Frequently asked questions from the previous class survey

- Is distcp across different clusters possible?
- For what application would sync be slow?
- Write pipeline: D1, D2, and D3 ... what if D2 fails?
- Uncompress and then MapReduce?
- How slow is slow?

Instructor: SHRIDEEP PALLICKARA

March 22, 2018

CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

Topics covered in this lecture

- Spark
  - Software stack
  - Interactive shells in Spark
  - Core Spark concepts
  - Resilient Distributed Datasets

HDFS WRAP-UP

HDFS does not split gzip files

- Single map will process 16 HDFS blocks
- Most of these blocks will not be local to the map
  - Loss of locality
  - Job is not granular ... takes much longer to run

The same story plays out if you were dealing with LZO files, but ...

- It is possible to preprocess LZO files using an indexer tool
- Build an index of split points
Bzip2
- This does provide a synchronization marker between blocks
- 48-bit approximation of \( \pi \)
- The marker is used to support splitting

Dealing with large, unbounded files [Log files]
1. Store the files uncompressed
2. Use compression format that supports
   - Splitting: Bzip2
   - Indexing to support splitting: LZO
3. Split the file into chunks in the application and compress each chunk separately
   - Choose chunk sizes such that the compressed chunks are approximately the size of an HDFS block

Using compression in MapReduce
- To compress the output of MapReduce job
  - In the job configure mapred.output.compress property to true
  - Use mapred.output.compression.codec to specify the codec
- Alternatively, we can do this using the FileOutputFormat

Using the FileOutputFormat
```
public class MaxTemperatureWithCompression {
    public static void main(String[] args) throws Exception {
        Job job = Job.getInstance();
        job.setJarByClass(MaxTemperature.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileOutputFormat.setCompressOutputClass(job, GzipCodec.class);
        job.setInputFormatClass(MaxTemperatureInputFormat.class);
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Main reason why Hadoop does not use Java Serialization
- Deserialization creates new instance of each object being deserialized
- Writable objects can be (and are often) reused
- Large MapReduce jobs often serialize/deserialize billions of records
- Savings from not having to allocate new objects is significant

APACHE SPARK
As distributed data analytics have grown common …

- Practitioners have sought easier tools for the task
- Apache Spark has emerged as one of the most popular
  - Extending and generalizing MapReduce

Spark: What is it?

- Cluster computing platform
  - Designed to be fast and general purpose
- Speed
  - Extends MapReduce to support more types of computations
    - Interactive queries, iterative tasks, and stream processing
- Why is speed important?
  - Difference between waiting for hours versus exploring data interactively

Spark: Influences and Innovations

- Spark has inherited parts of its API, design, and supported formats from other existing computational frameworks
  - Particularly DryadLINQ
- Spark’s internals, especially how it handles failures, differ from many traditional systems
- Spark’s ability to leverage lazy evaluation within memory computations makes it particularly unique

Where does Spark fit in the Analytics Ecosystem?

- Spark provides methods to process data in parallel that are generalizable
- On its own, Spark is not a data storage solution
  - Performs computations on Spark JVMs that last only for the duration of a Spark application
- Spark is used in tandem with:
  - A distributed storage system (e.g., HDFS, Cassandra, or S3)
    - To house the data processed with Spark
  - A cluster manager — to orchestrate the distribution of Spark applications across the cluster

Key enabling idea in Spark

- Memory resident data
  - Spark loads data into the memory of worker nodes
  - Processing is performed on memory-resident data

A look at the memory hierarchy

<table>
<thead>
<tr>
<th>Item</th>
<th>Time</th>
<th>Scaled time in human terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor cycle</td>
<td>0.5 ns</td>
<td>2 billion times slower</td>
</tr>
<tr>
<td>Cache access</td>
<td>1 ns</td>
<td>2 seconds</td>
</tr>
<tr>
<td>Memory access</td>
<td>70 ns</td>
<td>1.4 seconds</td>
</tr>
<tr>
<td>Context switch</td>
<td>5,000 ns</td>
<td>167 minutes</td>
</tr>
<tr>
<td>Disk access</td>
<td>7,000,000,000 ns</td>
<td>162 days</td>
</tr>
<tr>
<td>Quantum</td>
<td>100,000,000,000 ns</td>
<td>6.3 years</td>
</tr>
</tbody>
</table>

Spark covers a wide range of workloads

- Batch applications
- Iterative algorithms
- Queries
- Stream processing
- This has previously required multiple, independent tools

Spark covers a wide range of workloads

At its core, Spark is a computational engine

- Spark is responsible for several aspects of applications that comprise
  - Many tasks across many machines (compute clusters)
- Responsibilities include:
  1. Scheduling
  2. Distributions
  3. Monitoring

APIs

- Java, Python, Scala, and SQL
- Integrates well with other tools
  - Can run in Hadoop clusters
  - Access Hadoop data sources, including Cassandra

THE SPARK SOFTWARE STACK

The Spark stack

- Spark SQL: structured data
- Spark Streaming: real-time
- MLlib & MLL: machine learning
- GraphX: Graph processing

Spark Core

- Standalone Scheduler
- YARN
- Mesos

Benefits of tight integration [1/2]

- All libraries and higher-level components benefit from improvements at the lower layers
- E.g.: Spark’s core engine adds optimization? SQL and ML libraries automatically speed-up as well
Benefits of tight integration [2/2]
- Biggest advantage is ability to build applications that seamlessly combine different processing models
- An application may use ML to classify data in real time as it is being ingested
  - Analysts can query this resulting data, also in real time, via SQL (e.g.: join data with unstructured log-files)

Spark Core
- Basic functionality of Spark
  - Task scheduling, memory management, fault recovery, and interacting with storage systems
  - Also, the API that defines Resilient Distributed Datasets (RDDs)
    - Spark's main programming abstraction
      - Represents collection of data items dispersed across many compute nodes
        - Can be manipulated concurrently (parallel)

Spark SQL
- Package for working with structured data
- Allows querying data using SQL and HQL (Hive Query Language)
  - Data sources: Hive tables, Parquet, and JSON
- Allows intermixing queries with programmatic data manipulations support by RDDs
  - Using Scala, Java, and Python

Semi-structured data and Spark SQL
- Spark SQL defines an interface for a semi-structured data type, called DataFrames
- And as of Spark 1.6, a semi-structured, typed version of RDDs called Datasets
- Spark SQL is a very important component for Spark performance
- Much of what can be accomplished with Spark Core can be done by leveraging Spark SQL.

Spark Streaming
- Enables processing of live streams of data from sources such as:
  - Logfiles generated by production web servers
  - Messages containing web service status updates
- Uses the scheduling of the Spark Core for streaming analytics on minibatches of data
- Has a number of unique considerations, such as the window sizes used for batches

MLib
- Library that contains common machine learning functionality
- Algorithms include:
  - Classification, regression, clustering, and collaborative filtering
  - Low-level primitives
    - Generic gradient descent optimization algorithm
- Alternatives?
  - Mahout, sci-kit learn, VW, WEKA, and R among others
What about Spark ML?
- Still in the early stages, and has only existed since Spark 1.2
- Spark ML provides a higher-level API than MLlib
  - Goal is to allow users to more easily create practical machine learning pipelines
  - Spark MLlib is primarily built on top of RDDs and uses functions from Spark Core, while ML is built on top of Spark SQL DataFrames
- Eventually the Spark community plans to move over to ML and deprecate MLlib

Graph X
- Library for manipulating graphs
- Graph-parallel computations
- Extends Spark RDD API
  - Create a directed graph, with arbitrary properties attached to each vertex and edge

Cluster Managers
- Spark runs over a variety of cluster managers
- These include:
  - Hadoop YARN
  - Apache Mesos
  - Standalone Scheduler
    - Included within Spark

Storage Layers for Spark
- Spark can create distributed datasets from any file stored in HDFS
- Plus, other storage systems supported by the Hadoop API
  - Amazon S3, Cassandra, Hive, HBase, etc.

Spark Shells
- Interactive [Python and Scala]
- Similar to shells like Bash or Windows command prompt
- Ad hoc data analysis
- Traditional shells manipulate data using disk and memory on a single machine
- Spark shells allow interaction with data that is distributed across many machines
- Spark manages complexity of distributing processing
Several software were designed to run on the Java Virtual Machine.

- Languages that compile to run on the JVM and can interact with Java software packages but are not actually Java.
- There are a number of non-Java JVM languages:
  - The two most popular ones used in real-time application development: Scala and Clojure.

### Scala

- Has spent most of its life as an academic language.
- Still largely developed at universities.
- Has a rich standard library that has made it appealing to developers of high-performance server applications.
- Like Java, Scala is a strongly typed object-oriented language.
- Includes many features from functional programming languages that are not in standard Java.
- Interestingly, Java 8 incorporates several of the more useful features of Scala and other functional languages.

What is functional programming?

- When a method is compiled by Java, it is converted to instructions called byte code and ... Then largely disappears from the Java environment.
- Except when it is called by other methods.
- In a functional language, functions are treated the same way as data.
  - Can be stored in objects similar to integers or strings, returned from functions, and passed to other functions.

### What about Clojure?

- Based on Lisp.
- Javascript
  - Name was a marketing gimmick.
  - Closer to Clojure and Scala, than it is to Java.

### Core Spark Concepts

- Drivers
- SparkContext
- Executors
Spark in a nutshell

- Spark allows users to write a program for the driver (or master node) on a cluster computing system that can perform operations on data in parallel
- Spark represents large datasets as RDDs which are stored in the executors (or worker nodes)
- The objects that comprise RDDs are called partitions and may be (but do not need to be) computed on different nodes of a distributed system
- The Spark cluster manager handles starting and distributing the Spark executors across a distributed system

Drivers

- Every Spark application consists of a driver program
- Driver launches various parallel operations on the cluster
- Constituent elements
  - Application’s main function
  - Defines distributed datasets on the clusters
  - Applies operations to these datasets

Drivers

- Every Spark application consists of a driver program
- Driver manages a number of nodes, called executors
- Executors are responsible for running operations
- For example:
  - If we were running a count() operation on cluster
    - Different machines might count lines in different ranges of the file

Components for distributed execution in Spark

SparkContext

- Driver programs access Spark through a SparkContext object
  - Represents a connection to a computing cluster
- Within the shell?
  - Created as the variable sc
    - You can even print out sc to see the type
- Once you have a SparkContext, you can use it to build RDDs
  - And then run operations on the data …

Lot of Spark’s API revolves around passing functions to its operators

```
def hasPython(line):
    return "Python" in line
pythonLines = lines.filter(hasPython)
```

Also known as the lambda or => syntax
Lot of Spark’s API revolves around passing functions to its operators.

```java
JavaRDD<String> pythonLines = lines.filter(new Function<String, Boolean>() {
    Boolean call(String line) {
        return line.contains("Python");
    }
});
```

Resilient Distributed Dataset (RDD)

- RDD is an immutable distributed collection of objects.
- Each RDD is split into multiple partitions.
  - Maybe computed on different nodes in the cluster.
- Can contain any type of Java, Scala, or Python objects.
  - Including user-defined classes.

Creation of RDDs

1. Loading an external dataset
2. Distributing a collection of objects via the driver program.

```python
>>> lines = sc.textFile("README.md")
```

Once created, RDDs offer two types of operations.

- **Transformations**
  - Construct a new RDD from a previous one.
  - E.g., Filtering data that matches a predicate.
- **Actions**
  - Compute a result based on an RDD.
  - Return result to the driver program or save it in external storage system (HDFS).

Some more about RDDs

- Although you can define new RDDs anytime.
  - Spark computes them in a lazy fashion.
  - When?
    - The first time they are used in an action.
  - Loading lazily allows transformations to be performed before the action.
Lazy loading allows Spark to see the whole chain of transformations

- Allows it to **compute just the data needed** for the result.
- Example:
  ```python
  lines = sc.textFile("README.md")
  pythonLines = lines.filter(lambda line: "Python" in line)
  ```
- If Spark were to load and store all lines in the file, as soon as we wrote lines=sc.textFile()

March 22, 2018
CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

 RDD and actions

- **RDDs are recomputed** (by default) every time you run an action on them.
- If you wanted to **reuse** an RDD:
  - Ask Spark to **persist** it using RDD.persist()
  - After computing it the first time, Spark will store RDD contents in memory (partitioned across cluster machines)
  - Persisted RDD is used in future actions.

March 22, 2018
CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

Every Spark program and shell works as follows

1. **Create** some input RDD from external data
2. **Transform** them to define new RDDs using transformations like filter()
3. **Ask Spark** to persist() any intermediate RDDs that needs to be reused
4. **Launch actions** such as count(), etc. to kickoff a parallel computation

Computing is optimized and executed by Spark.

March 22, 2018
CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

The contents of this slide-set are based on the following references

- Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. Byron Ellis. Wiley. (Chapter 2)