Berkeley Data Analytics Stack (BDAS)
Overview

Ion Stoica
UC Berkeley
March 7, 2013

What is Big Data used For?

• Reports, e.g.,
  – Track business processes, transactions
• Diagnosis, e.g.,
  – Why is user engagement dropping?
  – Why is the system slow?
  – Detect spam, worms, viruses, DDoS attacks
• Decisions, e.g.,
  – Decide what feature to add
  – Decide what ad to show
  – Block worms, viruses, ...

Data is only as useful as the decisions it enables
Data Processing Goals

- **Low latency (interactive) queries on historical data**: enable faster decisions
  - E.g., identify why a site is slow and fix it

- **Low latency queries on live data (streaming)**: enable decisions on real-time data
  - E.g., detect & block worms in real-time (a worm may infect 1mil hosts in 1.3sec)

- **Sophisticated data processing**: enable “better” decisions
  - E.g., anomaly detection, trend analysis

Today’s Open Analytics Stack...

- ..mostly focused on large on-disk datasets: great for sophisticated **batch** applications, but **slow**
Goals

- Easy to combine batch, streaming, and interactive computations
- Easy to develop sophisticated algorithms
- Compatible with existing open source ecosystem (Hadoop/HDFS)

Our Approach: Support Interactive and Streaming Comp.

- Aggressive use of memory
- Why?
  1. Memory transfer rates >> disk or even SSDs
  - Gap is growing especially w.r.t. disk
  2. Many datasets already fit into memory
  - The inputs of over 90% of jobs in Facebook, Yahoo!, and Bing clusters fit into memory
  - E.g., 1TB = 1 billion records @ 1 KB each
  3. Memory density (still) grows with Moore’s law
  - RAM/SSD hybrid memories at horizon

High end datacenter node

<table>
<thead>
<tr>
<th>Data transfer rates</th>
<th>Capacity</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2-1GB/s (x10 disks)</td>
<td>10-30TB</td>
<td>16 cores</td>
</tr>
<tr>
<td>1-4GB/s (x4 disks)</td>
<td>1-4TB</td>
<td></td>
</tr>
<tr>
<td>40-60GB/s</td>
<td>128-512GB</td>
<td></td>
</tr>
<tr>
<td>10Gbps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Approach: Support Interactive and Streaming Comp.

- Increase *parallelism*
- Why?
  - Reduce work per node → improve latency

- Techniques:
  - Low latency parallel scheduler that achieve high locality
  - Optimized parallel communication patterns (e.g., shuffle, broadcast)
  - Efficient recovery from failures and straggler mitigation

Our Approach: Support Interactive and Streaming Comp.

- Trade between result *accuracy* and *response times*
- Why?
  - In-memory processing does not guarantee interactive query processing
    - E.g., ~10’s sec just to scan 512 GB RAM!
    - Gap between memory capacity and transfer rate increasing
- Challenges:
  - Accurately estimate error and running time for…
  - … arbitrary computations
Our Approach

- **Easy** to combine batch, streaming, and interactive computations
  - Single execution model that supports all computation models

- **Easy** to develop sophisticated algorithms
  - Powerful Python and Scala shells
  - High level abstractions for graph based, and ML algorithms

- **Compatible** with existing open source ecosystem (Hadoop/HDFS)
  - Interoperate with existing storage and input formats (e.g., HDFS, Hive, Flume, ..)
  - Support existing execution models (e.g., Hive, GraphLab)

Berkeley Data Analytics Stack (BDAS)

- Application
  - New apps: AMP-Genomics, Carat, ...

- Data Processing
  - in-memory processing
  - trade between time, quality, and cost

- Data Management

- Resource Management
  - Efficient data sharing across frameworks
  - Share infrastructure across frameworks (multi-programming for datacenters)
The Berkeley AMPLab

- "Launched" January 2011: 6 Year Plan
  - 8 CS Faculty
  - ~40 students
  - 3 software engineers
- Organized for collaboration:

The Berkeley AMPLab

- Funding:
  - DARPA data, NSF Expedition Grant
  - Industrial, founding sponsors
  - 18 other sponsors, including

Goal: next Generation of open source analytics stack for industry & academia:
- Berkeley Data Analytics Stack (BDAS)
Berkeley Data Analytics Stack (BDAS)

- Existing stack components...

- Data Processing
- Data Management
- Resource Management
Mesos [Released, vo.9]

- Management platform that allows multiple framework to share cluster
- Compatible with existing open analytics stack
- Deployed in production at Twitter on 3,500+ servers

One Framework Per Cluster Challenges

- Inefficient resource usage
  - E.g., Hadoop cannot use available resources from MPI’s cluster
  - No opportunity for stat. multiplexing
- Hard to share data
  - Copy or access remotely, expensive
- Hard to cooperate
  - E.g., Not easy for MPI to use data generated by Hadoop

Need to run multiple frameworks on same cluster
Solution: Mesos

- Common resource sharing layer
  - abstracts (“virtualizes”) resources to frameworks
  - enable diverse frameworks to share cluster

Dynamic Resource Sharing

- 100 node cluster
Spark [Release, v0.7]

- In-memory framework for **interactive** and **iterative** computations
  - Resilient Distributed Dataset (RDD): fault-tolerance, in-memory storage abstraction
- Scala interface, Java and Python APIs

Our Solution

- **Resilient Distributed Data Sets (RDD)**
  - Partitioned collection of records
  - Immutable
  - Can be created only through deterministic operations from other RDDs
- Handle of each RDD stores its **lineage**:
  - Lineage: sequence of operations that created the RDD
- Recovery: use lineage information to rebuild RDD
RDD Example

- Two-partition RDD $A = \{A_1, A_2\}$ stored on disk
  1) Apply $f()$ and cache → RDD $B$
  2) Shuffle, and apply $g()$ → RDD $C$
  3) Aggregate using $h()$ → $D$

\[
A_1 \xrightarrow{f()} B_1 \xrightarrow{g()} C_1 \xrightarrow{h()} D
\]

\[
A_2 \xrightarrow{f()} B_2 \xrightarrow{g()} C_2
\]

- $C_1$ lost due to node failure before $h()$ is computed

\[
A_1 \xrightarrow{f()} B_1 \xrightarrow{g()} C_1 \xrightarrow{h()} D
\]

\[
A_2 \xrightarrow{f()} B_2 \xrightarrow{g()} C_2
\]

\[
D = C.h()
\]
RDD Example

- $C_1$ lost due to node failure before $h()$ is computed
- Reconstruct $C_1$, eventually, on a different node

![Diagram of RDD example]

PageRank Performance

![Bar chart showing PageRank performance comparison between Hadoop and Spark]
Other Iterative Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time per Iteration (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Means Clustering</td>
<td>4.1</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>110</td>
</tr>
</tbody>
</table>

Spark Community

- 3000 people attended online training in August
- 500+ meetup members
- 14 companies contributing
Spark Streaming [Alpha Release]

- Large scale streaming computation
- Ensure exactly one semantics
- Integrated with Spark → unifies *batch, interactive, and streaming* computations!

<table>
<thead>
<tr>
<th>Spark Streaming</th>
<th>HIVE</th>
<th>Pig</th>
<th>Storm</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark Streaming</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Existing Streaming Systems

- Traditional streaming systems have a event-driven **record-at-a-time** processing model
  - Each node has mutable state
  - For each record, update state & send new records

- State is lost if node dies!

- Making stateful stream processing be fault-tolerant is challenging
Spark: Discretized Stream Processing

Run a streaming computation as a series of very small, deterministic batch jobs

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches
Performance

Can process 6 GB/sec (60M records/sec) of data on 100 nodes at sub-second latency

- Tested with 100 streams of data on 100 EC2 instances with 4 cores each

Comparison with Storm and S4

Higher throughput than Storm

- Spark Streaming: 670k records/second/node
- Storm: 115k records/second/node
- Apache S4: 7.5k records/second/node
Fast Fault Recovery

Recovering from faults/stragglers within 1 sec

[Chart showing interval processing time over time with a peak at 30 seconds indicating failure happens]

Sliding WordCount on 10 nodes with 30s checkpoint interval

Shark [Release, v0.2]

- HIVE over Spark: SQL-like interface (supports Hive 0.9)
  - up to 100x faster for in-memory data, and 5-10x for disk
- In tests on hundreds node cluster at Yahoo!

Diagram showing Shark, Spark, Spark Streaming, HIVE, Pig, Hadoop, Storm, MPI, Data Processing, Data Mgmt., Resource Mgmt.
Conviva Warehouse Queries (1.7 TB)

<table>
<thead>
<tr>
<th>Run time (seconds)</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shark</td>
<td>1.1</td>
<td>0.8</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>Shark (disk)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hive</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Spark & Shark available now on EMR!
**Tachyon** [Alpha Release, this Spring]

- High-throughput, fault-tolerant in-memory storage
- Interface compatible to HDFS
- Support for Spark and Hadoop

**BlinkDB** [Alpha Release, this Spring]

- Large scale approximate query engine
- Allow users to specify *error* or *time* bounds
- Preliminary prototype starting being tested at Facebook

---

**Diagram:**
- Tachyon and BlinkDB are highlighted.
- Dependencies and integration with other systems such as HDFS, Spark, Pig, Hadoop, and Mesos are shown.

---

*3/12/13*
SparkGraph [Alpha Release, this Spring]

- GraphLab API and Toolkits on top of Spark
- Fault tolerance by leveraging Spark

MLbase [In development]

- Declarative approach to ML
- Develop scalable ML algorithms
- Make ML accessible to non-experts
Compatible with Open Source Ecosystem

• **Support** existing interfaces whenever possible

---

**GraphLab API**

**Hive Interface and Shell**

**HDFS API**

**Compatibility layer for Hadoop, Storm, MPI, etc to run over Mesos**

---

**Accept inputs from Kafka, Flume, Twitter, TCP Sockets, ...**

**Support Hive API**

**Support HDFS API, S3 API, and Hive metadata**
Summary

Holistic approach to address next generation of Big Data challenges!

• Support *interactive* and *streaming* computations
  – In-memory, fault-tolerant storage abstraction, low-latency scheduling,...
• *Easy* to combine *batch*, *streaming*, and *interactive* computations
  – Spark execution engine supports all comp. models
• *Easy* to develop *sophisticated* algorithms
  – Scala interface, APIs for Java, Python, Hive QL, ...
  – New frameworks targeted to graph based and ML algorithms
• *Compatible* with existing open source ecosystem
• *Open source* (Apache/BSD) and fully committed to release *high quality* software
  – Three-person software engineering team lead by Matt Massie (creator of Ganglia, 5th Cloudera engineer)

Thanks!
Hive Architecture

Shark Architecture