

Apache Spark 소개

Apache Spark란?

조정우

Apache Spark란?

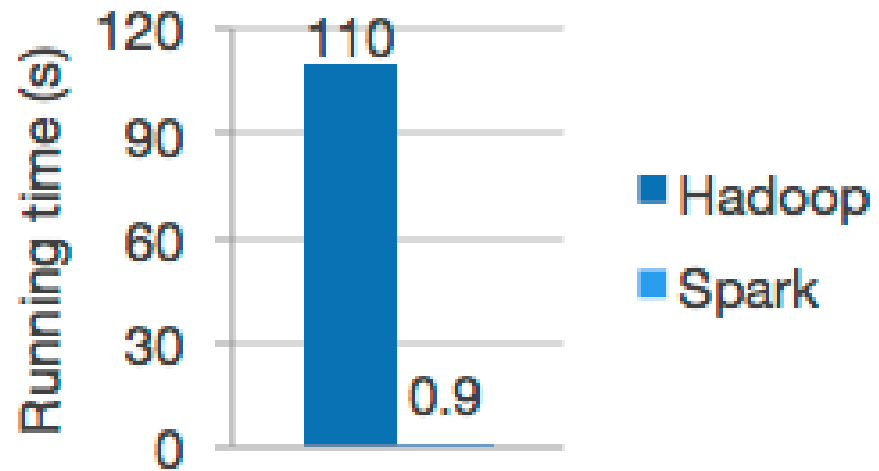
Apache Spark™ is a fast and general engine for large-scale data processing.

- ▶ 빅 데이터 처리 Engine
 - ▶ 데이터 전송/변환/분석
- ▶ 현재 버전 : 1.3.1(at 2015.06.08)
- ▶ Scala 로 개발된 Engine

Apache Spark 특징

- ▶ Speed

- ▶ Hadoop 보다 100배 이상 빠르다.



Logistic regression in Hadoop and Spark

Apache Spark 특징

- ▶ Ease of Use

- ▶ Java, Scala, Python을 이용하여 Application을 빠르게 작성 가능

Ex) Python 으로 WordCount 작성

```
text_file = spark.textFile("hdfs://...")
```

```
text_file.flatMap(lambda line : line.split())  
           .map(lambda word : (word, 1))  
           .reduceByKey(lambda a, b : a + b)
```

Apache Spark 특징

- ▶ Generality

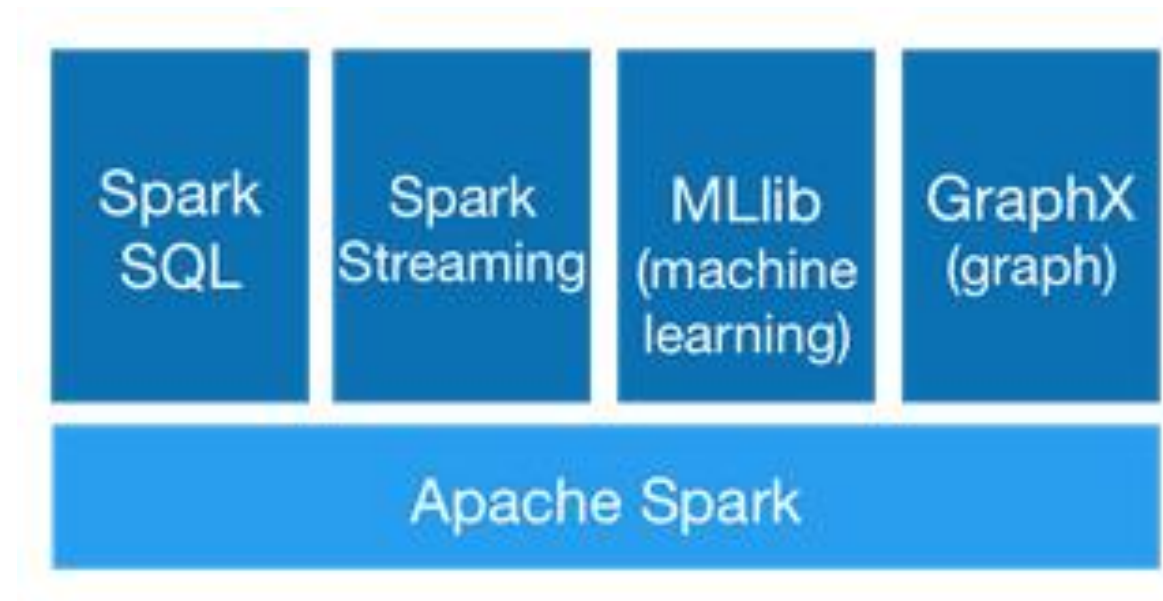
- ▶ **Spark SQL**

- ▶ HIVE 처럼 SQL 과 유사한 형식 지원

- ▶ Mllib

- ▶ Spark Streaming

- ▶ GraphX



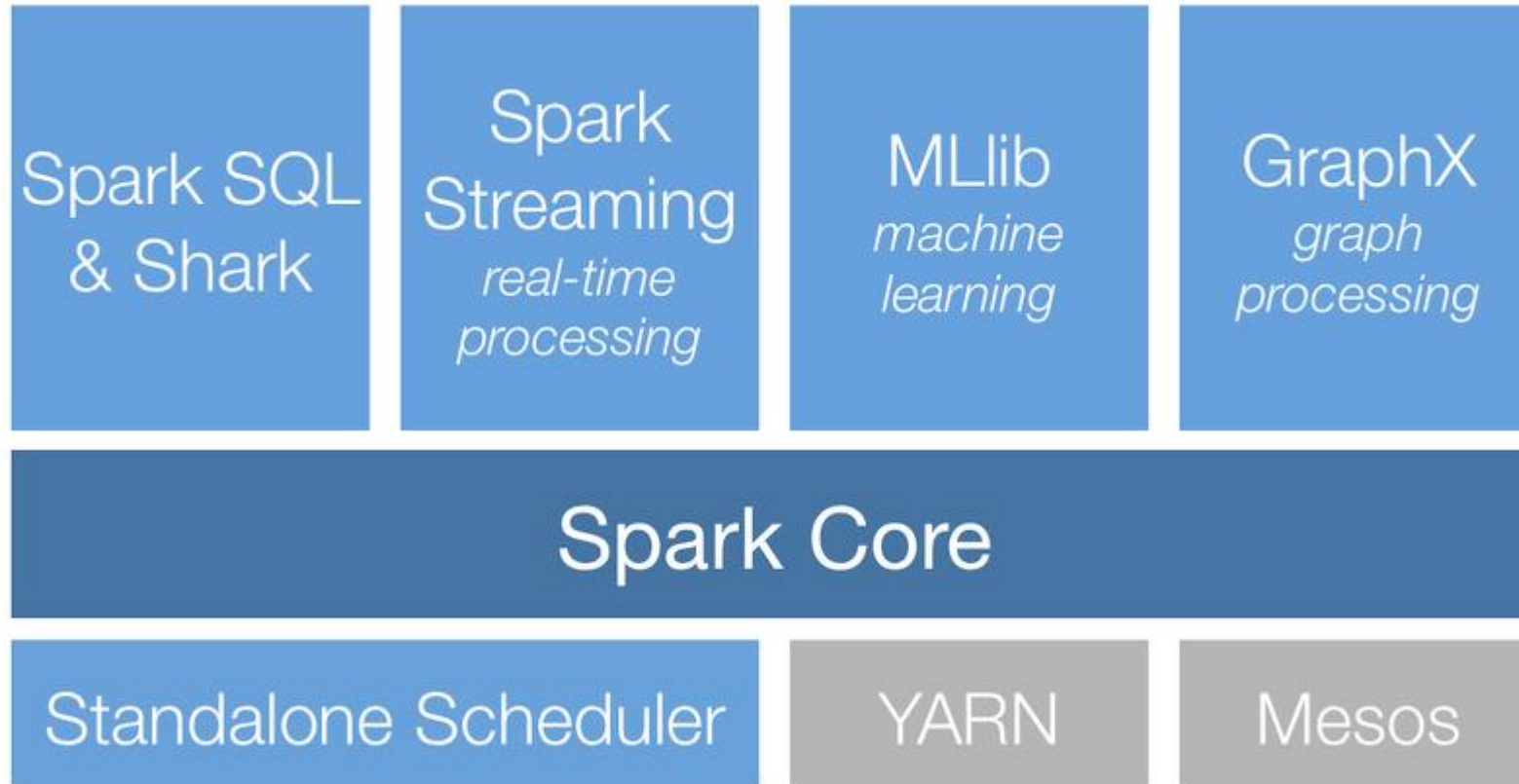
Apache Spark 특징

- ▶ Runs Everywhere
 - ▶ Standalone
 - ▶ EC2
 - ▶ Yarn
 - ▶ Apache Mesos
- ▶ Data Access
 - ▶ HDFS
 - ▶ Cassandra
 - ▶ Hbase
 - ▶ Hive



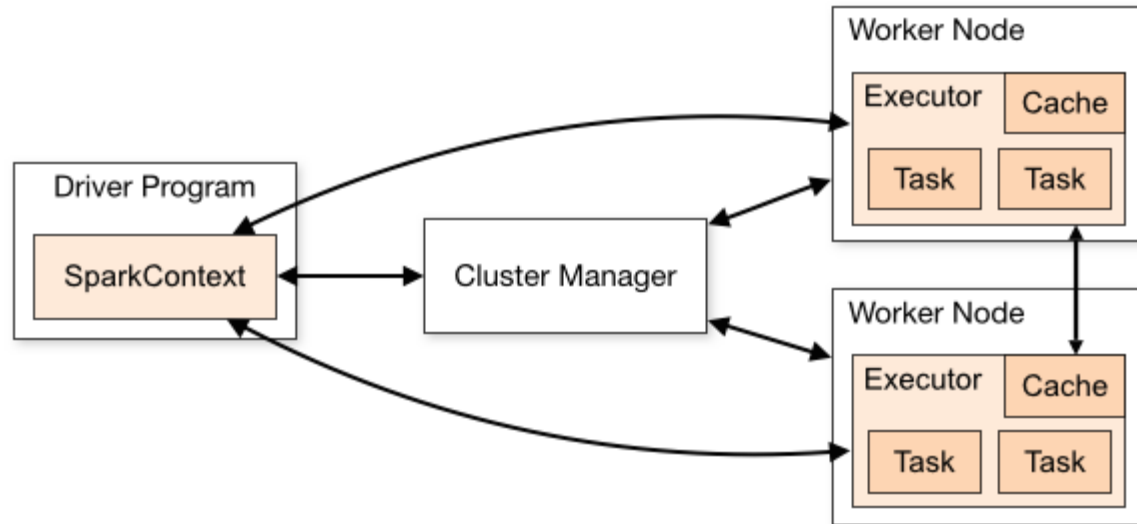
Apache Spark 구조

▶ Apache Spark Stack 구조



Apache Spark 구조

▶ Apache Spark Cluster 구조



1. SparkContext가 ClusterManager에 접속. 이 ClusterManager를 통해 Executor 할당.
2. Executor를 할당 받으면, 각각의 Executor들에게 수행할 코드를 전송.
3. Executor 내에 Task에서 로직 수행.

RDD란?

- ▶ Resilient Distributed Datasets
- ▶ 2012년 NSDI에서 발표
 - ▶ http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
 - ▶ 14장 짜리 논문
 - ▶ USENIX Symposium on Networked Systems Design & Implementation
 - ▶ 통신과 대형 처리 시스템
 - ▶ 2012년 - 빅데이터 세션에서 소개(Best Paper)

RDD 논문 구성

1. Introduction - 왜 RDD를 만들었는가?
2. Resilient Distributed Datasets - RDD 특성과 장점
3. Spark Programming Interface - RDD 예제
4. Representing RDD - RDD는 어떻게 표현되고, Lineage는?
5. Implementation - Job scheduling, Interpreter 통합, Memory관리, Checkpoint 구현
6. Evaluation - 성능평가(VS Hadoop)
7. Discussion - RDD는 제한된 Set임에도 왜 대부분의 프로그램 모델이 표현가능한가?
8. Related Work - RDD가 탄생하기까지 영향을 준 프로젝트
9. Conclusion - 결론

RDD가 탄생한 이유

MapReduce가 빅데이터 분석을 쉽게 만들어주긴 했음.

하지만 뭔가 부족함

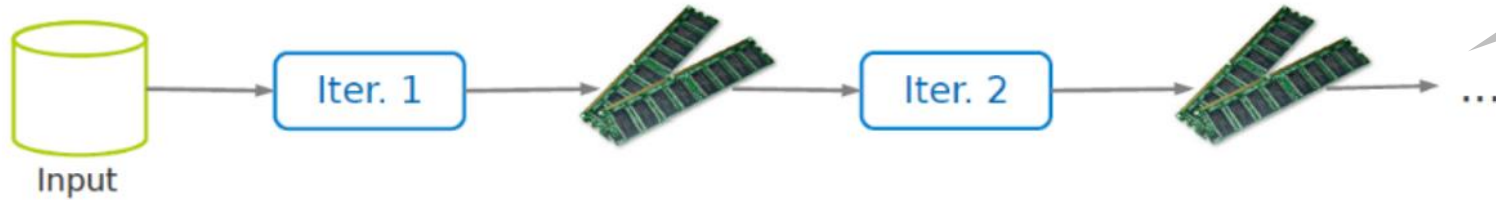
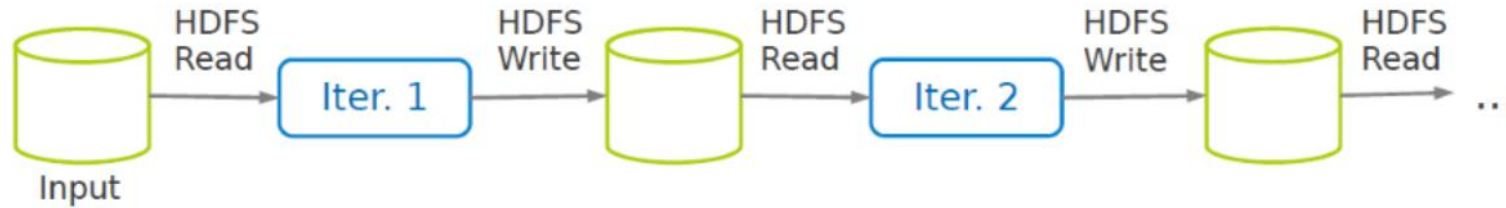
- ▶ 복잡하고, Multi-stage 한 처리 어려움(Machine Learning, Graph)
- ▶ Interactive하고 ad-hoc한 쿼리 어려움

효율적인 Data Sharing 도구가 필요

RDD가 탄생한 이유

▶ MapReduce가 iteration에서 느린 이유

각 iteration을 수행시 stage 간의 데이터 공유가 HDFS 를 이용하기 때문...



Super fast

RDD 문제점?

- ▶ 만약 fault가 발생하면?
Memory에 존재하는 데이터는 유실된다.
- ▶ 기존 Memory 사용 패러다임
 - ▶ Update(fine-grained update)
 - ▶ 데이터 유실을 막으려면 replicating & checkpointing 필요

RDD 문제점 해결!

- ▶ HDFS 는 어떤 파일 시스템인가?
Modify가 되지 않는 파일 시스템
Read-Only 파일 시스템.
- ▶ 그렇다면 Memory도 Read-Only로 사용하자.

This is RDD!!

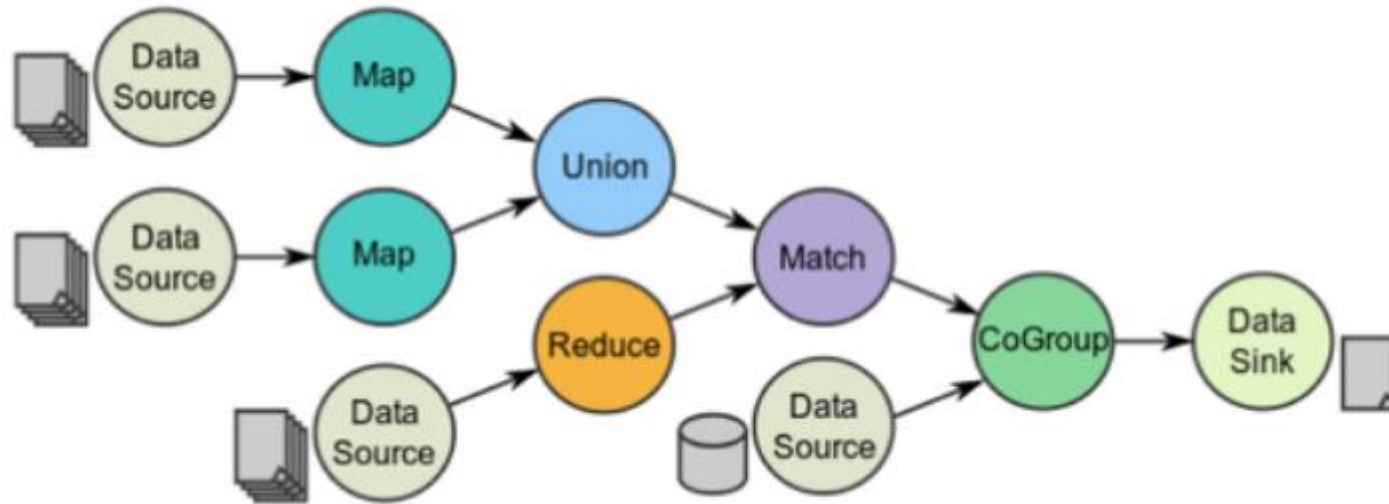
RDD 의 특징

- ▶ Immutable
 - ▶ 생성 후 수정 불가(Read-Only)
 - ▶ 부모(Storage or RDD)로 부터 어찌 만들어졌는지 계보(lineage) 만 기록해도 fault-tolerant
- ▶ Storage → RDD or RDD → RDD 가능

RDD 코딩 방법

실제로 계산되는 작업이 아니라
점점 더 나아가며 lineage 계보를

Directed Acyclic Graph(DAG) 로 디자인 해 나가는 것



RDD Operation

► Transformations & Actions

Transformations	<pre>map(f : T => U) : RDD[T] => RDD[U] filter(f : T => Bool) : RDD[T] => RDD[T] flatMap(f : T => Seq[U]) : RDD[T] => RDD[U] sample(fraction : Float) : RDD[T] => RDD[T] (Deterministic sampling) groupByKey() : RDD[(K, V)] => RDD[(K, Seq[V])] reduceByKey(f : (V, V) => V) : RDD[(K, V)] => RDD[(K, V)] union() : (RDD[T], RDD[T]) => RDD[T] join() : (RDD[(K, V)], RDD[(K, W)]) => RDD[(K, (V, W))] cogroup() : (RDD[(K, V)], RDD[(K, W)]) => RDD[(K, (Seq[V], Seq[W]))] crossProduct() : (RDD[T], RDD[U]) => RDD[(T, U)] mapValues(f : V => W) : RDD[(K, V)] => RDD[(K, W)] (Preserves partitioning) sort(c : Comparator[K]) : RDD[(K, V)] => RDD[(K, V)] partitionBy(p : Partitioner[K]) : RDD[(K, V)] => RDD[(K, V)]</pre>
Actions	<pre>count() : RDD[T] => Long collect() : RDD[T] => Seq[T] reduce(f : (T, T) => T) : RDD[T] => T lookup(k : K) : RDD[(K, V)] => Seq[V] (On hash/range partitioned RDDs) save(path : String) : Outputs RDD to a storage system, e.g., HDFS</pre>

RDD 의 Lazy Execution

```
text_file = spark.textFile("hdfs://...")
```

```
text_file.flatMap(lambda line : line.split())  
          .map(lambda word : (word, 1))  
          .reduceByKey(lambda a, b : a + b)
```

- ▶ Transformations 시에는 실제로 파일을 읽지 않음.
- ▶ Action 수행 시 파일을 읽음.

Lazy Execution 덕분에 Lineage(계보) 를 다 그려놓은 상태에서 수행가능.
즉, 자원이 배치된, 배치될 상황을 미리 고려해서 최적의 상태에서 수행.

Apache Spark 설치

- ▶ Standalone
- ▶ Cluster mode
- ▶ Yarn or Mesos
 - ▶ CPU나 Memory 자원을 관리(Resource Manager)

Apache Spark 수행방법

- ▶ spark-shell(REPL 방식)
 - ▶ 대화형 수행방식
 - ▶ 코드 한 줄씩 작성하여 바로 결과를 볼 수 있음.

- ▶ **spark-class**

- ▶ 상위버전에서 deprecated 예정
- ▶ uMON 수행샘플

```
spark-class org.apache.spark.deploy.yarn.Client  
--jar {수행 Class를 담고 있는 Jar파일 경로}  
--class {수행 Class Full Name}  
--arg {INPUT_PATH}  
--arg {OUTPUT_PATH}
```

- ▶ spark-submit

Apache Zeppelin

▶ <https://zeppelin.incubator.apache.org/>

▶ 특징

▶ Multiple Language 지원

▶ Scala, Python, SparkSQL, Hive 등

▶ Apache Spark Integration

▶ Spark Context나 SQL Context를 생성할 필요 없음.

▶ Data Visualization

▶ Collaboration

▶ 여러 사용자와 작업 공유

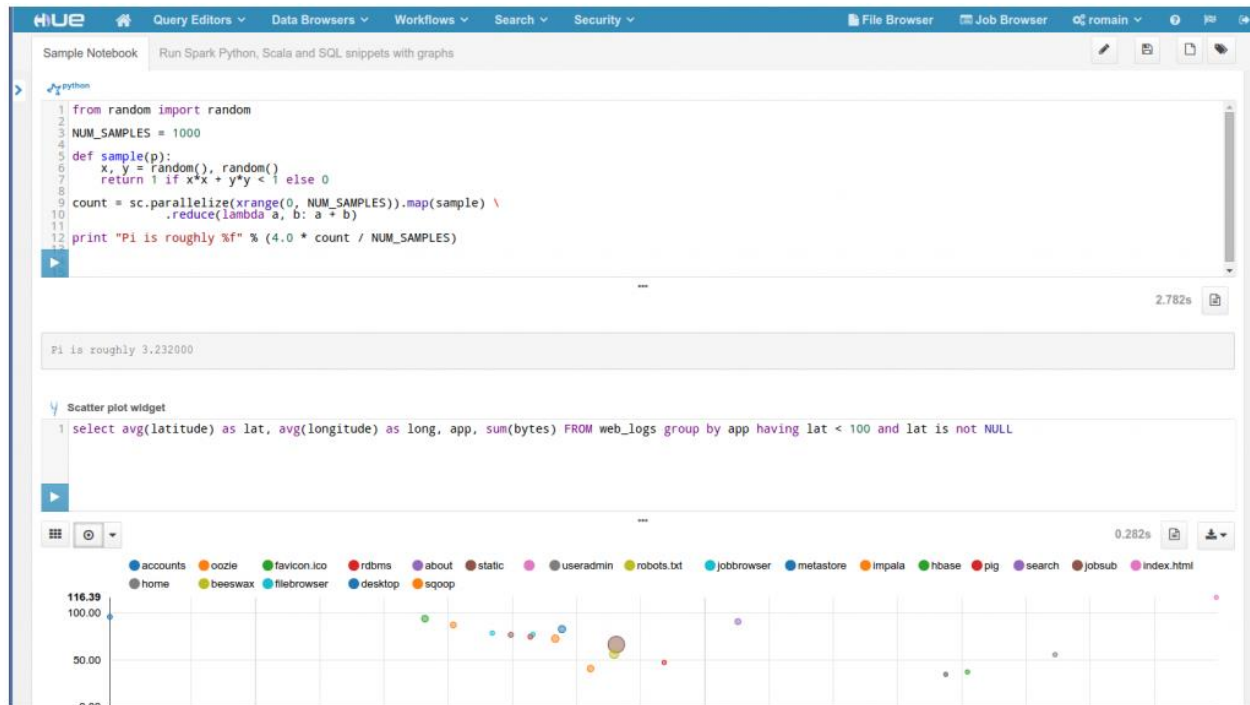
▶ Publish

▶ 데이터 분석 결과물을 URL로 제공하여 사용자 Website에서 Embedded 가능

▶ Open-Source

HUE

- ▶ <http://gethue.com/>
- ▶ 데이터 분석을 위한 Web interface
 - ▶ Spark는 Beta버전



The screenshot displays the HUE web interface. At the top, there is a navigation bar with menus for Query Editors, Data Browsers, Workflows, Search, and Security. Below this, a 'Sample Notebook' is open, showing a Python script for calculating Pi using a Monte Carlo method. The script is executed, and the output is displayed as 'Pi is roughly 3.232000'. Below the script, there is a 'Scatter plot widget' with a SQL query: 'select avg(latitude) as lat, avg(longitude) as long, app, sum(bytes) FROM web_logs group by app having lat < 100 and lat is not NULL'. The scatter plot shows data points for various applications, with a legend at the bottom identifying the colors for each app. The legend includes: accounts, oozie, favicon.ico, rdbms, about, static, useradmin, robots.txt, jobbrowser, metastore, impala, hbase, pig, search, jobsub, index.html, home, beeswax, filebrowser, desktop, and sqoop. The plot shows a distribution of points across the x and y axes, with a vertical axis ranging from 0.00 to 116.39.