

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

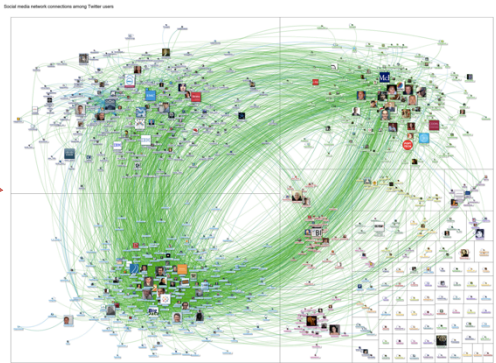
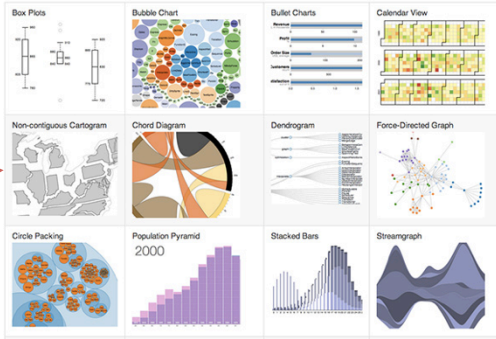
Lecture #3 Outline

- Motivation
- Introduction
- LAMA
- Functional Programming
- Coding Demo

Contents

- Motivation
- Introduction
- LAMA
- Functional Programming
- Coding Demo

Motivation



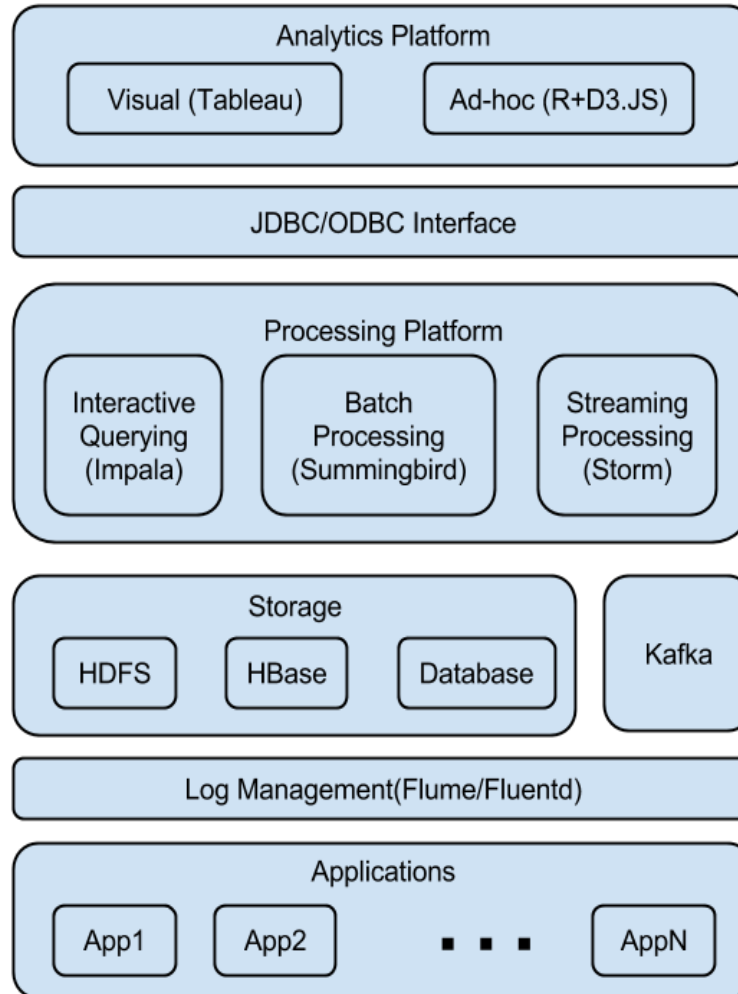
What Do We Need?

- Log Management System:
 - Collect application logs with reasonable latency
- Data processing platform:
 - Interactive, batch, streaming
- Analytics platform:
 - Visual, ad-hoc

Contents

- Motivation
- **Introduction**
- LAMA
- Functional Programming
- Coding Demo

Architecture







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

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)

Different Data Processing Engines

Engine	Open-Source Framework	Properties	Latency	Application
Batch Processing		<ul style="list-style-type: none">• Large data sets• High Throughput	Hours or Days	Hourly/Daily Statistics
Streaming Processing		<ul style="list-style-type: none">• Real-time• In-memory	Milliseconds	Real-time Counting
Interactive Querying		<ul style="list-style-type: none">• SQL-like query• In-memory	Minutes	Ad-Hoc SQL-like Data Analysis
Iterative Data Analysis		<ul style="list-style-type: none">• DAG execution• In-memory	Hours	Machine Learning

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

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

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

Impala Architecture

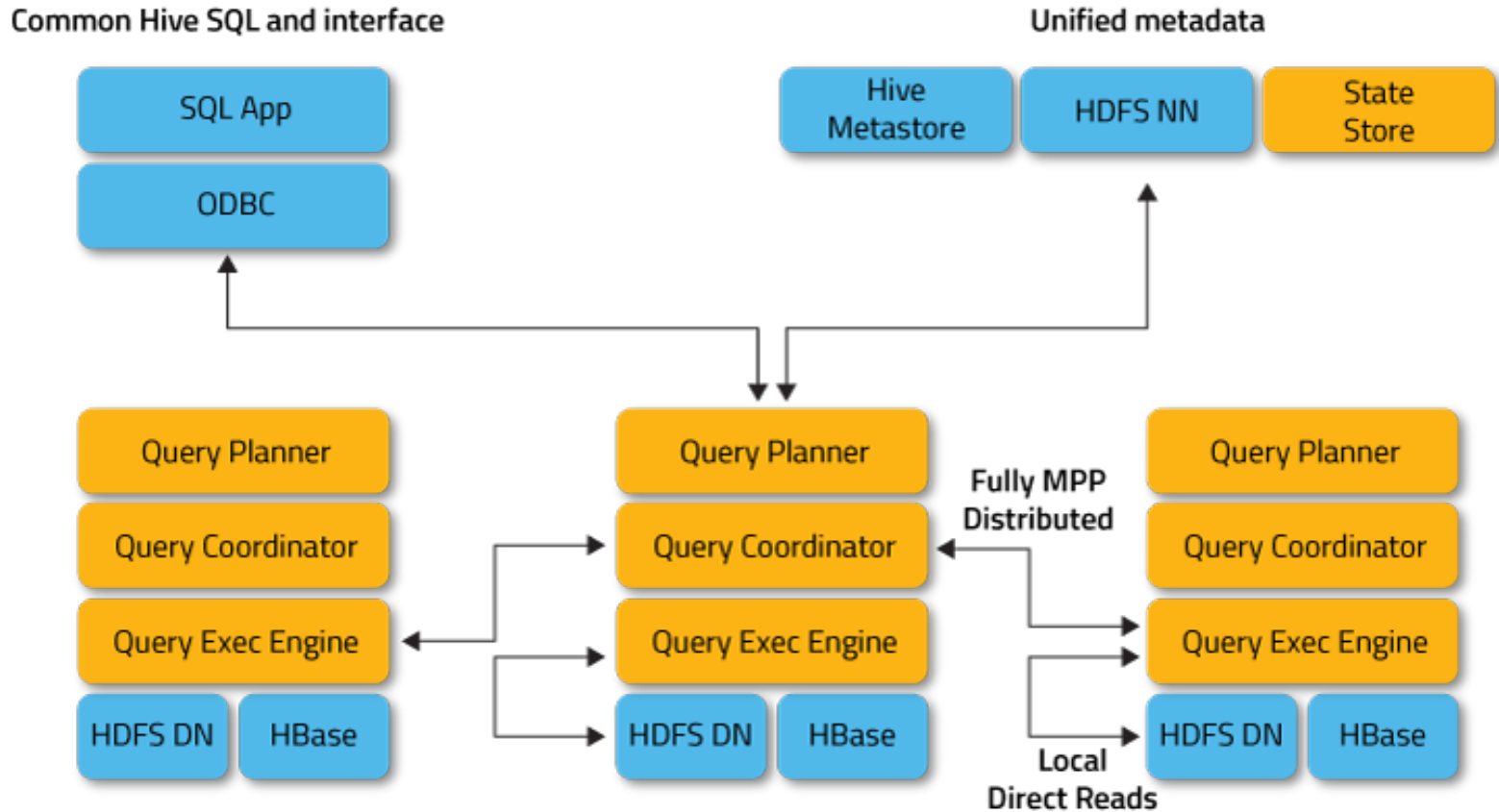


Figure credit: <http://impala.io/overview.html>
<https://github.com/LamaBigData/lama-demo>

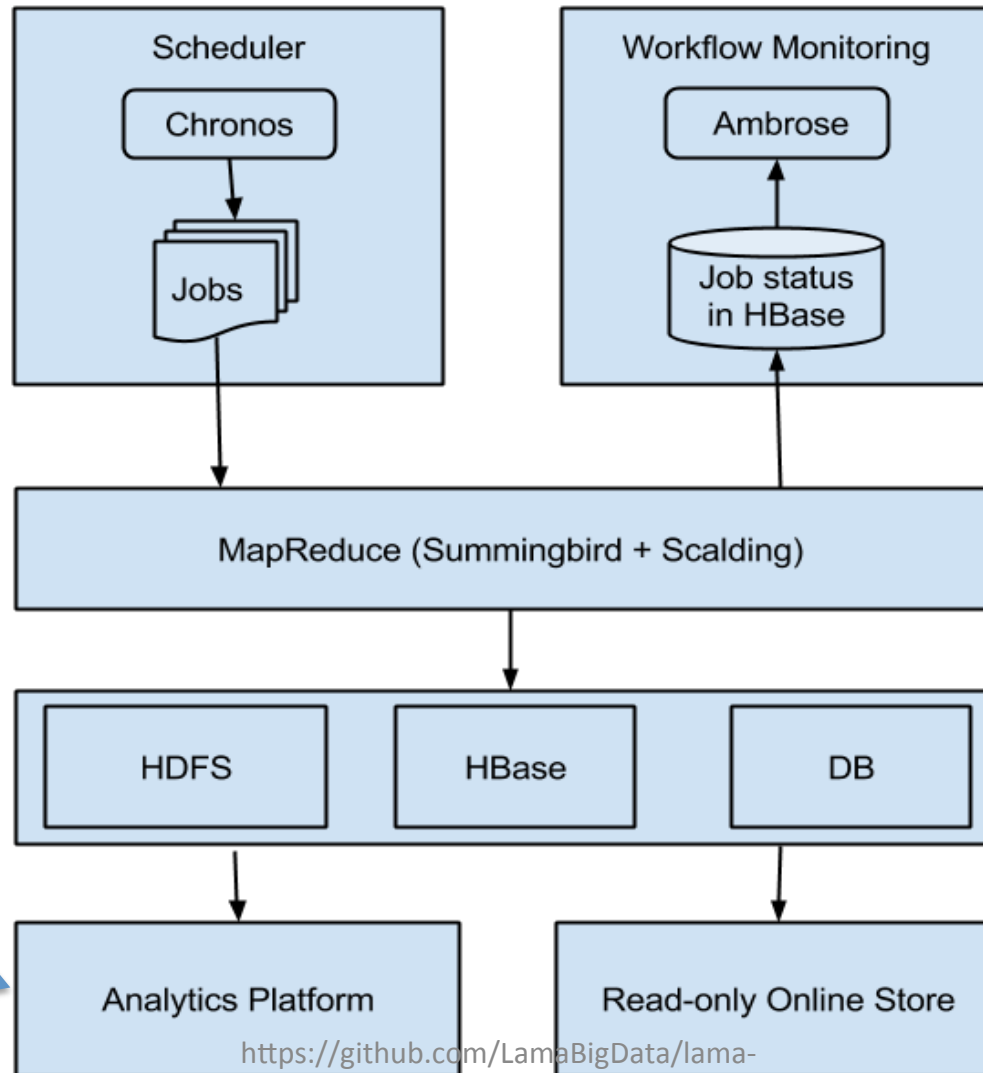
Data Processing Engines

- Interactive
 - Impala (open sourced), BigQuery (Google)
- Batch
 - Hadoop MapReduce, Spark, Hive/Pig, Cascading/Scalding
- Streaming
 - Storm, Spark streaming

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

Batch System Architecture



User can do visual or adhoc analysis on results computed from batch system

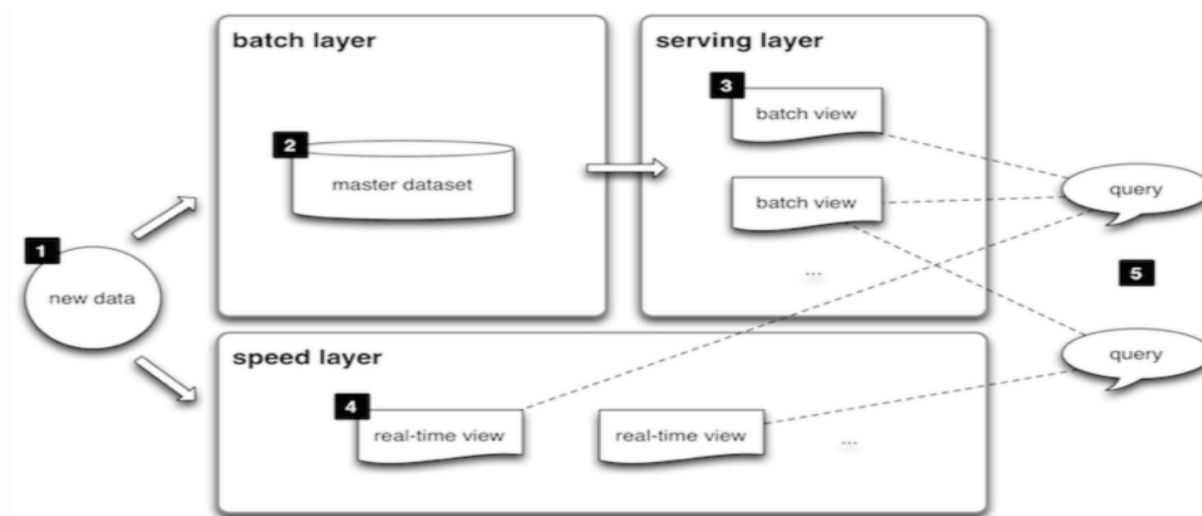
It's also common to upload results computed from batch system to online readonly store, in order for online services to lookup values

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!**

The Lambda Architecture

- Generic, scalable and fault-tolerant data processing architecture
- Proposed by Nathan Marz:
<http://lambda-architecture.net/>



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

Batch Layer

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

Table credit: <http://lambda-architecture.net/components/2014-06-30-batch-components/>

Speed Layer

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

Cloud-based (XaaS) Offerings

Technology	Does it fit	Maturity	Ease of use	API	Comments
AWS Kinesis	★★★	★★	★★	Java	introduced in 11/2013
Google Cloud Dataflow	★★	-	?	Java	introduced in 06/2014, not yet available

Table credit: <http://lambda-architecture.net/components/2014-06-30-speed-components/>

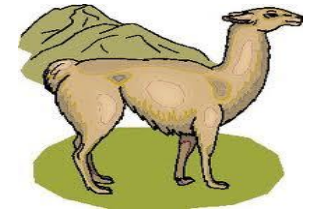
Serving Layer

Merge/Low-Latency Databases

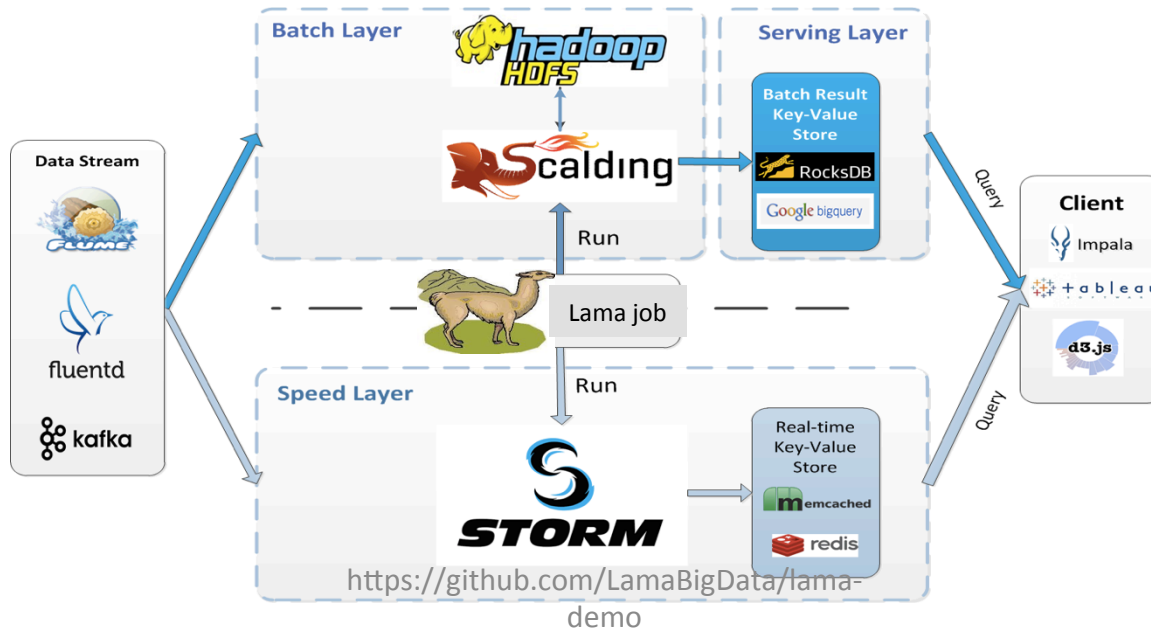
Technology	Does it fit	Maturity	Ease of use	API Language	Comments
ElephantDB	★★★	★	★	Clojure	
SploutSQL	★★★	★	★★	Java	
Voldemort (with a ReadOnly backend)	★★	★★	★★	Java	
HBase (bulk loading)	★★	★★	★★	Java	
Druid	★★★	★★	★	Java	originates from Metamarkets

Table credit: <http://lambda-architecture.net/components/2014-06-30-serving-components/>

Our Solution: LAMA



- LAMA: **Lambda Architecture Based Big Data Analytics System**
- Based on Twitter's open sourced [SummingBird](#) project!



Content

- Motivation
- Introduction
- **LAMA**
- Functional Programming
- Coding Demo

3 Select Sources

4 Select Stores

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

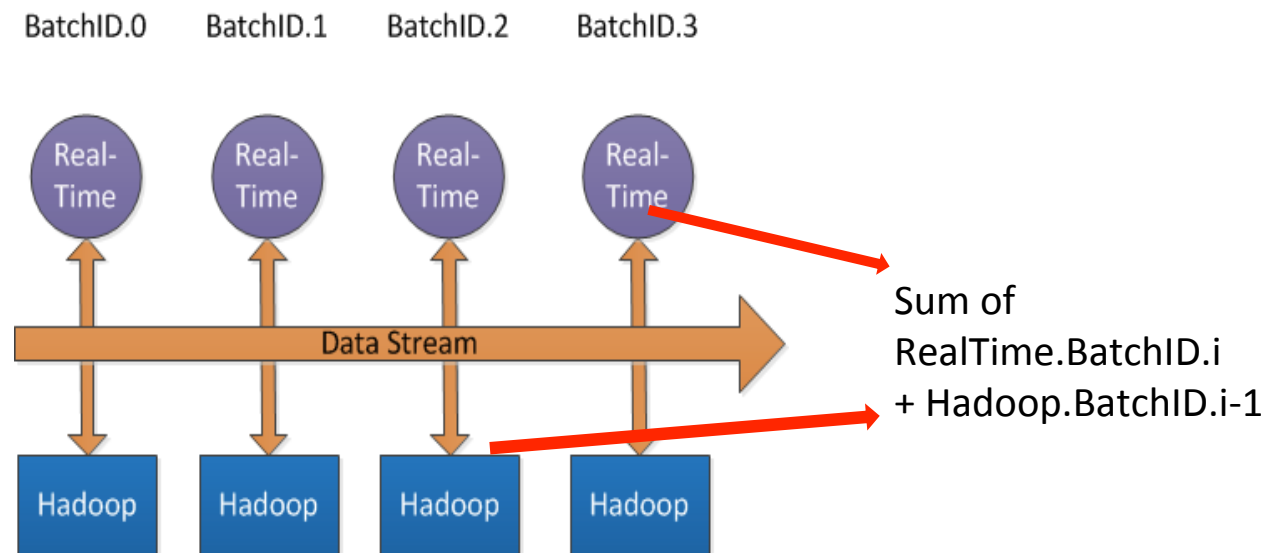
<https://github.com/LamaBigData/lama-demo>

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

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



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!

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)

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

Contents

- Motivation
- Introduction
- LAMA
- **Functional Programming**
- Coding Demo

Scala

- A **Scalable language**
- Created by [Martin Odersky](#) (founder of [Typesafe](#))
- Object-oriented
- Seamless Java interop
- Functions are objects
- Future proof
- Fun

Who Are Using Scala?

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

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)`

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?



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 unittesting
- 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.

Content

- Motivation
- Introduction
- **LAMA**
- Functional Programming
- Coding 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

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.

Job

It supports both batch (Scalding) and streaming (Storm), so just extend it to create your `WordCountScaldingJob` and `WordCountStormJob`

Job details have been abstracted!

```
22▼ trait WordCountJob[P <: Platform[P]] extends HTCJob[P] {  
23   /** The command line argument. */  
24   val args: Args  
25  
26▼  /**  
27   * Batcher here is used to declare how often this job will run.  
28   * Batcher.ofDays(1) means this job will run once every day. For  
29   * streaming job, it is only used for merging.  
30   */  
31   override implicit val batcher = Batcher.ofDays(1)  
32
```

Specify how to
schedule the job
running

Source

```
$ /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
```

```
51 // Define source of data, which are in TelLHTCLog in Tsv format  
52 val source: Source[String] = createSource(batchId =>  
53     new TimePathedTextSource(  
54         inputPath + inputTimePathPattern + "/*",  
55         dateRangeFrom(batchId))  
56     )
```

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

[https://github.com/LamaBigData/lama-
serno](https://github.com/LamaBigData/lama-
serno)

Monoid

```
23 implicit val featureMonoid = new Monoid[ImageFeatures] {
24     override def zero = ImageFeatures.newBuilder().build()
25
26     /**
27      * Merge two ImageFeatures, when one of them is zero, return another one,
28      * else, return merged ImageFeatures
29      * @param m1 ImageFeaures
30      * @param m2 ImageFeaures
31      * @exception CouldNotMergeException ImageFeatures messages could not merge
32      * @return merged ImageFeaures
33      */
34     override def plus(m1: ImageFeatures, m2: ImageFeatures) = {
35         if (!isNonZero(m1)) {
36             m2
37         } else if (!isNonZero(m2)) {
38             m1
39         } else {
40             if (m1.getImageId() != m2.getImageId()) {
41                 throw CouldNotMergeException(m1.getImageId(), m2.getImageId())
42             }
43             ImageFeatures.newBuilder().mergeFrom(m1).mergeFrom(m2).build()
44         }
45     }
46 }
47 }
```

In this example, we tells LAMA how to reduce two ImageFeatures protobuf into one.

Store

```
58 // Define store of aggregated data
59 val store: Store[UserInfo, Long] = new TextLineStore(outputPath)(
60     batcher, UserInfoOrder, UserInfoTsvInjection)
```

```
$ /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 replaced source, coz they serve very similar purposes

Injection

```
14  /**
15   * Convert the key-value pair to a TSV string, where key is the case class of
16   * UserInfor, and value is #actions in Long.
17   */
18  object UserInfoTsvInjection extends Injection[(UserInfo, Long), String] {
19    override def apply(record: (UserInfo, Long)): String = {
20      val (key, value) = record
21      new StringBuilder(key.toString(StringSeparator))
22        .append(StringSeparator)
23        .append(value)
24        .toString
25    }
26
27    override def invert(str: String) = Try {
28      str.split(StringSeparator) match {
29        case Array(uid, app, dayId, actions) =>
30          (UserInfo(uid, app, dayId.toLong), actions.toLong)
31      }
32    }
33  }
```

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

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

Business Logics

```
51▼ /**
52  * Split a sentence to a words array. This function only considers A-Z, a-z,
53  * 0-9 and _ as word characters. Besides split, it also convert all words into
54  * lower case and remove empty words or those start with a number.
55  * @param sentence The input sentence, which is the record type of the source.
56  * @return the list of all words in the sentence, may contain duplicate
57  * words.
58  */
59▼ def toWords(sentence: String): Array[String] = sentence.toLowerCase
60  .replaceAll("[^a-zA-Z0-9\\s]", " ")
61  .split("\\s+")
62  .filter(s => s != "" && !s(0).isDigit)
63
64▼ /**
65  * The actual job logic. Notice that you don't need to specify platform here.
66  * This job will work in any supported platforms.
67  * In this job logic, each sentence from source is splitted to a group of
68  * words with method [[toWords]], then each word is tupled with a "1", which
69  * indicates this word has appeared once.
70  * In sumByKey, the value "1" of the same word (key) will be added together,
71  * and we will have (word, count) tuples.
72  *
73  * @return This summer is used by Summingbird for planning the job.
74  */
75▼ override def job ← source
76  .flatMap { sentence => toWords(sentence).map(_ -> 1L) }
77  .sumByKey(store)
```

Just override the job function to
implement your own business logics

<https://github.com/LamaBigData/lama-deno>

Run in Different Platform?

- Batch (Scalding)

```
26 // Your batch job
27 class WordCountScaldingJob(override val args: Args)
28     extends WordCountJob[Scalding] with HTCScaldingJob {
29
30     override val source = Source.text(inputPath)
31     override val store =
32         Store.jdbc[String, Long, (String, Long)](connection, table)
33 }
```

- Streaming (Storm)

```
17 // Your streaming job
18 class WordCountStormJob(override val args: Args)
19     extends WordCountJob[Storm] with HTCStormJob {
20
21     override val source = Source.generator(generator)
22     override val store =
23         Storehaus.jdbc[String, Long](connection, table).fixedStore
24 }
```


How about Google's DataFlow?

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

```
35 // Your Google dataflow job
36 class WordCountDataFlowJob(override val args: Args)
37     extends WordCountJob[Scalding] with HTCDataFlowJob {
38
39     override val source = Source.bigQuery(input)
40     override val sink = Sink.bigQuery(output, schema)
41 }
```

Data Processing Engines

- Interactive
 - Impala (open sourced), BigQuery (Google)
- Batch
 - Hadoop MapReduce, Spark, Hive/Pig, Cascading/Scalding
- **Streaming**
 - Storm, Spark streaming

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

Supported Data Input Channels

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

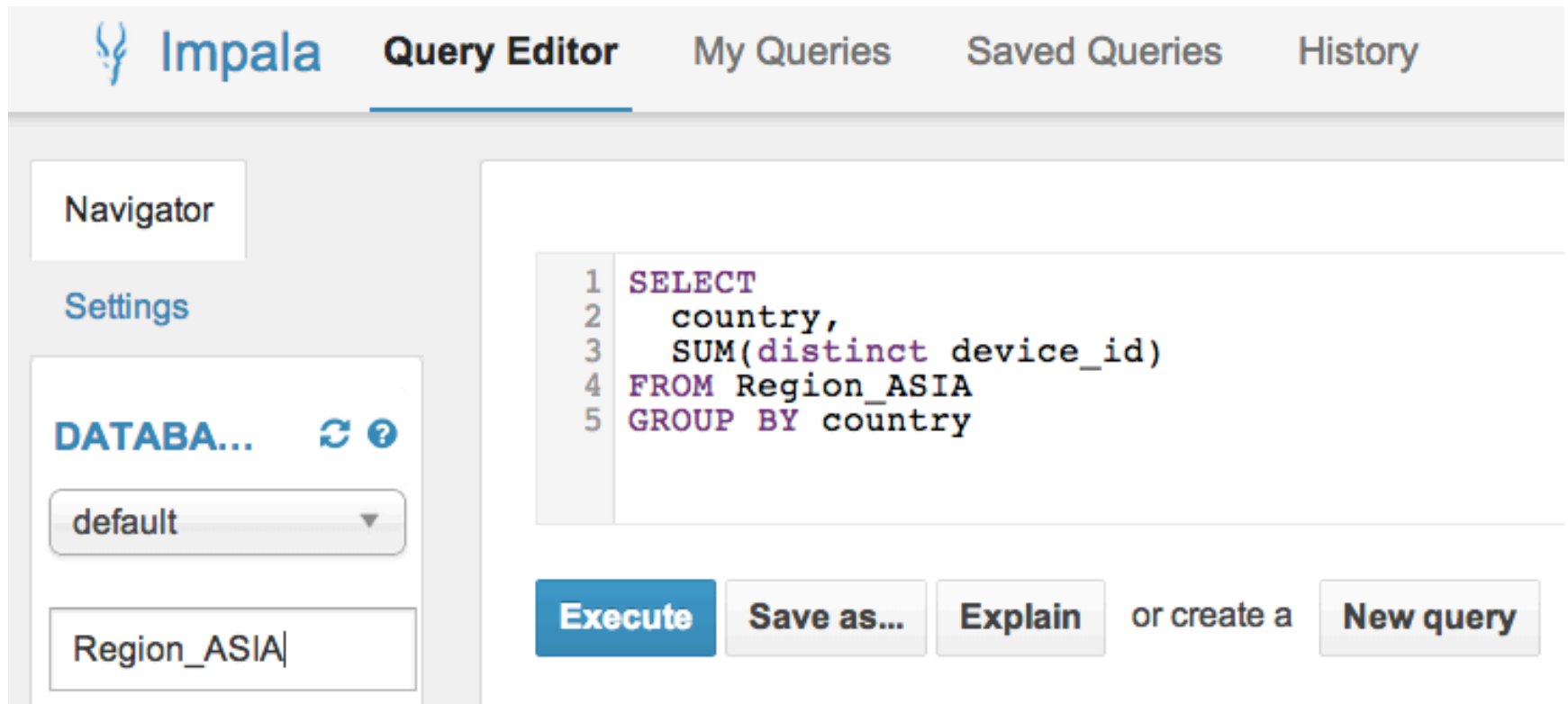
Possible Applications

- 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

Step 1: Write an Impala Query



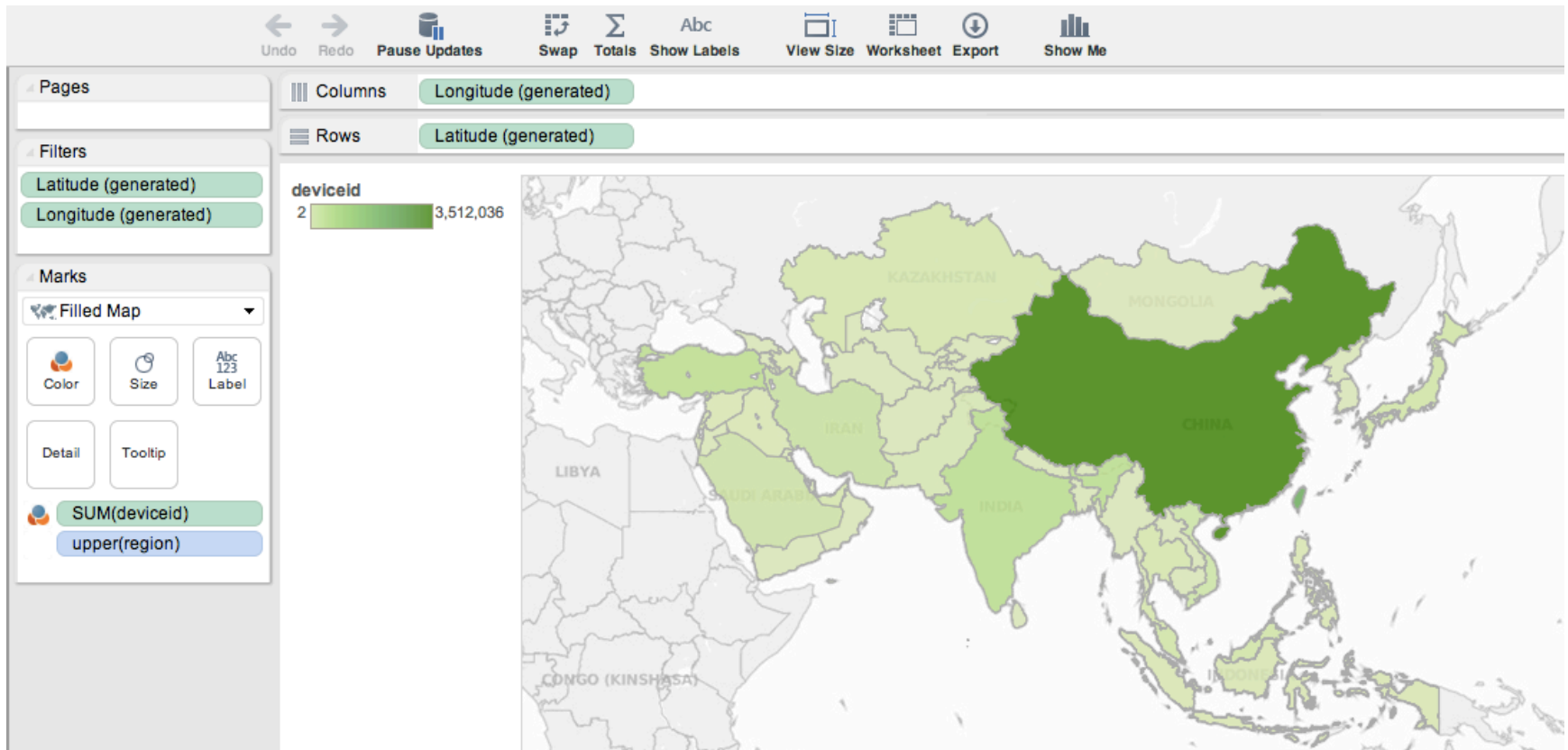
The screenshot shows the Impala Query Editor interface. At the top, there are navigation tabs: "Impala", "Query Editor" (which is active), "My Queries", "Saved Queries", and "History". On the left side, there is a "Navigator" section with "Settings" and a "DATABASE..." dropdown menu currently set to "default". Below that, a text input field contains "Region_ASIA". The main area is a code editor with a SQL query:

```
1 SELECT
2   country,
3   SUM(distinct device_id)
4 FROM Region_ASIA
5 GROUP BY country
```

At the bottom of the interface, there are several buttons: "Execute" (highlighted in blue), "Save as...", "Explain", "or create a", and "New 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



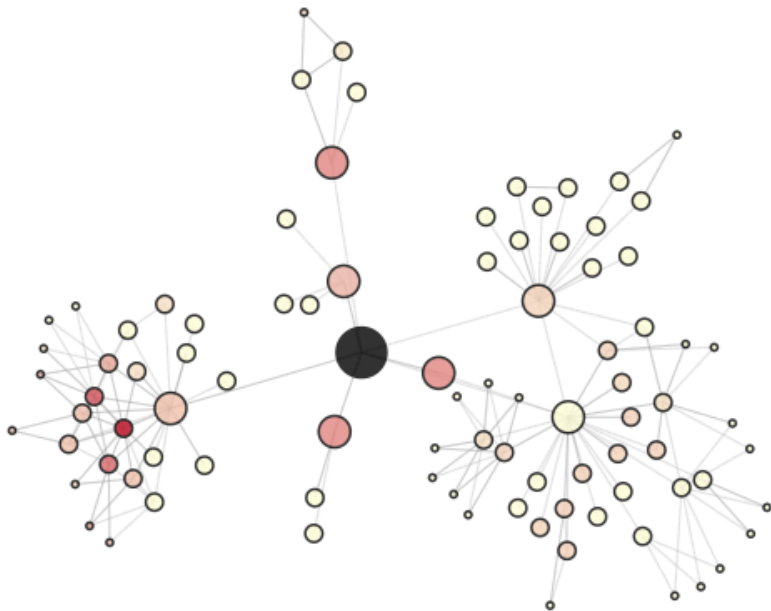
<https://github.com/LamaBigData/lama-demo>

Ad-hoc Analytics

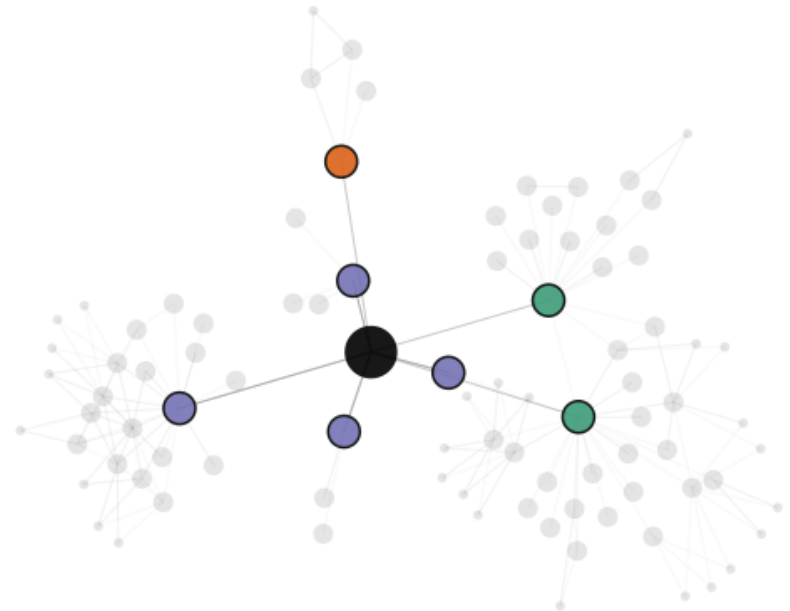
- **Example:** Tom is an engineer in HTC studio. One day, he came up with an idea of computing similarity between any two Apps from logs. By similarity, he means normalized common #users. Tom implemented his idea by writing a batch script using Scalding, ran it on one month of history logs, and visualize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

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:
 - <http://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
- Install Scala:
 - <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