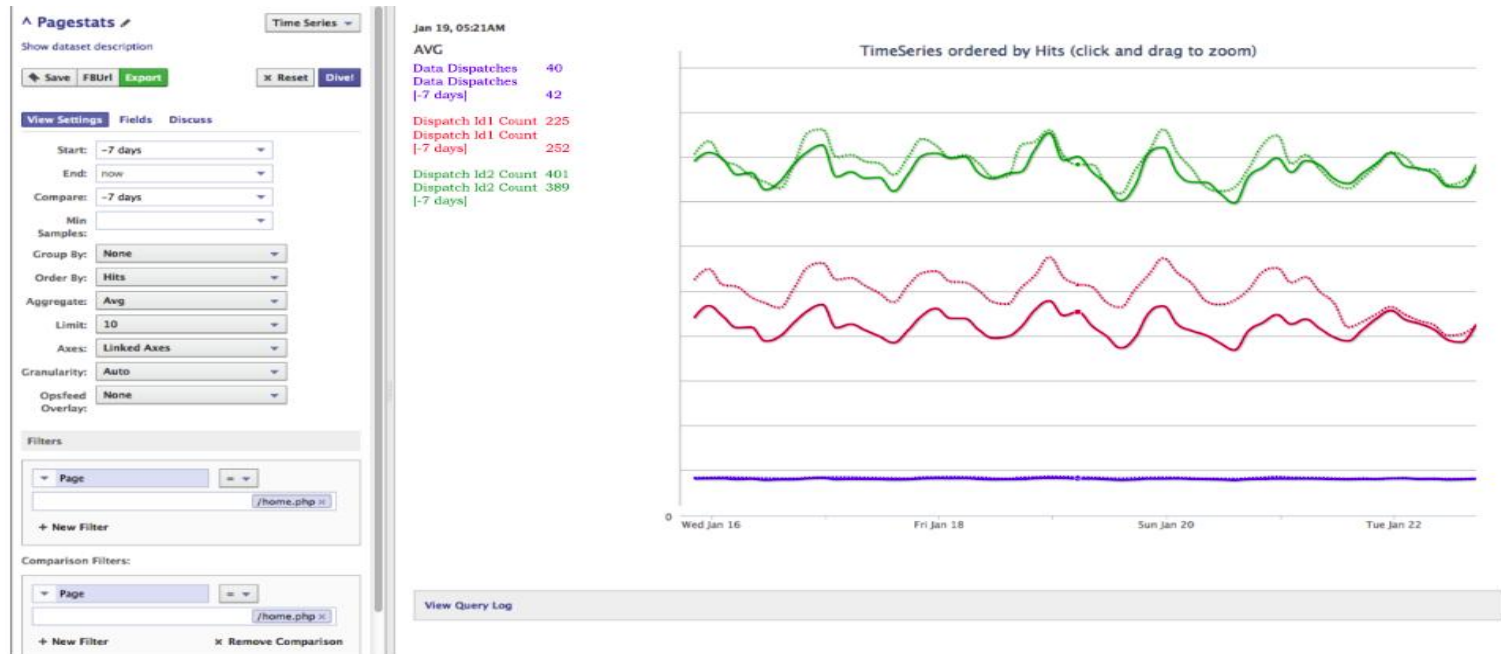


Figure 1: Scuba's web user interface. The query shown on the left side generates a time series graph with a week over week comparison of three columns related to Facebook page dispatches. The dotted lines represent the same days one week earlier. It is very easy to see daily and weekly cyclical behavior with these graphs.

# Why name Scuba ?

- Start by high-level aggregate queries to identify interesting phenomena in data.
- Dive deeper to find base data points of interest.



# Use Cases

## 1. Performance Monitoring

Use Dashboards to visualise and monitor performance metrics like CPU load, network throughput.

## 2. Trend Analysis

Look for trends in the data content. Eg. Extract sets of words from user posts and look for spikes in word frequencies over time.

## 3. Pattern Mining

Look for patterns based on different dimensions.



# Scuba Overview

## Data Model

Datatypes supported -

- Integer: Timestamp is also stored as Integer,
- String,
- Set of Strings: Represents say, words in a FB post,
- Vector of Strings: Ordered and used to stack traces.

Floats not supported.

Timestamp mandatory for each row.



# Data Layout

Data Type	Compression type	Representation in row
Integer	Variable length	1-8 bytes
String	Dictionary	Index, uses number of bits for max dictionary index
String (alternate)	Uncompressed	4 bytes length + actual string
Sets of String	Dictionary	Fibonacci encoding of deltas between sorted indexes
Vectors of String	Dictionary	2 bytes count + 1 byte index size + each index

**Figure 3: Data types and compression methods in Scuba.**

Figure from L. Abraham et al, *Scuba: Diving into Data at Facebook*, Proceedings of the VLDB Endowment, Vol 6, No. 11, August 2th-30th, 2013. Riva Del Garda, Trento, Italy.



**UNIVERSITY OF WATERLOO**  
FACULTY OF MATHEMATICS

- Data is stored in row order.
- No create table statement.
- Table is created on each leaf node whenever the leaf first receives data for it.
- Table may exist on some leaves.
- Different schemas on leaves possible.
- Single table image presented to users. Missing columns - NULL values.
- Columns may be sparsely populated.
- No complex schema evolution commands.

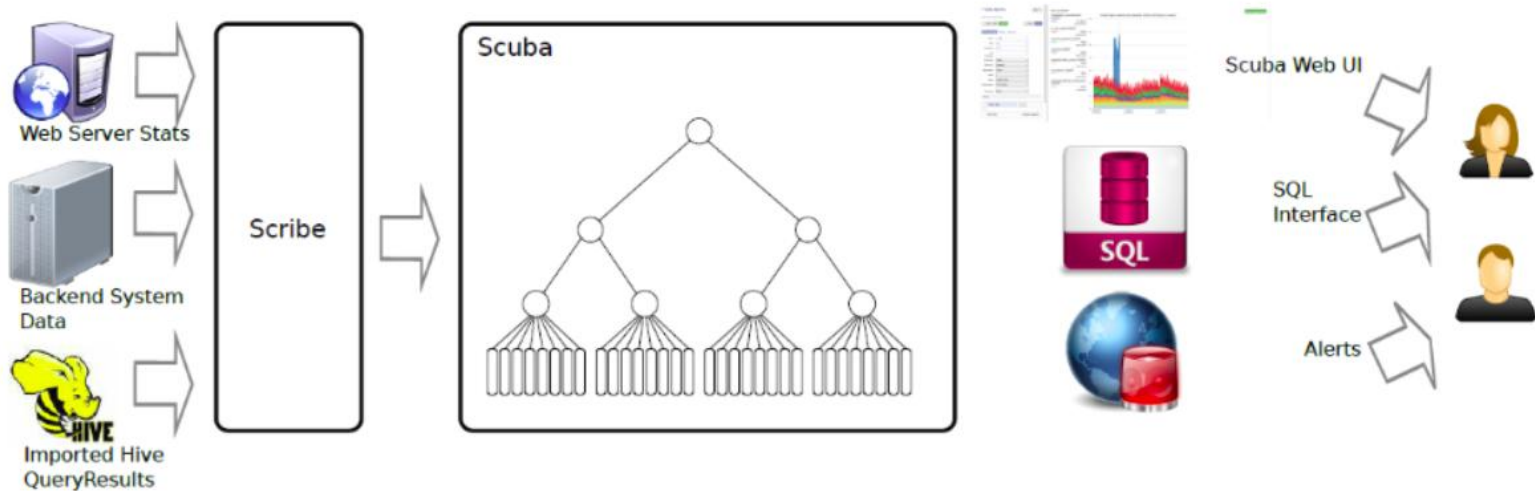


# Data ingestion, distribution and lifetime

- Event occurs -> log entry written to Scribe.
- Scuba chooses two leaves at random. Batch of new rows are sent to the leaf with more free memory, via Thrift API.
- Table rows end up partitioned randomly across all the leaves.
- A gzip compressed copy of the file is stored to disk for persistence.
- Columns are compressed and new rows are added to the table in memory.
- This total elapsed time is usually within a minute.
- Subsampling of data is supported.



# SCUBA OVERVIEW: *Data ingestion and distribution*



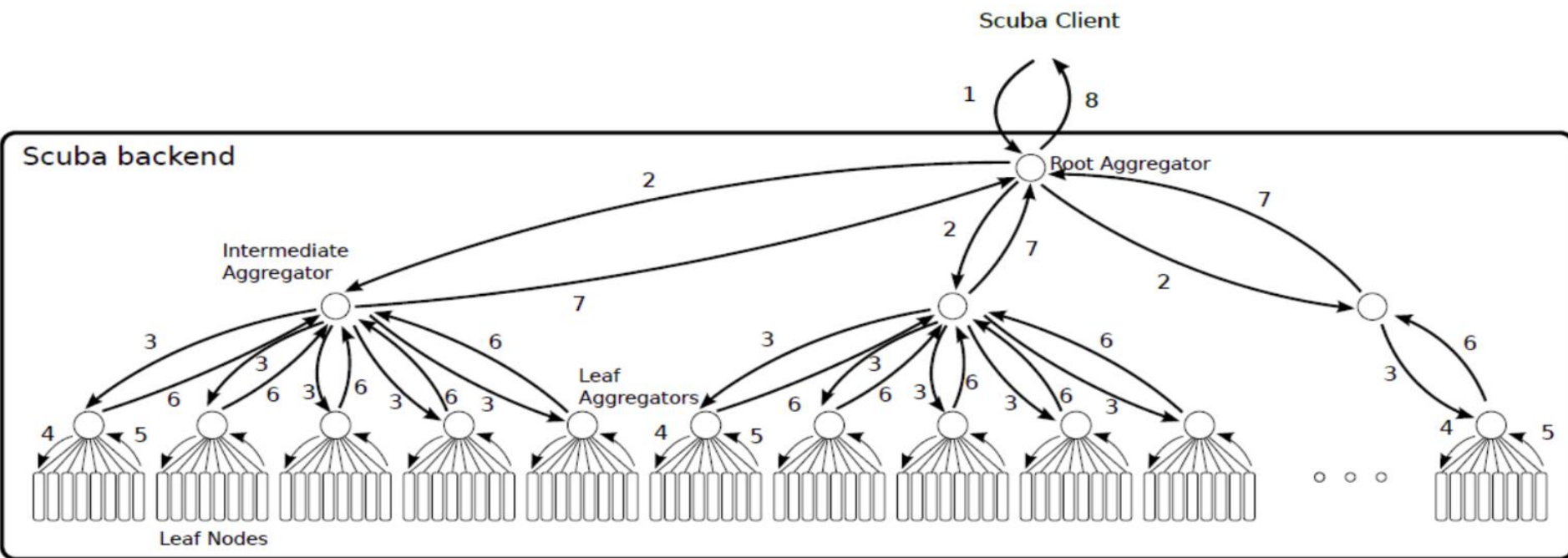
**Figure 2: Scuba system architecture: from data ingestion on the left to user queries on the right.**

# Query Model

- Three query interfaces are supported
  - Web-based interface
  - Command line interface
  - Thrift-based API.
- WHERE clause must contain a time range.
- Joins are not supported.



# Query Execution



**Figure 7: Step-wise breakdown of executing a Scuba query.**

# Experimental Evaluation

- Aggregation cost
  - Independent of the amount of data at each leaf.
  - Function of query and cluster size.
  - Grows logarithmically on scaleup for query with large aggregations.
- Time to scan data at each leaf
  - Proportional to the amount of data at each leaf.
  - Independent of the number of machines.



- Throughput( Queries/sec )
  - Increases with the increase in number of clients until the CPUs saturate.
  - After that flattens.
- Response Time
  - Each query response time increases in proportion to the number of clients.



## Related Work( Other systems for real-time analysis )

1. **HyPer**: Stores data in memory, single large expensive machine. Does not use compression.

**Scuba**: Cluster of cheap small machines and easily scalable.

2. **Shark and Impala**: Analysis over data in HIVE. Cache data in memory during query processing. Suffer long latency in importing data to HIVE.



**3. Powerdrill and Dremel:** Google's data management analytical systems. Highly distributed, scale well, primary data copy lives on disk.

**4. Druid and rrdtool/MRTG:** Imports data quickly, Aggregation on import and provides fast query response time. Cannot drill down to raw original data.

**Scuba:** Stack traces down to the actual code change.

Above systems neither trade accuracy for response time which is main requirement at Facebook nor provide a way to expire data automatically.



# Future Work mentioned in paper

- Try columnar based approach.
- Experiment with BlinkDB techniques like precomputing stratified samples.
- Revisiting TQuel to reason about time and intervals.



# Conclusion

- Provides Automatic pruning of data.
- Stores data after sampling.
- No schema declaration needed.
- Table can contain rows with different schemas.
- Dozen different ways to visualise data.
- Queries run with best effort availability.
- Not a complete SQL database.



# Discussion

- Why are Intermediate Aggregators required ? What purpose do they solve ? Why not connect root aggregators directly to the leaf aggregators ?
- Latency and throughput are main concerns of Scuba. So why is compression done ? Won't it increase the latency of queries ?
- Joins are not supported. If joins are required, data needs to be combined from multiple sources before importing. Isn't it too inefficient for analytical queries ?
- Why multiple leaf servers on a single machine ? Why not use single machine with less resources ?

