**Frequently asked questions from the previous class survey**

- Where is the block metadata stored?
- Control plane traffic?
- In the parallel copy situation, how is replication handled by destination?
- When exactly are blocks visible? After 1 block is written OR only after the sync?
- With compressed data, is it possible that the number of splits being processed by different mappers is different?
- In a MR job, where is the compression being performed?

**Topics covered in this lecture**

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

**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
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 (2 GHz)</td>
<td>1 second</td>
</tr>
<tr>
<td>Cache access</td>
<td>1 ns (1 GHz)</td>
<td>2 seconds</td>
</tr>
<tr>
<td>Memory access</td>
<td>70 ns</td>
<td>1.40 seconds</td>
</tr>
<tr>
<td>Context switch</td>
<td>5,000 ns (5 μs)</td>
<td>167 minutes</td>
</tr>
<tr>
<td>Disk access</td>
<td>7,000,000 ns (7 ms)</td>
<td>162 days</td>
</tr>
<tr>
<td>Quantum</td>
<td>100,000,000 ns (100 ms)</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

APIs

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

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

The Spark Software Stack
**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

**Spark Streaming**

- Enables processing of live streams of data
- Sources
  - Logfiles generated by production webservers
  - Messages containing web service status updates
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

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 incorporate 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
### 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

### 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 event print out `sc` to see the the type.
- Once you have a SparkContext, you can use it to build RDDs.
  - And then run operations on the data...

### Executors
- Driver programs manage 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

### Lot of Spark’s API revolves around passing functions to its operators
```python
def hasPython(line):
    return "Python" in line

pythonLines = lines.filter(hasPython)
```

Also known as the **lambda or =>** syntax.
RESILIENT DISTRIBUTED DATASET [RDD]

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:
  - `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()`?
  - Would waste a lot of storage space, since we immediately filter out a lot of lines
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.

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.

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]