Conquering Big Data with BDAS (Berkeley Data Analytics)

Ion Stoica
UC Berkeley / Databricks / Conviva
Extracting Value from Big Data

Insights, diagnosis, e.g.,
» Why is user engagement dropping?
» Why is the system slow?
» Detect spam, DDoS attacks

Decisions, e.g.,
» Decide what features to add to a product
» Personalized medical treatment
» Decide when to change an aircraft engine part
» Decide what ads to show

Data only as useful as the decisions it enables
What do We Need?

Interactive queries: enable faster decisions
  » E.g., identify why a site is slow and fix it

Queries on streaming data: enable decisions on real-time data
  » E.g., fraud detection, detect DDoS attacks

Sophisticated data processing: enable “better” decisions
  » E.g., anomaly detection, trend analysis
Our Goal

Support *batch*, *streaming*, and *interactive* computations…
… in a unified framework

*Easy* to develop *sophisticated* algorithms (e.g., graph, ML algos)
The Berkeley AMPLab

January 2011 – 2017
» 8 faculty
» > 40 students
» 3 software engineer team

Organized for collaboration

AMPCamp3
(August, 2013)

3 day retreats
(twice a year)

220 campers
(100+ companies)
The Berkeley AMPLab

Governmental and industrial funding:

Goal: Next generation of open source data analytics stack for industry & academia: Berkeley Data Analytics Stack (BDAS)
Data Processing Stack

Data Processing Layer

Resource Management Layer

Storage Layer
Hadoop Stack

- HDFS, S3, ...
- Hadoop Yarn
- Hadoop MR
- Storm
- Impala
- Pig
- Hive
- Hadoop MR
BDAS Stack

Spark

BlinkDB

Shark SQL

GraphX

MLBase

MLlib

Spark Streaming

HDFS, S3, …

Mesos

Tachyon
How do BDAS & Hadoop fit together?

- Spark Streaming
- BlinkDB
- Shark SQL
- GraphX
- MLBase
- MLlib

Spark

Hadoop Yarn

HDFS, S3, …
How do BDAS & Hadoop fit together?

Spark
Spark Streaming
BlinkDB
Shark SQL
GraphX
MLbase
ML library
Hive
Pig
Impala
Storm
Hadoop MR
Hadoop Yarn
HDFS, S3, …
How do BDAS & Hadoop fit together?

- Spark
- BlinkDB
- GraphX
- MLbase
- Hive
- Pig
- Impala
- Storm

Hadoop Yarn

HDFS, S3, …
Apache Mesos

Enable multiple frameworks to share same cluster resources (e.g., Hadoop, Storm, Spark)

Twitter’s large scale deployment
   » 10,000+ servers,
   » 500+ engineers running jobs on Mesos

Third party Mesos schedulers
   » AirBnB’s Chronos
   » Twitter’s Aurora

Mesosphere: startup to commercialize Mesos
Apache Spark

Distributed Execution Engine
» Fault-tolerant, efficient in-memory storage
» Low-latency large-scale task scheduler
» Powerful prog. model and APIs: Python, Java, Scala

Fast: up to 100x faster than Hadoop MR
» Can run sub-second jobs on hundreds of nodes

Easy to use: 2-5x less code than Hadoop MR

General: support interactive & iterative apps
Fault Tolerance

Need to achieve

- High throughput reads and **writes**
- Efficient memory usage

Replication

- Writes bottlenecked by network
- Inefficient: store multiple replicas

Persist update logs

- Big data processing can generate massive logs

Our solution: Resilient Distributed Datasets (RDDs)

- Partitioned collection of **immutable** records
- Use **lineage** to reconstruct lost data
RDD Example

Two-partition RDD $A=\{A_1, A_2\}$ stored on disk

1) filter and cache $\rightarrow$ RDD $B$
2) join $\rightarrow$ RDD $C$
3) aggregate $\rightarrow$ RDD $D$
RDD Example

$C_1$ lost due to node failure before reduce finishes
RDD Example

$C_1$ lost due to node failure before reduce finishes

Reconstruct $C_1$, eventually, on different node
Spark Streaming

Existing solutions: *record-by-record* processing

Low latency

Hard to
  » Provide fault tolerance
  » Mitigate straggler
Spark Streaming

Implemented as sequence of micro-jobs (<1s)
- Fault tolerant
- Mitigate stragglers
- Ensure exactly one semantics

Spark & SparkStreaming: batch, interactive, and streaming computations
Shark

Hive over Spark: full support for HQL and UDFs
Up to 100x when input is in memory
Up to 5-10x when input is on disk
Running on hundreds of nodes at Yahoo!
Not Only General, but Fast

Graphs showing performance comparisons:
- **Streaming**: Storm outperforms Spark.
- **Interactive (SQL)**: Hive has the highest response time, followed by Impala (disk), Impala (mem), Shark (mem), and Shark (disk).
- **Batch (ML, Spark)**: Hadoop has the highest time per iteration, followed by Spark.
## Spark Distribution

**Includes**
- Spark (core)
- Spark Streaming
- GraphX (alpha release)
- MLlib

**In the future:**
- Shark
- Tachyon
Explosive Growth

2,500+ Spark meetup users

180+ contributors from 30+ companies

1st Spark Summit
  » 450+ attendees
  » 140+ companies

2nd Spark Summit
  » June 30 – July 2
Explosive Growth

Databricks: founded in 2013 to commercialize Spark Platform

Included in all major Hadoop Distributions
  » Cloudera
  » MapR
  » Hortonworks (technical preview)

Enterprise support: Cloudera, MapR, Datastax

Spark and Shark available on Amazon’s EMR
BlinkDB

Trade between query performance and accuracy using sampling

Why?
» In-memory processing doesn’t guarantee interactive processing
  • E.g., ~10’s sec just to scan 512 GB RAM!
  • Gap between memory capacity and transfer rate increasing
Key Insight

Computations don’t always need **exact** answers

- Input often **noisy**: exact computations do **not** guarantee exact answers

- **Error** often acceptable if **small** and **bounded**

Best scale ± 200g error

Speedometers ± 2.5 % error
(www.edmunds.com)

OmniPod Insulin Pump ± 0.96 % error
(www.ncbi.nlm.nih.gov/pubmed/22226273)
Approach: Sampling

Compute results on samples instead of full data

» Typically, error depends on sample size \((n)\) not on original data size, i.e., error \(\alpha \frac{1}{\sqrt{n}}\)

Can trade between answer’s latency and accuracy and cost
BlinkDB Interface

SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco' AND 'dt=2012-9-2'
WITHIN 1 SECONDS 234.23 ± 15.32
BlinkDB Interface

SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco' AND 'dt=2012-9-2'
WITHIN 2 SECONDS  

\[239.46 \pm 4.96\]

SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco' AND 'dt=2012-9-2'
ERROR 0.1 CONFIDENCE 95.0%
Quick Results

Dataset
  » 365 mil rows, 204GB on disk
  » 600+ GB in memory (deserialized format)

Query: query computing 95-th percentile

25 EC2 instances with
  » 4 cores
  » 15GB RAM
Query Response Time

10x as response time is dominated by I/O
Query Response Time

Error Bars

(0.02\%) (0.07\%) (1.1\%) (3.4\%) (11\%)

Fraction of full data
BlinkDB Overview

**TABLE**

**Original Data**

**Sampling Module**

**Offline-sampling:** Optimal set of samples across different dimensions (columns or sets of columns) to support ad-hoc exploratory queries
BlinkDB Overview

Sample Placement: Samples striped over 100s or 1,000s of machines both on disks and in-memory.
BlinkDB Overview

HiveQL/SQL Query

SELECT foo (*)
FROM TABLE
WITHIN 2

Query Plan
Sample Selection

Sample Selection

Sampling Module

Original Data

TABLE

On-Disk Samples

In-Memory Samples

In-Memory Samples

On-Disk Samples
**BlinkDB Overview**

HiveQL/SQL Query

Original Data

Sampling Module

Query Plan

Sample Selection

Online sample selection to pick best sample(s) based on query latency and accuracy requirements

On-Disk Samples

In-Memory Samples

SELECT foo (*) FROM TABLE WITHIN 2
BlinkDB Overview

HiveQL/SQL Query

Original Data

New Query Plan

Sample Selection

Error Bars & Confidence Intervals

Hadoop/Spark

Result

182.23 ± 5.56
(95% confidence)

Parallel query execution on multiple samples striped across multiple machines.
BlinkDB Challenges

Which set of samples to build given a storage budget?

Which sample to run the query on?

How to accurately estimate the error?
BlinkDB Challenges

Which set of samples to build given a storage budget?

Which sample to run the query on?

How to accurately estimate the error?
How to Accurately Estimate Error?

Close formulas for limited number of operators
  » E.g., count, mean, percentiles

What about user defined functions (UDFs)?
  » Use bootstrap technique
**Bootstrap**

Quantify accuracy of a sample estimator, $f()$

- Distribution $X$ → $f(X)$
  - can’t compute $f(X)$ as we don’t have $X$

- Random sample

  - $|S| = N$
  - $S$ → $f(S)$
    - what is $f(S)$’s error?

- Sampling with replacement

  - $S_1$ → $f(S_1)$
    - use $f(S_1)$, ..., $f(S_K)$ to compute
      - estimator: $\text{mean}(f(S_i))$
      - error, e.g.: $\text{stdev}(f(S_i))$

  - $|S_i| = N$

  - ...
Bootstrap for BlinkDB

Quantify accuracy of a query on a sample table

Original Table $T$ $\implies Q(T)$ $Q(T)$ takes too long!

Sample $S$, $|S| = N$ $\implies Q(S)$ what is $Q(S)$’s error?

Sampling with replacement

$S_1$ $\implies Q(S_1)$

$\vdots$

$S_k$ $\implies Q(S_k)$

$|S_i| = N$

Use $Q(S_1), \ldots, Q(S_k)$ to compute estimator quality assessment, e.g., $\text{stdev}(Q(S_i))$
How Do You Know Error Estimation is Correct?

Assumption: $f()$ is Hadamard differentiable
   » How do you know an UDF is Hadamard differentiable?
   » Sufficient, not necessary condition

Only **approximations** of true error distribution (true for closed formula as well)

Previous work doesn’t address error estimation correctness
How Bad it Is?

Workloads
» Conviva: 268 real-world 113 had custom User-Defined Functions
» Facebook

Closed Forms/Bootstrap fails for
» 3 in 10 Conviva Queries
» 4 in 10 Facebook Queries

Need runtime diagnosis!
Error Diagnosis

Compare bootstrapping with ground truth for small samples

Check whether error improves as sample size increases
Ground Truth (Approximation)

<table>
<thead>
<tr>
<th>$T$</th>
<th>Original Table</th>
</tr>
</thead>
</table>

Ground Truth (Approximation)

\[ \hat{\xi} = \text{stddev}(Q(S_j)) \]

Estimator quality assessment
Ground Truth and Bootstrap

$T$

Original Table

Bootstrap on Individual Samples

$S_1$

$S_{11}$ $\rightarrow Q(S_{11})$

$\cdots$

$S_{1k}$ $\rightarrow Q(S_{1k})$

$
\xi^*_1 = \text{stdev}(Q(S_{1j}))
$

$S_p$

$S_{p1}$ $\rightarrow Q(S_{p1})$

$\cdots$

$S_{pk}$ $\rightarrow Q(S_{pk})$

$
\xi^*_p = \text{stdev}(Q(S_{pj}))
$
Ground Truth vs. Bootstrap

\[ \tilde{\xi}_i = \text{mean}(\xi_{ij}) \]

\[ \xi^*_{i1} = \text{std}(Q(S_{i1j})) \ldots \xi^*_{ip} = \text{std}(Q(S_{ipj})) \]

Relative Error

Sample Size

\( n_1 \quad n_2 \quad n_3 \)

Bootstrap

Ground Truth
Ground Truth vs. Bootstrap

\[ \tilde{\xi}_i = \text{mean}(\xi_{ij}) \]

\[ \xi^*_i = \text{stddev}(Q(S_{ij})) \]

\[ \xi^*_p = \text{stddev}(Q(S_{pj})) \]
Ground Truth vs. Bootstrap

\[ \hat{\xi}_i = \text{mean}(\xi_{ij}) \]

\[ \xi^{*}_{i1} = \text{stddev}(Q(S_{i1j})) \]

\[ \xi^{*}_{ip} = \text{stddev}(Q(S_{ipj})) \]
How Well Does it Work in Practice?

Evaluated on Conviva Query Workload

Diagnostic predicted that 207 (77%) queries can be approximated
  » False Negatives: 18
  » False Positives: 3 (conditional UDFs)
Overhead

Bootstrap and Diagnostic overheads can be very large
  » Diagnostics requires to run 30,000 small queries!

Optimization
  » Pushdown filter
  » One pass execution

Can reduce overhead by orders of magnitude
Query + Bootstrap + Diagnosis

Query Response Time

- Diagnosis overhead
- Bootstrap overhead
- Query Response Time

1020 seconds

Fraction of full data

1 10^{-1} 10^{-2} 10^{-3} 10^{-4} 10^{-5}

Bootstrap + Diagnosis 53x more than query itself!
Optimization: Filter Pushdown

Perform filtering before resampling
  » Can dramatically reduce I/O

Assume $n \ll N$
Optimization: One Pass Exec.

For each resample add new column
  » Specify how many times each row has been selected
  » Query generate results for each resamples in one pass

Compute all k results in one pass!

Poissoned resampling: construct all k resamples in one pass!
Query + Bootstrap + Diagnosis
(with Filter Pushdown and Single Pass)

Query Response Time

- Diagnosis overhead
- Bootstrap overhead
- Query Response Time

Less than 70% overhead (80x reduction)
Open Source

BlinkDB is open-sourced and released at http://blinkdb.org
Open Source

Used regularly by 100+ engineers at Facebook Inc.
Lesson Learned

Focus on novel usage scenarios

Be paranoid about simplicity
  » Very hard to build real complex systems in academia

Basic functionality first, performance opt. next
Example: Mesos

Focus on novel usage scenarios
  » Let multiple frameworks share same cluster

Be paranoid about simplicity
  » Just enforce allocation policy across frameworks, e.g., fair sharing
  » Let frameworks decide which slots they accept, which tasks to run, and when to run them
  » First release: 10K code

Basic functionality first, performance opt. next
  » First, support arbitrary scheduling, but inefficient
  » Latter added filters to improve performance
Example: Spark

Focus on novel usage scenarios
  » Interactive queries and iterative (ML) algorithms

Be paranoid about simplicity
  » Immutable data; avoid complex consistency protocols
  » First release: 2K code

Basic functionality first, performance opt. next
  » First, no automatic check-pointing
  » Latter to add automatic checkpoint
Example: BlinkDB

Focus on novel usage scenarios
  » Approximate computations; error diagnosis

Be paranoid about simplicity
  » Use bootstrap as generic technique; no support for close formula

Basic functionality first, performance opt. next
  » First, straightforward error estimation, very expensive
  » Latter, optimizations to reduce overhead
  » First, manual sample generation
  » Later, automatic sample generation (to do)
Summary

BDAS: address next Big Data challenges

Unify batch, interactive, and streaming

Enable users to trade between
  » Response time, accuracy, and cost

Explosive adoption
  » 30+ companies, 180+ individual contributors (Spark)
  » Spark platform included in all major Hadoop distros
  » Enterprise grade support and professional services for both Mesos and Spark platform