Big Data Analytics
Map-Reduce and Spark

Slides by:
Joseph E. Gonzalez
jegonzal@cs.berkeley.edu

Guest Lecturer:
Vikram Sreekanti
From SQL to Big Data (with SQL)

- A few weeks ago...
  - Databases
  - (Relational) Database Management Systems
  - SQL: Structured Query Language

- Today
  - More on databases and database design
  - Enterprise data management and the data lake
  - Introduction to distributed data storage and processing
  - Spark
Data in the Organization

A little bit of buzzword bingo!
Inventory

How we like to think of data in the organization
The reality…

Sales (Asia)

Inventory

Sales (US)

Advertising
Operational Data Stores

- Capture the **now**
- Many different databases across an organization
- Mission critical... be careful!
  - Serving live ongoing business operations
  - Managing inventory
- Different formats (e.g., currency)
  - Different schemas (acquisitions ...)
- Live systems often don’t maintain history

We would like a consolidated, clean, historical snapshot of the data.
Data Warehouse

Collects and organizes historical data from multiple sources

Data is periodically ETL-ed into the data warehouse:

- Extracted from remote sources
- Transformed to standard schemas
- Loaded into the (typically) relational (SQL) data system
**Extract → Transform → Load (ETL)**

**Extract & Load:** provides a snapshot of operational data

- Historical snapshot
- Data in a single system
- Isolates analytics queries (e.g., Deep Learning) from business critical services (e.g., processing user purchase)
- Easy!

**Transform:** clean and prepare data for analytics in a unified representation

- **Difficult** → often requires specialized code and tools
- Different schemas, encodings, granularities
Data Warehouse

Collects and organizes historical data from multiple sources

How is data organized in the Data Warehouse?
## Example Sales Data

<table>
<thead>
<tr>
<th>pname</th>
<th>category</th>
<th>price</th>
<th>qty</th>
<th>date</th>
<th>day</th>
<th>city</th>
<th>state</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corn</td>
<td>Food</td>
<td>25</td>
<td>25</td>
<td>3/30/16</td>
<td>Wed.</td>
<td>Omaha</td>
<td>NE</td>
<td>USA</td>
</tr>
<tr>
<td>Corn</td>
<td>Food</td>
<td>25</td>
<td>8</td>
<td>3/31/16</td>
<td>Thu.</td>
<td>Omaha</td>
<td>NE</td>
<td>USA</td>
</tr>
<tr>
<td>Corn</td>
<td>Food</td>
<td>25</td>
<td>15</td>
<td>4/1/16</td>
<td>Fri.</td>
<td>Omaha</td>
<td>NE</td>
<td>USA</td>
</tr>
<tr>
<td>Galaxy</td>
<td>Phones</td>
<td>18</td>
<td>30</td>
<td>1/30/16</td>
<td>Wed.</td>
<td>Omaha</td>
<td>NE</td>
<td>USA</td>
</tr>
<tr>
<td>Galaxy</td>
<td>Phones</td>
<td>18</td>
<td>50</td>
<td>4/1/16</td>
<td>Fri.</td>
<td>Omaha</td>
<td>NE</td>
<td>USA</td>
</tr>
<tr>
<td>Peanuts</td>
<td>Food</td>
<td>2</td>
<td>45</td>
<td>3/31/16</td>
<td>Thu.</td>
<td>Seoul</td>
<td>Korea</td>
<td></td>
</tr>
</tbody>
</table>

- **Big** table: many columns and rows
- Substantial redundancy → expensive to store and access
- Make mistakes while updating
- Could we organize the data more efficiently?
### Multidimensional Data Model

#### Sales Fact Table

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>26</td>
</tr>
</tbody>
</table>

#### Locations

<table>
<thead>
<tr>
<th>locid</th>
<th>city</th>
<th>state</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omaha</td>
<td>Nebraska</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Seoul</td>
<td></td>
<td>Korea</td>
</tr>
<tr>
<td>5</td>
<td>Richmond</td>
<td>Virginia</td>
<td>USA</td>
</tr>
</tbody>
</table>

#### Products

<table>
<thead>
<tr>
<th>pid</th>
<th>pname</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Corn</td>
<td>Food</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Galaxy 1</td>
<td>Phones</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Peanuts</td>
<td>Food</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Time

<table>
<thead>
<tr>
<th>timeid</th>
<th>Date</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/30/16</td>
<td>Wed.</td>
</tr>
<tr>
<td>2</td>
<td>3/31/16</td>
<td>Thu.</td>
</tr>
<tr>
<td>3</td>
<td>4/1/16</td>
<td>Fri.</td>
</tr>
</tbody>
</table>

#### Dimension Tables

- Multidimensional “Cube” of data
Multidimensional Data Model

**Sales Fact Table**

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>26</td>
</tr>
</tbody>
</table>

**Locations**

<table>
<thead>
<tr>
<th>locid</th>
<th>city</th>
<th>state</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omaha</td>
<td>Nebraska</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Seoul</td>
<td></td>
<td>Korea</td>
</tr>
<tr>
<td>5</td>
<td>Richmond</td>
<td>Virginia</td>
<td>USA</td>
</tr>
</tbody>
</table>

**Products**

<table>
<thead>
<tr>
<th>pid</th>
<th>pname</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Corn</td>
<td>Food</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Galaxy 1</td>
<td>Phones</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Peanuts</td>
<td>Food</td>
<td>2</td>
</tr>
</tbody>
</table>

**Time**

<table>
<thead>
<tr>
<th>timeid</th>
<th>Date</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/30/16</td>
<td>Wed.</td>
</tr>
<tr>
<td>2</td>
<td>3/31/16</td>
<td>Thu.</td>
</tr>
<tr>
<td>3</td>
<td>4/1/16</td>
<td>Fri.</td>
</tr>
</tbody>
</table>

**Dimension Tables**

- **Fact Table**
  - Minimizes redundant info
  - Reduces data errors
- **Dimensions**
  - Easy to manage and summarize
  - Rename: Galaxy1 → Phablet
- **Normalized Representation**
- **How do we do analysis?**
  - **Joins!**
The Star Schema
The Star Schema

This looks like a star ...
The Star Schema

This looks like a star...
The Snowflake Schema

This looks like a snowflake …?
Which schema illustration would best organize this data?

Data(calid, student_name, year, major, major_grade_req, asg1_name, asg1_pts, asg1_score, asg1_grader_name, asg2_name, asg2_pts, asg2_score, asg2_grader_name, avg_grade)

Data(calid, student_name, year, major, major_grade_req,
asg1_name, asg1_pts, asg1_score, asg1_grader_name
asg2_name, asg2_pts, asg2_score, asg2_grader_name
avg_grade)

Grades(calid, asg_name, grader_id, score)
Graders(grader_id, grader_name)
Student(calid, name, year, major_name, avg_grade)
Majors(major_name, grade_req)
Assignments(asg_name, asg_pts)
Online Analytics Processing (OLAP)

Users interact with multidimensional data:

- Constructing ad-hoc and often complex SQL queries
- Using graphical tools that to construct queries
- Sharing views that summarize data across important dimensions
Cross Tabulation (Pivot Tables)

- Aggregate data across pairs of dimensions
  - **Pivot Tables**: graphical interface to select dimensions and aggregation function (e.g., SUM, MAX, MEAN)
  - **GROUP BY** queries
- Related to contingency tables and marginalization in stats.
- What about many dimensions?

<table>
<thead>
<tr>
<th>Item</th>
<th>Color</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desk</td>
<td>Blue</td>
<td>2</td>
</tr>
<tr>
<td>Desk</td>
<td>Red</td>
<td>3</td>
</tr>
<tr>
<td>Sofa</td>
<td>Blue</td>
<td>4</td>
</tr>
<tr>
<td>Sofa</td>
<td>Red</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>Desk</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Sofa</td>
<td>4</td>
</tr>
<tr>
<td>Red</td>
<td>Desk</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Sofa</td>
<td>5</td>
</tr>
<tr>
<td>Sum</td>
<td>Desk</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Sofa</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Desk</th>
<th>Sofa</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Red</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Sum</td>
<td>5</td>
<td>9</td>
<td>14</td>
</tr>
</tbody>
</table>
Cube Operator

- Generalizes cross-tabulation to higher dimensions.

In SQL:

```sql
SELECT Item, Color, SUM(Quantity) AS QtySum
FROM Furniture
GROUP BY CUBE (Item, Color);
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Color</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desk</td>
<td>Blue</td>
<td>2</td>
</tr>
<tr>
<td>Desk</td>
<td>Red</td>
<td>3</td>
</tr>
<tr>
<td>Sofa</td>
<td>Blue</td>
<td>4</td>
</tr>
<tr>
<td>Sofa</td>
<td>Red</td>
<td>5</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>14</td>
</tr>
<tr>
<td>*</td>
<td>Blue</td>
<td>6</td>
</tr>
<tr>
<td>*</td>
<td>Red</td>
<td>8</td>
</tr>
</tbody>
</table>
**OLAP Queries**

- **Slicing**: selecting a value for a dimension

  ![Slicing Diagram]

- **Dicing**: selecting a range of values in multiple dimension

  ![Dicing Diagram]
- **Rollup:** Aggregating along a dimension

- **Drill-Down:** de-aggregating along a dimension
Reporting and Business Intelligence (BI)

- Use high-level tools to interact with their data:
  - Automatically generate SQL queries
  - Queries can get big!

- Common!
Data Warehouse

Collects and organizes historical data from multiple sources

So far ...

- Star Schemas
- Data cubes
- OLAP Queries
Data Warehouse

Collects and organizes historical data from multiple sources

- How do we deal with semi-structured and unstructured data?
- Do we really want to force a schema on load?
How do we **clean** and **organize** this data?

depends on use ...

---

How do we **load and process** this data in a relational system?

depends on use...

Can be difficult...

Requires thought...

---

It is Terrible!
Data Lake*

- Store a copy of all the data
  - in one place
  - in its original “natural” form
- Enable data consumers to choose how to transform and use data.
  - Schema on Read

*Still being defined… [Buzzword Disclaimer]
The Dark Side of Data Lakes

- Cultural shift: *Curate ➔ Save Everything!*
  - Noise begins to dominate signal

- Limited data governance and planning
  - **Example:** `hdfs://important/joseph_big_file3.csv_with_json`
  - **What** does it contain?
  - **When** and **who** created it?

- No cleaning and verification ➔ lots of dirty data

- New tools are more complex and old tools no longer work

Enter the data scientist
A Brighter Future for Data Lakes

- Data scientists bring new skills
  - Distributed data processing and cleaning
  - Machine learning, computer vision, and statistical sampling

- Technologies are improving
  - SQL over large files
  - Self describing file formats & catalog managers

- Organizations are evolving
  - Tracking data usage and file permissions
  - New job title: data engineers
How do we **store** and **compute** on large unstructured datasets

- **Requirements:**
  - Handle very **large files** spanning **multiple computers**
  - Use **cheap** commodity devices that **fail frequently**
  - **Distributed data processing** quickly and easily

- **Solutions:**
  - **Distributed file systems** → spread data over multiple machines
    - Assume machine **failure** is common → **redundancy**
  - **Distributed computing** → load and process files on multiple machines concurrently
    - Assume machine **failure** is common → **redundancy**
    - **Functional programming** computational pattern → **parallelism**
Distributed File Systems
Storing very large files
Fault Tolerant Distributed File Systems

How do we store and access very large files across cheap commodity devices?

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

[Big File]

Machine 1  Machine 2  Machine 3  Machine 4

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

- Split the file into smaller parts.
- How?
  - Ideally at record boundaries
  - What if records are big?

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

[Big File]

[Machine 1]
B
C

[Machine 2]
A
B
C

[Machine 3]
A
C

[Machine 4]
A
B

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

- Split large files over multiple machines
  - Easily support massive files spanning machines

- Read parts of file in parallel
  - Fast reads of large files

- Often built using cheap commodity storage devices

Cheap commodity storage devices will fail!
Fault Tolerant Distributed File Systems

Failure Event

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

Failure Event

Machine 1
B C D

Machine 2
A B C

Machine 3
A C D

Machine 4
A B D

Big File

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

Failure Event

Big File

Machine 2

A B C

Machine 4

A B D

[Ghemawat et al., SOSP'03]
Fault Tolerant Distributed File Systems

Failure Event

[Ghemawat et al., SOSP'03]
Map-Reduce Distributed Aggregation
Computing are very large files
How would you compute the number of occurrences of each word in all the books using a team of people?
Simple Solution
Simple Solution

1) Divide Books Across Individuals
Simple Solution

1) Divide Books Across Individuals

2) Compute Counts Locally

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>2</td>
</tr>
<tr>
<td>Bird</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>0</td>
</tr>
<tr>
<td>Bird</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Simple Solution

1) Divide Books Across Individuals

2) Compute Counts Locally

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>2</td>
</tr>
<tr>
<td>Bird</td>
<td>7</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

3) Aggregate Tables

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>2</td>
</tr>
<tr>
<td>Bird</td>
<td>8</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
The Map Reduce Abstraction

Example: Word-Count

Map(docRecord) {
    for (word in docRecord) {
        emit (word, 1)
    }
}

Reduce(word, counts) {
    emit (word, SUM(counts))
}

Map: Deterministic
Reduce: Commutative and Associative

[Dean & Ghemawat, OSDI'04]
Key properties of Map And Reduce

- **Deterministic Map**: allows for re-execution on failure
  - If some computation is lost we can always re-compute
  - Issues with samples?

- **Commutative Reduce**: allows for re-order of operations
  - Reduce(A,B) = Reduce(B,A)
  - Example (addition): A + B = B + A
  - Is floating point math commutative?

- **Associative Reduce**: allows for regrouping of operations
  - Reduce(Reduce(A,B), C) = Reduce(A, Reduce(B,C))
  - Example (max): max(max(A,B), C) = max(A, max(B,C))
Executing Map Reduce
Executing Map Reduce

Distributing the Map Function
Executing Map Reduce

Distributing the Map Function
Executing Map Reduce

The map function applied to a local part of the big file.

Run in Parallel.

Output is cached for fast recovery on node failure.
Executing
Map
Reduce

Reduce function can be run on many machines ...
Run in Parallel

Map

Machine 1

B
C
D

Machine 2

A
B
C

Machine 3

A
C
D

Machine 4

A
B
D

Map

Reduce

apple 1

Reduce

the 2

Reduce

cat 1

Reduce

big 1

Reduce

dog 1

Map

Reduce

apple 1

Reduce

the 3

Reduce

cat 5

Reduce

big 2

Reduce
dog 2
If part of the file or any intermediate computation is lost we can simply **recompute it** without recomputing everything.
Interacting with Data @ Scale

Map-Reduce
Interacting With the Data

- Good for smaller datasets
  - Faster more natural interaction
  - Lots of tools!

Can we send the computation to the data? Yes!

\[ \Sigma = \bigoplus_{r \in \text{Data}} f_\theta(r) \]
Statistical Query Pattern
Common Machine Learning Pattern

- Computing aggregates of user defined functions
- Data-Parallel computation

\[
\sum = \bigoplus_{r \in \text{Data}} f_\theta(r)
\]

\(f_\theta\): User defined function [UDF]
\(\bigoplus\): User defined aggregate [UDA]

Algorithm

Query: $f_\theta(r)$

Response: $\sum_{r \in \text{Data}} f_\theta(r)$

Iterative Algorithm

Query: $f_\theta(r)$

Response: $\sum_{r \in \text{Data}} f_\theta(r)$
Interacting With the Data

Good for smaller datasets
- Faster more natural interaction
- Lots of tools!

Good for bigger datasets and compute intensive tasks

\[
\sum = \bigoplus_{r \in \text{Data}} f_\theta(r)
\]

Query: \( f_\theta \)

Response: \( \sum \)

Compute Locally

Algorithm

Cluster Compute

Request Data Sample

Sample of Data

Algorithm
Map Reduce Technologies
Hadoop

- First open-source map-reduce software
  - Managed by Apache foundation

- Based on Google’s
  - Google File System
  - MapReduce

- Companies formed around Hadoop:
  - Cloudera
  - Hortonworks
  - MapR
Hadoop

- Very active open source ecosystem
- Several key technologies
  - **HDFS**: Hadoop File System
  - **MapReduce**: map-reduce compute framework
  - **YARN**: Yet another resource negotiator
  - **Hive**: SQL queries over MapReduce
  - ...
In-Memory Dataflow System

Developed at the UC Berkeley AMP Lab


What Is Spark

- Parallel execution engine for big data processing
- **General**: efficient support for multiple workloads
- **Easy** to use: 2-5x less code than Hadoop MR
  - High level API’s in Python, Java, and Scala
- **Fast**: up to 100x faster than Hadoop MR
  - Can exploit in-memory when available
  - Low overhead scheduling, optimized engine
Spark Programming Abstraction

- Write programs in terms of transformations on distributed datasets

- Resilient Distributed Datasets (RDDs)
  - Distributed collections of objects that can be stored in memory or on disk
  - Built via parallel transformations (map, filter, ...)
  - Automatically rebuilt on failure

Slide provided by M. Zaharia
RDD: Resilient Distributed Datasets

- Collections of objects partitioned & distributed across a cluster
  - Stored in RAM or on Disk
  - Resilient to failures

- Operations
  - Transformations
  - Actions
Operations on RDDs

- Transformations $f(RDD) \Rightarrow RDD$
  - Lazy (not computed immediately)
  - E.g., “map”, “filter”, “groupBy”

- Actions:
  - Triggers computation
  - E.g. “count”, “collect”, “saveAsTextFile”
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

Diagram:

- Driver
- Worker
- Worker
- Worker
- Transformed RDD
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining
Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()  
messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

**Cache your data ➔ Faster Results**

Full-text search of Wikipedia
- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk
Abstraction: *Dataflow Operators*

- **map**
- **filter**
- **groupBy**
- **sort**
- **union**
- **join**
- **leftOuterJoin**
- **rightOuterJoin**

- **reduce**
- **count**
- **fold**
- **reduceByKey**
- **groupByKey**
- **cogroup**
- **cross**
- **zip**

- **sample**
- **take**
- **first**
- **partitionBy**
- **mapWith**
- **pipe**
- **save**

...
Abstraction: Dataflow Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...
Language Support

**Python**
```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

**Scala**
```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

**Java**
```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
    Boolean call(String s) {
        return s.contains("error");
    }
}).count();
```

**Standalone Programs**
Python, Scala, & Java

**Interactive Shells**
Python & Scala

**Performance**
Java & Scala are faster due to static typing