A real-time Lambda Architecture using Hadoop & Storm
Nathan Bijnens
Big Data Engineer @ Virdata
@nathan_gs
Computing Trends

Past

- Computation (CPUs) Expensive
- Disk Storage Expensive
- DRAM Expensive
- Coordination Easy (Latches Don’t Often Hit)

Current

- Computation Cheap (Many Core Computers)
- Disk Storage Cheap (Cheap Commodity Disks)
- DRAM / SSD Getting Cheap
- Coordination Hard (Latches Stall a Lot, etc)

Source: [Immutability Changes Everything](https://example.com) - Pat Helland, RICON2012
Nathan Marz

- Ex-Backtype & Twitter
- Startup in Stealthmode

Creator of
- Storm
- Cascalog
- ElephantDB

Coined the term Lambda Architecture.
a Data System
Data is more than Information

Not all information is equal.

Some information is derived from other pieces of information.
Eventually you will reach the most ‘raw’ form of information.

This is the information you hold true, simply because it exists. Let’s call this ‘data’, very similar to ‘event’.
Events: Before

Events used to **manipulate** the master data.
Today, events are the master data.
Let’s store everything.
Data System

Data is **Immutable**.
Data is **Time Based.**
Capturing change

Traditionally

INSERT INTO contact (name, city) VALUES ('Nathan', 'Antwerp')
UPDATE contact SET city = 'Cologne' WHERE name = 'Nathan'
Capturing change

in a Data System

INSERT INTO contact (name, city, timestamp) VALUES ('Nathan', 'Antwerp', 2008-10-11 20:00Z)
INSERT INTO contact (name, city, timestamp) VALUES ('Nathan', 'Cologne', 2014-04-29 10:00Z)
The data you query is often **transformed**, aggregated, ... 

Rarely used in it’s original form.
Query = function ( all data )
Query: Number of people living in each city

<table>
<thead>
<tr>
<th>Person</th>
<th>City</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nathan</td>
<td>Antwerp</td>
<td>2008-10-11</td>
</tr>
<tr>
<td>John</td>
<td>Cologne</td>
<td>2010-01-23</td>
</tr>
<tr>
<td>Dirk</td>
<td>Antwerp</td>
<td>2012-09-12</td>
</tr>
<tr>
<td>Nathan</td>
<td>Cologne</td>
<td>2014-04-29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antwerp</td>
<td>1</td>
</tr>
<tr>
<td>Cologne</td>
<td>2</td>
</tr>
</tbody>
</table>
Query

All Data

Precomputed View

Query
Layered Architecture

- Batch Layer
- Speed Layer
- Serving Layer
Layered Architecture

Incoming Data

Hadoop

ElephantDB

Cassandra

Query
Batch Layer
Batch Layer

Incoming Data

Hadoop

ElephantDB
Batch Layer

Unrestrained computation.

The batch layer can calculate anything, given enough time...
No need to De-Normalize.

The batch layer stores the data normalized, the generated views are often, if not always denormalized.
Horizontally scalable.
Batch Layer

High Latency.

Let’s for now pretend the update latency doesn’t matter.
Functional computation, based on immutable inputs, is idempotent.
Batch Layer

Stores a master copy of the data set

... append only
Batch Layer

Hadoop
Batch: view generation

Master Dataset → MapReduce → View #1
Master Dataset → MapReduce → View #2
Master Dataset → MapReduce → View #3
MapReduce

1. Take a large data set and divide it into subsets

   ![Diagram of data division](image)

2. Perform the same function on all subsets

   ![Diagram of function application](image)

3. Combine the output from all subsets

   ![Diagram of output combination](image)

Output
MapReduce
Catch errors as quickly as they happen. Validate on write vs on read.

Catch errors as quickly as they happen. Validate on write vs on read.
CSV is actually a serialization language that is just poorly defined.
Use a format with a schema

- Thrift
- Avro
- Protocolbuffers

Could be combined with Parquet.

Added bonus: it’s faster and uses less space.
Read Only database

No random writes required.
Every iteration produces the views from scratch.
Batch View Databases

Pure Lambda databases

- ElephantDB
- SploutSQL

Databases with a batch load & read only views

- Voldemort

Other databases that could be used

- ElasticSearch/Solr: generate the lucene indexes using MapReduce
- Cassandra: generate sstables
- ...
Eventually consistent

Without the associated complexities.
Batch Layer

We are not done yet...

Data absorbed into Batch Views

Not yet absorbed.

Just a few hours of data.
Speed Layer
Speed Layer

Incoming Data

Hadoop

ElephantDB

Cassandra
Speed Layer

Stream processing.
Continuous computation.
Storing a limited window of data.

Compensating for the last few hours of data.
All the complexity is isolated in the Speed Layer.

If anything goes wrong, it’s auto-corrected.
You have a choice between:

- **Availability**
  - Queries are eventual consistent

- **Consistency**
  - Queries are consistent
Some algorithms are hard to implement in real-time. For those cases we could estimate the results.
Storm
Message passing
Distributed processing
Horizontally scalable.
Incremental algorithms
Storm

Fast.
Storm

Grouping

Grouping

Grouping
Data Ingestion

Queues & Pub/Sub models are a natural fit.
Data Ingestion

- Kafka
- Flume
- Scribe
- *MQ
- ...
The views need to be stored in a random writable database.
The logic behind a R/W database is much more complex than a read-only view.
Speed Layer Views

The views are stored in a Read & Write database.

- Cassandra
- Hbase
- Redis
- SQL
- ElasticSearch
- ...

NoSQL Matter 2014 - A real-time (Lambda) Architecture using Hadoop & Storm - #nosql14
Serving Layer
Serving Layer

- Hadoop
- ElephantDB
- Cassandra
- Query

Incoming Data
Serving Layer

Random reads.
This layer queries the batch & real-time views and merges it.
How to query an Average?
Side note: CQRS
CQRS

Source: [martinfowler.com/bliki/CQRS.html](http://martinfowler.com/bliki/CQRS.html) - Martin Fowler
CQRS & Event Sourcing

Event Sourcing

- Every command is a new event.
- The event store keeps all events, new events are appended.
- Any query loops through all related events, even to produce an aggregate.

source: CQRS Journey - Microsoft Patterns & Practices
Lambda Architecture
The Lambda Architecture can discard any view, batch and real-time, and just recreate everything from the master data.
Mistakes are corrected via recomputation.

Write bad data? Remove the data & recompute.
Bug in view generation? Just recompute the view.
Data storage is highly optimized.
Immutability changes everything.
Questions?

@nathan_gs #nosql14
nathan@nathan.gs / slideshare.net/nathan_gs
lambda-architecture.net / @LambdaArch / #LambdaArch
Virdata is the cross-industry cloud service/platform for the Internet of Things. Designed to elastically scale to monitor and manage an unprecedented amount of devices and applications using concurrent persistent connections, Virdata opens the door to numerous new business opportunities.

Virdata combines Publish-Subscribe based Distributed Messaging, Complex Event Processing and state-of-the-art Big Data paradigms to enable both historical & real-time monitoring and near real-time analytics with a scale required for the Internet of Things.
Acknowledgements

I would like to thank Nathan Marz for writing a very insightful book, where most of the ideas in this presentation come from.

Parts of this presentation has been created while working for datacrunchers.eu, I thank them for the opportunities to speak about the Lambda Architecture both at clients and at conferences. DataCrunchers is the first Big Data agency in Belgium.

Schema’s & Pictures:

- Computing Trends: Immutability Changes Everything - Pat Helland, RICON2012
- MapReduce #1: PolybasePass2012.pptx - David J. DeWitt, Microsoft Gray Systems Lab
- MapReduce #2: Introduction to MapReduce and Hadoop - Shivnath Babu, Duke
- CQRS: martin Fowler.com/bliki/CQRS.html - Martin Fowler
- CQRS & Event Sourcing: CQRS Journey - Adam Dymitruk, Josh Elster & Mark Seemann, Microsoft Patterns & Practices
Thank you

@nathan_qs
nathan@nathan.gs