PART A. BIG DATA TECHNOLOGY
3. DISTRIBUTED COMPUTING
MODELS FOR SCALABLE BATCH
COMPUTING
SECTION 1: MAPREDUCE

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

• PA1
  • Your port ranges are announced
  • Guideline for the on-line students is available on the course web page (Assignments.html)
• GEAR session 1 papers are available
  • How to write a good critical review
• Quiz 1 (Feb 7 ~ 9, Friday, Saturday, Sunday)
  • Accommodation Request
• CompositeInputFormat () with the sorted data (CS435 Week 4-B)
  • An InputFormat capable of performing joins over a set of data sources sorted and partitioned the same way. A user may define new join types by setting the property mapred.join.define.<ident> to a classname.
  • In the expression mapred.join.expr, the identifier will be assumed to be a ComposableRecordReader.mapred.join.keycomparator can be a classname used to compare keys in the join.
  • This requires data preprocessing to sort dataset
How do we write a critical review?

- **Objective**
  - Identify and explain the argument that the authors are making
  - Provide your own analysis about the authors’ argument

- Your review should not exceed 3 pages
  - Your list of references does not count towards the page limit

- Team Submission

- **Do not summarize the paper**
  - You are not re-writing somebody else's research paper
  - You are analyzing the author's argument

---

How do we write a critical review?

- Follow the structural guidelines available at: [https://www.cs.colostate.edu/~cs535/GEARSessions.html](https://www.cs.colostate.edu/~cs535/GEARSessions.html)

1. Introduction
2. Short Summary
3. Your Analysis
4. Conclusion
5. References
How do we write a critical review?  |  1. Introduction

- a. Identify the research problem(s)
  - Objectives and challenges
- b. Identify the research question(s)
  - *Your review much include items a and b.*

How do we write a critical review?  |  2. Short Summary

- The summary does not need to be comprehensive. Present only what the readers need to know to understand your argument.
  - *Your review much include this item.*
How do we write a critical review? | 3. Your Analysis

- You can analyze **whether or not** you find the author's argument compelling.
- Example questions for evaluating their arguments
  - Theoretical questions
    - How does the author encapsulate the phenomena or approach? Does it provide sufficient theoretical background (if needed)?
  - Definitional questions
    - Are all the concepts in the text clear? Does the author define a concept vaguely to allow it to travel across different situations?
  - Evidence questions
    - Was the author's evidence (e.g. experiments, proofs) sufficient to support their argument? Do they have a specific evidence to prove the more general point?
    - Does the author underemphasize or ignore evidence that is contrary to their argument?
    - Is the evidence credible? Is the setup of experiment reasonable?
    - Can we identify a bias in the evidence?
  - Implication/Policy relevant questions
    - What are the implications of this argument? How has the author dealt with this issue?

How do we write a critical review? | 4. Conclusion

- a. Reflect on how you have proven your argument.
- b. What are the **advantages** of the proposed approach?
- c. What are the **weaknesses** of the proposed approach?
- d. What problems are **explicitly or implicitly left as future research questions**?
- **Your review must include items a, b, c, and d.**
How do we write a critical review? | References

- If your argument uses other articles, you should list those in a separate reference section and cross-reference that within your text.

Topics of Today's Class

- 3. Distributed Computing Models for Scalable Batch Computing
  - MapReduce –II
  - Introduction to Spark

- Reading for the Week 3
How MapReduce Works

Programming components of MapReduce

- Driver
- Mapper
- Reducer
- InputFormat
- Combiner
- Partitioner
- OutputFormat

http://www.cs.colostate.edu/~cs535
Data locality optimization

- Hadoop tries to run the map task on a node where the input data resides in HDFS
  - Minimizes usage of cluster bandwidth

- If all replication nodes are running other map tasks
  - The job scheduler will look for a free map slot on a node in the same rack
Data movement in Map tasks

Shuffle

- The process by which the system performs the sort and transfers the map outputs to the reducers as inputs
  - MapReduce guarantees that the input to every reducer is sorted by key
Combiner functions

- Minimize data transferred between map and reduce tasks

- Users can specify a *combiner function*
  - To be run on the map output
  - To replace the map output with the combiner output

Combiner example

- Example (from the previous max temperature example)—Without combiner
  - The first map produced,
    - (1950, 0), (1950, 20), (1950, 10)
  - The second map produced,
    - (1950, 25), (1950, 15)
  - The reduce function is called with a list of all the values,
    - (1950, [0, 20, 10, 25, 15])
  - Output will be,
    - (1950, 25)

  We may express the function as,
  - \( \max(0, 20, 10, 25, 15) \)
    - \( = \max( \max(0, 20, 10), \max(25, 15)) \)
    - \( = \max(20, 25) = 25 \)
Combiner example

• Example (from the previous max temperature example)—With combiner
  • The first map produced,
    • (1950, 0), (1950, 20), (1950, 10). \(\Rightarrow\) (1950, 20)
  
  • The second map produced,
    • (1950, 25), (1950, 15) \(\Rightarrow\) (1950, 25)
  
  • The reduce function is called with a list of all the values,
    • (1950, [20, 25])
  
  • Output will be,
    • (1950, 25)

Combiner function

• Run a **local** reducer over Map output

• Reduce the amount of data shuffled between the mappers and the reducers

• Combiner cannot replace the reduce function
  • Why?
Combiner function : Requirements

• Function should be **commutative and associative**

  • Finding Maximum number (Yes/No)
  • Finding distribution (Yes/No)
  • Calculating Sum (Yes/No)
  • Finding an average (Yes/No)

• Finding an average (Yes/No) : if your combiner deliver the count of items, it is still possible
YARN Framework

YARN (MapReduce 2)

- To provide the *scalability* to MapReduce
  - Splitting responsibility of the jobtracker
    - Scheduling
    - Task progress monitoring

- MapReduce is one type of YARN application
**YARN (MapReduce 2)**

- **Resource manager**
  - Manages the use of resources across the cluster

- **Node manager**
  - Launches and monitors the compute containers on machines in the cluster

- **Application master**
  - Manages the lifecycle of applications running on the cluster
  - Application master negotiates with the resource manager for cluster resources
    - Number of container and certain memory limit
    - Node managers oversee containers not to use more resources than allocated

---

**A MapReduce job using YARN**

1. Run job
2. Get new Application
3. Copy job resources
4. Submit application
5a. Start container
5b. Launch MR AppMaster
6. Initialize job
7. Retrieve input splits
8. Allocate resources
9a. Start container
9b. Launch Task JVM
10. Retrieve job resources
11. Run MapTask or reduce task

---

http://www.cs.colostate.edu/~cs535
Progress and status updates

- Task reports its progress and status back to its application master
  - Every 3 seconds over the umbilical interface

- The client polls the application master every second
  - mapreduce.client.progressmonitor.pollinterval
In-Memory Cluster Computing: Apache Spark

Introduction
This material is built based on


- Spark programming guide
  - https://spark.apache.org/docs/2.2.0/
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

Distributed processing with the Spark framework
Inefficiencies for emerging applications: (1) Data reuse

- Data reuse is common in many iterative machine learning and graph algorithms
  - e.g. PageRank, K-means clustering, and logistic regression

<table>
<thead>
<tr>
<th>Type</th>
<th>Access Time</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>10 nanoseconds</td>
<td>1</td>
</tr>
<tr>
<td>DRAM</td>
<td>50-150 nanoseconds</td>
<td>x 5-15</td>
</tr>
<tr>
<td>HDD</td>
<td>9-15 milliseconds</td>
<td>x 900 ~ 1,500</td>
</tr>
</tbody>
</table>

Inefficiencies for emerging applications: (2) Interactive data analytics

- User runs multiple ad-hoc queries on the same subset of the data
### Existing approaches

- **Hadoop**
  - Writing output to an external stable storage system
    - e.g. HDFS
    - Substantial overheads due to data replication, disk I/O, and serialization

- **Pregel**
  - Iterative graph computations

- **HaLoop**
  - Iterative MapReduce interface

- **Pregel/HaLoop** support specific computation patterns
  - e.g. looping a series of MapReduce steps

### A unified stack

- **Spark** contains multiple closely integrated components
  - **Spark core**
    - Computational engine
    - Scheduling, distributing, and monitoring applications

- **Spark Streaming**
  - Processes live streams of data

- **MLlib**
  - Machine learning functionality
  - ML algorithms (classification, regression, clustering and collaborative filtering)
  - Model evaluation
  - Data import
A unified stack

- **GraphX**
  - Library for manipulating graphs
  - Performs graph-parallel computations
  - Extends the Spark RDD API

- **Cluster Managers**
  - Spark can run over a variety of cluster managers
  - Hadoop YARN, Apache Mesos, and Spark built-in cluster manager (Standalone scheduler)

---

Running a simple example

```scala
import org.apache.spark.sql.SparkSession
object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

https://spark.apache.org/docs/latest/quick-start.html
Scala Tutorial: https://www.tutorialspoint.com/scala/
Self-contained Application | sbt build file

```scala
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.12"

// additional libraries
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.4"
```

https://spark.apache.org/docs/latest/quick-start.html

Scala build and run

```bash
# Your directory layout should look like this
$ find .
.
./build.sbt
./src
./src/main
./src/main/scala
./src/main/scala/SimpleApp.scala

# Package a jar containing your application
$ sbt package
...
[info] Packaging {..}/target/scala-2.12/simple-project_2.12-1.0.jar

# Use spark-submit to run your application
$ YOUR_SPARK_HOME/bin/spark-submit \
--class "SimpleApp" \
--master local[4] \
target/scala-2.12/simple-project_2.12-1.0.jar \
...
```

https://spark.apache.org/docs/latest/quick-start.html

http://www.cs.colostate.edu/~cs535
In-Memory Cluster Computing: Apache Spark

RDD (Resilient Distributed Dataset)

- Read-only, memory resident partitioned collection of records
  - A fault-tolerant collection of elements that can be operated on in parallel

- RDDs are the core unit of data in Spark
  - Most Spark programming involves performing operations on RDDs
Creating RDDs

- Loading an external dataset

```scala
val lines = sc.textFile("/path/to/README.md")
```

- Parallelizing a collection in your driver program

```scala
val lines = sc.parallelize(List("pandas", "i like pandas"))
```


---

Creating RDDs

1: `val lines = sc.textFile("data.txt")`
2: `val lineLengths = lines.map(s => s.length)`
3: `val totalLength = lineLengths.reduce((a, b) => a + b)`

- Line 1: defines a base RDD from an external file
  - This dataset is not loaded in memory

- Line 2: defines `lineLengths` as the result of map transformation
  - It is not immediately computed

- Line 3: performs reduce and compute the results
Creating RDDs

- If you want to use `lineLengths` again later

```scala
lineLengths.persist()
```

Spark Programming Interface to RDD

- **transformations**
  - Operations that create RDDs
    - Return pointers to new RDDs
    - e.g. `map`, `filter`, and `join`
  - RDDs can only be created through deterministic operations on either
    - Data in stable storage
    - Other RDDs

```scala
1: val lines = sc.textFile("data.txt")
2: val lineLengths = lines.map(s => s.length)
3: val totalLength = lineLengths.reduce((a, b) => a + b)
```
Spark Programming Interface to RDD

• **actions**
  - Operations that return a value to the application or export data to a storage system
    - e.g. count: returns the number of elements in the dataset
    - e.g. collect: returns the elements themselves
    - e.g. save: outputs the dataset to a storage system

```scala
1: val lines = sc.textFile("data.txt")
2: val lineLengths = lines.map(s => s.length)
3: val totalLength = lineLengths.reduce((a, b) => a + b)
```

• **persist**
  - Indicates which RDDs they want to **reuse in future operations**
  - Spark keeps persistent RDDs **in memory** by default
  - If there is not enough RAM
    - It can spill them to disk
  - Users are allowed to,
    - store the RDD only on disk
    - replicate the RDD across machines
    - specify a persistence priority on each RDD

```scala
lineLengths.persist() /n```
Questions?