Extreme Computing

Distributed Data-Parallel Programming

Amir Shaikhha, Fall 2023
Part 1

Programming Models/Languages
THE INTERNET IN 2023 EVERY MINUTE

- 271,309 visits to ChatGPT
- 241.2M emails sent
- 18.8M text messages sent
- 694,000 video hours viewed
- 347,222 tweets
- 2.4M Google searches
- 6.94M emoji sent
- 3.02M photos created with smartphones
- 34,247 Slack messages
- 11,834 chats on Microsoft Teams
- 6.3M total Zoom meeting minutes
- 10.4M viewing minutes
- 11,035 fake accounts removed
- 3.47M snaps created

Mainstream Languages for Data Scientists

Python

R

Tableau
Mainstream Languages for Data Scientists (cont.)

Pros
✓ Rapid Development
✓ Large community

Cons
❖ What to do with large datasets?

Rewrite from scratch 😞
Is there any language without this issue?
Why Scala is related to BigData?

How Technologies Are Connected

https://insights.stackoverflow.com/survey/2019
Mainstream Big Data models

How to store, manage and process Big Data by harnessing large clusters of commodity nodes

• MapReduce family: simpler, more constrained
  
  ![Hadoop Logo](image1)  ![Hive Logo](image2)

• Dataflow family: enables more complex processing & data, optimization opportunities

  ![Spark Logo](image3)  ![Google Pregel Logo](image4)  ![Microsoft Dryad Logo](image5)
The Hadoop Ecosystem

<table>
<thead>
<tr>
<th>Storm, Flink (Streaming)</th>
<th>Giraph, Hama (Graph)</th>
<th>Pig, Hive (Query)</th>
<th>Other Hadoop libraries</th>
</tr>
</thead>
</table>

- **Hadoop MapReduce**
  Distributed Batch Processing Framework

- **YARN**
  Cluster resource management

- **HDFS**
  Hadoop Distributed File System
3.2.1 Apache Spark

Apache Spark is another top project of Apache Software Foundation for Big Data analysis. Differently from Hadoop, in which intermediate data are always stored in distributed file systems, Spark stores data in RAM memory and queries it repeatedly so as to obtain better performance for some classes of applications compared to Hadoop [13]. A Spark application is defined as a set of independent stages running on a pool of worker nodes. A stage is a set of tasks executing the same code on different partitions of input data.

Spark and Hadoop are considered the leading open source Big Data systems and thus are supported by every major Cloud providers. As shown in Figure 4, different libraries have been built on top of Spark: Spark SQL for dealing with SQL and Data Frames, MLlib for machine learning, GraphX for graph-parallel computation, Spark Streaming for building streaming applications. The execution of a generic Spark application on a cluster is driven by a central coordinator (i.e., the main process of the application), which can connect with different cluster managers, such as Apache Mesos, YARN, or Spark Standalone (i.e., a cluster manager available as part of the Spark distribution). Ambari can be used for provisioning, managing, and monitoring Spark clusters.

Spark Core
Processing Engine

Mesos / YARN / Standalone
Cluster Resource Management

HDFS / Amazon S3 / OpenStack Swift / Cassandra
Distributed File System & Storage

Even though in some classes of applications Spark is considered a better alternative to Hadoop, in many others it has limitations that make it complementary to Hadoop. The main limitation of Spark is that datasets should fit in RAM memory. In addition, it does not provide its own distributed storage system, which is a fundamental requirement for Big Data applications. To overcome this lack, Spark has been designed to run on top of several data sources, such as distributed file systems (e.g., HDFS), Cloud object storages (e.g., Amazon S3, OpenStack Swift) and NoSQL databases (e.g., Cassandra).

Spark's real-time processing capability is increasingly being used into applications that requires to extract insights quickly from data, such as recommendation and monitoring systems. For this reason, several big companies exploit Spark for data analysis purpose: SK Telecom analyzes mobile usage patterns of customers, Ebay uses Spark for log aggregation, and Kelkoo for product recommendations.
PROGRAMMING MODELS
The vision...
Sample function: convert all text to upper case

Splits may be stored at diff. nodes

Huge data file

split 1 -> Map -> out 1
split 2 -> Map -> out 2
split 3 -> Map -> out 3
split 4 -> Map -> out 4
split 5 -> Map -> out 5

Result

convertUpper()
The vision (2)
More complicated: the word-count problem
• Huge file → extract frequencies of words
• Example

Logic will get you from A to B. Imagination will take you everywhere.

Extracted frequencies:
• <Logic,1>, <will,2>, <get,1>, <you,2>, …
The vision (3)

Sample application: the word-count example

Huge data file

split 1 → Map → Reduce → out 1
split 2 → Map → Reduce → out 2
split 3 → Map → Reduce → out 3
split 4 → Map → Reduce
split 5 → Map

Merging of results grouped by word

Local computation

Result
MapReduce programming model

• Data model: everything is a <key,value> pair
• Programming model - two core functions
  – Map(key,value): Invoked for every split of the input data. Value corresponds to the split.
  – Reduce(key,list(values)): Invoked for every unique key emitted by Map. List(values) corresponds to all values emitted from ALL mappers for this key.
• These are second-order functions
  – Map(key,value, MapperClassName)
  – Reduce(key,list(values), ReducerClassName)
→ parallelism and deployment handled by the system
MapReduce programming model (2)

- The word-count problem
  - Input: Text file, broken in splits
  - Output: Frequency of each word observed in the file
  - Map(key, value): value: a split of the text file
    
    
    
    for each word in value
    emit pair <word, +1>

  - Reduce(key, list(values)): Key: word, values: list of (+1’s)
    count=0
    for each value in list(values)
    count += value
    emit pair <key, count>
for each word in value
emit pair <word,+1>

for each value in list(values)
count+=value
emit pair<word,count>
Dataflow programming model

- MapReduce simple but weak for some reqs.
  - Cannot define complex processes
  - Everything file-based, no distributed memory
  - Procedural → difficult to optimize

- Dataflow
  - Processing expressed as a DAG, tree, graph with cycles, …
  - Vertices: processing tasks
  - Edges: Communication
    - DAG: Spark, Dryad
    - Tree: Dremel
    - Directed graph with cycles: Pregel
Dataflow programming model (2)

- Describing the processing tasks
  - Declarative languages, e.g., Dremel
    
    ```
    SELECT DocId AS Id, COUNT(Name.Language.Code) WITHIN Name AS Cnt FROM t 
    WHERE REGEXP(Name.Url, '^http');
    ```

  - Functional programming, e.g., Spark
    
    ```
    val wordCounts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)
    ```

  - Domain-specific languages, e.g., Pregel
    
    ```
    class PageRankVertex : public Vertex<double, void, double> {
        public: virtual void Compute(MessageIterator* msgs) {
            const int64 n = GetOutEdgeterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        }
    };
    ```
Why Spark? (1)

Scala Collections  

Similar API 😊
Which programming language is this?

```java
Integer totalAgeReduce = roster.stream()
    .map(Person::getAge)
    .reduce(0, (a, b) -> a + b);

Map<String, List<String>> a = words.stream().collect(
    Collectors.groupingBy(w -> sortChars(w)));
```
PLs that have a functional collection interface like Scala

C++, C#, F#, Clojure, Haskell, Java8, JavaScript, Perl, PHP, Python, Ruby, Scheme, Smalltalk, Standard ML, OCAML, ...

See

https://en.wikipedia.org/wiki/Map_(higher-order_function)
Fault Tolerance

• Essential for scaling out
• The main reason behind the success of MapReduce in Google
• Requires writing intermediate data to disk
Fault Tolerance in Spark

- **Data**
  - Immutable
  - In-memory
- **Operations = Functional transformations**
- **Fault tolerance = Replay operations**
Why Spark? (2)

- Compared to Hadoop MapReduce, improves efficiency through:
  - General execution graphs
  - In-memory storage

Up to $10 \times$ faster on disk, $100 \times$ in memory

Logistic regression in Hadoop and Spark

http://spark.apache.org/
Why Spark? (3)
Why Spark? (4)
Learn Scala

Scala School!
From ø to Distributed Service

Other Languages:
한국어
Русский
簡体中文

About
Scala school started as a series of lectures at Twitter to prepare experienced engineers to be productive Scala programmers. Scala is a relatively new language, but draws on many familiar concepts. Thus, these lectures assumed the audience knew the concepts and showed how to use them in Scala. We found this an effective way of getting new engineers up to speed quickly. This is the written material that accompanied those lectures. We have found that these are useful in their own right.

Lessons

Basics
Values, functions, classes, methods, inheritance, try-catch-finally. Expression-oriented programming

Basics continued
Case classes, objects, packages, apply, update, Functions are Objects (uniform access principle), pattern matching.

Collections
Lists, Maps, functional combinators (map, foreach, filter, zip, folds)

Pattern matching & functional composition
More functions! PartialFunctions, more Pattern Matching

Type & polymorphism basics
Basic Types and type polymorphism, type inference

https://twitter.github.io/scala_school/
QUESTIONS?