Popular Open Source Data Processing Frameworks

Giri Iyengar

Cornell University

gi43@cornell.edu

April 25, 2018

Agenda for the week

- Pig
- Spark
- Storm
- BlinkDB
- Druid

Overview

Storm

Apache Storm

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and is a lot of fun to use!

Storm

Apache Storm

Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a **million tuples processed per second per node**. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Storm Topology

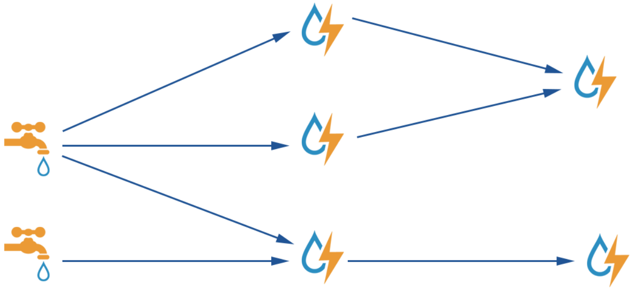


Figure: Storm Topology

Storm Concepts

- **Tuples** - Named list of dynamically typed values. Main data-type of Storm
- **Spouts** - Generates Streams of Tuples by reading files, queues, databases etc
- **Bolts** - Consume any number of input streams and output any number of stream
- **Topology** - A configuration of Spouts and Bolts that does some processing

Storm

- Integrates with many Queues such as Kestrel, Kafka, RabbitMQ, Kinesis
- Simple API
- Scalable
- Fault-Tolerant. Workers are automatically restarted if they fail.
- Guaranteed Tuple processing
- Supports multiple languages

Storm Concept Definitions

Topologies

The logic for a realtime application is packaged into a Storm topology. A Storm topology is analogous to a **MapReduce** job. One key difference is that a MapReduce job eventually finishes, whereas a **topology runs forever** (or until you kill it, of course). A topology is a graph of spouts and bolts that are connected with stream groupings.

Storm Concept Definitions

Streams

The stream is the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Streams are defined with a schema that names the fields in the stream's tuples. By default, tuples can contain integers, longs, shorts, bytes, strings, doubles, floats, booleans, and byte arrays. You can also define your own serializers so that custom types can be used natively within tuples.

- **Tuple**: streams are composed of tuples
- **OutputFieldsDeclarer**: used to declare streams and their schemas
- **Serialization**: Information about Storm's dynamic typing of tuples and declaring custom serializations

Storm Concept Definitions

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.

Storm Concept Definitions

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 (you can do this particular stream transformation in a more scalable way with three bolts than with two).

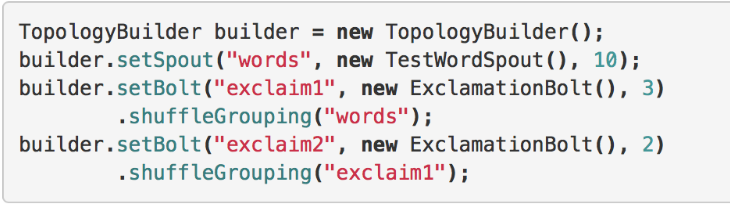
Bolts can emit more than one stream.

Storm Concept Definitions

Workers

Topologies execute across one or more worker processes. Each worker process is a physical JVM and executes a subset of all the tasks for the topology. For example, if the combined parallelism of the topology is 300 and 50 workers are allocated, then each worker will execute 6 tasks (as threads within the worker). Storm tries to spread the tasks evenly across all the workers.

Topology example



```java
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("words", new TestWordSpout(), 10);
builder.setBolt("exclaim1", new ExclamationBolt(), 3)
        .shuffleGrouping("words");
builder.setBolt("exclaim2", new ExclamationBolt(), 2)
        .shuffleGrouping("exclaim1");
```

Word count example

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new RandomSentenceSpout(), 5);

builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

Overview

BlinkDB

Queries with Bounded Errors and Bounded Response Times on Very Large Data

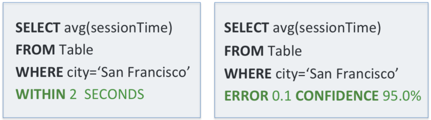
BlinkDB

BlinkDB Performance

BlinkDB has been demonstrated live at VLDB 2012 on a 100 node Amazon EC2 cluster answering a range of queries on 17 TBs of data in less than 2 seconds (over 200x faster than Hive), within an error of 2-10%.

BlinkDB

- Massively Parallel, Approximate query engine
- Trade-off query accuracy vs query time
- Uses bootstrapping and LBOB underneath



```
SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
WITHIN 2  SECONDS
```

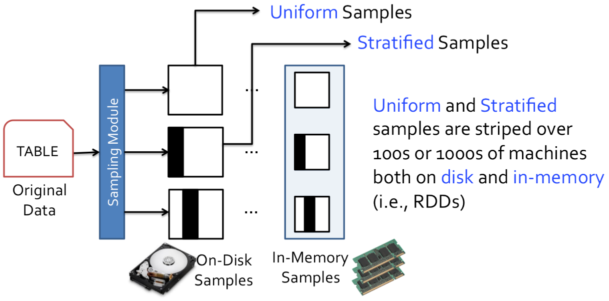
Queries with Time Bounds

```
SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
ERROR 0.1 CONFIDENCE 95.0%
```

Queries with Error Bounds

BlinkDB Overview

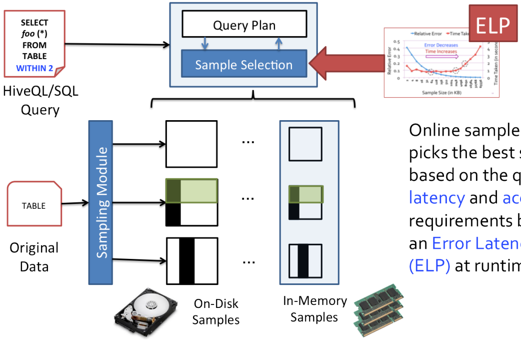
BlinkDB features an Offline Sampling Module that creates Uniform and Stratified samples from underlying data within a given storage budget. The sets of column(s) to stratify on are decided by solving a MILP optimization problem that takes into account the frequently occurring column(s) in the GROUP BY and WHERE clauses of the queries. This module is also responsible for periodically updating, deleting and refreshing the samples to minimize the statistical bias.


Uniform Samples
Stratified Samples
TABLE
Original Data
Sampling Module
...
On-Disk Samples
In-Memory Samples
Uniform and Stratified samples are striped over 100s or 1000s of machines both on disk and in-memory (i.e., RDDs)

1 of 3

BlinkDB Overview

When a query comes into the system, the Sample Selection Module then picks an appropriately sized sample based on its response time or accuracy requirements. This is done by creating an Error Latency Profile (ELP) for the query at runtime by operating the query on small samples of data and extrapolating the error/response time on larger samples.

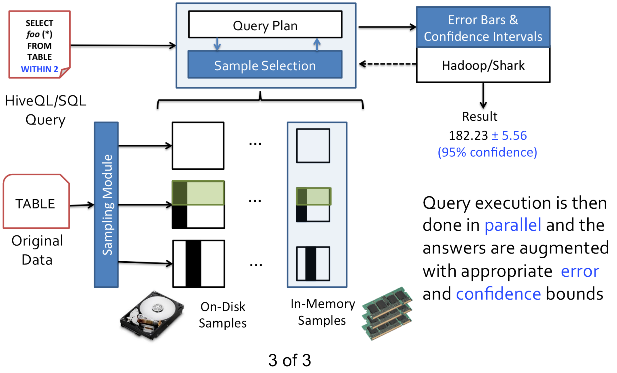


Online sample selection picks the best sample(s) based on the query's latency and accuracy requirements by creating an Error Latency Profile (ELP) at runtime

2 of 3

BlinkDB Overview

Finally, the queries are executed in parallel on the chosen samples and the answers are augmented by appropriate error and confidence bounds by leveraging statistical closed forms or statistical bootstrap techniques.



Overview

Druid

Druid

Druid is an open-source data store designed for sub-second queries on real-time and historical data. It is primarily used for business intelligence (OLAP) queries on event data. Druid provides low latency (real-time) data ingestion, flexible data exploration, and fast data aggregation. Existing Druid deployments have scaled to trillions of events and petabytes of data. Druid is most commonly used to power user-facing analytic applications.

Druid Key Features

- **Sub-second OLAP Queries** Druid's unique architecture enables rapid multi-dimensional filtering, ad-hoc attribute groupings, and extremely fast aggregations.
- **Real-time Streaming Ingestion** Druid employs lock-free ingestion to allow for simultaneous ingestion and querying of high dimensional, high volume data sets. Explore events immediately after they occur.
- **Power Analytic Applications** Druid has numerous features built for multi-tenancy. Power user-facing analytic applications designed to be used by thousands of concurrent users.

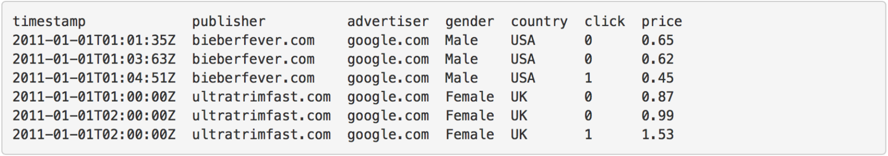
Druid Key Features

- **Cost Effective** Druid is extremely cost effective at scale and has numerous features built in for cost reduction. Trade off cost and performance with simple configuration knobs.
- **Highly Available** Druid is used to back SaaS implementations that need to be up all the time. Druid supports rolling updates so your data is still available and queryable during software updates. Scale up or down without data loss.
- **Scalable** Existing Druid deployments handle trillions of events, petabytes of data, and thousands of queries every second.

Druid Features

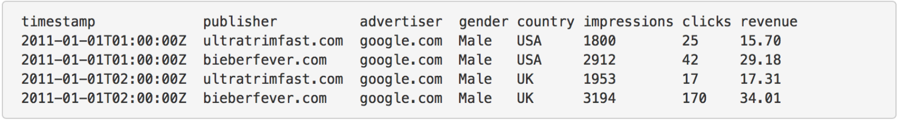
- Column oriented distributed Datastore
- Approximate Algorithms (e.g. HyperLogLog+)

Druid Data Handling: Roll-ups



timestamp	publisher	advertiser	gender	country	click	price
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	USA	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	USA	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	USA	1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	1	1.53

Druid Data Handling: Roll-ups

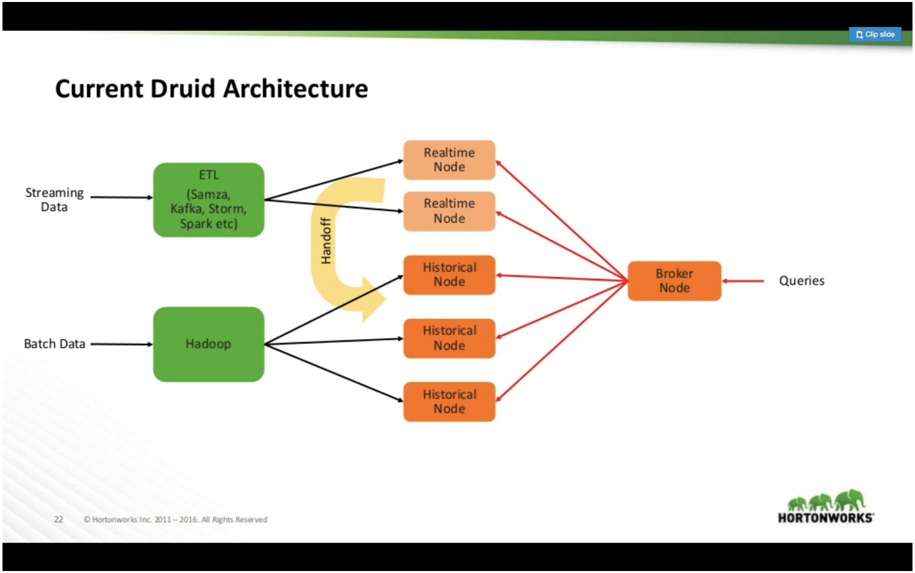


timestamp	publisher	advertiser	gender	country	impressions	clicks	revenue
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Male	USA	1800	25	15.70
2011-01-01T01:00:00Z	bieberfever.com	google.com	Male	USA	2912	42	29.18
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Male	UK	1953	17	17.31
2011-01-01T02:00:00Z	bieberfever.com	google.com	Male	UK	3194	170	34.01

Druid

- Data stored column-wise
- Only rolled-up data is stored
- Both real-time and batch data are combined when answering a query

Druid: Example of Lambda Architecture in Action



Druid Cluster

Druid Cluster Node types

- Historical Nodes
- Real-time Nodes
- Broker Nodes
- Co-ordinator Nodes

Depends on external systems such as Zookeeper to manage intra-cluster communications and co-ordination

thank you
danke
gracias
merci
obrigado
спасибо
teşekkür ederim
go raibh maith agat
dziękuję
謝謝
ありがとう
감사합니다