Why Spark?

Why Spark?

- In this chapter, you will learn
  - The problems with traditional large-scale computing systems
  - How Spark addresses those problems
  - Typical big data questions Spark can be used to answer

Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

The Cost Of Computing

- Perhaps you have heard of Moore's Law
  - Processor performance increases approximately twofold every 2 years
  - With prices falling in the long term
- But have you heard of Kryder's Law?
  - If not, Google it
  - The density of hard drives increases by a factor of 1,000 every 10.5 years (doubling every 13 months)
  - With prices falling in the long term
- The implication of both laws is that it is becoming increasingly cost effective to store and process data large quantities of data
  - As an alternative to discarding or summarizing that data

Big Data

- Enterprises often have to work with unwieldy data
  - Large amounts of data
  - Data in awkward formats
  - Often streaming in faster than can be consumed
- Known as the 3 V's
  - What do they stand for?
    - Velocity
    - Variety
    - Volume
  - What's important about them?
    - Scalability

Velocity

- We are generating data faster than ever
- Then:
  - The velocity with which we could process data was a function of the speed and number of data input operators we had on staff
- Now:
  - Automated processes and devices are generating data from all kinds of sources, millions of times faster than could human data input operators
    - Mobile phones pinging cell towers and generate GPS information
    - Utility meters monitoring activity in your home to regulate resources
    - Traffic signals monitoring throughput to more effectively utilize the roadways

Variety

- We are producing a wide variety of data
  - Social network connections
  - Server and application log files
  - Motion sensors
  - Electronic medical records
  - Images, audio, and video
  - RFID and wireless sensor network events
  - Product ratings on shopping and review Web sites
- Not all of these data representations map cleanly to the relational mode

Volume

- Not only is the speed with which we process data increasing, more importantly is the sheer volume of data being generated by all of these devices
  - Every day Google processes about 24 petabytes of data
  - Every minute Facebook stores 110 million comments
  - Every second banks process more than 10,000 credit card transactions
- There is simply too much data to store cost effectively in a traditional RDBMS
- http://lintool.github.io/my-data-is-bigger-than-your-data/

Traditional Large-Scale Computation

- What has been the traditional response to problems posed by the 3 V's?
    - Buying bigger, more expensive computers
        - But even that does not scale sufficiently, and tends to be prohibitively expensive
    - Data summarization, keeping the high value data and discarding the details
        - Which itself takes time, money, and risks losing valuable details
            - You may not learn much from a single tweet, but you can learn quite a lot from a stream of 1,000,000 tweets over time

What's Wrong With Big Computers?

- They tend to be
  - Disproportionately expensive to their scale
  - Not scalable enough for the amounts of data we are working with today

What's Wrong With Data Summarization?

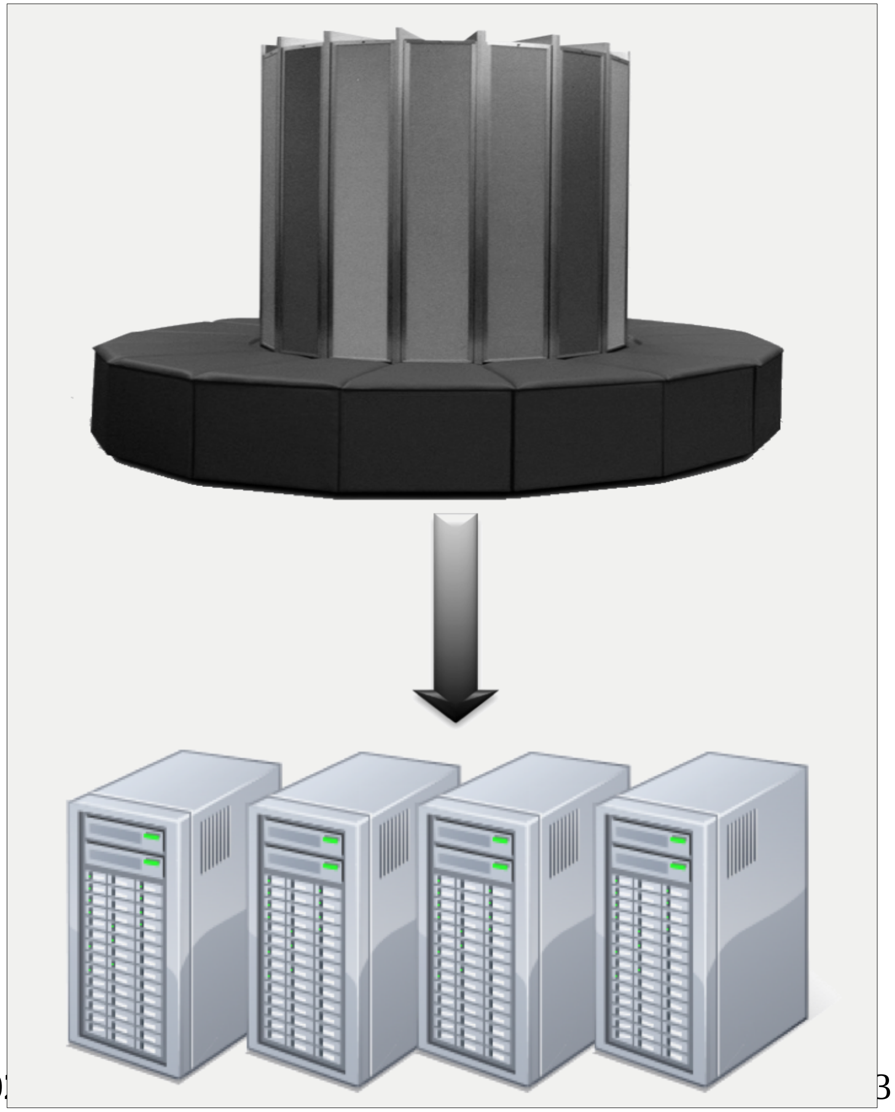
- There could be real nuggets in the lost detail
- More data == Deeper understanding
  - “There’s no data like more data”
    - http://citeseerx.ist.psu.edu/showciting?cid=3965783
    - Which is likely to yield a deeper understanding of your data; an analysis of a dataset consisting of 10 elements or of a dataset consisting of 1000 elements?
      - Coin flip analogy
  - “It’s not who has the best algorithms that wins. It’s who has the most data”
    - https://github.com/chao787/RNote/blob/master/Blog/machine_learning/Large_Scale_ML.org
- We need to preserve this detail, but in ways that are cost effective

We Need A Better Approach

- We’re generating too much data to process with traditional tools
- We need a new approach to the problems posed by the 3 V's, one that addresses 2 key concerns
  - How can we store large amounts of data, reliably, at a reasonable cost?
  - How can we analyze all the data we have stored, in a reasonable amount of time?

Distributed Systems

- The better solution: More computers
  - Distributed systems
    - Use multiple machines for a single job
- Grace Hopper:
  - "In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, we didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers"

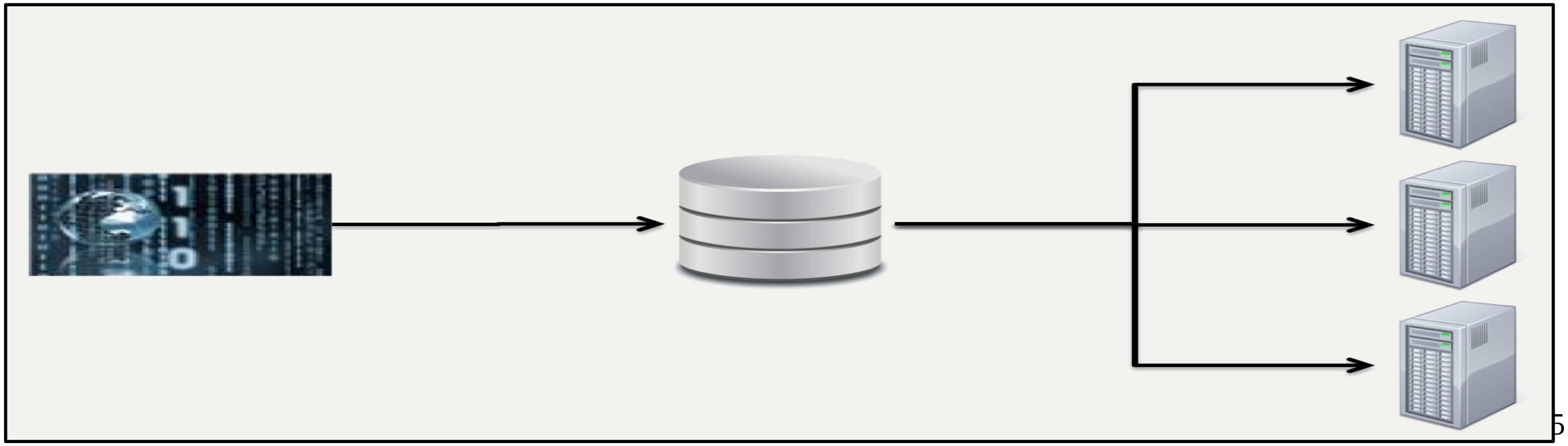


Distributed Systems: Challenges

- Unfortunately, distribution can complicate the handling of big data
  - Programming for traditional distributed systems is complex
  - Data exchange takes time (network/disk latency)
  - Data exchange requires synchronization, which introduce bottlenecks
  - Bandwidth is limited
  - It distracts from an emphasis on the problem domain
  - It is often based on misconceptions
    - Fallacies of Distributed Computing
  - It is difficult to deal with partial failures of the system
    - Ken Arnold, CORBA (Common Object Request Broker Architecture) designer:
      - “Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure”

Distributed Systems: Data Bottleneck (1 Of 2)

- Consider a typical scenario with data stored in a central location, on a SAN (Storage Area Network)
- At compute time, data is copied to the compute nodes
- This is fine for relatively limited amounts of data, but not for the voluminous amounts of data we generate today



Distributed Systems: Data Bottleneck (2 Of 2)

- Getting the data from the storage device to the processors has become the bottleneck, not the process power itself
- Do the math
  - Typical disk data transfer rate: 75MB/sec
  - Time taken to transfer 100GB of data to the processor: Approximately 22 minutes!
- But modern systems today typically work with much more data than this
  - Terabytes, even Petabytes
- A better idea would be to send the processor to the data

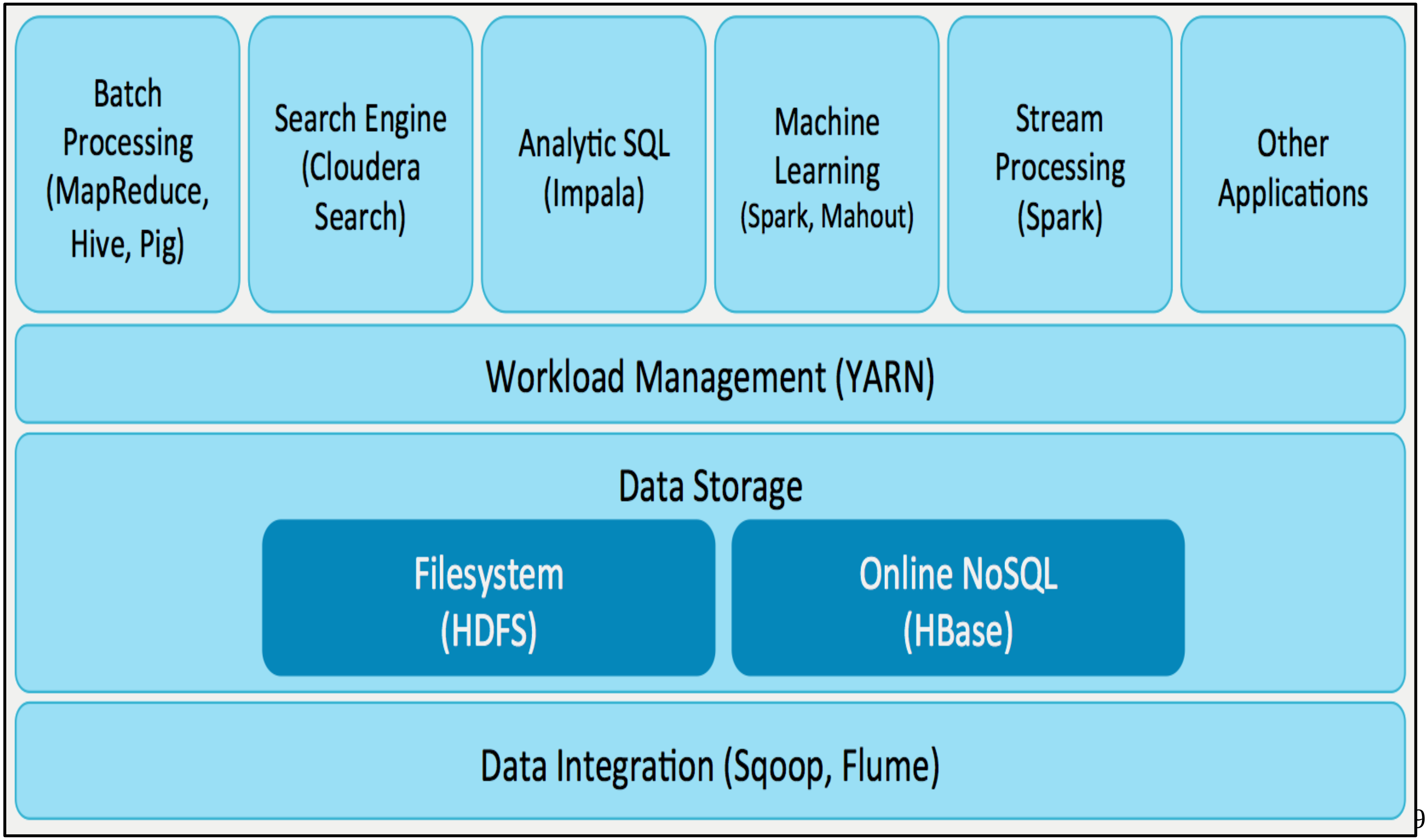
Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

What Is Apache Hadoop? (1 Of 2)

- An open-source framework for the storage, processing and analysis of large data sets
- Key Features:
  - Scalable
    - Designed to scale up from a single server to clusters comprised of thousands of nodes, each offering local computation and storage
  - High-Performance
    - Through the parallelism of distributed computing platforms
  - Fault-Tolerant
    - Designed to detect and handle failures transparently
  - Cost-Effective
    - Rather than rely on expensive hardware to deliver high-availability, Hadoop runs on standard server technology
- https://hadoop.apache.org/
-

What Is Apache Hadoop? (2 Of 2)


Batch Processing (MapReduce, Hive, Pig)
Search Engine (Cloudera Search)
Analytic SQL (Impala)
Machine Learning (Spark, Mahout)
Stream Processing (Spark)
Other Applications
Workload Management (YARN)
Data Storage
Filesystem (HDFS)
Online NoSQL (HBase)
Data Integration (Sqoop, Flume)

The Origins Of Hadoop

- Hadoop started with research that originated at Google in 2003
  - Google's objective was to index the entire World Wide Web
  - But the web was growing too quickly
  - Google had reached the limits of scalability of RDBMS technology in terms of its ability to store, sort, organize and analyze large quantities of data
  - This research led to a new approach toward distributing and analyzing large quantities of data
    - http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf
    - http://research.google.com/archive/mapreduce-osdi04.pdf

What Was This New Approach Proposed By Google?

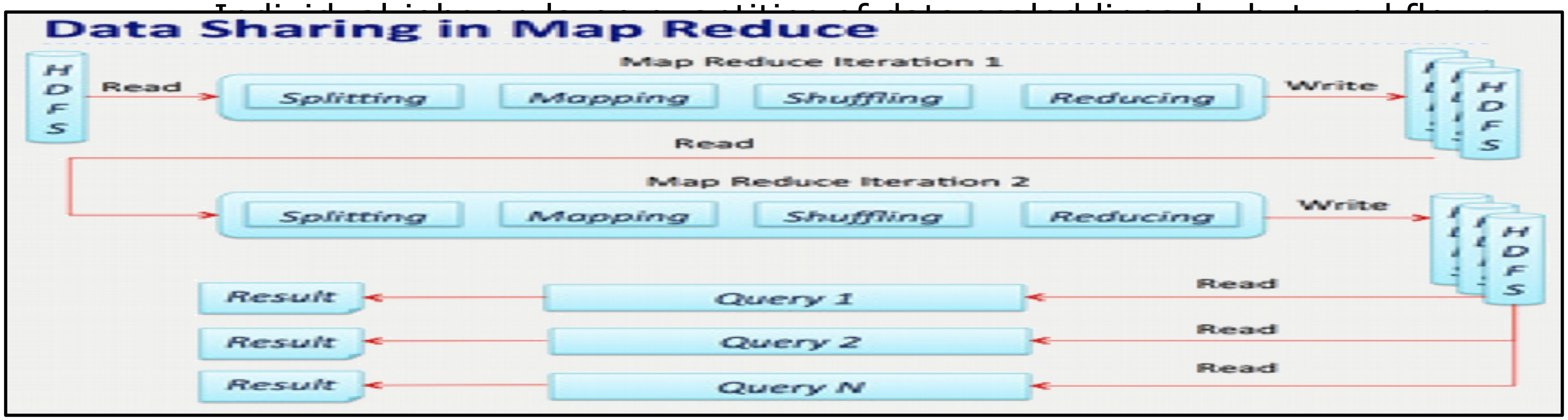
- Two big ideas
  - Google File System (GFS)
    - A distributed file system, central elements of which were
      - Spread the data over a cluster (of standard hardware)
      - Compute where the data lives, not vice versa
        - Known as data locality
      - Replicate the data for increased reliability and availability
      - Shield the developer from the complexity of the underlying infrastructure
        - Known as Fault Tolerance
  - MapReduce
    - A programming model for decomposing a problem into discrete units that can be processed in parallel across multiple computers
      - In ways that isolate the developer from the complexities of working with distributed data

Hadoop Was The First Open Source Iteration Of This New Approach

- Designed for the architectures prevalent at the time (limited RAM, lots of disk storage), Hadoop provided a platform that was
  - Fault tolerant
  - Economical
  - Scalable
- Unfortunately, while state of the art at the time, and a major improvement over its predecessors, the approach taken by Hadoop had significant limits

Limits Of Hadoop (1 Of 2)

- Latency
  - Hadoop jobs generate lots of intermediate processing, involving heavy disk and network utilization (think bottleneck), which degrades performance, particularly when applied to complex problems involving multiple tasks
    - [illegible]



Limits Of Hadoop (2 Of 2)

- A complex programming model
  - Initially, you had to be a programmer to use Hadoop, the consequences of which were
    - Non-programmers (e.g. Data Analysts) were locked out of the technology
    - Application development was expensive
- In time, a set of tools were introduced, known collectively as the Hadoop ecosystem, designed to deliver the benefits of the technology to a wider audience
  - Flume - Streaming data into Hadoop
  - Hive or Impala - Querying data in Hadoop
  - Pig - Manipulating data in Hadoop
  - Giraph - Graph analysis
  - Mahout - Machine learning
  - Sqoop - Exchanging data with RDBMS systems
  - HBase - NoSQL storage
  - Oozie - Workflow managers
- Unfortunately, this meant that Hadoop developers had to familiarize themselves with a multiplicity of tools, each with different objectives and interfaces to exploit the technology to its fullest extent

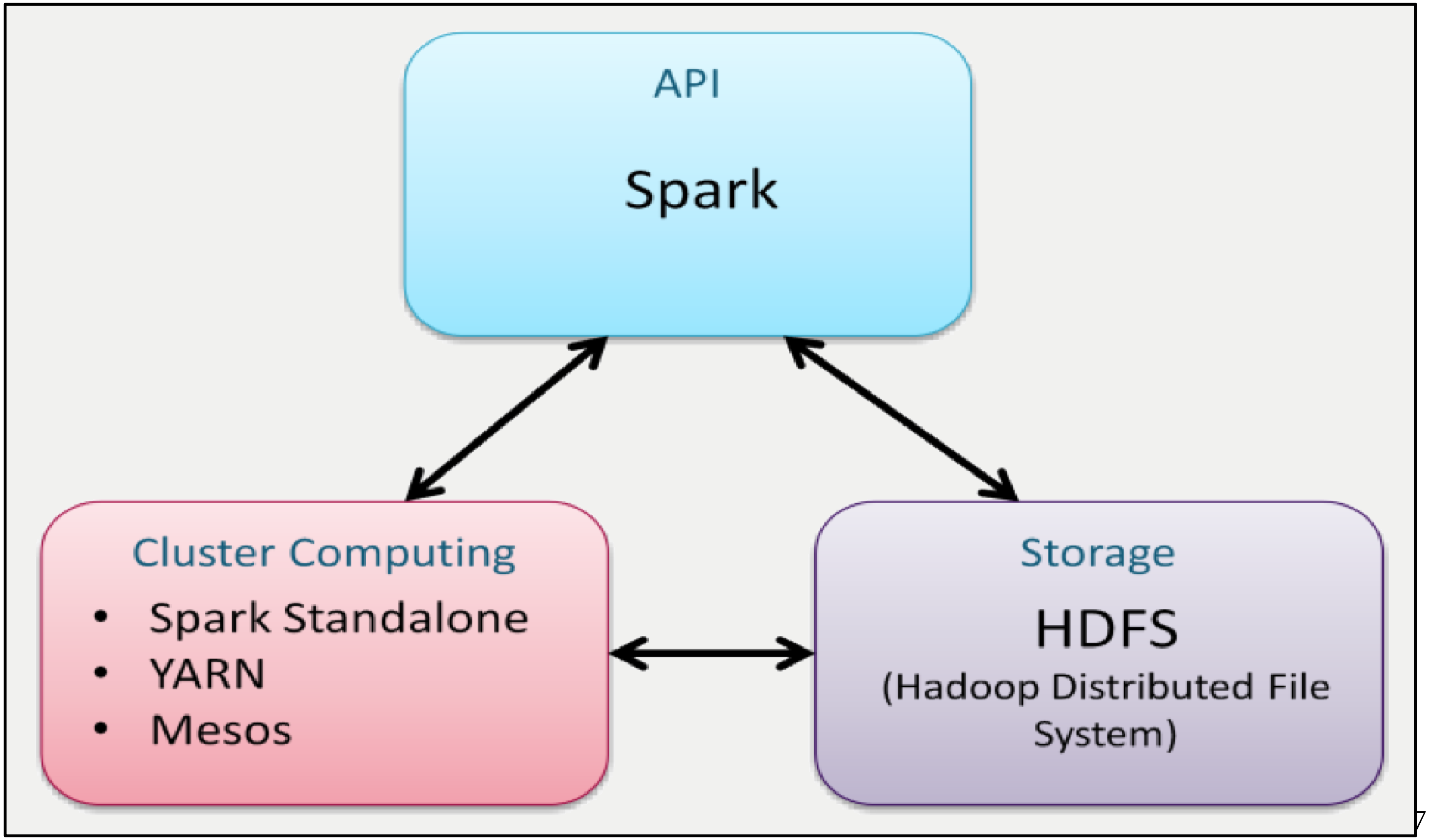
Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

What Is Apache Spark?

- Apache Spark is a fast, general engine for large-scale data processing on a cluster
- Originally developed at AMPLab at UC Berkeley
  - https://amplab.cs.berkeley.edu
  - Started as a research project in 2009
  - The creators later founded Databricks to commercialize Spark
    - Cloudera partners with Databricks to provide enterprise-level support for Spark
- Open source Apache project
  - Committers from Yahoo, Databricks, UC Berkeley, Intel, Groupon, Cloudera
  - One of the most active and fastest growing Apache projects

Key Parts Of The Spark Framework (1 Of 4)



Key Parts Of The Spark Framework (2 Of 4)

- Spark API
  - High level programming API
    - Programmers focus on the problem domain
    - Spark handles the plumbing/infrastructure
      - How to divide tasks into steps that can be run in parallel
      - Task coordination
      - Copying data where it is needed
      - Network communication
      - Temporal dependencies
    - Resulting in greater developer productivity
  - Supporting programs written in
    - Scala
      - Most of Spark was written in Scala
    - Python
    - Java

Key Parts Of The Spark Framework (3 Of 4)

- Cluster Computing
  - Spark programs run across a cluster of networked computers using a cluster resource management framework, providing
    - Workload distribution
    - Resource sharing
    - Lifecycle management
  - Master/Slave architecture with slaves managed by the master
    - Shared nothing architecture eliminates most temporal dependencies (bottlenecks)
  - Currently supported resource managers:
    - Spark Standalone
      - For clusters running Spark exclusively
    - YARN (Yes Another Resource Negotiator)
    - Mesos
      - Spark was originally implemented on Mesos

Key Parts Of The Spark Framework (4 Of 4)

- Storage
  - Spark data is stored in a distributed file system
    - Hadoop's Distributed File System (HDFS) by convention, though others are supported
      - e.g. Amazon S3
  - Data is distributed/replicated when it is stored, providing
    - Availability/Fault Tolerance
      - Because there are multiple copies of the data, should one copy not be available
      - Eliminates Single Point Of Failure
    - Reliability
      - Because there are multiple copies of the data, should one copy of the data be corrupt
    - Scalability
      - Because the data is distributed across multiple storage devices
    - Performance/Efficiency
      - Because sending the processor to the data improves performance, and having multiple copies of the data increases the likelihood this will happen
      - Because the data is distributed across multiple nodes, facilitating parallel processing

How Does Spark Provide Scalability?

- Spark is a coordinated, distributed system of standard computers
  - Some are masters, supervising activity and resources
  - Most are slaves storing and processing data
- Just add nodes to the cluster to increase scalability
  - Facebook and Yahoo both run clusters in excess of 4400 nodes
  - The latest version of YARN offers linear scalability to approximately 6600 nodes, each with multiple cores
  - The latest version of Mesos offers linear scalability to tens of thousands of nodes
- This simplifies capacity planning greatly

Node Failure Is Inevitable

- Assumptions:
  - 1000 computers
  - Mean Time Between Failure: 3 years
- Consequences:
  - 1 disk failure per day
- Case Study: Google
  - At Google there are so many computers and node failure so commonplace that Google uses velcro to mount their servers into racks, rather than bolting them in, so as to make them easier to replace

What Kinds Of Guarantees Should A Fault Tolerant System Offer?

- When a task/node fails ...
  - System performance should degrade gracefully
  - The workload should be assumed automatically by other, still functioning components of the system
  - No data should be lost
  - The outcome of the job in which it was working should not be affected
  - The failure should be detected automatically and the task reassigned automatically to another node of the system
  - There should not be an impact on other tasks/nodes running as part of the same job
  - When the task/node restarts, it should be automatically added back to the system and assigned new responsibilities
- All without the need for human intervention

How Does Spark Provide Reliability/Availability/Fault Tolerance?

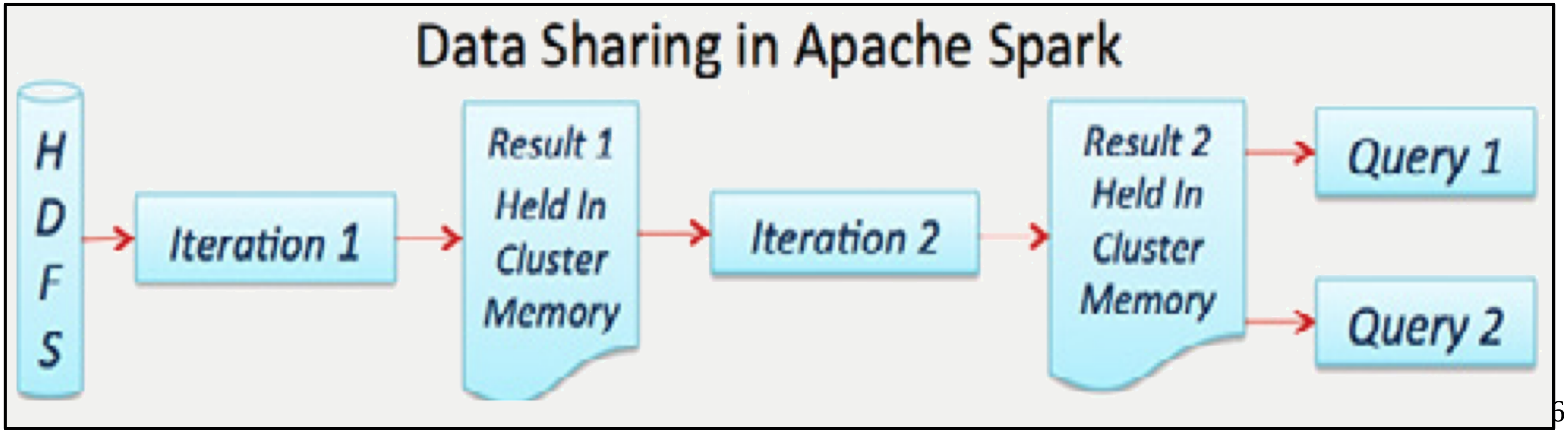
- In a word, redundancy
- Files stored in HDFS are replicated
  - If a node goes down, there are copies of the data available elsewhere in the cluster
- Processing in Spark is replicated
  - Different machines process different parts of the data in parallel
  - If a node goes down, there are other nodes in the cluster available to handle the processing of the failed node

How Does Spark Provide Performance?

- Data locality
  - Sending the processor to the data rather than the data to the processor
- Parallelism
  - Harnessing the horsepower of many CPUs at once
- Shared nothing architecture
  - Isolating components from one another eliminates bottlenecks
- It also turns out that the redundancy that provides Fault Tolerance also provides opportunities to improve performance
  - Running a task on the replica that has the most resources available
- Aggressive use of memory in lieu of disk storage

How Does Spark Improve On Hadoop? (1 Of 3)

- Improved performance
  - Less reliance on disk technology (as RAM has become cheaper and more plentiful)
  - Aggressive use of memory and optimization through caching
  - Resulting in
    - Dramatic improvements in performance/responsiveness
    - Opening the door to its application in interactive contexts


Data Sharing in Apache Spark
HDFS
Iteration 1
Result 1 Held In Cluster Memory
Iteration 2
Result 2 Held In Cluster Memory
Query 1
Query 2

How Does Spark Improve On Hadoop? (2 Of 3)

- A simpler programming model
    - Often, what you can achieve in 40 lines of MapReduce code can be implemented in 5 lines of Spark code

How Does Spark Improve On Hadoop? (3 Of 3)

- A more consistent, “unified” programming model
- Benefits of an integrated stack over a loosely coupled ecosystem
  - Upper level components benefit from improvements at the lower layers
    - e.g. When Spark’s core engine adds an optimization, SQL and machine learning libraries automatically speed up as well
  - Costs associated with running an integrated stack are minimized, because instead of running 5–10 independent software systems, an organization needs to run only one system
    - Including deployment, maintenance, testing, support, and others
    - This also means that each time a new component is added to the Spark stack, every organization that uses Spark will immediately be able to try this new component
  - Improves the ability of an organization to build applications that seamlessly combine different processing models
    - One application could incorporate machine learning, ad-hoc query analytics, ETL & ELT data manipulation for batch or real-time processing
    - All the while, the IT team only has to maintain a single system

Who Uses Spark?

- Yahoo!
  - Personalization and ad analytics (having evolved from MapReduce)
- Conviva
  - Real-time video stream optimization, providing real-time (rather than batch oriented) analytics about the delivery of streaming content to content providers
- Technicolor
  - Real-time analytics for telco clients
- Ooyala
  - Cross-device personalized video experience
  - http://engineering.ooyala.com/blog/fast-spark-queries-memory-datasets
- Intel
  - http://www.intel.com/content/www/us/en/healthcare-it/big-data-real-time-healthcare-analytics-whitepaper.html
- Plus
  - Groupon, TrendMicro, Autodesk, Nokia, Shopify, Clearstory, ...
- See Bibliography.odp
- https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark

Common Applications Of Spark

- ETL
  - “How do we refactor the data to look right for subsequent analysis?”
- Text mining
  - “What percentage of our customer emails include the word 'refund' or 'apology'”
  - “From 2012 to 2014, the frequency of the word “apology” in response to traveler complaints increased by seven times”
    - http://www.nytimes.com/2015/10/20/business/lost-bags-at-140-characters-and-airlines-respond.html?hpw&rref=technology&action=click&pgtype=Homepage&module=well-region&region=bottom-well&WT.nav=bottom-well
- Index building
  - “How can we build a keyword index for a web site?”
- Graph creation and analysis
  - “Who is the opinion leader in a population?”
- Pattern recognition
  - “Is that Joel robbing that bank in this photograph?”
- Recommendation
  - “Which products will this customer enjoy?”
- Prediction models
  - “How can we prevent a catastrophic service outage rather than merely reacting to it?”
- Classification
  - “How can we tell which emails are spam and which are not?”
- Risk assessment
  - “How likely is this borrower to pay back a loan?”

What Do These Problems Share In Common?

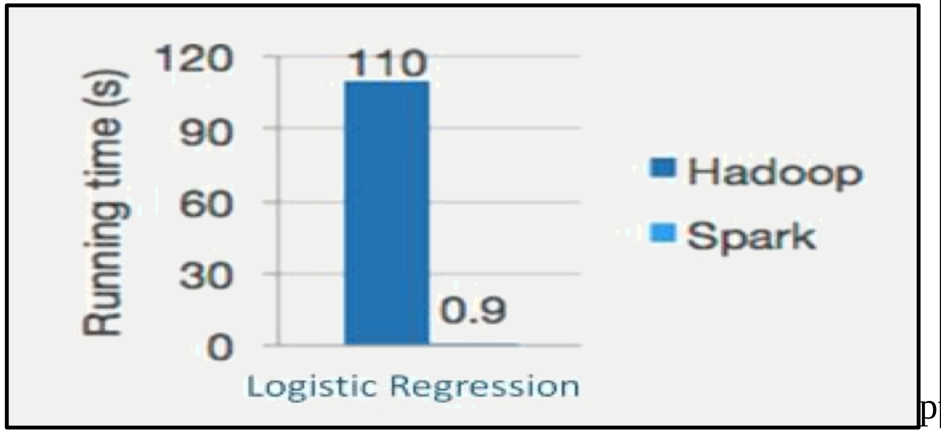
- The nature of their data
- 3 V's
  - Volume
  - Velocity
  - Variety

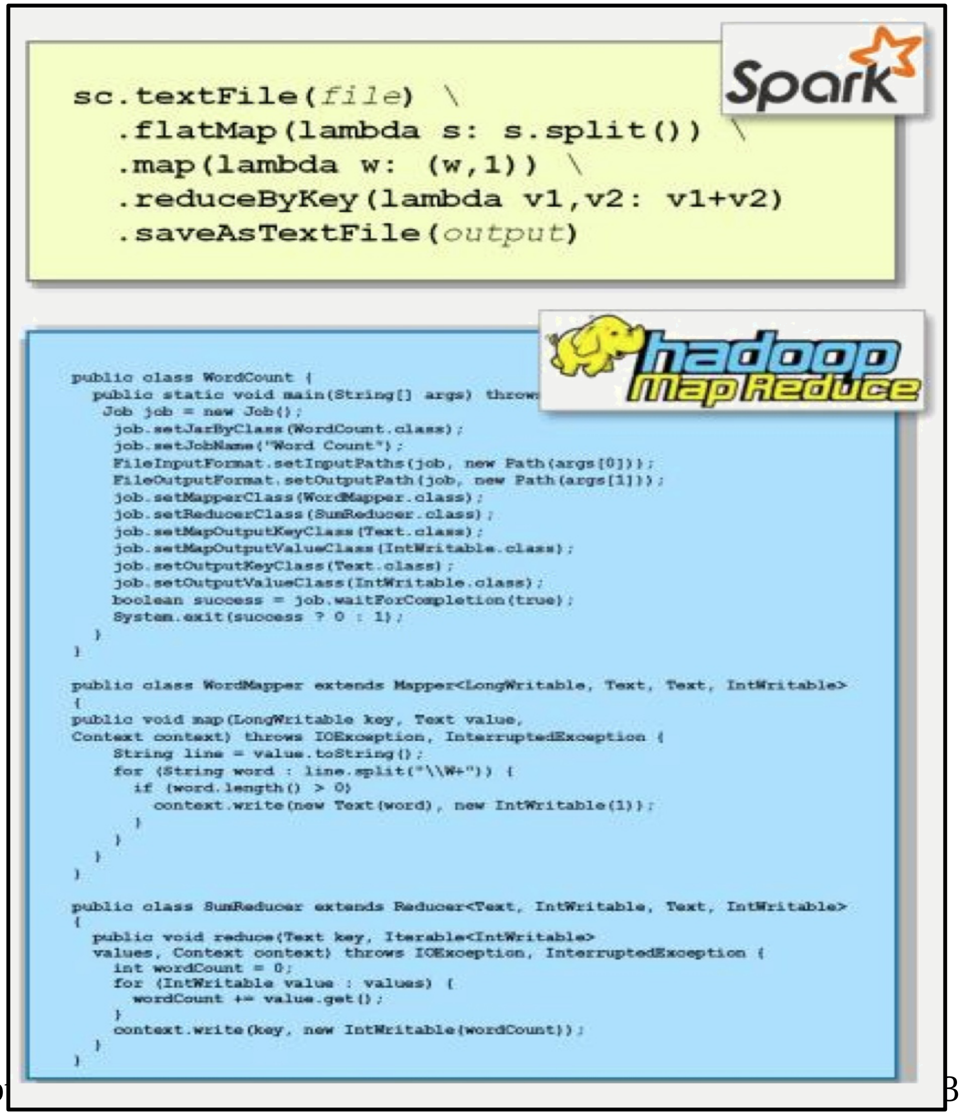
Benefits Of Using Spark

- The ability to analyze data in ways that, until recently were considered impractical or impossible
- On standard hardware
- More quickly that would be the case without Spark
- Regardless of data format
- With near-linear scalability
- Allowing you to ask even bigger questions
- And if you call in the next 30 minutes ...
-

Spark Vs. MapReduce

- Spark takes the concepts of MapReduce to the next level, providing
  - A higher level API
    - Resulting in greater productivity
  - A consistent stack (in contrast to an inconsistent ecosystem)
    - Resulting in greater productivity
  - Low latency
    - Resulting in near real-time processing
  - Performance
    - Resulting in 100-fold performance gains for in-memory data storage
    - Resulting in 10-fold performance gains for disk based data storage
- http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html





```
sc.textFile(file) \
  .flatMap(lambda s: s.split()) \
  .map(lambda w: (w,1)) \
  .reduceByKey(lambda v1,v2: v1+v2)
  .saveAsTextFile(output)
```

```
public class WordCount {
  public static void main(String[] args) throw
   Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}

public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
public void map(LongWritable key, Text value,
Context context) throws IOException, InterruptedException {
    String line = value.toString();
    for (String word : line.split("\\W+")) {
      if (word.length() > 0)
        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}

public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
  public void reduce(Text key, Iterable<IntWritable>
  values, Context context) throws IOException, InterruptedException {
    int wordCount = 0;
    for (IntWritable value : values) {
      wordCount += value.get();
    }
    context.write(key, new IntWritable(wordCount));
  }
}
```

Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

Summary

- Traditional large-scale computing involved complex processing on small amounts of data
- Exponential growth in data drove development of distributed computing
- Distributed computing is difficult!
- Spark addresses big data distributed computing challenges
  - Bring the computation to the data
  - Provide fault tolerance
  - Provide scalability
  - Hide the ‘plumbing’ so developers can focus on the data
  - Cache data in memory

Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

Review

- What are the 3 V's?

Review Answer

- What are the 3 V's?
  - Volume
    - We are generating more data than ever
  - Variety
    - In varied formats not well understood by traditional technologies
  - Velocity
    - Faster than we can manage with traditional technologies

Review

- List 3 challenges of distributed computing

Review Answer

- List 3 challenges of distributed computing
  - Programming for traditional distributed systems is complex
  - Data exchange takes time (network/disk latency)
  - Data exchange requires synchronization, which introduce bottlenecks
  - Bandwidth is limited
  - It distracts from an emphasis on the problem domain
  - It is often based on misconceptions
    - Fallacies of Distributed Computing
  - It is difficult to deal with partial failures of the system
    - Ken Arnold, CORBA (Common Object Request Broker Architecture) designer:
      - "Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure"

Review

- What 2 big ideas did Hadoop introduce?

Review Answer

- What 2 big ideas did Hadoop introduce?
    - Distribute data as when it is stored
    - Run the computation where the data resides

Review

- What kinds of guarantees should a fault tolerant system offer?

Review Answer

- What kinds of guarantees should a fault tolerant system offer?
  - When a task/node fails ...
    - System performance should degrade gracefully
    - The workload should be assumed automatically by other, still functioning components of the system
    - No data should be lost
    - The outcome of the job in which it was working should not be affected
    - The failure should be detected automatically and the task reassigned automatically to another node of the system
    - There should not be an impact on other tasks/nodes running as part of the same job
    - When the task/node restarts, it should be automatically added back to the system and assigned new responsibilities
  - All without the need for human intervention

Review

- What are the 3 big advantages Spark offers over Hadoop?

Review Answer

- What are the 3 big advantages Spark offers over Hadoop?
  - Improved performance
  - A simpler programming model
  - A more consistent, “unified” programming model

Chapter Topics

- Problems With Traditional Large-Scale Systems
- Introducing Hadoop
- Introducing Spark
- Summary
- Review
- References

References

- The following offer more information on topics discussed in this chapter
  - Amplab UC Berkeley
    - https://amplab.cs.berkeley.edu/
  - Databricks
    - http://databricks.com
  - The Apache Spark Stack
    - http://www.spark-stack.org/
  - Spark Survey
    - https://gigaom.com/2015/01/27/a-few-interesting-numbers-about-apache-spark/
  - For information on the newest Spark release (1.5)
    - http://www.oreilly.com/pub/e/3285