DATA516/CSED516 Scalable Data Systems and Algorithms Lecture 4 Spark, MapReduce, Hive

### Announcements

- HW1 is graded and posted (thanks Kexuan!)
- Project proposals due this Friday!
   Working in team? Only one of you submits
- HW2 (Spark) due on Monday

# Distributed or Parallel Query Processing

- Clusters:
  - More servers  $\rightarrow$  more in main memory
  - More servers  $\rightarrow$  more computing power
  - Clusters are now cheaply available in the cloud
  - *<u>Distributed</u>* query procesing
- Multicores:
  - The end of Moore's law
  - Parallel query processing

### Outline

- Spark
- MapReduce and critique
- Fault Tolerance
- Hive (short)

Next lecture: Parallel databases

# Spark

# Motivation

- Limitations of relational database systems:
  - Single server (at least traditionally)
  - SQL is a limited language (eg no iteration)
- Spark:
  - Distributed system
  - Functional language (Java/Scala) good for ML
- Implementation:
  - Extension of MapReduce
  - Distributed physical operators

### **Review: Single Client**

### E.g. data analytics



### **Review: Client-Server**



### **Review:** Three-tier



### **Review: Distributed Database**



# Programming in Spark

- A Spark program consists of:
  - Transformations (map, reduce, join...). Lazy
  - Actions (count, reduce, save...). Eager
- Eager: operators are executed immediately
- Lazy: operators are not executed immediately

   A operator tree is constructed in memory instead
   Similar to a relational algebra tree

### **Collections in Spark**

RDD<T> = an RDD collection of type T

- Distributed on many servers, not nested
- Operations are done in parallel
- Recoverable via lineage; more later

Seq<T> = a sequence

- Local to one server, may be nested
- Operations are done sequentially

### Example from paper, new syntax Search logs stored in HDFS

// First line defines RDD backed by an HDFS file
lines = spark.textFile("hdfs://...")

// Now we create a new RDD from the first one
errors = lines.filter(x -> x.startsWith("Error"))

```
// Persist the RDD in memory for reuse later
errors.persist()
errors.collect()
errors.filter(x -> x.contains("MySQL")).count()
```

### Example from paper, new syntax Search logs stored in HDFS

```
// First line defines RDD backed by an HDFS file
lines = spark.textFile("hdfs://...")
// Now we create a new RDD from the first one
errors = lines.filter(x -> x.startsWith("Error"))
                                 Transformation: Not executed yet...
// Persist the RDD in memory for reuse later
errors.persist()
errors.collect()
errors.filter(x -> x.contains("MySQL")).count()
```

### Example from paper, new syntax Search logs stored in HDFS



### **Anonymous Functions**

### A.k.a. lambda expressions, starting in Java 8

### errors = lines.filter(x -> x.startsWith("Error"))

### **Chaining Style**

#### The RDD s:

Error Vvarning Vvarning Error Abort Abort Error Error Vvarning Error	Error Warnir	ng Warning	Warning Error	Abort	Abort	Error	Error	Warning	Error
--	--------------	------------	---------------	-------	-------	-------	-------	---------	-------

The RDD s:

Parallel step 1

Error	Warning	Warning	Error	Abort	Abort	Error	Error	Warning	Error
filter("ERROR")									
▼	▼	▼	▼	▼	▼	▼	▼	▼	▼

Parallel step 1









# More on Programming Interface

Large set of pre-defined transformations:

 Map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, crossProduct,

Small set of pre-defined actions:

• Count, collect, reduce, lookup, and save

Programming interface includes iterations

Transformations:				
map(f : T -> U):	RDD <t> -&gt; RDD<u></u></t>			
<pre>flatMap(f: T -&gt; Seq(U)):</pre>	RDD <t> -&gt; RDD<u></u></t>			
<pre>filter(f:T-&gt;Bool):</pre>	RDD <t> -&gt; RDD<t></t></t>			
<pre>groupByKey():</pre>	RDD<(K,V)> -> RDD<(K,Seq[V])>			
<pre>reduceByKey(F:(V,V)-&gt; V):</pre>	RDD<(K,V)> -> RDD<(K,V)>			
union():	(RDD <t>,RDD<t>) -&gt; RDD<t></t></t></t>			
join():	(RDD<(K,V)>,RDD<(K,W)>) -> RDD<(K,(V,W))>			
<pre>cogroup():</pre>	(RDD<(K,V)>,RDD<(K,W)>)-> RDD<(K,(Seq <v>,Seq<w>))&gt;</w></v>			
<pre>crossProduct():</pre>	(RDD <t>,RDD<u>) -&gt; RDD&lt;(T,U)&gt;</u></t>			

Actions:			
<pre>count():</pre>	RDD <t> -&gt; Long</t>		
<pre>collect():</pre>	RDD <t> -&gt; Seq<t></t></t>		
<pre>reduce(f:(T,T)-&gt;T):</pre>	RDD <t> -&gt; T</t>		
<pre>save(path:String):</pre>	Outputs RDD to a storage system e.g., HDFS		

# More Complex Example

```
val points = spark.textFile(...)
                                 .map(parsePoint).persist()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
    val gradient = points.map{ p =>
        p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
    }.reduce((a,b) => a+b)
    w -= gradient
}
```



Image from: http://spark.apache.org/

# Spark SQL vs Functional Prog. API

- Spark's original functional programming API
   General
  - But limited opportunities for automatic optimization
- Spark SQL simultaneously
  - Makes Spark accessible to more users
  - Improves opportunities for automatic optimizations

### Three Java-Spark APIs

- RDDs: Sytnax: JavaRDD<T>
  - T = anything, basically untyped
  - Distributed, main memory
- Data frames: Dataset<Row>
  - <Row> = a record, dynamically typed
  - Distributed, main memory or external (e.g. SQL)
- Datasets: Dataset<Person>
  - <Person> = user defined type
  - Distributed, main memory (not external)

### DataFrames

- Like RDD: immutable distributed collection
- Organized into *named columns* 
  - Just like a relation
  - Elements are untyped objects called Row's
- Similar API as RDDs with additional methods
  - people = spark.read().textFile(...);
    ageCol = people.col("age");
    ageCol.plus(10); // creates a new DataFrame

### Datasets

- Like DataFrames, but elements must be typed
- E.g.: Dataset<People> rather than Dataset<Row>
- Can detect errors during compilation time
- DataFrames are aliased as Dataset<Row> (as of Spark 2.0)

# Datasets API: Sample Methods

- Functional API
  - <u>agg(Column</u> expr, <u>Column</u>... exprs)
     Aggregates on the entire Dataset without groups.
  - <u>groupBy</u>(String col1, String... cols)
     Groups the Dataset using the specified columns, so that we can run aggregation on them.
  - join(Dataset<?> right)
     Join with another DataFrame.
  - orderBy(Column... sortExprs)
     Returns a new Dataset sorted by the given expressions.
  - <u>select</u>(<u>Column</u>... cols)
     Selects a set of column based expressions.
- "SQL" API
  - SparkSession.sql("select \* from R");
- Look familiar?

# Recap: Programming in Spark

- A Spark/Scala program consists of:
  - Transformations (map, reduce, join...). Lazy
  - Actions (count, reduce, save...). Eager
- RDD<T> = an RDD collection of type T
  - Partitioned, recoverable (through lineage), not nested
- Seq<T> = a sequence

- Local to a server, may be nested

## Outline

- Spark
- MapReduce and critique
- Fault Tolerance
- Hive (short)

Next lecture: Parallel databases

### MapReduce: References

- Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. OSDI'04
- D. DeWitt and M. Stonebraker. Mapreduce a major step backward. In Database Column (Blog), 2008.

# MapReduce

- Google:
  - Started around 2000
  - Paper published 2004
  - Discontinued September 2019
- Free variant: Hadoop
- MapReduce = high-level programming model and implementation for large-scale parallel data processing

# Distributed File System (DFS)

- For very large files: TBs, PBs
- Each file partitioned into *chunks* (64MB)
- Each chunk replicated (≥3 times) why?
- Implementations:
  - Google's DFS: GFS, proprietary
  - Hadoop's DFS: HDFS, open source

### MapReduce

• Describe the **input** and **output** to map reduce

• Describe the **Map** function

• Describe the **Reduce** function
#### MapReduce

- Describe the input and output to map reduce
  - Input: a bag of (inputkey, value) pairs
  - Output: a bag of (outputkey, value) pairs
- Describe the **Map** function

• Describe the **Reduce** function

#### MapReduce

- Describe the input and output to map reduce
  - Input: a bag of (inputkey, value) pairs
  - Output: a bag of (outputkey, value) pairs
- Describe the **Map** function
  - Input: (input key, value)
  - Ouput: bag of (intermediate key, value)
- Describe the **Reduce** function

#### MapReduce

- Describe the input and output to map reduce
  - Input: a bag of (inputkey, value) pairs
  - Output: a bag of (outputkey, value) pairs
- Describe the Map function
  - Input: (input key, value)
  - Ouput: bag of (intermediate key, value)
- Describe the **Reduce** function
  - Input: (intermediate key, bag of values)
  - Output: bag of output (values)

### Step 1: the MAP Phase

User provides the MAP-function:

- Input: (input key, value)
- Ouput: bag of (intermediate key, value)

System applies the map function in parallel to all (input key, value) pairs in input file

## Step 2: the REDUCE Phase

User provides the **REDUCE** function:

- Input: (intermediate key, bag of values)
- Output: bag of output (values)

System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

## Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

# Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

map(String key, String value): // key: document name // value: document contents for each word w in value:

EmitIntermediate(w, "1");

### Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

```
      map(String key, String value):
      reduce

      // key: document name
      // key:

      // value: document contents
      // value

      for each word w in value:
      // value

      EmitIntermediate(w, "1");
      for each
```

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

#### **Documents:**

#### Relation

#### did1 did2

Hive – A Petabyte Scale Data Warehouse Using Hadoop					
Ashish Thuseo, Joydeep Sen Sarma, Namit Jain, Zheng S and Raghet	hao, Prasad Chakka, Ning Zhang, Surosh Antony, Hao Liu hari Murthy				
Facebook Date In	Pastwater Team				
where - the state of the state	date A is not surfar dividing labeling as a hashing in the state of the state of the state of the labeling as a state of the state of the state of the labeling as a state of the state of				
1 PURSUATES 1	<text><text><text><section-header></section-header></text></text></text>				

system with their care types and functions. The query	Objectionposter jets interface and expensing that
language in very similar in NQL and therefore can be easily anderstood by anyone familiar with NQL. There are some manues in the data model, type system and HireQL that are	implementation through the priCity-colorpation method present in the XeeDe interface. More datable on these interfaces can be found on the Elece webi [2], but the basis interesty here
different from traditional databases and that have been matiruled by the respectment gained at Pacobook. We will highlight these and other details in this section.	is that any arbitrary data format and types muraled therein can be plugged into How by providing a jar that contains the implementations for the XerDe and ObjectInspector interfaces.
A. Data Model and Type System	All the native XorDes and complex types supported in line are also implementations of these interfaces. As a result once
Ximilar to instituted databases, Here stores data in tables, where each table sensitive of a number of serve, and much new sensitive of a specified number of enhance. Each othere has an annutated bear. The true in either a minimize tree or a	the proper exercisions have been made between the table and the jar, the query layer insis there are par with the native types and iterativ. As an example, the following statement with a iteration the XeeDs and Obsolutionscien interfaces to the
complex type. Canonity, the following primitive types are suggested	deterbated casher[2]) so that it is available to Haloop and then presends to conste the table with the castern unde
<ul> <li>- angent - regulat types, ingla types, instanti, types, inspiral (19(4)) All integer types are used.</li> <li>- Floating point numbers - Environgle provision), shabbi(dashle provision)</li> <li>- Nime</li> </ul>	alii jar. jar. jop limmai jar; CREATE TABLE 62 ROW FORMAT SIEDE 'som optionnal MyReGP;
Here also satively supports the following complex types • Associative arrays - map key type, value type:	Note that, if possible, the table schema could also be previded by composing the complex and primitive types.
Xino - simo manager (ger     Xino - simo - life manager field type, _ >	# Query Language The Hore must language Hourds,) commission of a solvest of
These complex logues are tomplated and can be composed by provide logues of arbitrary amount points, for example, her-impriving, strategically, g2 into $>$ expression as lost of strategical points, g2 into $>$ expression as lost of particular arrays. The maximum point point points are part together into a centre table calculated point addies with the district distance. For example, the following intervent remains a fully if with a complex subsets.	SQL and some evications that we have found used in our survivanues. To define all SQL instances like its set chaose sub- queries, survivan types of pure – inner, lief user, split and of enter pure, currences produces, group type, and quantum primitive and models for the set purpose purposes of the set of the set of the set of the set purpose of the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the
CREATE TABLE (b) story, if float, is for map using, simility I sat, p2 and 12,	sporying the system right array. Useful mutualata howeving sepablistics like show tables and describe are also process and to are exclusive data cambilities to interest every shows (downly
Query expressions are access fields within the structs using a <sup>11</sup> -spreads. Values in the associative array and this can be accreared using 11 operation. In the protons reample, 11.1001 gives the first element of the lost and 11.0002 key1 gives the timal accessibility with key1 in this accession accession. Finally,	the plant lock vary different item what you would use in a traditional EDBMG. These are used institution ag-maly separity postimistics are supported in a jour products and the join here in he specified using the AMM join sprine such as
the p2 field of this sizest can be accessed by 12 3(72 key?) p2. With these constraints Have is able to memory structures of	NULECT (Lal as al., GM as al. PROM (LINEW COVID al GM2)
advinary complexity. The tables usualed in the manuer describe above are	invited of the more indifferent

. . .

DATA516/CSED516 - Fall 2021

#### Documents:

#### Relation

alai											
Hive - A Petabyte Scale Data Warehouse Using	i	i	1	r	0	1	1	5	5		

-11 -1 **A** 

Had	loop
Ashish Thusso, Joydeep Sen Sarma, Namit Jain, Zheng S and Raghed	tao, Prasad Chakka, Ning Zhang, Sarosh Antony, Hao Liu hara Murthy
Facebook Data In	Fasturities Team
	<text><text><text><text><text></text></text></text></text></text>
set in 2007 to a 70018 data set today. The infrastructure at that time was so inadequare that some duly data processing jobs were taking more than a day to process and the situation aray just goting worse with every passing day. We had an argum need for infrastructure that could scale along with our	concepts like tables, columns, cosst, and partitions. It supports all the major primitive types – integers, floats, doubles and strings – as well as a complex types such as maps, lists and structs. The latter can be needed addrardly to construct more couplex types. In addition, Niko allows suces to extand the

using with their and hear and therings. The same	Orienteensing into interfere and managing the
language in very similar in N/L and therefore can be easily understood by anyone familiar with N/L. There are some manues in the data model, iver voters and Bire/CL that are	implementation decouple for get/Diperforqueiter motion present in the Sorthe interface. More details on these interface can be found on the Erec wide [11], but the basis intercours here
different from traditional databases and that have been maintained by the respectances gained at Facebook. We will haddlade three and other details in the section.	is that any arbitrary data format and types rescaled therein on he plugged unit. Here by providing a jur that contains the intermentations for the XeeBs and Orienteenseties interface
A. Data Model and Type System	All the native Soften and complex types supported in Hir-
Ximilar to inskinoid databases, How stores data in inbles, where each table associate of a number of serve, and such new consists of a specified number of columns. Each scheme has	the proper associations have been made between the inble as the jae, the query layer institution on par with the native type and itematic As an example, the following statement allth
an associated type. The type is other a primitive type or a complex type. Canonity, the following primitive types are supported • Interes - Install Inters), pull Inters, smallini 2 Inters.	pe unidating the Xeelle and Objecknepecter interfaces to it desirbated cacher[1] to their it is available to Badeep an line proceeds to create the table with the caciom unde-
inspin(1 byte) All integer types are signed. • Floating point mathews - Ensityingle previous), double(double-previous)	alii jar (jarstophonai jar) CREATETABLEG ROW FORMAT MEREE (sona mpionai MyNeGe)
• Same	Note that, if remaining the table subsets could also be receipted
Here also satively supports the following complex types • Associative arrays - map key type, value type? • Look - Tel: Associative arrays	by composing the complex and primitive types.
<ul> <li>Xinuch - sinuci life name: Reld type,</li></ul>	# Query Language
These complex layers are inequired and one her composed in gene of arbitrary measures (spin reample). For example, hereinquireness, whereinquires, planet-planet, planet-planet is structure but in here contains two integer fields meaned planet planet in the mean and the set integer field means of the planet planet planet is the planet p	[32] and same evinesian flat as here found sould are measurement. Therefore a subscription of EUC stars in terms (i.e. in some of a stars, eight and stars and stars in the stars influence and measurements in the star in the star in the stars and a stars and a star in the star in the stars in the stars. The stars is a stars in the s
CHEATE TABLE (1)s using if this, is becomplosing, similarly int, p2 at (1);	sporying the system right array. Useful motadata horeven aspathlities like show inflex and describe are also process as so are explain plan capabilities to impact query plans (theng
(buty preprinting an anxiety many time in the branch story a <sup>™</sup> sparator. Values in the associative aways and loin can be avorened using [] queues. In the pervision reample, [13,00] gives the first element of the list and (13,00][hey] gives the	the pairs near very answer into the pro-trian set in inditional EDBMO. These are some limitations e.g. ex- spanity penducties are supported in a join producte and it joins have to be specified using the ANM join sprine such as
timit associated with key' in that associative array. Feadly, the p2 field of this struct can be accessed by 11.102%key'[p2]. With these constraints How is able to support structures of advisor constraints.	NULECT (i.e.) as al, (2.b) as a2 PROM of HORV COV(() (2 = C.b2))
The lables avoid in the manner describe dover are serialized and description units defail versions and	instead of the more inalitional
descriptions already protect in How Henever, there are	MERCE (Lat mod, Gibl and
programs or may even be legacy data. Here provides the flexibility is incorporate that data into a table without hering	W00000101-42-423-62
to transform the data, which can save substantial amount of time for large data sets. As we will describe in the later	Another Institution is in here inserts are done. Here current does not support inserting into an existing table or da
sections, this can be achieved by providing a jar that implements the XetTe jara interface in How In such	partition and all inserts overweite the existing dat Accordingly, we make this explosi is our system as follows:
situations the type information can also be provided by that jar	NOTIFIC CONTRACTOR TANKS & C.

. . .

did2

Did	Word
did1	Scalable
did1	analysis
did1	on
did1	large
did1	
did2	system
did2	with

select	word, count(*)
from	Data
group by	word

#### Relation

Did	Word
did1	Scalable
did1	analysis
did1	on
did1	large
did1	
did2	system
did2	with

select	word, count(*)
from	Data
group by	word

map = group by
reduce = count(...) (or sum(...) or...)

#### Relation

Did	Word
did1	Scalable
did1	analysis
did1	on
did1	large
did1	
did2	system
did2	with

select	word, count(*)		Relati	on
from	Data		Did	Word
group by	word		did1	Scalable
5 1 5			did1	analysis
			did1	on
			did1	large
			did1	
map = grou	o by		did2	system
<pre>reduce = count() (or sum() or)</pre>		did2	with	

MapReduce = Group-by-aggregate







#### Examples from the paper

Discuss in class how to implement in MR

- Distributed grep
- Count URL access frequency: (URL, count)
- Reverse web-link graph: (URL, (list of URLs))
- Inverted index: (word, (list of URLs)

#### Jobs v.s. Tasks

- A MapReduce Job
  - One simple "query", e.g. count words in docs
  - Complex queries may require many jobs
- A Map <u>Task</u>, or a Reduce <u>Task</u>
  - A group of instantiations of the map-, or reducefunction, to be scheduled on a single worker

#### Workers

- A worker is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node

### Fault Tolerance

- If one server fails once every year...
   ... then a job with 10,000 servers will fail in less than one hour
- MapReduce handles fault tolerance by writing intermediate files to disk:
  - Mappers write file to local disk
  - Reducers read the files (=reshuffling); if the server fails, the reduce task is restarted on another server



# Choosing Parameters in MR

- Number of map tasks (M):
  - Default: one map task per chunk
  - E.g. data = 64TB, chunk = 64MB  $\rightarrow$  M = 10<sup>6</sup>
- Number of reduce tasks (R):
  - No good default; set manually R << M</li>
  - E.g. R = 500 or 5000
- In general, MapReduce had very many parameters that required expertise to tune

#### **MapReduce Execution Details**



#### Discussion

Why doesn't MR determine the number of reduce tasks R dynamically, after all map tasks finish?

#### Discussion

Why doesn't MR determine the number of reduce tasks R dynamically, after all map tasks finish?

Because each map tasks needs to write its output into R file; so R must be known before the map tasks start

#### **MapReduce Phases**



- The combiner function performs an optimization that you already know
- Which one?

- The combiner function performs an optimization that you already know
- Which one?
- Pushing aggregates down

- The combiner function performs an optimization that you already know Temp=
- Which one?

select server, word, count(\*) as c from **Data** group by server, word

- Pushing aggregates down:
  - Each mapper groups by word

- The combiner function performs an optimization that you already know Temp=
- Which one?

select server, word, count(\*) as c from Data group by server, word

- Pushing aggregates down:
  - Each mapper groups by word
  - Reducers perform final group-by

Output = select word, sum(c) from **Temp** group by word

# Implementation

- There is one master node
- Master partitions input file into <u>M splits</u>, by key
- Master assigns *workers* (=servers) to the *M map* tasks, keeps track of their progress
- Workers write their output to local disk, partition into *R regions*
- Master assigns workers to the *R reduce tasks*
- Reduce workers read regions from the map workers' local disks

Blog by DeWitt and Stonebraker

• "Schemas are good"

- "Schemas are good"
- "Indexes"

- "Schemas are good"
- "Indexes"
- "Skew" (MR mitigates it somewhat, how?)

- "Schemas are good"
- "Indexes"
- "Skew" (MR mitigates it somewhat, how?)
- The M \* R problem what is it?

- "Schemas are good"
- "Indexes"
- "Skew" (MR mitigates it somewhat, how?)
- The M \* R problem what is it?
- "Parallel databases uses push (to sockets) instead of pull" – what's the point?

#### Outline

- Spark
- MapReduce and critique
- Fault Tolerance
- Hive (short)

Next lecture: Parallel databases
- Traditional RDBMs:
  - Major concern: recover after failure
  - FT: not a concern
- Massively distributed systems:
  - Probability of failure increases w/ no. of workers and length of job

Example:

- if a server fails once/year...
- ... a job with 10000 servers fails once/hour

How is fault tolerance handled in each system?

• MapReduce: if a worker fails then

• Spark:

How is fault tolerance handled in each system?

- MapReduce: if a worker fails then
  - All its completed map tasks need re-executed
  - Its in-progress reduce task needs re-executed

• Spark:

How is fault tolerance handled in each system?

- MapReduce: if a worker fails then
  - All its completed map tasks need re-executed
  - Its in-progress reduce task needs re-executed: this is possible because the map tasks still have intermediate data on their local disks
- Spark:

How is fault tolerance handled in each system?

- MapReduce: if a worker fails then
  - All its completed map tasks need re-executed
  - Its in-progress reduce task needs re-executed: this is possible because the map tasks still have intermediate data on their local disks
- Spark: will discuss next

# Approach

New abstraction: Resilient Distributed Datasets

#### RDD properties

- Parallel data structure
- Can be persisted in memory
- Fault-tolerant
- Users can manipulate RDDs with rich set of operators

## **Resilient Distributed Datasets**

- RDD = Resilient Distributed Dataset
  - Distributed, immutable.
  - Records lineage = expression that says how that relation was computed = a relational algebra plan
- Spark stores intermediate results as RDD
- If a server crashes, its RDD in main memory is lost. However, the driver (=master node) knows the lineage, and will simply recompute the lost partition of the RDD



If any server fails before the end, then Spark must restart



If any server fails before the end, then Spark must restart





If any server fails before the end, then Spark must restart





in memory

or on disk



#### SELECT count(\*) FROM R, S WHERE R.B > 200 and S.C < 100 and R.A = S.A

## Example





## **RDD** Details

- An RDD is a partitioned collection of records
  - RDD's are typed: RDD[Int] is an RDD of integers
  - Records are Java/Python objects
- An RDD is read only
  - This means no updates to individual records
  - This is to contrast with in-memory key-value stores
- To create an RDD
  - Execute a deterministic operation on another RDD
  - Or on data in stable storage
  - Example operations: map, filter, and join

#### **RDD** Materialization

• Users control persistence and partitioning

• Persistence

Materialize this RDD in memory

• Partitioning

- Users can specify key for partitioning an RDD

#### Outline

- Spark
- MapReduce and critique
- Fault Tolerance
- Hive (short)

Next lecture: Parallel databases

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)

- Note: this is similar to Google's FlumeJava

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:
  - Column pruning

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:
  - Column pruning
  - Predicate push-down

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:
  - Column pruning
  - Predicate push-down
  - Partition pruning

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:
  - Column pruning
  - Predicate push-down
  - Partition pruning
  - Map-side join = "broadcast join" (discuss in class)

- Facebook's implementation of SQL over MR
- Supports subset of SQL
- Uses MapReduce runtime (pros/cons?)
   Note: this is similar to Google's FlumeJava
- Optimizations:
  - Column pruning
  - Predicate push-down
  - Partition pruning
  - Map-side join = "broadcast join" (discuss in class)
  - Join reordering

## Discussion

- Parallel database systems: since the 80s
  Will discuss next lecture
- MapReduce: around 2000
- Hive: built on MapReuce
- Spark: "better" MapReduce around 2010
- Snowflake, Aurora: cloud, parallel databases; around 2015 (next lecture)

Quick comparison (next slides)

#### MapReduce v.s. Spark

- Job = Map+Reduce
  Job = any query
- Language = Java
  Language ≈ RA
- Data = untyped
- Optimization = no

- Data = has schema
- Optimization = yes but limited: missing stats on base data

#### Spark v.s. RDBMS (e.g. Snowflake)

- Query language = its own proprietary
- Optimizer = limited
- Runtime = its own proprietary
- External functions = yes; very useful in ML

- Query language = SQL
- Optimizer = full scale
- Runtime = efficient SQL query engine
- External functions = no