Apache Spark Lessons Learned

Madhu Siddalingaiah
madhu@madhu.com

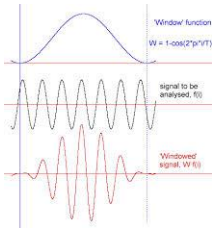
Contents

About the Presenter

1989

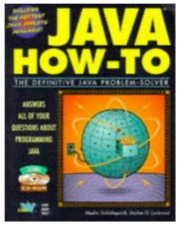
B. Sc. Physics
EE, CS, Math

Engineering age



1995

1996



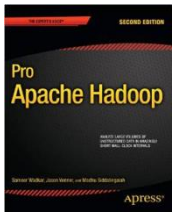
Internet age

2009

Big data age

2011

2013



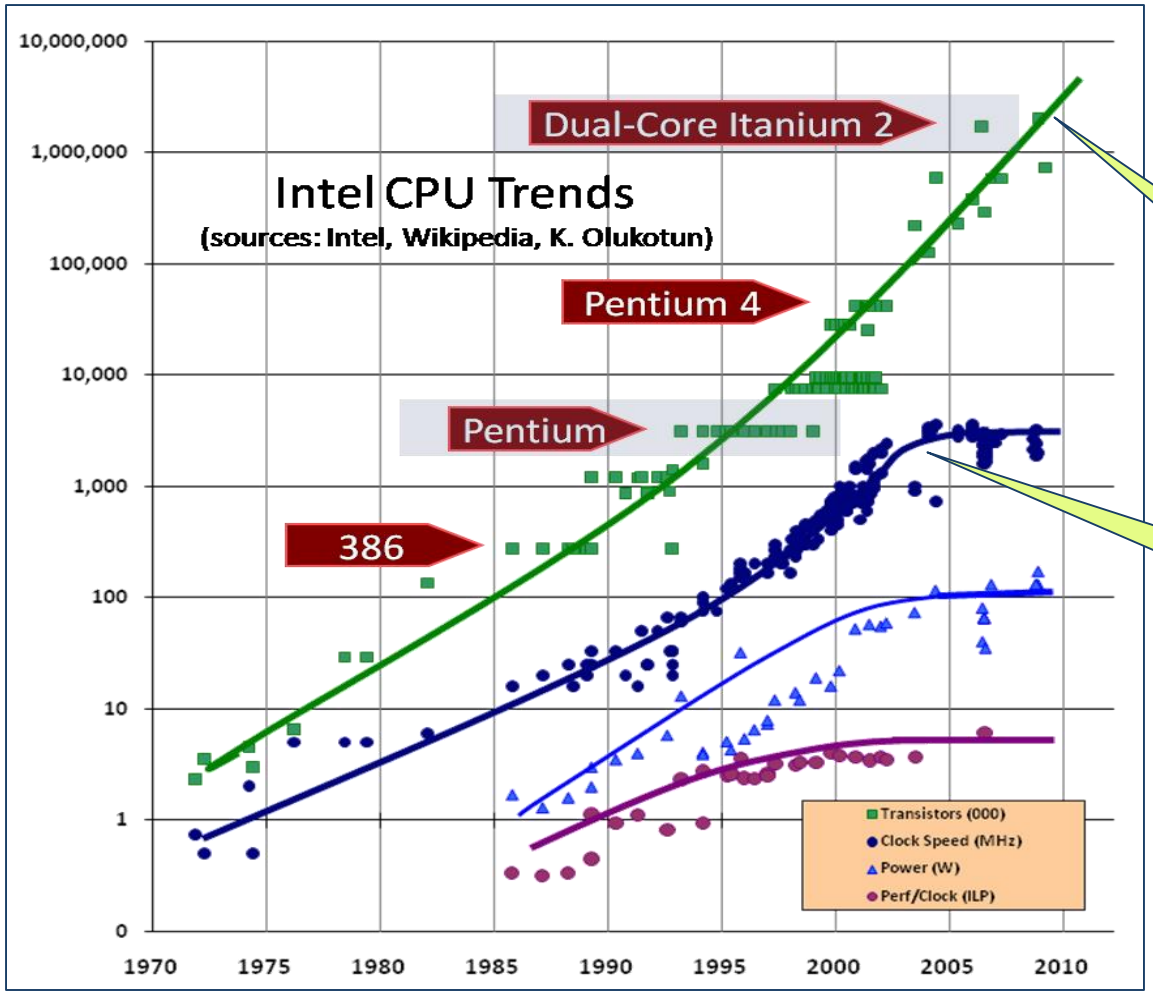
Sameer Wadkar
Madhu Siddalingaiah
Jason Venner

1 How Did We Get Here?

Big Data

- **Scale of data that cannot be efficiently processed with conventional technology**
  - Requires a new approach
- **Big data is characterized by**
  - Volume
    - Big today, even bigger in the future
  - Velocity
    - High data rate
  - Variety
    - Unstructured data from unconventional sources
    - Natural language, emails, tweets, images, video, audio

Moore's Law

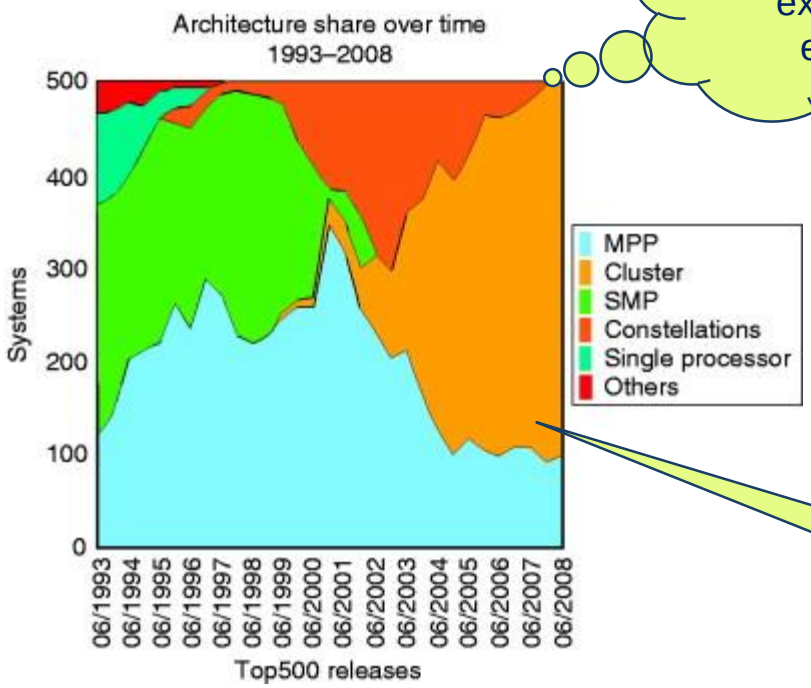


Scale out with multiple cores

Clock rates peaked in 2004

Scale Out

- **Many approaches developed over time**
  - Most have not survived
  - Currently clusters dominate



Multiple parallel architecture extinction events

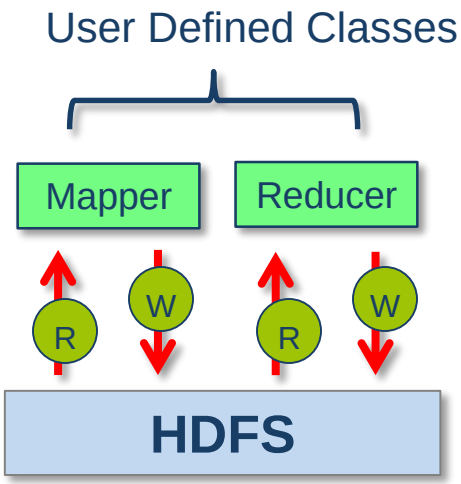
Clusters seem to be winning

Hadoop

- **Parallel processing based on MapReduce concept**
  - Influenced by proprietary Google technology
  - “MapReduce: simplified data processing on large clusters”*
    - Jeffrey Dean and Sanjay Ghemawat
- **Developed by Doug Cutting and Mike Cafarella in 2008**
  - Originally developed as part of Nutch open source engine
  - Working at Yahoo at the time
- **An open source platform for large-scale computing**
  - Parallel distributed data storage
    - Hadoop Distributed File System (HDFS)
  - Parallel distributed processing
    - MapReduce API

MapReduce Weakness 1: Big Bang Approach

- **Parallel programming is easy!**
  - Divide and conquer!
- **Hadoop is batch oriented**
  - Read, map, write, read, reduce, write
  - All that I/O is expensive
  - Iterative algorithms don't work well
  - Streaming is not in scope
- **Big data problems generally I/O bound**
  - Hard drive data rate: 100 MB/s/drive
  - Map only data rate: 50 MB/s/drive
  - Map reduce data rate: < 25 MB/s/drive
    - Shuffle/sort takes time



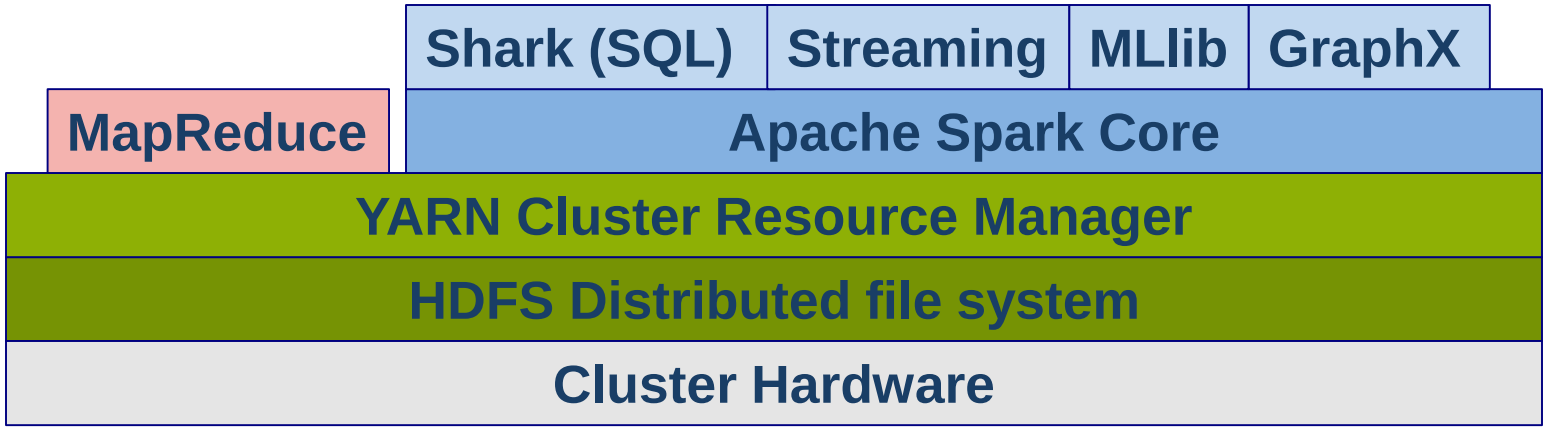
MapReduce Weakness 2: Source Code Quality

Software developers talk about "technical debt" in older code bases — a collection of decisions on design and implementation over many years, all made to solve problems of the moment and to address the urgent requirements of the past. ***MapReduce is, in that sense, deeply in arrears.***

— *Mike Olson, Chief Strategist/Co-founder Cloudera*

Apache Spark

- **General purpose large scale parallel processing**
  - More general than MapReduce, Storm, or Hama
  - Started as a research project at UC Berkeley AmpLab
- **High performance**
  - 10x to 100x performance increase over Hadoop
  - Your mileage will vary



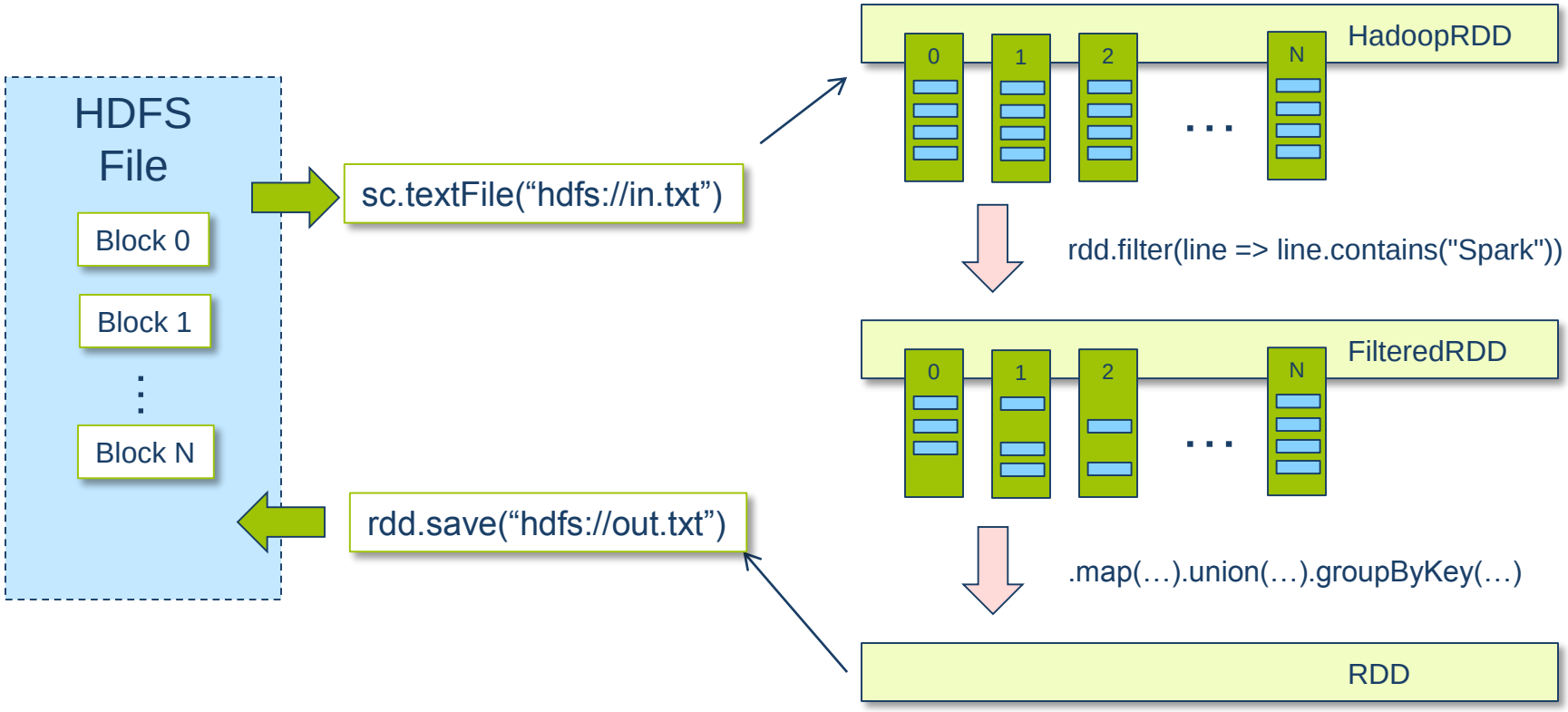
2 Spark Internals

Resilient Distributed Dataset (RDD)

- **Fundamental Spark abstraction**
  - Compare with Hadoop MapReduce
- **RDD is a read-only collection of distributed objects**
  - Can reside and operate in memory
    - Memory is much faster than disk
    - Can reuse intermediate results
  - Supports a rich set of *transformations* and *actions*
    - `map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, crossProduct, mapValues, sort, partitionBy, count, collect, reduce, lookup, save`
- **Written in Scala**
  - Scala is a statically typed functional/OO language for the JVM
  - Spark public API supports Java, Scala, Python

RDD Partitions

- **RDD contains a list of *partitions***
  - Partitions contain a list of JVM objects



Trivial Map Example

- **Multiplies a set of value by 2 (in parallel)**

Input data set with 4 partitions

```
object MapExample {
  def main(args: Array[String]): Unit = {
    // disable logging
    BasicConfigurator.configure(new NullAppender());
    val sc = new SparkContext("local", "Map Example")
    val input = sc.parallelize(List(1,2,3,4,5,6,7,8), 4)
    println("Number of partitions: " + input.partitions.length);
    val mapped = input.map(x => 2*x)
    mapped.collect.foreach(println)
  }
}
```

```
Number of partitions: 4
2
4
6
8
10
12
14
16
```

RDD Internals

spark/core/src/main/scala/org/apache/spark/rdd/RDD.scala

```
  /**
   * Return a new RDD by applying a function to all elements of this RDD.
   */
  def map[U: ClassTag](f: T => U): RDD[U] = new MappedRDD(this, sc.clean(f))
```

spark/core/src/main/scala/org/apache/spark/rdd/MappedRDD.scala

```
class MappedRDD[U: ClassTag, T: ClassTag](prev: RDD[T], f: T => U)
  extends RDD[U](prev) {

  override def getPartitions: Array[Partition] = firstParent[T].partitions

  override def compute(split: Partition, context: TaskContext) =
    firstParent[T].iterator(split, context).map(f)
}
```

Scheduling and Execution

- **Transformations are lazy**
  - Nothing really happens until an action is executed
  - Transformations produce a Directed Acyclic Graph (DAG)
- **DAGScheduler splits graph into stages of tasks**
  - Submit stages when ready
- **TaskScheduler launches tasks via cluster manager**
  - Retry failed tasks
- **Much more detail**
  - Where the magic happens
  - https://www.youtube.com/watch?v=49Hr5xZyTEA

Fault Tolerance

- **How to manage node failure?**
  - MapReduce tasks write data to disk after each stage (checkpointing)
    - If a task fails, another task is launched on another node
    - Simple, but inefficient and slow
- **Spark RDDs maintain a *lineage***
  - Like a transaction log of how an RDD was derived
    - Automatically reconstructed if needed
- **MapReduce is *pessimistic***
  - Expects nodes to fail
- **Spark is *optimistic***
  - Expects nodes to succeed

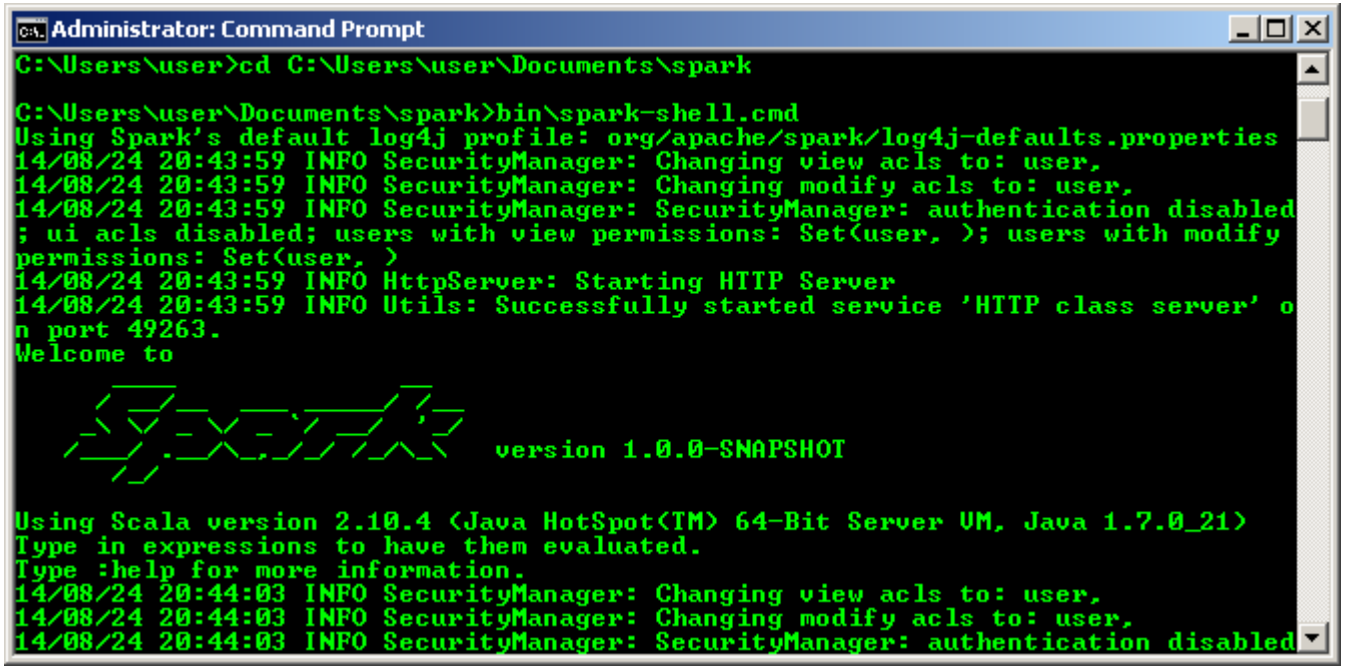
3 Development Process

Getting Started

- **Download binary release**
  - Easiest way
  - Add `spark/spark-1.0.2-bin-hadoop2/lib/spark-assembly-1.0.2-hadoop2.2.0.jar` to your CLASSPATH
  - Works in Eclipse
- **Maven**
  - groupId: org.apache.spark
  - artifactId: spark-core_2.10
  - version: 1.0.2

Spark Shell

- **Interactively access Spark API**
  - Great way to learn Spark!
- **`spark/bin/spark-shell` or `spark\bin\spark-shell.cmd`**



Build From Source

- **Useful if you want bleeding edge features or bug fixes**
  - Else wait until the next quarter for a release
- **Easy to build on any platform**
  - With sbt (Simple Build Tool) or maven
- **SBT steps**
  - Download/install sbt (Windows only)
  - Build requires a lot of memory, adjust `sbt/conf/sbtconfig.txt`:
    - `-Xmx1024M, -XX:MaxPermSize=256m, -XX:ReservedCodeCacheSize=128m`
  - `git clone https://github.com/apache/spark`
  - `cd spark`
  - `sbt assembly` (Windows)
  - `sbt/sbt assembly` (non-Windows)

Language Choices

- **Spark supports Scala, Java, and Python**
  - To varying degrees

	Advantages	Disadvantages
Scala	• Full API support • High performance • Type inference (concise) • Interactive (Spark shell)	• Not as well known • Language still evolving
Java	• Good API support • Interoperable with Scala • High performance • Well established	• More verbose syntax • Lack of type inference can be painful!
Python	• Interactive • Concise syntax • Well established • Dynamically typed	• Lower compute performance • API support still evolving • Dynamically typed

Running at Scale

- **Spark runs as a YARN application**
  - Works with Hadoop
  - Interoperates with HDFS, Hive external tables
- **Easy to deploy as Amazon EC2 cluster**
  - Run spark/ec2/spark-ec2 script
    - Does not work on Windows ☹
  - Run on demand or spot instances
  - Specify node type, number of nodes
  - Quick, low cost proof of concept

4 Summary

Will Spark Work for You?

- **If MapReduce works for you…**
  - Very likely Spark will work better
  - Particularly for iterative algorithms
    - Machine learning
- **Give it try!**
  - Risk is low
  - Spin up an Amazon EC2 cluster
    - Run Spark EC2 setup scripts
    - Use low cost spot instances
- **It's not a silver bullet**
  - Nothing is
  - Expect to see more parallel computing models in the future

Spark Business Value

- **Faste*r***
  - Uses memory and disk more efficiently
  - In the cloud, *time is money*
- **Open source**
  - No licensing fees
  - Approachable, high quality source code
- **Popular**
  - Over 100 active contributors
- **Supported by many vendors**
  - Cloudera, Hortonworks, MapR

madhu@madhu.com

LinkedIn
www.linkedin.com/in/msiddalingaiah