Programming at Scale: Dataflow

Chris Rossbach and Calvin Lin

cs380p
Review: Scale: Goal
Review: Design Space

Throughput vs. Latency

- Internet
- Private data center

Shared something vs. Shared nothing
Review: Design Space

- Throughput
- Latency
- Internet
- Private data center
- Shared nothing
- Shared something
- Transaction
Review: Design Space

Latency

Throughput

Internet

Private data center

Shared nothing

Shared something

Transaction

Grid
Review: Design Space

Throughput vs. Latency

Internet vs. Private data center

Shared nothing vs. Shared something

Grid

Transaction

Search
Review: Design Space

- Throughput
- Latency

Internet

Private data center

Shared nothing

Shared something

Transaction

Search

Grid

MapReduce

Spark

Dryad
Review: Design Space
You are an engineer at:
Hare-brained-scheme.com
You are an engineer at: Hare-brained-scheme.com

Your boss, comes to your office and says:

“We’re going to be super rich! We just need a program to search for strings in text files...”
Your boss, comes to your office and says:

“We’re going to be super rich! We just need a program to search for strings in text files…”

Input: <search_term>, <files>
You are an engineer at:
Hare-brained-scheme.com

Your boss, comes to your office and says:

“We’re going to be super rich! We just need a program to search for strings in text files...”

Input: <search_term>, <files>
Output: list of files containing <search_term>
public class StringFinder {
    int main(...) {
        foreach(File f in getInputFiles()) {
            if(f.contains(searchTerm))
                results.add(f.getFileName);
        }
        System.out.println("Files:" + results.toString());
    }
}
Infrastructure is hard to get right

[Diagram showing a web server connecting to multiple StringFinder nodes with indexed data]
Infrastructure is hard to get right

1. How do we distribute the searchable files on our machines?
Infrastructure is hard to get right

1. How do we distribute the searchable files on our machines?
2. What if our webserver goes down?
Infrastructure is hard to get right

1. How do we distribute the searchable files on our machines?
2. What if our webserver goes down?
3. What if a StringFinder machine dies? How would you know it was dead?
1. How do we distribute the searchable files on our machines?
2. What if our webserver goes down?
3. What if a StringFinder machine dies? How would you know it was dead?
4. What if marketing comes and says, “well, we also want to show pictures of the earth from space too! Ooh..and the moon too!”
StringFinder was the easy part!

You really need general infrastructure.
Many different tasks
Want to use hundreds or thousands of PC’s
Continue to function if something breaks
Must be easy to program...
Dataflow Engines

Programming model + infrastructure
Write programs that run on lots of machines
  Automatic parallelization and distribution
Fault-tolerance
I/O and jobs Scheduling
Status and monitoring
MapReduce

- Input & Output: sets of <key, value> pairs
- Programmer writes 2 functions:
  - `map (in_key, in_value) -> list(out_key, intermediate_value)`
    - Processes <k,v> pairs
    - Produces intermediate pairs
  - `reduce (out_key, list(interm_val)) -> list(out_value)`
    - Combines intermediate values for a key
    - Produces a merged set of outputs
public void map() {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    if(itr.countTokens() >= N) {
        while(itr.hasMoreTokens()) {
            word = itr.nextToken() + "|" + key.getFileName();
            output.collect(word, 1);
        }
    }
}

Input: a line of text, e.g. "mistakes were made" from myfile.txt

Output:
mistakes|myfile.txt
were |myfile.txt
made |myfile.txt
public void reduce() {
    int sum = 0;
    while (values.hasNext()) {
        sum += values.next().get();
    }
    output.collect(key, sum);
}

Input: a <term,filename> pair, list of occurrences (e.g. {1, 1,..1})
Output:
    mistakes | myfile.txt    10
    were     | myfile.txt    45
    made     | myfile.txt    2
Example: K-Means Mapper

```java
/*
 * Map: find minimum distance center for point, emit to reducer
 */
@override
public void map(LongWritable key, Text value,
               OutputCollector<DoubleWritable, DoubleWritable> output,
               Reporter reporter) throws IOException {
    String line = value.toString();
    double point = Double.parseDouble(line);
    double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters.get(0);
    // Find the minimum center from a point
    for (double c : mCenters) {
        min1 = c - point;
        if (Math.abs(min1) < Math.abs(min2)) {
            nearest_center = c;
            min2 = min1;
        }
    }
    // Emit the nearest center and the point
    output.collect(new DoubleWritable(nearest_center),
                   new DoubleWritable(point));
}
```
/*
 * Reduce: collect all points per center and calculate
 * the next center for those points
 */

@override
public void reduce(
    DoubleWritable key, Iterator<DoubleWritable> values,
    OutputCollector<DoubleWritable, Text> output, Reporter reporter)
    throws IOException {
    double newCenter;
    double sum = 0;
    int no_elements = 0;
    String points = "";
    while (values.hasNext()) {
        double d = values.next().get();
        points = points + " " + Double.toString(d);
        sum = sum + d;
        ++no_elements;
    }
    // We have a new center now
    newCenter = sum / no_elements;
    // Emit new center and point
    output.collect(new DoubleWritable(newCenter), new Text(points));
}
How Does Parallelization Work?
Execution
Key idea → shuffle == sort!
Task Granularity And Pipelining

| map tasks | >> | machines | -- why?
Task Granularity And Pipelining

| map tasks | >> | machines | -- why?
Minimize fault recovery time
Pipeline map with other tasks
Easier to load balance dynamically
The end of your career at:
Hare-brained-scheme.com
The end of your career at: Hare-brained-scheme.com

Your boss, comes to your office and says:
The end of your career at: Hare-brained-scheme.com

Your boss, **comes to your office and says:**

“*I can’t believe you used *MapReduce*!!!*
The end of your career at:
Hare-brained-scheme.com

Your boss, comes to your office and says:

“I can’t believe you used MapReduce!!!
You’re fired...”
The end of your career at: Hare-brained-scheme.com

Your boss, , comes to your office and says:

“I can’t believe you used **MapReduce!!!**

*You’re fired...*”

Why might he say this?
MapReduce: A major step backwards | The Database Column


on Jan 17 in Database architecture, Database history, Database innovation posted by DeWitt

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of low-end processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
Why is MapReduce backwards?
Why is MapReduce backwards?

Backwards step in programming paradigm
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
  It’s just a group-by aggregate engine
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
   It’s just a group-by aggregate engine
Missing most DBMS features
   Schema, foreign keys, ...
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
  It’s just a group-by aggregate engine
Missing most DBMS features
  Schema, foreign keys, ...
Incompatible with most DBMS tools
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
   It’s just a group-by aggregate engine
Missing most DBMS features
   Schema, foreign keys, ...
Incompatible with most DBMS tools

So why is it such a big success?
MapReduce and Dataflow
MapReduce and Dataflow

- MR is a *dataflow* engine
MapReduce and Dataflow

- MR is a *dataflow* engine
MapReduce and Dataflow

- MR is a *dataflow* engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark
MapReduce and Dataflow

• MR is a **dataflow** engine
• Lots of others
  • Dryad
  • DryadLINQ
  • Dandelion
  • CIEL
  • GraphChi/Pregel
  • Spark
MapReduce vs Dryad (and others...)

DAG instead of BSP
Interface variety
  Memory FIFO
  Disk
  Network
Flexible Modular Composition
Dryad (2007): 2-D Piping

- Unix Pipes: 1-D
  
grep | sed | sort | awk | perl
Dryad (2007): 2-D Piping

- Unix Pipes: 1-D
  ```
  grep | sed | sort | awk | perl
  ```

- Dryad: 2-D
  ```
  grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}
  ```
Dataflow Engines

job graph

data plane

Files, TCP, FIFO

control plane

cluster
Dataflow Job Structure

Input files
- grep
- sed

Channels
- grep
- sed

Stage
- sort
- awk
- perl

Output files
- awk

Vertices (processes)
- grep
- sed
- sort
Dataflow Job Structure

Input files

Channels

Vertices (processes)

Stage

Output files

How to implement?
Channels

Finite streams of items

- distributed filesystem files (persistent)
- SMB/NTFS files (temporary)
- TCP pipes (inter-machine)
- memory FIFOs (intra-machine)
Channels

Finite streams of items

- distributed filesystem files (persistent)
- SMB/NTFS files (temporary)
- TCP pipes (inter-machine)
- memory FIFOs (intra-machine)

Key idea:
Encapsulate data movement behind channel abstraction → gets programmer out of the picture
Commodity clusters: important platform

  **In industry:** search, machine translation, ad targeting, ...
  **In research:** bioinformatics, NLP, climate simulation, ...

Cluster-scale models (e.g. MR) de facto standard

  Fault tolerance through replicated durable storage
  Dataflow is the common theme
Spark (2012) Background

Commodity clusters: important platform

*In industry:* search, machine translation, ad targeting, ...
*In research:* bioinformatics, NLP, climate simulation, ...

Cluster-scale models (e.g. MR) de facto standard
Fault tolerance through replicated durable storage
Dataflow is the common theme

*Multi-core*

*Iteration*
Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:
Motivation

Programming models for clusters transform data flowing from stable storage to stable storage.

E.g., MapReduce:

Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures.
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to
   \[ \Sigma_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|} \]

```scala
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank(links.size)))
  }.reduceByKey(_ + _)
}
```
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[ \sum_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|} \]

\begin{Verbatim}
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
    links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
\end{Verbatim}
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[ \Sigma_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|} \]

```scala
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
    links.map(dest => (dest, rank / links.size))
  }.reduceByKey(_ + _)
}
```

Solution: augment data flow model with “resilient distributed datasets” (RDDs)
Programming Model

• Resilient distributed datasets (RDDs)
  • Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  • Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  • Can be \textit{cached} across parallel operations
Programming Model

• Resilient distributed datasets (RDDs)
  • Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  • Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  • Can be *cached* across parallel operations

• Parallel operations on RDDs
  • Reduce, collect, count, save, ...
Programming Model

• Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  - Can be cached across parallel operations

• Parallel operations on RDDs
  - Reduce, collect, count, save, ...

• Restricted shared variables
  - Accumulators, broadcast variables
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

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

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

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

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

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

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(' \t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t)(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo"))
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\"\t\")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

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

```python
driver = spark.textFile("hdfs://...")
errors = driver.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")) . count
cachedMsgs.filter(_.contains("bar")) . count
```
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\"\t\t\")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
```

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
RDD Fault Tolerance

• RDDs maintain *lineage* information that can be used to reconstruct lost partitions

• Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
  .map(_.split('t')(2))
  .persist()
```

![Diagram showing the lineage of RDDs]
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage
Data-Parallel Computation Systems

Application
Language
Execution
Storage

SQL
Parallel Databases
Data-Parallel Computation Systems

- Application
- SQL
- Language
- Execution
- Storage
- Parallel Databases
- Map-Reduce
- GFS
- BigTable
- SQL
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

SQL

Sawzall

Map-Reduce

GFS

BigTable

Sawzall
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage

**Parallel Databases**

- SQL
- Sawzall

**Map-Reduce**

- Sawzall
- GFS
- BigTable

Sample code:

```plaintext
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```
Data-Parallel Computation Systems

- Application
  - SQL
  - Sawzall
- Language
  - Sawzall
- Execution
  - Parallel Databases
  - Map-Reduce
- Storage
  - GFS
  - BigTable
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

Map-Reduce

GFS

HDFS

BigTable

Hadoop

S3

SQL

Sawzall

Sawzall
Data-Parallel Computation Systems

Application

Language
SQL
Sawzall
≈SQL
Pig, Hive

Execution
Parallel Databases
Map-Reduce
GFS
BigTable
Hadoop
HDFS
S3
```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

SQL

Sawzall

Map-Reduce

GFS

BigTable

≈SQL

Pig, Hive

Hadoop

HDFS

S3
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

- SQL
- Sawzall
- ≈SQL

Map-Reduce

- GFS
- BigTable

Hadoop

- HDFS
- S3

Storage Systems

- Cosmos
- Azure
- SQL Server

Databases

- DryadLINQ
- Dryad
- Scope

SQL

≈SQL

Sawzall

Pig, Hive
Data-Parallel Computation Systems

- Application
- SQL
- Sawzall
- Map-Reduce
- GFS
- BigTable
- ≈SQL
- Pig, Hive
- Hadoop
- HDFS
- S3
- LINQ, SQL
- DryadLINQ
- Scope
- Dryad
- Cosmos, HPC, Azure
- Cosmos
- Azure
- SQL Server
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

Map-Reduce

GFS

BigTable

Sawzall

Pig, Hive

HDFS

S3

≈SQL

DryadLINQ

Spark

Dryad

Hadoop

LINQ, SQL

SQL

≈SQL

LINQ, SQL

HPC, Azure

Cosmos

Azure

SQL Server
Summary

Dataflow key enabler for cluster-scale parallelism

Key issues become runtime’s responsibility
- Data movement
- Scheduling
- Fault-tolerance
Example: Counting Words...

map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key, 
   Iterator intermediate_values):
   // output_key: a word
   // output_values: a list of counts
   int result = 0;
   for each v in intermediate_values:
      result += ParseInt(v);
   Emit(AsString(result));

MapReduce handles all the other details!
Redundant Execution

Slow worker can throttle performance: why?
What makes a worker slow?

Solution:
Redundant Execution

Slow worker can throttle performance: why?

What makes a worker slow?
- Other Jobs on machine (how could we fix)
- Bad disks, soft errors
- Exotica (processor caches disabled!)

Solution: spawn backups near end of phase
MapReduce is sub-optimal
MapReduce is sub-optimal

Modern DBMSs: hash + B-tree indexes to accelerate data access.

Indexes are user-defined
Could MR do this?
MapReduce is sub-optimal

Modern DBMSs: hash + B-tree indexes to accelerate data access.
   Indexes are user-defined
   Could MR do this?
No query optimizer! (oh my, terrible...but good for researchers! 😊)
MapReduce is sub-optimal

Modern DBMSs: hash + B-tree indexes to accelerate data access.
   Indexes are user-defined
   Could MR do this?
No query optimizer! (oh my, terrible...but good for researchers! ☹)
Skew: wide variance in distribution of keys
   E.g. “the” more common than “zyzzyva”
MapReduce is sub-optimal

Modern DBMSs: hash + B-tree indexes to accelerate data access.
- Indexes are user-defined
- Could MR do this?

No query optimizer! (oh my, terrible...but good for researchers! 😊)

Skew: wide variance in distribution of keys
- E.g. “the” more common than “zyzzyva”

Materializing splits
- N=1000 mappers → M=500 keys = 500,000 local files
- 500 reducer instances “pull” these files
- DBMSs push splits to sockets (no local temp files)
MapReduce: !novel && feature-poor
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
MapReduce: novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
MapReduce: novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
- Indexing
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
- Indexing
- Update operator
MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
- Indexing
- Update operator
- Transactions
MapReduce: !novel && & feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
- Indexing
- Update operator
- Transactions
- Integrity constraints, referential integrity
MapReduce: novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:
- Indexing
- Update operator
- Transactions
- Integrity constraints, referential integrity
- Views
DryadLINQ = LINQ + Dryad

```csharp
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
               where IsLegal(c.key)
               select new { Hash(c.key), c.value};
```
DryadLINQ = LINQ + Dryad

```csharp
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
              where IsLegal(c.key)
              select new { Hash(c.key), c.value};
```

Data

```
collection
results
```
DryadLINQ = LINQ + Dryad

```
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
               where IsLegal(c.key)
               select new { Hash(c.key), c.value};
```

Data → Query plan (Dryad job) → collection → results
DryadLINQ = LINQ + Dryad

```csharp
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
              where IsLegal(c.key) 
              select new { Hash(c.key), c.value};
```

Vertex code

Data

Query plan (Dryad job)

(collection)

(results)
Programming Model
Programming Model

Where
Programming Model

Where
Programming Model

Where

Select
Programming Model

Where

Select
Programming Model

Where
Select
GroupBy
Programming Model

Where
Select
GroupBy
Programming Model

Where
Select
GroupBy
OrderBy
Programming Model

Where
Select
GroupBy
OrderBy
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize
Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize
```csharp
public static IQueryable<Pair> Histogram(IQueryable<LineRecord> input, int k)
{
    var words = input.SelectMany(x => x.line.Split(' '));
    var groups = words.GroupBy(x => x);
    var counts = groups.Select(x => new Pair(x.Key, x.Count()));
    var ordered = counts.OrderByDescending(x => x.count);
    var top = ordered.Take(k);
    return top;
}
```

<table>
<thead>
<tr>
<th>“A line of words of wisdom”</th>
</tr>
</thead>
<tbody>
<tr>
<td>[“A”, “line”, “of”, “words”, “of”, “wisdom”]</td>
</tr>
<tr>
<td>[[“A”], [“line”], [“of”, “of”], [“words”], [“wisdom”]]</td>
</tr>
<tr>
<td>[ {“A”, 1}, {“line”, 1}, {“of”, 2}, {“words”, 1}, {“wisdom”, 1}]</td>
</tr>
<tr>
<td>[{“of”, 2}, {“A”, 1}, {“line”, 1}, {“words”, 1}, {“wisdom”, 1}]</td>
</tr>
<tr>
<td>[{“of”, 2}, {“A”, 1}, {“line”, 1}]</td>
</tr>
</tbody>
</table>
Example: Histogram

```csharp
public static IQueryable<Pair> Histogram(IQueryable<IQueryable<LineRecord>> input, int k)
{
    var words = input.SelectMany(x => x.line.Split(' '));
    var groups = words.GroupBy(x => x);
    var counts = groups.Select(x => new Pair(x.Key, x.Count()));
    var ordered = counts.OrderByDescending(x => x.count);
    var top = ordered.Take(k);
    return top;
}
```

```
[ "A", "line", "of", "words", "of", "wisdom" ]

[ [ "A" ], [ "line" ], [ "of", "of" ], [ "words" ], [ "wisdom" ] ]

[ [ "A", 1 ], [ "line", 1 ], [ "of", 2 ], [ "words", 1 ], [ "wisdom", 1 ] ]

[ [ "of", 2 ], [ "A", 1 ], [ "line", 1 ], [ "words", 1 ], [ "wisdom", 1 ] ]

[ [ "of", 2 ], [ "A", 1 ], [ "line", 1 ] ]
```
RDDs

- Immutable, partitioned, logical collection of records
  - Need not be materialized
  - Contains information to rebuild a dataset
- Partitioning can be based on a key
- Built using bulk transformations on other RDDs
- Can be cached for future reuse
RDDs

• Immutable, partitioned, logical collection of records
  • Need not be materialized
  • Contains information to rebuild a dataset
  • Partitioning can be based on a key
  • Built using bulk transformations on other RDDs
  • Can be cached for future reuse

Transformations (define a new RDD)
map
filter
sample
union
groupByKey
reduceByKey
join
persist/cache
...
RDDs

• Immutable, partitioned, logical collection of records
  • Need not be materialized
  • contains information to rebuild a dataset
  • Partitioning can be based on a key
  • Built using bulk transformations on other RDDs
  • Can be cached for future reuse

Transformations
(define a new RDD)
map
filter
sample
union
groupByKey
reduceByKey
join
persist/cache
...

Parallel operations
(return a result to driver)
reduce
collect
count
save
lookupKey
...

RDDS

- Immutable, partitioned, logical collection of records
  - Need not be materialized
  - Contains information to rebuild a dataset
  - Partitioning can be based on a key
  - Built using bulk transformations on other RDDs
  - Can be cached for future reuse

Transformations (define a new RDD)
- map
- filter
- sample
- union
- groupByKey
- reduceByKey
- join
- persist/cache
- ...

Parallel operations (return a result to driver)
- reduce
- collect
- count
- save
- lookupKey
- ...

Diagram:
- Where
- Select
- GroupBy
- OrderBy
- Aggregate
- Join
- Apply
- Materialize
## RDDs vs Distributed Shared Memory

<table>
<thead>
<tr>
<th>Concern</th>
<th>RDDs</th>
<th>Distr. Shared Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>Fine-grained</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Writes</td>
<td>Bulk transformations</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Consistency</td>
<td>Trivial (immutable)</td>
<td>Up to app / runtime</td>
</tr>
<tr>
<td>Fault recovery</td>
<td>Fine-grained and low-overhead using lineage</td>
<td>Requires checkpoints and program rollback</td>
</tr>
<tr>
<td>Straggler mitigation</td>
<td>Possible using speculative execution</td>
<td>Difficult</td>
</tr>
<tr>
<td>Work placement</td>
<td>Automatic based on data locality</td>
<td>Up to app (but runtime aims for transparency)</td>
</tr>
</tbody>
</table>
Review: What is GroupBy?
Review: What is GroupBy?

Group a collection by key
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements → key
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements \( \rightarrow \) key

```csharp
var res = ints.GroupBy(x => x);
```
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements → key

```csharp
var res = ints.GroupBy(x => x);
```

```plaintext
10 30 20 10 20 30 10
```
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements $\rightarrow$ key

```csharp
var res = ints.GroupBy(x => x);
```

```
10 30 20 10 20 30 10
```

```
10 10 10 30 30 20 20
```
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements → key

```csharp
var res = ints.GroupBy(x => x);
```

```csharp
foreach (T elem in ints)
{
    key = KeyLambda(elem);
    group = GetGroup(key);
    group.Add(elem);
}
```
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements → key

```
var res = ints.GroupBy(x => x);
```

```csharp
da3f0f
foreach (T elem in PF(ints))
{
    key = KeyLambda(elem);
    group = GetGroup(key);
    group.Add(elem);
}
```
Review: What is GroupBy?

Group a collection by key
Lambda function maps elements $\rightarrow$ key

```csharp
var res = ints.GroupBy(x => x);
```

```csharp
foreach (T elem in PF(ints))
{
    key = KeyLambda(elem);
    group = GetGroup(key);
    group.Add(elem);
}
```
Why is MapReduce backwards?

Map == group-by
Reduce == aggregate
Why is MapReduce backwards?

Map == group-by
Reduce == aggregate

```sql
SELECT job, COUNT(*) as "numemps"
FROM employees
WHERE salary > 1000
GROUP BY job;
```
Why is MapReduce backwards?

Map == group-by
Reduce == aggregate

SELECT job, COUNT(*) as "numemps"
    FROM employees
    WHERE salary > 1000
    GROUP BY job;

• Where is the aggregate in this example?
• Is the DBMS analogy clear?
Why is MapReduce backwards?
Why is MapReduce backwards?

Schemas are good (what’s a schema?)
Why is MapReduce backwards?

Schemas are good (what’s a schema?)
Separation of schema from app is good (why?)
Why is MapReduce backwards?

Schemas are good (what’s a schema?)
Separation of schema from app is good (why?)
High-level access languages are good (why?)
Why is MapReduce backwards?

Schemas are good (what’s a schema?)
Separation of schema from app is good (why?)
High-level access languages are good (why?)