Big Data Analytics
Architectures, Algorithms
and Applications
Part #3: Analytics Platform

Simon Wu
HTC (Prior: Twitter & Microsoft)

Edward Chang 張智威
HTC (Prior: Google & U. California)
Three Lectures

• Lecture #1: Scalable Big Data Algorithms
  – Scalability issues
  – Key algorithms with application examples

• Lecture #2: Intro to Deep Learning
  – Autoencoder & Sparse Coding
  – Graph models: CNN, MRF, & RBM

• Lecture #3: Analytics Platform [by Simon Wu]
  – Intro to LAMA platform
  – Code lab
Lecture #3 Outline

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
Contents

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
Motivation

https://github.com/LamaBigData/lama-demo
What Do We Need?

• Log Management System:
  – Collect application logs with reasonable latency

• Data processing platform:
  – Interactive, batch, streaming

• Analytics platform:
  – Visual, ad-hoc

https://github.com/LamaBigData/lama-demo
Contents

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
Logging Management

• Open-sourced systems:
  – Flume
  – FluentD (recommended by GCE)

• Supported file formats:
  – Textline, SequenceFile, Structured data in Thrift/Protobuf/Avro
  – Compressed data in Lzo/Gzip/Snappy
  – However, logs saved in **Parquet** (columnar format) give us significant performance gains over other choices

https://github.com/LamaBigData/lama-demo
Data Processing Engines

• Interactive
  – Impala (open sourced), BigQuery (Google)

• Batch
  – Hadoop MapReduce, Spark, Hive/Pig, Cascading/Scalding

• Streaming
  – Storm, Spark streaming

• Hybrid
  – Google’s Dataflow (managed, in beta)

https://github.com/LamaBigData/lama-demo
# Different Data Processing Engines

<table>
<thead>
<tr>
<th>Engine</th>
<th>Open-Source Framework</th>
<th>Properties</th>
<th>Latency</th>
<th>Application</th>
</tr>
</thead>
</table>
| Batch Processing        | Hadoop MapReduce      | • Large data sets  
  • High Throughput | Hours or Days | Hourly/Daily Statistics         |
| Streaming Processing    | Storm                 | • Real-time  
  • In-memory | Milliseconds | Real-time Counting               |
| Interactive Querying    | Impala/Impala Impala  | • SQL-like query  
  • In-memory | Minutes      | Ad-Hoc SQL-like Data Analysis   |
| Iterative Data Analysis | Spark                 | • DAG execution  
  • In-memory | Hours         | Machine Learning                 |

https://github.com/LamaBigData/lama-demo
Analytics Platform

• Visual Analytics:
  – Template dashboard
  – Customized visual graphs and pivotal tables
  – Tableau will be our choice!
  – Best for Execs/PMs/Sales, even for Engineers

• Ad-hoc Analytics:
  – R + DS.js + GGPlot2
  – More sophisticated DM/ML analytics on big data
  – Best for Engineers

https://github.com/LamaBigData/lama-demo
Interactive Querying Engine

• Built upon an open-sourced distributed SQL query engine (Impala)
• Logs saved in HDFS in columnar-format (Parquet)
• Query in SQL-like syntax
• Benchmark results show Impala+Parquet outperforms the various other open source alternatives

https://github.com/LamaBigData/lama-demo
Who is Going to Need it?

• More used to SQL-like querying
• Impatient enough to see results, i.e., in minutes instead of hours
• Quickly testing ideas through visual analytics on short or medium-long period of history logs
• Ideal choice for PMs/Sales/Execs, even for engineers

https://github.com/LamaBigData/lama-demo
Impala Architecture

Figure credit: http://impala.io/overview.html
https://github.com/LamaBigData/lama-demo
Data Processing Engines

• Interactive
  – Impala (open sourced), BigQuery (Google)

• Batch
  – Hadoop MapReduce, Spark, Hive/Pig, Cascading/Scalding

• Streaming
  – Storm, Spark streaming

https://github.com/LamaBigData/lama-demo
Batch Processing Engine

• Capable of processing much longer period of history logs with higher latency, usually in hours or longer
• Capable of conducting very sophisticated analytics using DM/ML techniques using MapReduce
• Cron-scheduled for processing new logs

https://github.com/LamaBigData/lama-demo
Batch System Architecture

User can do visual or ad hoc analysis on results computed from batch system.

It’s also common to upload results computed from batch system to online readonly store, in order for online services to lookup values.

https://github.com/LamaBigData/lama-demo
One Problem?

• What if we want to see metrics from all history to current moment?
  – Batch: high throughput, but high latency
  – Streaming: low throughput, but low latency

• The Lambda Architecture is the solution!

https://github.com/LamaBigData/lama-demo
The Lambda Architecture

- Generic, scalable and fault-tolerant data processing architecture
- Proposed by Nathan Marz:
  [http://lambda-architecture.net/](http://lambda-architecture.net/)

[Diagram of Lambda Architecture]

[GitHub Link: https://github.com/LamaBigData/lama-demo]
## Batch Layer

<table>
<thead>
<tr>
<th>Technology</th>
<th>Does it fit</th>
<th>Maturity</th>
<th>Ease of use</th>
<th>Language</th>
<th>Platforms</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop MapReduce</td>
<td>★★★★</td>
<td>★★★</td>
<td>★</td>
<td>Java</td>
<td>Hadoop</td>
<td>Very low-level, not re-usable</td>
</tr>
<tr>
<td>Spark</td>
<td>★★★★</td>
<td>★★</td>
<td>★★★</td>
<td>Scala, Java, Python</td>
<td>Spark</td>
<td>In-memory</td>
</tr>
<tr>
<td>Hive</td>
<td>★★★★</td>
<td>★★★★★</td>
<td>★★★</td>
<td>HiveQL, Java</td>
<td>Hadoop</td>
<td>Support planned for Tez</td>
</tr>
<tr>
<td>Spark SQL</td>
<td>★★★★</td>
<td>★</td>
<td>★★</td>
<td>SQL, Scala, Java, Python</td>
<td>Spark</td>
<td>Successor of Shark</td>
</tr>
<tr>
<td>Pig</td>
<td>★★★★</td>
<td>★★★★★</td>
<td>★★★</td>
<td>Pig Latin, Java</td>
<td>Hadoop</td>
<td>Support planned for Tez</td>
</tr>
<tr>
<td>Spork</td>
<td>★★★★</td>
<td>★</td>
<td>★★★</td>
<td>Pig Latin, Java</td>
<td>Spark</td>
<td></td>
</tr>
<tr>
<td>Cascading/Scalding</td>
<td>★★★★</td>
<td>★★</td>
<td>★★</td>
<td>Java, Scala</td>
<td>Hadoop</td>
<td></td>
</tr>
<tr>
<td>Cascalog</td>
<td>★★★★</td>
<td>★</td>
<td>★</td>
<td>Clojure</td>
<td>Hadoop</td>
<td></td>
</tr>
<tr>
<td>Grunch/SCrunch</td>
<td>★★★★</td>
<td>★★</td>
<td>★</td>
<td>Java, Scala</td>
<td>Hadoop</td>
<td>Support planned for Spark and Tez</td>
</tr>
<tr>
<td>Pangool</td>
<td>★★★★</td>
<td>★</td>
<td>★</td>
<td>Java</td>
<td>Hadoop</td>
<td></td>
</tr>
</tbody>
</table>

Table credit: [http://lambda-architecture.net/components/2014-06-30-batch-components/](http://lambda-architecture.net/components/2014-06-30-batch-components/)
## Speed Layer

<table>
<thead>
<tr>
<th>Technology</th>
<th>Does it fit</th>
<th>Maturity</th>
<th>Ease of use</th>
<th>Language</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Storm</td>
<td>★★★</td>
<td>★★★</td>
<td>★</td>
<td>Clojure</td>
<td>originates from Twitter</td>
</tr>
<tr>
<td>Apache Spark Streaming</td>
<td>★★★</td>
<td>★</td>
<td>★★★</td>
<td>Scala/Java/Python</td>
<td>originates from AMPLab</td>
</tr>
<tr>
<td>Apache Samza</td>
<td>★★★</td>
<td>★</td>
<td>★</td>
<td>Scala/Java</td>
<td>originates from LinkedIn</td>
</tr>
<tr>
<td>Apache S4</td>
<td>★★★</td>
<td>★</td>
<td>★</td>
<td>Java</td>
<td>originates from Yahoo!</td>
</tr>
<tr>
<td>Spring XD</td>
<td>★★★</td>
<td>★</td>
<td>★★★</td>
<td>Java</td>
<td>originates from Pivotal</td>
</tr>
</tbody>
</table>

## Cloud-based (XaaS) Offerings

<table>
<thead>
<tr>
<th>Technology</th>
<th>Does it fit</th>
<th>Maturity</th>
<th>Ease of use</th>
<th>API</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Kinesis</td>
<td>★★★</td>
<td>★★★</td>
<td>★</td>
<td>Java</td>
<td>introduced in 11/2013</td>
</tr>
<tr>
<td>Google Cloud Dataflow</td>
<td>★★</td>
<td>-</td>
<td>?</td>
<td>Java</td>
<td>introduced in 06/2014, not yet available</td>
</tr>
</tbody>
</table>

Serving Layer

### Merge/Low-Latency Databases

<table>
<thead>
<tr>
<th>Technology</th>
<th>Does it fit</th>
<th>Maturity</th>
<th>Ease of use</th>
<th>API Language</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElephantDB</td>
<td>★★★</td>
<td>★</td>
<td>★</td>
<td>Clojure</td>
<td></td>
</tr>
<tr>
<td>SploutSQL</td>
<td>★★★</td>
<td>★</td>
<td>★★★</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>Voldemort (with a ReadOnly backend)</td>
<td>★★</td>
<td>★★★</td>
<td>★★★</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>HBase (bulk loading)</td>
<td>★★</td>
<td>★★★</td>
<td>★★★</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>Druid</td>
<td>★★★</td>
<td>★★★</td>
<td>★</td>
<td>Java</td>
<td>originates from Metamarkets</td>
</tr>
</tbody>
</table>
Our Solution: LAMA

- **LAMA**: Lambda Architecture Based Big Data Analytics System
- Based on Twitter’s open sourced **SummingBird** project!

https://github.com/LamaBigData/lama-demo
Content

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
How Does SummingBird Work?

1. Only define job logic once

2. Select a platform
   - Generate topology automatically
   - No need to care about low-level details

3. Select Sources
   - Text/TSV/CSV
   - protobuf
   - memcached
   - redis

4. Select Stores
   - Text/TSV/CSV
   - protobuf
   - memcached
   - redis

https://github.com/LamaBigData/lama-demo
Perspectives

• For batch & streaming, same job logic, different running platform
  – Why? Since both batch and streaming DO mapreduce in different ways

• For iterative data analysis, at least same language & APIs, no need to learn different ones

• DOES not support interactive querying platform, which SHOULD be in SQL-like querying language

https://github.com/LamaBigData/lama-demo
Key Concept in SummingBird: BatchID

- BatchID: Used for job scheduling, data merging, fault tolerance
- Batcher here is used to declare how often this job will run
- Batcher.ofDays(1) means this job will run once every day

BatchID.0  BatchID.1  BatchID.2  BatchID.3

Data Stream

Real-Time  Real-Time  Real-Time  Real-Time

Hadoop  Hadoop  Hadoop  Hadoop

Sum of
RealTime.BatchID.i
+ Hadoop.BatchID.i-1

https://github.com/LamaBigData/lama-demo
Why not directly using SummingBird?

• Immature
  – No job concept (which means you have to take care of job runner by yourself)
  – APIs are hard to use
  – Users have to deal with batchID by themselves
  – Too few input/output formats and databases supported
  – No DAG job supported
  – No deployment, monitoring
  – No support for Spark platform
  – No Support for Google’s GFS/BigQuery/DataStore
  – No support for Google’s Dataflow

• More like a prototype than a product!

https://github.com/LamaBigData/lama-demo
Lama’s Contributions

- Simplified API
- BatchID Management
- More Flexible Inputs and Outputs
- DAG Job Scheduling
- Easy Deployment & Monitoring
- Visualization
- Support Google’s GCS/BigQuery/DataStore/Dataflow
- Support Spark (ready by EOQ1)

https://github.com/LamaBigData/lama-demo
Perspectives

- Based on Twitter’s Summingbird
  - Scala DSL
  - OOP
  - Cascading in low level
- Thanks to Scala DSL, writing MapReduce jobs is like writing native programming codes
- All components of the job can be unit tested

https://github.com/LamaBigData/lama-demo
Contents

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
Scala

- A Scalable language
- Created by Martin Odersky (founder of Typesafe)
- Object-oriented
- Seemless Java interop
- Functions are objects
- Future proof
- Fun
Who Are Using Scala?

- Twitter
- LinkedIn
- Foursquare
- Box
- Quora
- Tumblr
- Git
- Yammer
- ...

https://github.com/LamaBigData/lama-demo
Best Scala Lectures

• Scala school:  
  http://twitter.github.io/scala_school/

• Effective scala:  
  http://twitter.github.io/effectivescala/
Basic Data Structures

• List: An ordered collection of elements
  – scala> val numList = List(1, 2, 3, 4)
• Set: No duplicates
  – scala> val numSet = Set(1, 2)
• Tuple: A group of simple logical collections
  – scala> val record = ("Peter", "male", 30)
• Map: mapping association
  – scala> val wordCount = Map("foo" -> 30)
• Option: A container that may or may not hold sth
  – scala> wordCount.get("foo") (Some(30))
  – scala> wordCount.get("bar") (None)

https://github.com/LamaBigData/lama-demo
Functional Combinators

• **map**: evaluates a function over each element in the list
  
  — scala> numList.map{i: Int => i * 2}

• **flatMap**: combines map and flatten
  
  — scala> numList.map{i: Int => List(i*2, i*2 + 1)}

• **foreach**: similar with map without returning anything
  
  — scala> numList.foreach{i: Int => i * 2}

• **filter**: removes any elements whose logic is false
  
  — scala> numList.filter{i: Int => i % 2 == 0}

• Your own!
  
  — scala> def isEven(i: Int): Boolean = i % 2 == 0
  — scala> def myFilter(num: List[Int], fn: Int => Boolean): List[Int]
  — scala> def myFilter(num: List[Int], fn: Int => Boolean): List[Int] = num.flatMap{i: Int => if (fn(i)) Some(i) else None}

Can you implement it?

https://github.com/LamaBigData/lama-demo
Basics Continued

• Object:
  – It’s used to hold single instances of classes

• Companion object:
  – When an object has the same name with the class
  – You can put your static methods here for easy sharing and unit testing

• Case class
  – It’s used to conveniently store and match on the class contents

• Trait
  – Similar with *interfaces* in Java, but with partial implementations
  – May not have constructor parameters
  – Define object types by specifying the signature of the supported methods.

https://github.com/LamaBigData/lama-demo
Content

• Motivation
• Introduction
• LAMA
• Functional Programming
• Coding Demo

https://github.com/LamaBigData/lama-demo
A LAMA Job Example

• Example: Kevin is an engineer in studio team working on NLP projects. He needs to count word frequencies from a certain period of history related news logs, where each news is a very long article. He has to parse each article, split them into words, and then count number of words. Logs are saved in a Hadoop cluster in cloud. So he wrote a batch job using LAMA as shown in next page

https://github.com/LamaBigData/lama-demo
How to Write a LAMA Job?

• **Source**: This is how you read/transform raw data into your desired format

• **Store**: This is the place where your aggregation logics happen, so define the key, value, and serialized data format here

• **Monoid**: This instructs the store how to aggregate your keys

• **Injection**: This instructs the store how to serialize data

• **Job**: This is the main place where you combine all modules together in order to compute your metrics constantly.

https://github.com/LamaBigData/lama-demo
It supports both batch (Scalding) and streaming (Storm), so just extend it to create your WordCountScaldingJob and WordCountStormJob.

```scala
trait WordCountJob[P <: Platform[P]] extends HTCJob[P] {
  /** The command line argument. */
  val args: Args

  /**
   * Batcher here is used to declare how often this job will run.
   * Batcher.ofDays(1) means this job will run once every day. For streaming job, it is only used for merging.
   */
  override implicit val batcher = Batcher.ofDays(1)
}
```

Specify how to schedule the job running:

https://github.com/LamaBigData/lama-demo
Source

We have implemented most source APIs. You can also customize yours.

```
$ /usr/hadoop/usr_log/2015/01/23/part-000.txt
$ /usr/hadoop/usr_log/2015/01/24/part-000.txt
$ /usr/hadoop/usr_log/2015/01/25/part-000.txt
$ /usr/hadoop/usr_log/2015/01/26/part-000.txt
```
In this example, we tell LAMA how to reduce two ImageFeatures protobuf into one.
Store

$ /usr/hadoop/usr_count/2015/01/23/part-000.txt
$ /usr/hadoop/usr_count/2015/01/24/part-000.txt
$ /usr/hadoop/usr_count/2015/01/25/part-000.txt
$ /usr/hadoop/usr_count/2015/01/26/part-000.txt

Note: In our latest LAMA APIs, store has completely replace source, coz they serves very similar purposes

https://github.com/LamaBigData/lama-demo
Injection

```java
/**
 * Convert the key-value pair to a TSV string, where key is the case class of
 * UserInfor, and value is #actions in Long.
 */

object UserInfoTsvInjection extends Injection[((UserInfo, Long), String)] {
  override def apply(record: (UserInfo, Long)): String = {
    val (key, value) = record
    new StringBuilder(key.toString(StringSeparator))
      .append(StringSeparator)
      .append(value)
      .toString
  }

  override def invert(str: String) = Try {
    str.split(StringSeparator) match {
      case Array(uid, app, dayId, actions) =>
        (UserInfo(uid, app, dayId.toLong), actions.toLong)
    }
  }
}
```

Injection tells LAMA how to serialize/de-serialize key-value pairs

https://github.com/LamaBigData/lama-demo
Business Logics

Just override the job function to implement your own business logics.

https://github.com/LamaBigData/lama-demo
Run in Different Platform?

• Batch (Scalding)

```java
// Your batch job
class WordCountScallingJob(override val args: Args) 
extends WordCountJob[Scalding] with HTScaldingJob {

override val source = Source.text(inputPath)
override val store = 
  Store.jdbc[String, Long, (String, Long)](connection, table)
}
```

• Streaming (Storm)

```java
// Your streaming job
class WordCountStormJob(override val args: Args) 
extends WordCountJob[Storm] with HTCStormJob {

override val source = Source.generator(generator)
override val store = 
  Storehaus.jdbc[String, Long](connection, table).fixedStore
}
```

https://github.com/LamaBigData/lama demo
How about Google’s DataFlow?

• Google’s DataFlow is a managed batch +streaming platform?
• Not free
• LAMA supports it too

```scala
// Your Google dataflow job
class WordCountDataFlowJob(override val args: Args)
  extends WordCountJob[Scalding] with HTCDataFlowJob {
    override val source = Source.bigQuery(input)
    override val sink = Sink.bigQuery(output, schema)
}
```

https://github.com/LamaBigData/lama-demo
Data Processing Engines

• Interactive
  – Impala (open sourced), BigQuery (Google)

• Batch
  – Hadoop MapReduce, Spark, Hive/Pig, Cascading/Scalding

• Streaming
  – Storm, Spark streaming

https://github.com/LamaBigData/lama-demo
Streaming Processing System

• We built a realtime computation system using Storm
  – A distributed realtime computation system
  – Simple, fast, scalable, fault-tolerant, and very reliable
  – Throughput up to 1M tuples processed per second per node
• Events are sent from services through Kafka, which connects with Storm’s bolts

https://github.com/LamaBigData/lama-demo
Supported Data Input Channels

- Kafka
- Kestrel
- Flume
- Majority of key-value stores, such as Memcache, Redis, MongoDB
- MySQL

https://github.com/LamaBigData/lama-demo
Possible Applications

• Realtime analytics
• Online machine learning
• Continuous computation
• Distributed ETL
Visual Analytics

• **Example**: Alice is a PM in HTC studio. One day, it’s urgent for her to get numbers of HTC cell phones sold in Asian countries in past month. Alice uses our interactive querying system to get what she needs by following steps like:
  – Write a SQL-like query and execute it in Impala
  – Wait for several minutes (just some time for drinking a cup of coffee)
  – See visualized reports in Tableau

https://github.com/LamaBigData/lama-demo
Step 1: Write an Impala Query

* In this example, I used Apache Hue as an example of query IDE. We can certainly do the same thing in Tableau but I can’t access it at the time of writing the deck.
Step 2: Visualize Results in Tableau

https://github.com/LamaBigData/lama-demo
Ad-hoc Analytics

• **Example**: Tom is an engineer in HTC studio. One day, he came up with an idea of computing similarity between any two Apps from logs. By similarity, he means normalized common #users. Tom implemented his idea by writing a batch script using Scalding, ran it on one month of history logs, and visualize his app network results using D3.js to get high-level understanding.

https://github.com/LamaBigData/lama-demo
App Similarity Network

Overall Network

Highlights for one App

* Network graph is visualized using D3.js
  https://github.com/LamaBigData/lama-demo
Coding Lab

• Demo repository:
  – https://github.com/LamaBigData/lama-demo

• Q/A contact:
  – Skype account @ LamaBigData

• Lab hour:
  – Mon, 1/26, 19:10~21:00
Environment Setup

• Install Git:

• Install Scala:
  – [http://scala-lang.org/download/2.10.3.html](http://scala-lang.org/download/2.10.3.html)

• Clone the repository:
  
  ```
  $ git clone https://code.google.com/p/lama-demo-spain
  ```

• Compile codes
  
  ```
  $ ./sbt compile
  ```

• Follow README for the rest

https://github.com/LamaBigData/lama-demo