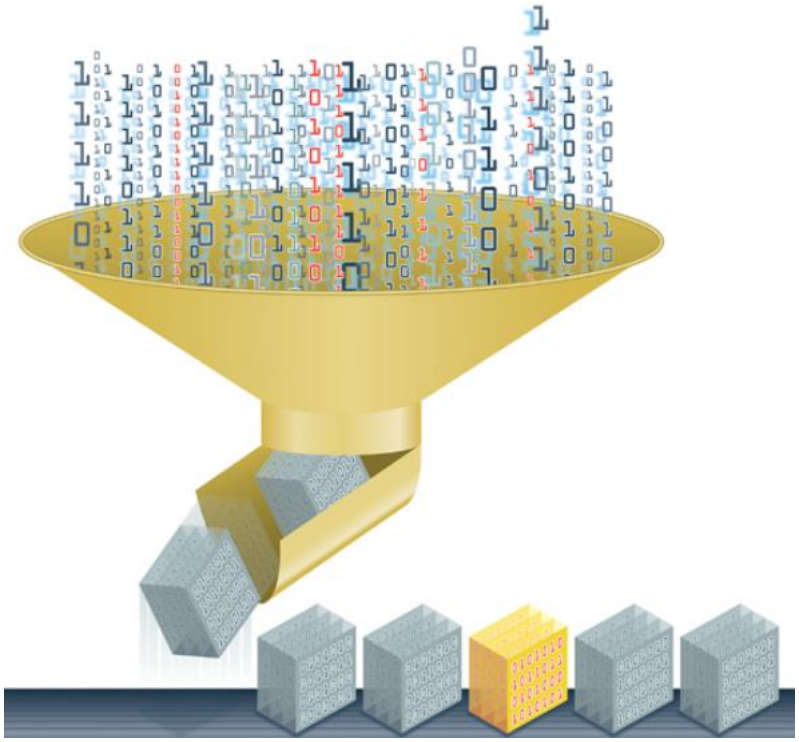
Big Data Fundamentals for Data Scientists

Presented by: Derek Kane

Overview of Topics

- Preface
- The Age of Big Data
- Knowing the Big Data Ecosystem
  - Hadoop / HDFS / MapReduce
  - HBase (NoSQL)
  - Hive / Pig / Mahout
  - ZooKeeper / Sqoop / Oozie
  - Kafka / Storm
- Utilizing Big Data with R
- Practical Application Example
  - Taxi Cab Analytics in R
    - Hadoop / MapReduce
    - Transferring data to HDFS
    - Querying the Hive

Preface



- The “Big Data” space is truly the wild west of analytics and one of the pinnacle achievements of the modern information era.
- As such, the landscape is rapidly evolving and software vendors are each offering their own implementations of the core technologies that drive “Big Data”.
- There are a lot of different technologies and options the data scientists have to choose from and can be quite daunting when deciding on a platform.
- Furthermore, business leaders are relying on our expertise when deciding the “Big Data ecosystem” to setup for the end to end analytics spectrum.

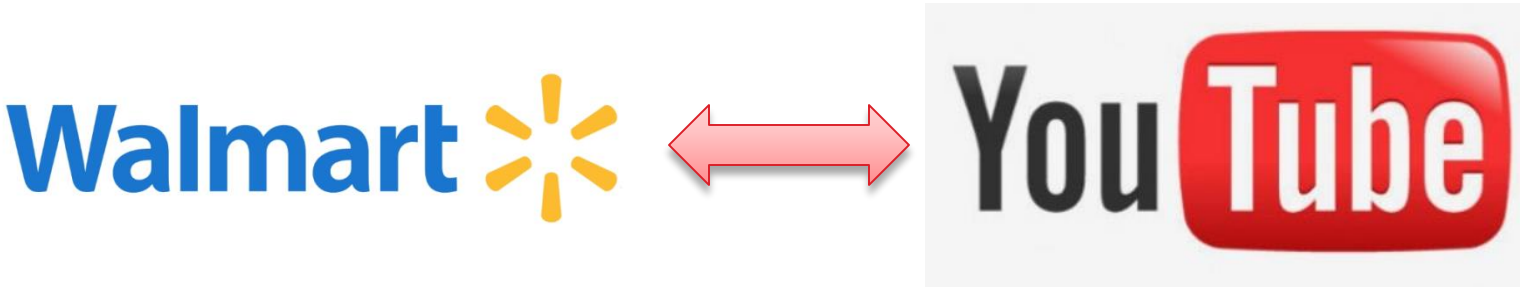
Preface

- The intention of this lecture is to provide awareness of the underlying techniques of working with Big Data technologies from a data scientists perspective exploring Big Data machine learning with R.
- With a functional understanding of the mechanics of Big Data techniques, Data Science professionals will be poised to be able to deploy the analytical techniques regardless of the hardware/software technologies deployed.
- We will spend some time going over some of the notable pioneering technologies from the Apache Software Foundation and their IT underpinnings.
- However, a proper evaluation needs to be made with the CIO and aligned to the organizations analytical goals when selecting the appropriate software/hardware deployment.

Key Point: Not all Big Data projects require the complete ecosystem!!!

Preface

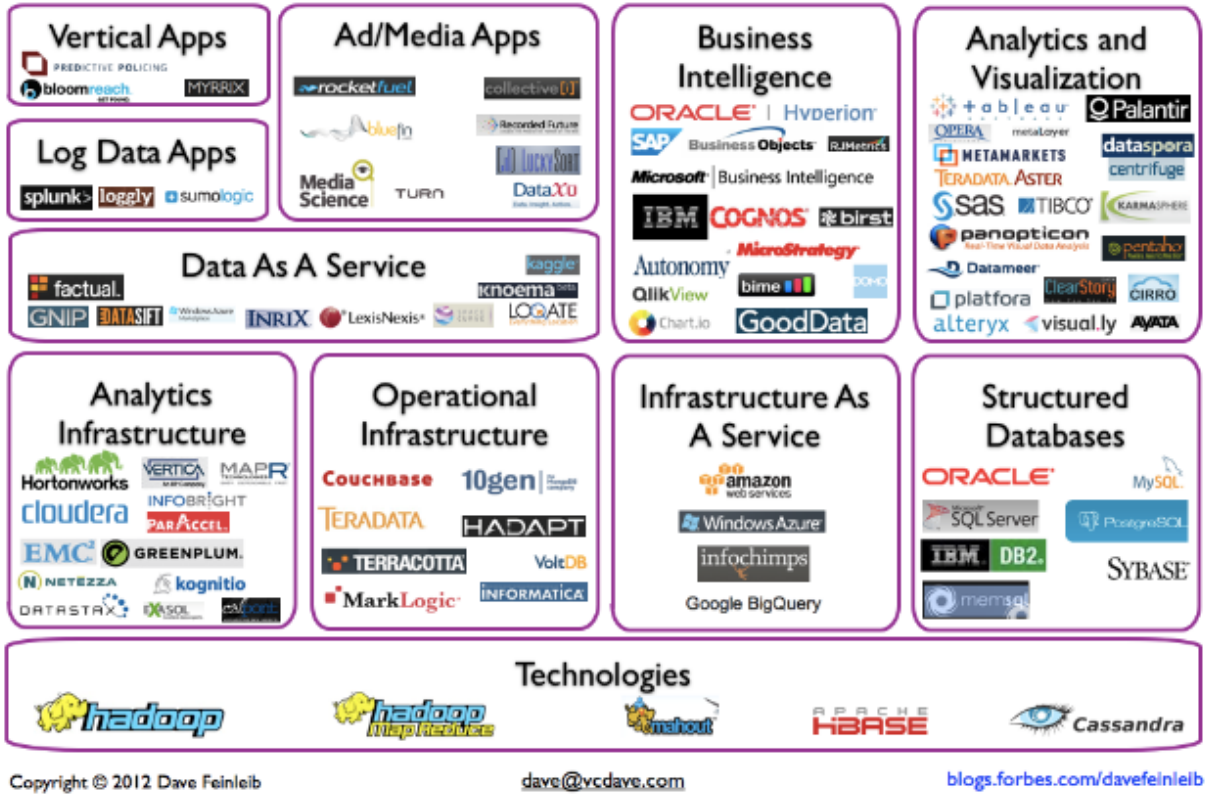
- Ex. Wal-Mart will have different storage and analytical needs then Google's YouTube business.



- Interesting Fact: Walmart handles more than 1 million customer transactions every hour, which is imported into databases estimated to contain more than 2.5 petabytes (2560 terabytes) of data – the equivalent of 167 times the information contained in all the books in the US Library of Congress.

Preface

- Which technologies do you actually need to navigate this exciting new landscape?
- Here was the landscape of technologies in 2012.

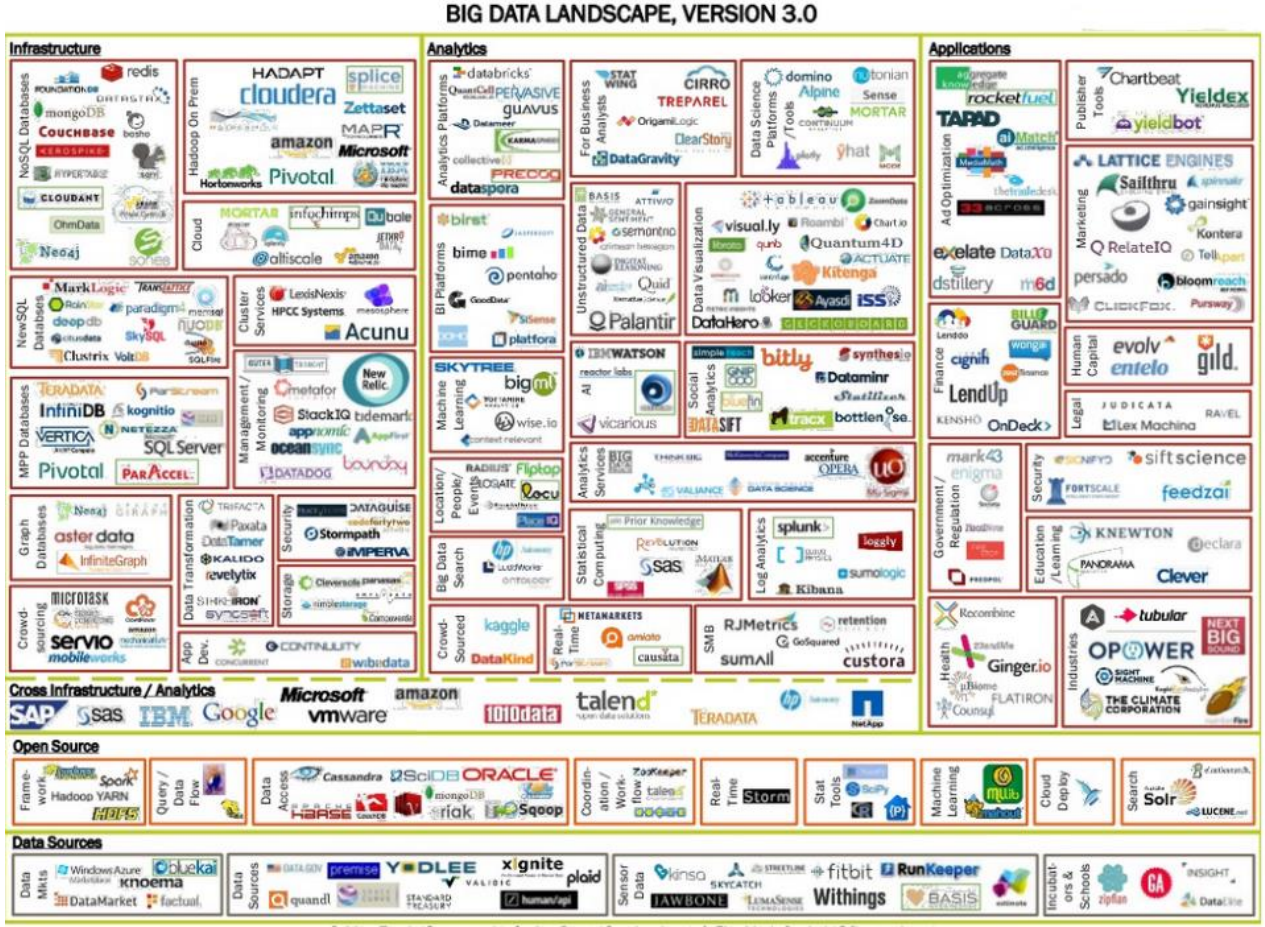


dave@vcdave.com

blogs.forbes.com/davefeinleib

Preface

- Here is a more current iteration of the landscape. Are you getting a little more nervous? Don't worry. We will sift through the noise and get to the heart of Big Data.



The Age of Big Data

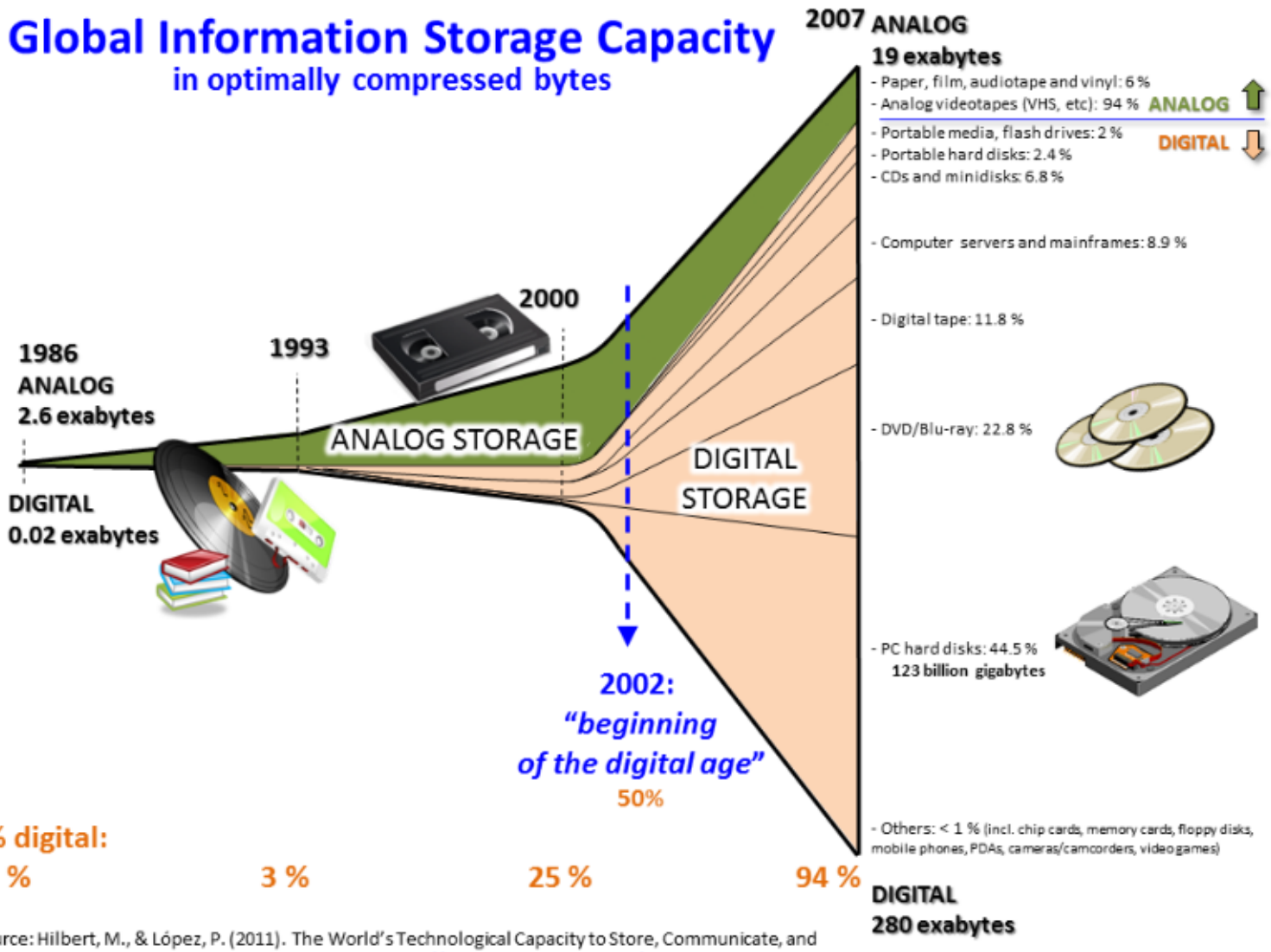
Data Explosion in the 21st Century

Data Explosion in the 21st Century

- Developed economies make increasing use of data-intensive technologies.
- There are 4.6 billion mobile-phone subscriptions worldwide and there are between 1 billion and 2 billion people accessing the internet.
- Between 1990 and 2005, more than 1 billion people worldwide entered the middle class which means more and more people who gain money will become more literate which in turn leads to information growth.

Global Information Storage Capacity

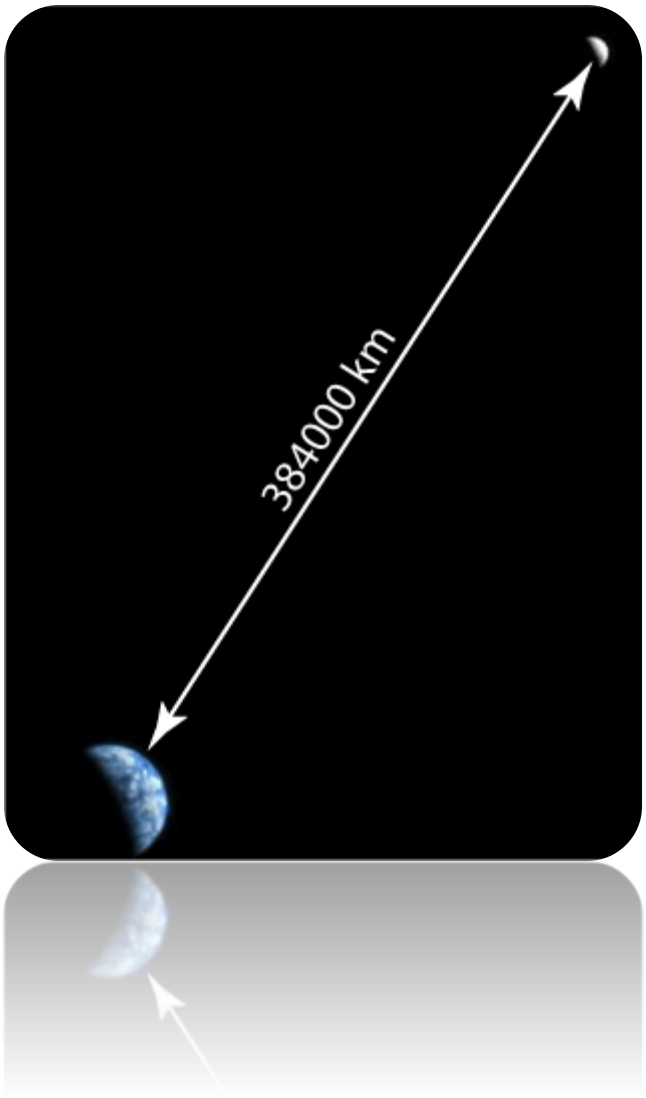
in optimally compressed bytes



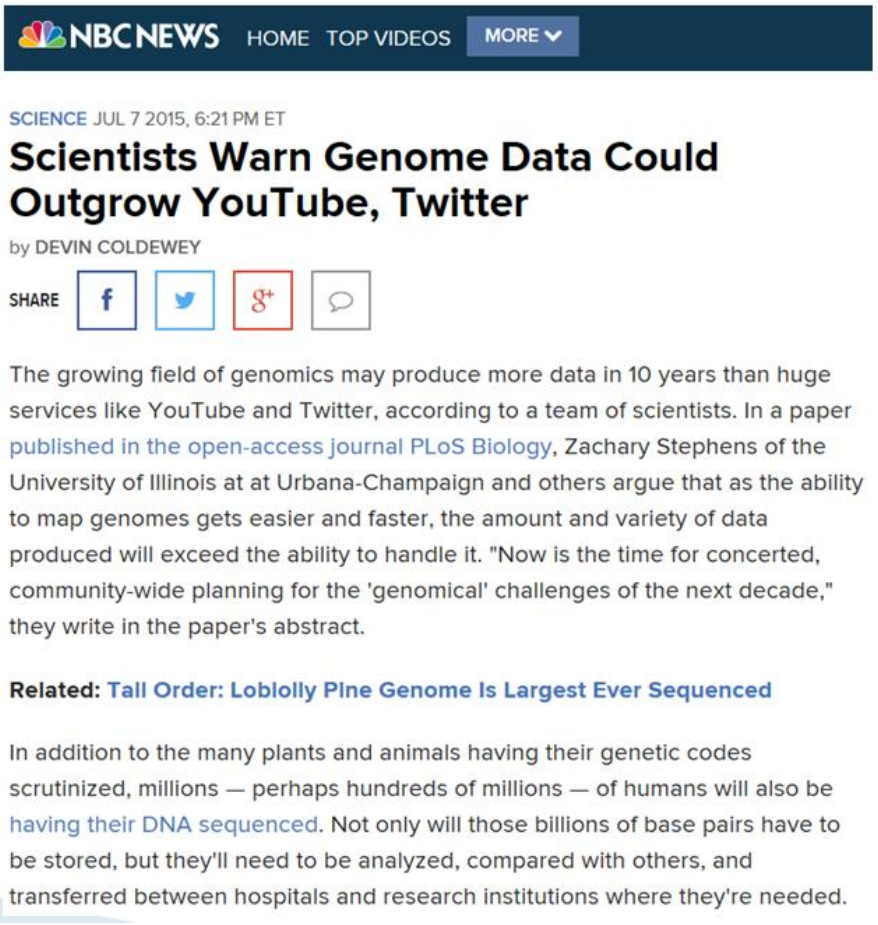
Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

Data Explosion in the 21st Century

- In 2007, the amount of digital data represents about 61 cd's worth of information for every human on the planet.
- If we were to stack these 404 billion cd's on top of one another, this stack would reach the moon and a quarter of the distance beyond.
- According to Cisco IP traffic forecast, global mobile data traffic will grow 131% between 2008-2013 reaching 2 Exabyte's per month in 2013.
- A single Exabyte is roughly equivalent to the entire US Library of Congress text duplicated 100,000 times.
- By the end of 2016, annual global traffic will reach 1.3 Zettabytes or 110.3 Exabytes per month.



Data Explosion in the 21st Century



NBC NEWS HOME TOP VIDEOS MORE

SCIENCE JUL 7 2015, 6:21 PM ET

Scientists Warn Genome Data Could Outgrow YouTube, Twitter

by DEVIN COLDEWEY

SHARE

The growing field of genomics may produce more data in 10 years than huge services like YouTube and Twitter, according to a team of scientists. In a paper published in the open-access journal PLoS Biology, Zachary Stephens of the University of Illinois at at Urbana-Champaign and others argue that as the ability to map genomes gets easier and faster, the amount and variety of data produced will exceed the ability to handle it. "Now is the time for concerted, community-wide planning for the 'genomical' challenges of the next decade," they write in the paper's abstract.

Related: Tall Order: Loblolly Pine Genome Is Largest Ever Sequenced

In addition to the many plants and animals having their genetic codes scrutinized, millions — perhaps hundreds of millions — of humans will also be having their DNA sequenced. Not only will those billions of base pairs have to be stored, but they'll need to be analyzed, compared with others, and transferred between hospitals and research institutions where they're needed.

- As our computing power steadily increases, the data explosion will continue to drive innovations within all aspects of our lives.
- The Human Genome Project has launched a new era surrounding our understanding of biology and medical diseases.
- This includes identifying the underlying genetic markers for some of natures cruelest genetic mutations.
- As we shift our focus towards leveraging this knowledge towards saving lives, there will be an explosion of data that will need to be managed.

Data Explosion in the 21st Century

- This voluminous data will require a more robust infrastructure in order to capitalize on the benefits that genomic data has to offer.
- The article suggests that the broader scientific community is ill prepared to handle data reaching into the Exabyte threshold.
- This example showcases that as we are getting smarter and generating significant volumes of data, we need to ensure that our infrastructures are keeping up.

The data could reach into the exabytes — millions of times the storage space of ordinary computers — putting genomics in league with YouTube, Twitter and ultra-high-resolution astronomy sites.

Related: Bowhead Whale's Genome Could Hold Clues to Longevity

The infrastructure for that just doesn't exist right now, the team warns — even basic standards need hammering out, so competing companies and sequencers can work seamlessly together. In a Nature editorial, others dissented, saying the estimates may be high — but that the problem is still very real.

Data Explosion in the 21st Century

This volume of information can be separated into 2 distinct data types: Structured and Unstructured Data.

Structured – Data is organized in a highly mechanized or manageable manner. Some examples include data tables, OLAP cubes, XML format, etc…

Unstructured – Raw and unorganized data which can be cumbersome and costly to work with. Examples include News Articles, Social Media, Video, Email, etc..

Merrill Lynch projected that 80-90% of all potential usable information exists in unstructured form. In 2010, Computer World claimed that unstructured information might account for 70-80% of all data in an organization.

The Cisco IP traffic forecast projects that video will account for 61% of the total internet data in 2015.

Data Explosion in the 21st Century

"Your recent Amazon purchases, Tweet score and location history makes you 23.5% welcome here."

- Software AG, Oracle Corporation, IBM, Microsoft, SAP, EMC, HP and Dell have spent more than $15 billion on software firms only specializing in data management and analytics.
- In 2010, this industry on its own was worth more than $100 billion and was growing at almost 10 percent a year: about twice as fast as the software business as a whole.

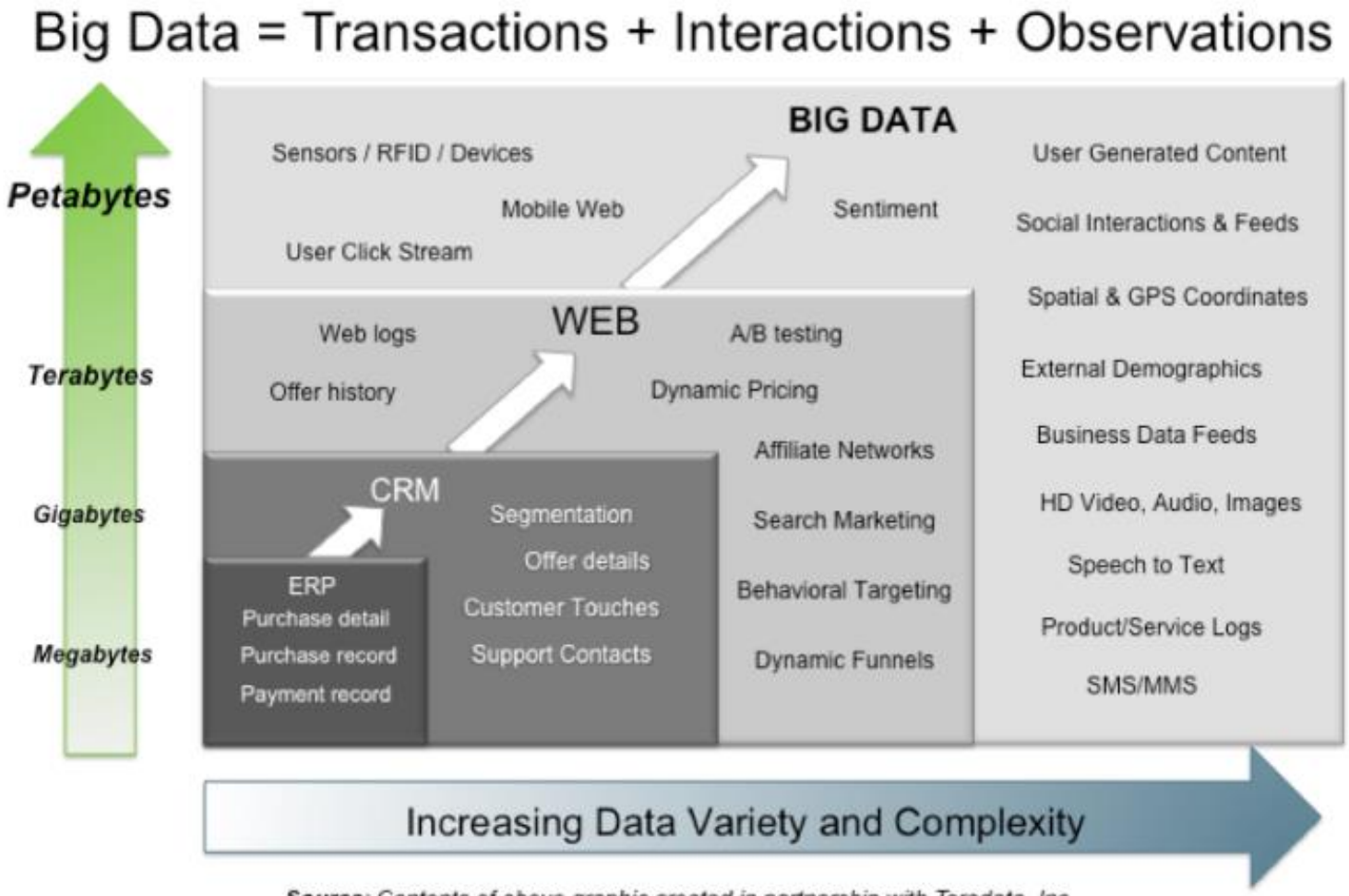
What Exactly is Big Data?

Big Data is a collection of datasets that are so large that they become difficult to process with traditional database management tools. This requires a new set of tools and techniques to manage the data.

The “Big Data” concept is somewhat ambiguous and is used interchangeably with unstructured data analysis, social media/ web analytics, and growing data needs that exceeds the organizational capabilities.

A simple guideline for determining whether or not an organization is in the “Big Data” realm can be based on the size of the datasets they are working with. Generally, datasets that are many terabytes to petabytes in size would fall into the “Big Data” family.

What Exactly is Big Data?



***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

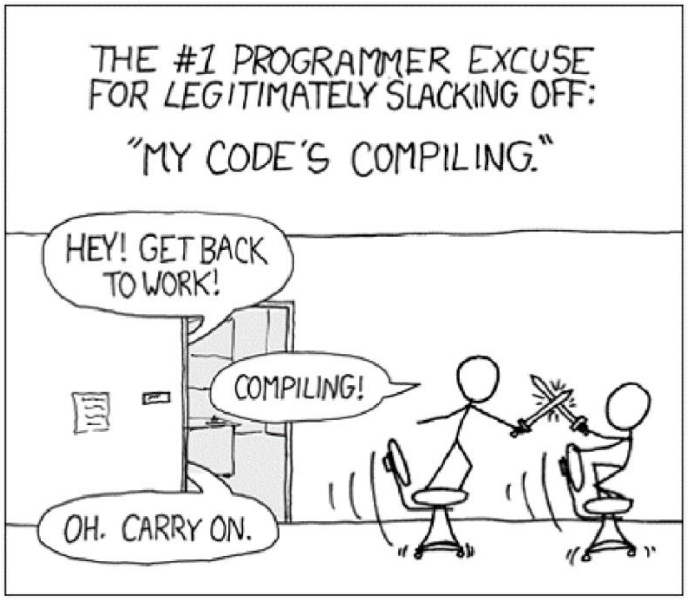
Is Your Problem Big Enough for Big Data?

When to use Big Data Technologies?:

- Conventional processing tools won't work on your data.
- Your data is really BIG.
- Won't fit/process in your favorite database or file-system.
- Your data is really diverse !
- Semi-structured – JSON, XML, Logs, Images, Sounds.
- You're a whiz at programming.

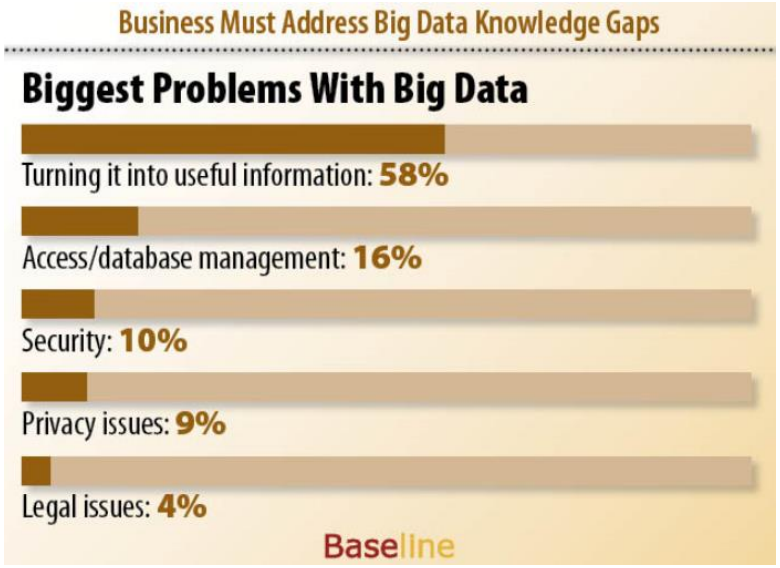
When not to use Big Data Technologies?

- !(When to use Big Data Technologies?:)
- You're in a hurry !!!!



My Job is Reducing – "Andrie de Vries & Michele Usuelli"

Why Should I Care About Big Data?



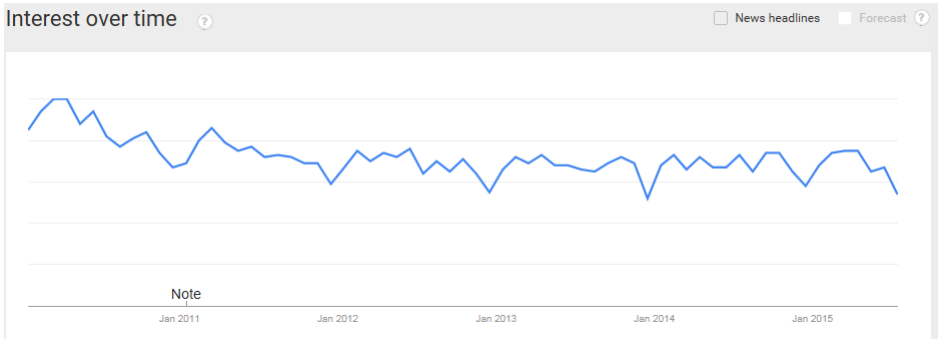
- Many people consider Big Data solutions to be a new way to do data warehousing when the volume of data exceeds the capacity or cost limitations for relational database systems.
- However, it can be difficult to fully grasp what Big Data solutions really involve, what hardware and software they use, and how and when they are useful.
- It is important that we are able to break through any misconceptions regarding Big Data and drive towards realizable value.

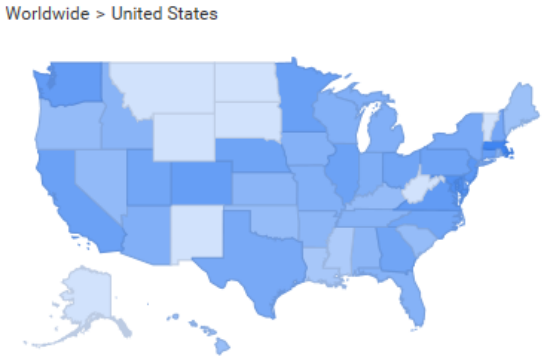
Why Should I Care About Big Data?

- There are some basic questions, the answers to which will help you understand where Big Data solutions are useful—and how you might approach the topic in terms of implementing your own solutions:
  - Why do I need a Big Data solution?
  - What problems do Big Data solutions solve?
  - How is a Big Data solution different from traditional database systems?
  - Will a Big Data solution replace my relational databases?

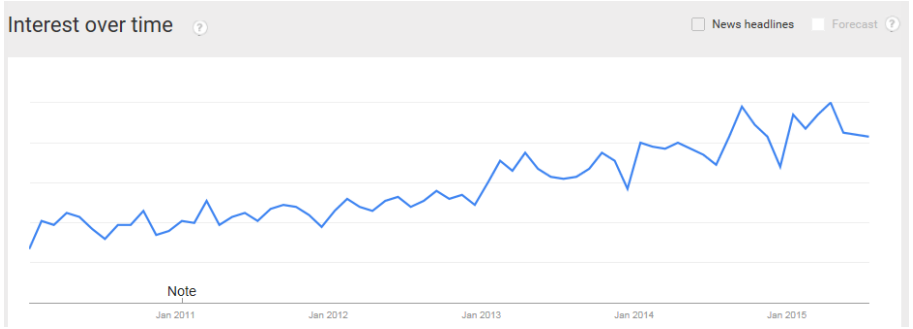
Google Trends Analysis

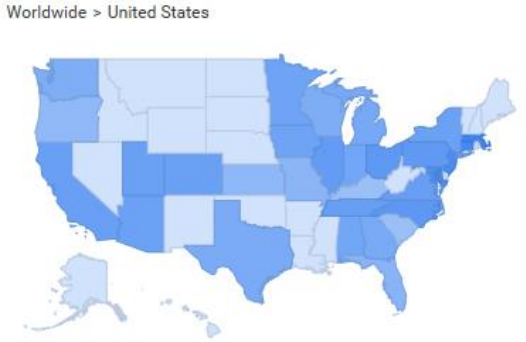
Business Intelligence





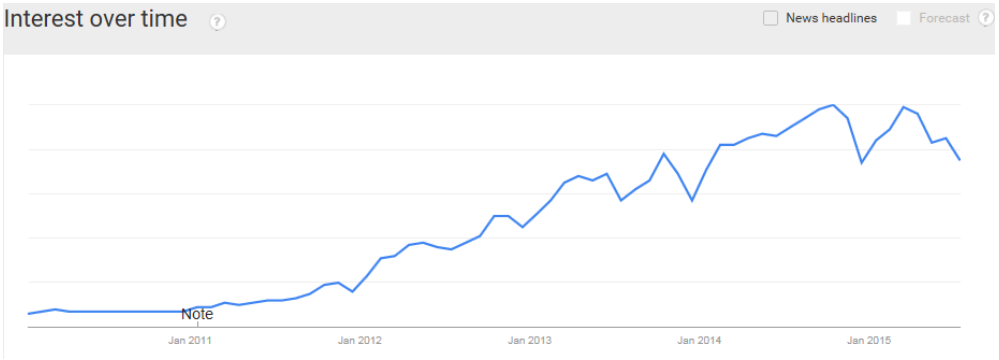
Business Analytics

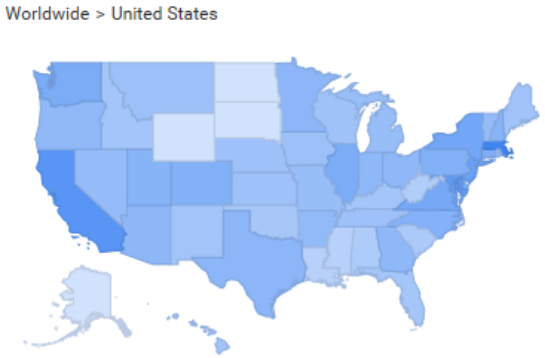




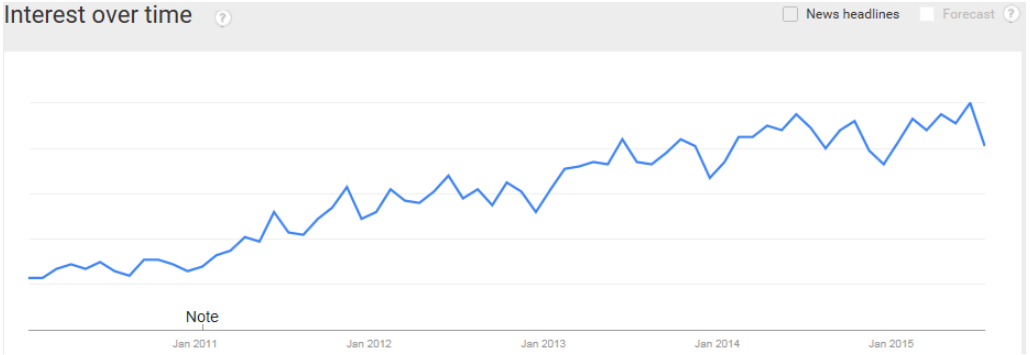
Google Trends Analysis

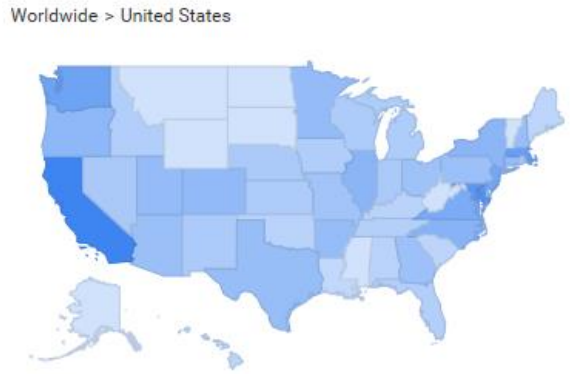
Big Data





Hadoop





Why Do I Need a Big Data Solution?

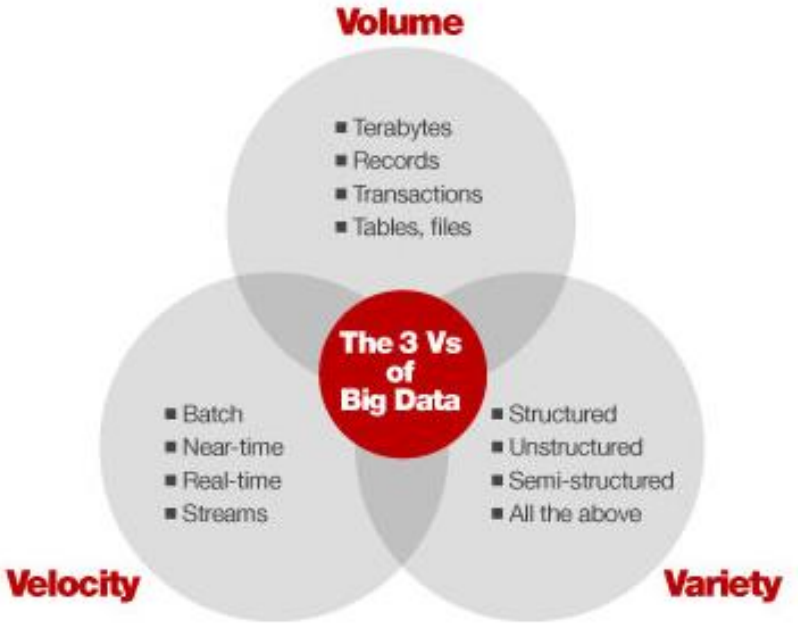
- Organizations will need a Big Data solution to enable them to survive in an increasingly competitive market where the sources and the requirements to store data are growing at exponential rate.

Big Data solutions are typically used for:

- Storing huge volumes of unstructured or semi-structured data.
- Finding hidden insights in large stores of data from multiple devices. (Ex. Internet of Things)
- Extracting vital management information.

- Big Data solutions provide a way to help you discover value, which often cannot be measured just through traditional business methods such as cost and revenue analysis.

What Problems Do Big Data Solutions Solve?



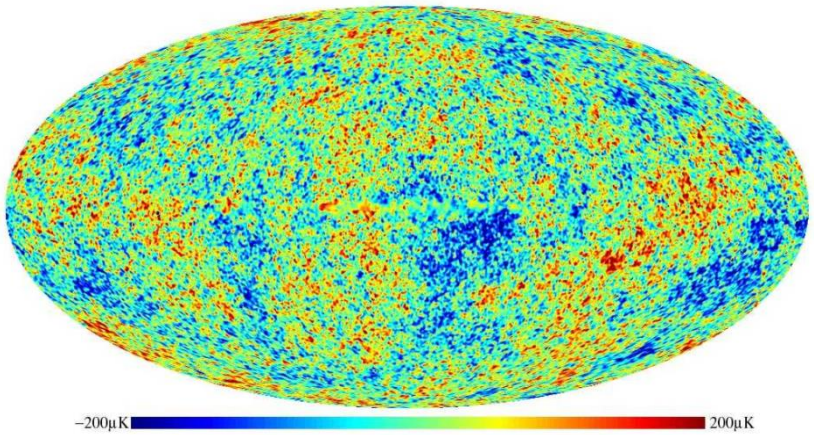
Big Data is often described as a solution to the "three V's problem":

- Volume: Big Data solutions are designed and built to store and process hundreds of petabytes of data in a way that can dramatically reduce storage cost. Processing systems must be scalable to handle increasing volumes.
- Variety: Organizations often collect unstructured data, which is not in a format that suits relational database systems. This means that applying schemas to the data before or during storage is no longer a practical proposition.
- Velocity: Data is being collected at an increasing rate from many new types of devices and from a fast-growing number of users. The design and implementation of storage must be able to manage this efficiently, and processing systems must be able to return results within an acceptable timeframe.

What Problems Do Big Data Solutions Solve?

- The combination of all these factors means that, in some circumstances, a Big Data batch processing solution may be a more practical proposition than a traditional relational database system.
- However, as Big Data solutions have continued to evolve it has become clear that they can also be used in a fundamentally different context.
- They can be leveraged to quickly get insights into data, and to provide a platform for further investigation in a way that just isn't possible with traditional data storage, management, and querying tools.

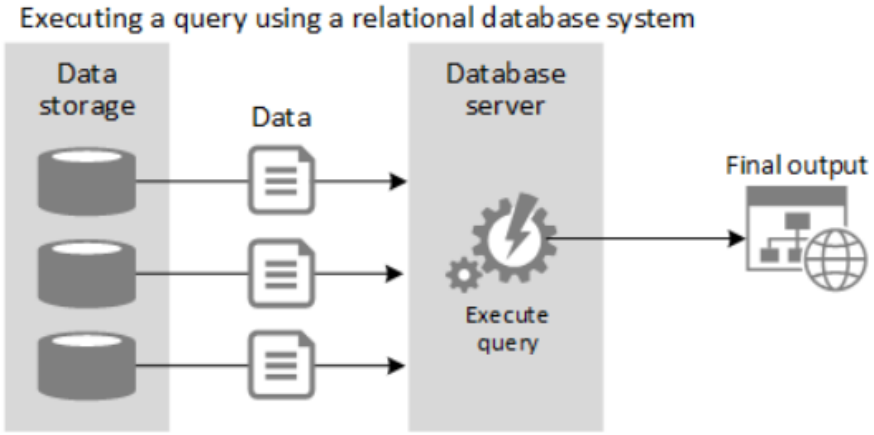
What Problems Do Big Data Solutions Solve?



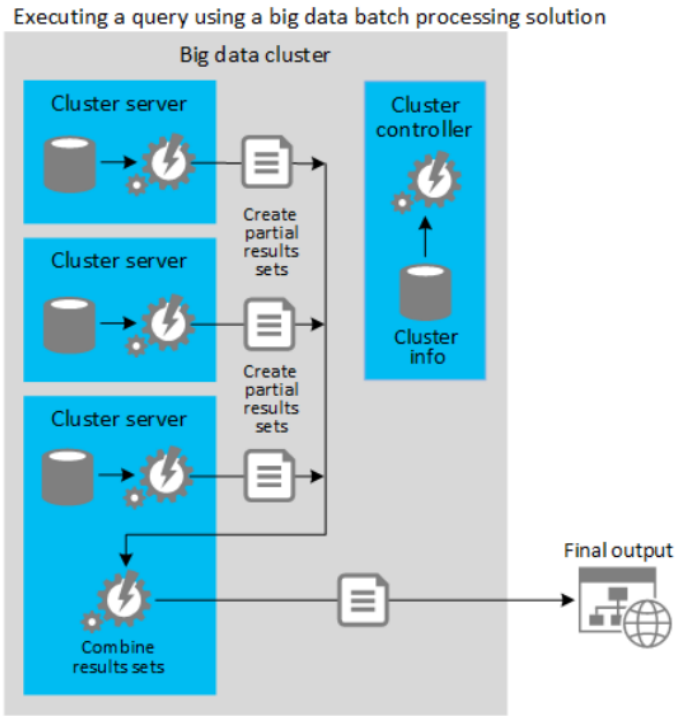
- Big Data solutions aren't all about business topics such as customer sentiment or web server log file analysis.
- They have many diverse uses around the world and across all types of applications.
  - Police forces are using Big Data techniques to predict crime patterns.
  - Researchers are using them to explore the human genome.
  - Particle physicists are using them to search for information about the structure of matter.
  - Astronomers are using them to plot the entire universe.
- That's what we in the business would call "Big Data".

How is a Big Data Solution Different From Traditional Database Systems?

- Traditional database systems typically use a relational model where all the data is stored using predetermined schemas, and linked using the values in specific columns of each table.
- Requiring a schema to be applied when data is written may mean that some information hidden in the data is lost.
- There are some more flexible mechanisms, such as the ability to store XML documents and binary data, but the capabilities for handling these types of data are usually quite limited.



How is a Big Data Solution Different From Traditional Database Systems?



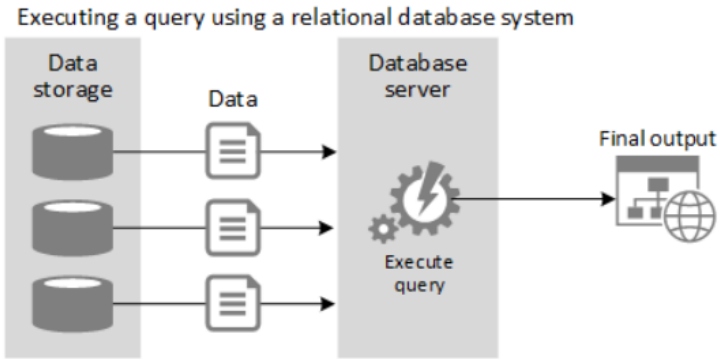
- Big Data solutions do not force a schema onto the stored data.
- Instead, you can store almost any type of structured, semi-structured, or unstructured data and then apply a suitable schema when you query this data.
- Big Data solutions store the data in its raw format and apply a schema only when the data is read, which preserves all of the information within the data.

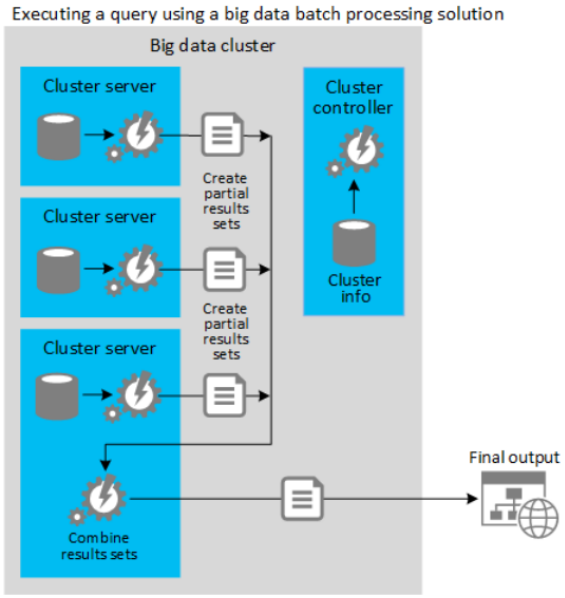
How is a Big Data Solution Different From Traditional Database Systems?

- Traditional database systems typically consist of a central node where all processing takes place, which means that all the data from storage must be moved to the central location for processing.
- The capacity of this central node can be increased only by scaling up, and there is a physical limitation on the number of CPUs and memory, depending on the chosen hardware platform.
- The consequence of this is a limitation of processing capacity, as well as network latency when the data is moved to the central node.
- In contrast, Big Data solutions are optimized for storing vast quantities of data using simple file formats and highly distributed storage mechanisms, and the initial processing of the data occurs at each storage node.
- This means that, assuming you have already loaded the data into the cluster storage, the bulk of the data does not need to be moved over the network for processing.

How is a Big Data Solution Different From Traditional Database Systems?

- This figure shows some of the basic differences between a relational database system and a Big Data solution in terms of storing and querying data.





- Notice how both relational databases and Big Data solutions use a cluster of servers; the main difference is where query processing takes place and how the data is moved across the network.

How is a Big Data Solution Different From Traditional Database Systems?

- Modern data warehouse systems typically use high speed fiber networks, in-memory caching, and indexes to minimize data transfer delays.
- However, in a Big Data solution only the results of the distributed query processing are passed across the cluster network to the node that will assemble them into a final results set.
- Under ideal conditions, performance during the initial stages of the query is limited only by the speed and capacity of connectivity to the co-located disk subsystem, and this initial processing occurs in parallel across all of the cluster nodes.

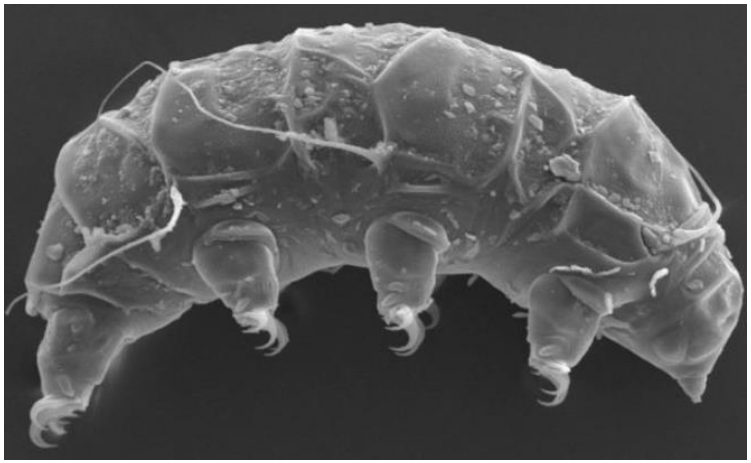
How is a Big Data Solution Different From Traditional Database Systems?

- The ability to work with highly distributed data and simple file formats also opens up opportunities for more efficient and more comprehensive data collection.
- For example, services and applications can store data in any of the predefined distributed locations without needing to preprocess it or execute queries that can absorb processing capacity.
- Data is simply appended to the files in the data store.
- Any processing required on the data is done when it is queried, without affecting the original data and risking losing valuable information.

How is a Big Data Solution Different From Traditional Database Systems?

- Queries to extract information in a Big Data solution are typically batch operations that may take some time to return a final result.
- However, when you consider the volumes of data that Big Data solutions can handle, the fact that queries run as multiple tasks on distributed servers does offer a level of performance that may not be achievable by other methods.
- While it is possible to perform real-time queries, typically you will run the query and store the results for use within your existing BI tools and analytics systems.
- This means that Big Data queries are typically not executed repeatedly as part of an application's execution—and so batch operation is not a major disadvantage.

How is a Big Data Solution Different From Traditional Database Systems?



A resilient species of water bear that is able to survive the vacuum of outer space.

- Big Data systems are also designed to be highly resilient against failure of storage, networks, and processing.
- The distributed processing and replicated storage model is fault-tolerant, and allows easy re-execution of individual stages of the process.
- The capability for easy scaling of resources also helps to resolve operational and performance issues.

How is a Big Data Solution Different From Traditional Database Systems?

- The following table summarizes the major differences between a Big Data solution and existing relational database systems.

Feature	Relational Database Systems	Big Data Solutions
Data types and formats	Structured	Semi-structured and unstructured
Data integrity	High—transactional updates	Depends on the technology used.
Schema	Static—required on write	Dynamic—optional on read and write
Read and write pattern	Fully repeatable read/write	Write once, repeatable read
Storage volume	Gigabytes to terabytes	Terabytes, petabytes, and beyond
Scalability	Scale up with more powerful hardware	Scale out with additional servers
Data processing distribution	Limited or none	Distributed across the cluster
Economics	Expensive hardware and software	Commodity hardware and open source software

Will a Big Data Solution Replace My Relational Database?

- Big Data batch processing solutions offer a way to avoid storage limitations, or to reduce the cost of storage and processing, for huge and growing volumes of data.
- This is especially true where this data might not be part of a vital business function.
- But this isn't to say that relational databases are no longer essential.
- Continual development of the hardware and software for this core business function provides capabilities for storing very large amounts of data.

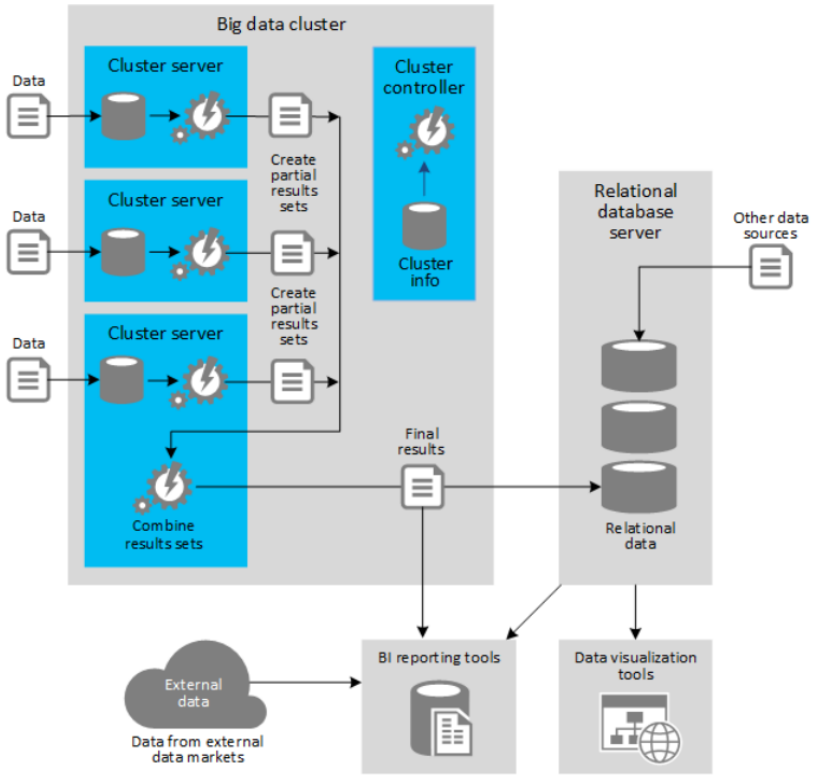
Will a Big Data Solution Replace My Relational Database?

A Crocodile Plover in a symbiotic relationship.

- In fact, the relational database systems we use today and the more recent Big Data batch processing solutions are complementary mechanisms.
- Big Data batch processing solutions are extremely unlikely ever to replace the existing relational database—in the majority of cases they complement and augment the capabilities for managing data and generating BI.
- For example, it's common to use a Big Data query to create a result set that is then stored in a relational database for use in the generation of BI, or as input to another process.

Will a Big Data Solution Replace My Relational Database?

- Big Data is also a valuable tool when you need to handle data that is arriving very quickly, and which you can process later.
- You can dump the data into the storage cluster in its original format, and then process it when required using a query that extracts the required result set and stores it in a relational database, or makes it available for reporting.
- This complimentary based system allows for us to deploy our full arsenal of machine learning techniques and an infrastructure highly suitable for Data Science.

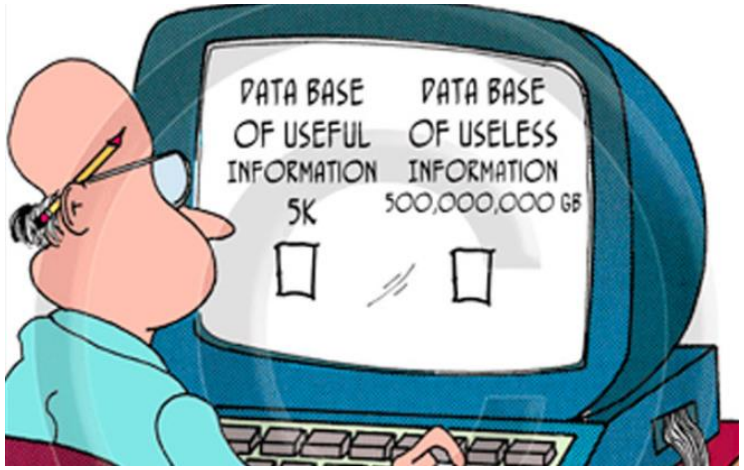


How Do Big Data Solutions Work?

- In the days before Structured Query Language (SQL) and relational databases, data was typically stored in flat files, often is simple text format, with fixed width columns.
- Application code would open and read one or more files sequentially, or jump to specific locations based on the known line width of a row, to read the text and parse it into columns to extract the values.
- Results would be written back by creating a new copy of the file or a separate results file.

	A	B	C	D	E	F	G	H	I
1	Header 1	Header 2	Header 3	Header 4	Header 5	Header 6	Header 7	Header 8	Header 9
2	2013000153	2013000153	VTS		1/1/2013 0:01	1	240	-74.004517	40.721241
3	2013001117	2013001114	VTS		1/1/2013 0:06	1	960	-73.995415	40.759789
4	2013002494	2013002491	VTS		1/1/2013 0:10	4	540	-73.988754	40.722275
5	2013000562	2013000560	CMT	N	1/1/2013 0:12	2	1336	-73.952782	40.791897
6	2013000118	2013000118	VTS		1/1/2013 0:14	1	2160	-73.995392	40.754562
7	2013001783	2013001780	VTS		1/1/2013 0:17	3	180	-74.002182	40.726425
8	2013005459	2013005456	VTS		1/1/2013 0:18	1	1200	-73.985664	40.763676
9	2013001304	2013001301	CMT	N	1/1/2013 0:20	2	329	-74.006737	40.73164
10	2013006603	2013006599	VTS		1/1/2013 0:22	1	480	-74.003532	40.734226
11	2013003205	2013003202	CMT	N	1/1/2013 0:24	3	323	-73.961975	40.763859
12	2013002411	2013002408	CMT	N	1/1/2013 0:26	1	496	-73.972183	40.749954
13	2013007943	2013007939	VTS		1/1/2013 0:28	1	360	-73.993263	40.73584
14	2013008173	2013008169	VTS		1/1/2013 0:30	5	720	-73.994347	40.760849
15	2013003010	2013003007	VTS		1/1/2013 0:31	2	780	-73.959175	40.714935
16	2013008633	2013008629	CMT	N	1/1/2013 0:33	1	577	-73.972603	40.757713
17	2013003994	2013003991	CMT	N	1/1/2013 0:34	2	428	-73.958527	40.715004
18	2013008974	2013008970	CMT	N	1/1/2013 0:38	2	848	-73.986359	40.745609
19	2013008993	2013008989	VTS		1/1/2013 0:38	1	420	-73.978683	40.720047
20	2013005268	2013005265	CMT	N	1/1/2013 0:40	1	572	-73.993614	40.745876
21	2013008157	2013008153	CMT	N	1/1/2013 0:42	1	237	-73.975235	40.732994

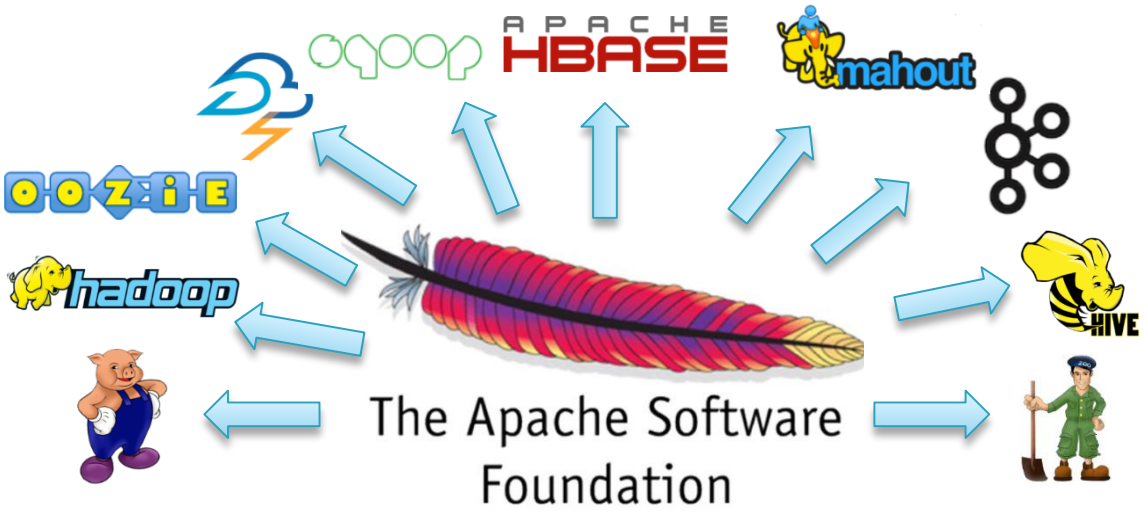
How Do Big Data Solutions Work?



- Modern relational databases put an end to all this, giving us greater power and additional capabilities for extracting information simply by writing queries in a standard format such as SQL.
- The database system hides all the complexity of the underlying storage mechanism and the logic for assembling the query that extracts and formats the information.
- However, as the volume of data that we collect continues to increase, and the native structure of this information is less clearly defined, we are moving beyond the capabilities of even enterprise-level relational database systems.

The Apache Software Foundation

- At the core of many Big Data implementations is an open source technology named Apache Hadoop. Hadoop was developed by Yahoo and the code was then provided as open source to the Apache Software Foundation.



- Established in 1999, the ASF is a US 501(c)(3) charitable organization, funded by individual donations and corporate sponsors. The all-volunteer board oversees more than 350 leading Open Source projects.

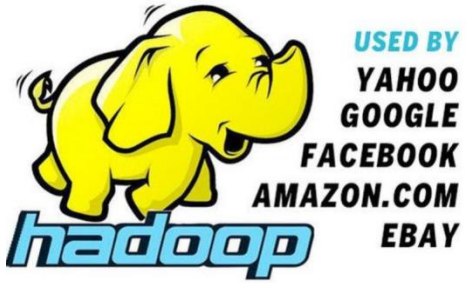
Why Hadoop?

Source: http://www.indeed.com/jobtrends?q=HTML5%2C+hadoop%2C+SAS&l

What is Hadoop?

Doug Cutting, Hadoop Pioneer

- Hadoop was created by Doug Cutting and Mike Cafarella in 2005.
- The name "Hadoop" was given by one of Doug Cutting's sons to that son's toy elephant. Doug used the name for his open source project because it was easy to pronounce and to Google.
- As of 2013, Hadoop adoption is widespread. For example, more than half of the Fortune 50 use Hadoop.



What is Hadoop?

- The Apache Hadoop software library is a framework (written in Java) that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- It is designed to scale up from single computers to thousands of machines, each offering local computation and storage.
- Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

What is Hadoop?

A line change in hockey swaps out the tired players for fresh bodies.

- All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.
- If there is a hardware that needs replacement, we can swap out the defective piece and replace it with a higher performing machine.
- The Hadoop architecture allows for new hardware to be installed and scaled up without having to rewrite the underling code.

Key Point: Once you have your Hadoop system operational, you can expand and build up the performance seamlessly. The hard work is done!

Hadoop on the Cloud

Hadoop can be deployed in traditional on-site datacenters but has also been implemented in public cloud spaces such as:

- Microsoft Azure
- Amazon Web Services
- Google Compute Engine
- IBM Bluemix.
- And Many Others…

Hadoop on the Cloud

- Here are some examples of Cloud-Based platforms:
- Azure HDInsight is a service that deploys Hadoop on Microsoft Azure. HDInsight uses a Windows-based Hadoop distribution that was jointly developed with Hortonworks and allows programming extensions with .NET.
- It is possible to run Hadoop on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3).

Interesting Example: The New York Times used 100 Amazon EC2 instances and a Hadoop application to process 4 TB of raw image TIFF data into 11 million finished PDFs in the space of 24 hours at a computation cost of about $240.

Hadoop on the Cloud

What is Hadoop?

The Apache Hadoop core consists of:

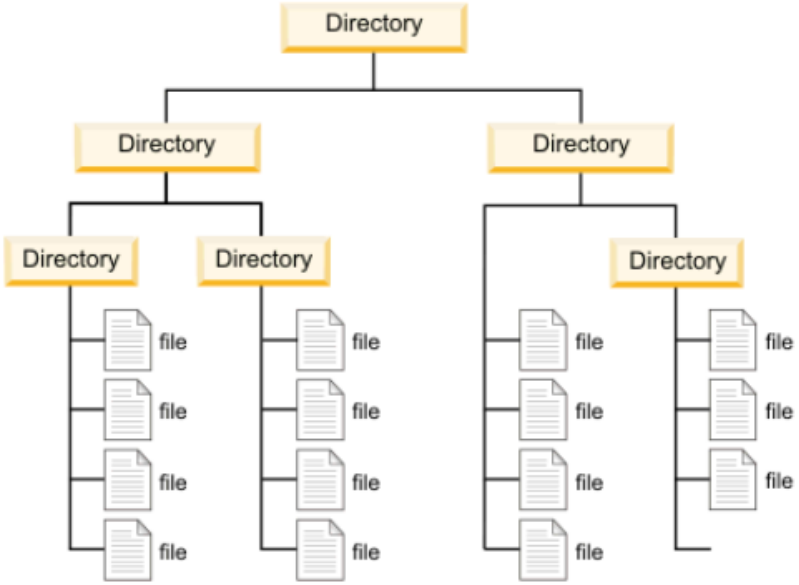
- The Hadoop Common Package (Kernel)
- A storage component - HDFS (Hadoop Distributed File System)
- A data processing component - Map/Reduce Framework (either MapReduce/MR1 or YARN/MR2)
- A runtime resource manager that allocates tasks, and executes queries (such as map/reduce jobs) and other applications. (Ex. YARN)
- Other resources, tools, and utilities that run under the control of the resource manager (YARN) to support tasks such as managing data and running queries or other jobs on the data.

Note: This approach allows the data to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are connected via high-speed networking.

What is Hadoop?

- The Hadoop Common package contains the necessary Java Archive (JAR) files and scripts needed to start Hadoop.
- Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (ssh) be set up between nodes in the cluster.
- The package also provides source code, documentation, and a contribution section that includes projects from the Hadoop Community.

What is HDFS?

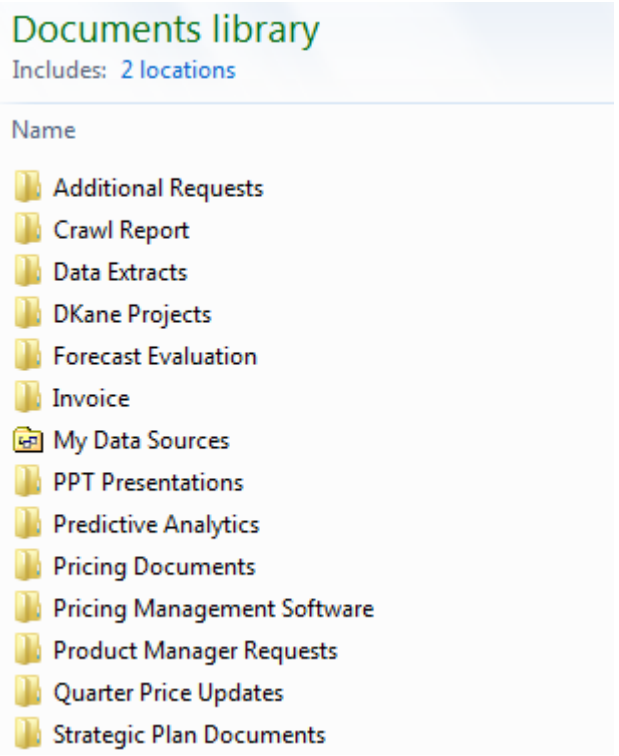


Example of a simple file system.

- The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant.
- HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
- HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

What is HDFS?

- HDFS supports a traditional hierarchical file organization.
- A user or an application can create directories and store files inside these directories.
- The file system namespace hierarchy is similar to most other existing file systems.
- One can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas.
- HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.



What is HDFS?

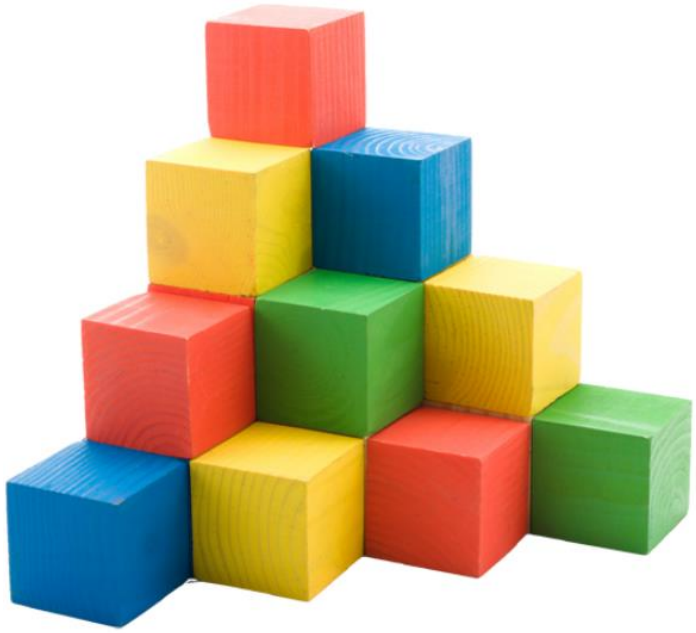
HDFS Assumptions and Goals:

- Simple Coherency Model
  - HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.

- "Moving Computation is Cheaper than Moving Data"
  - A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

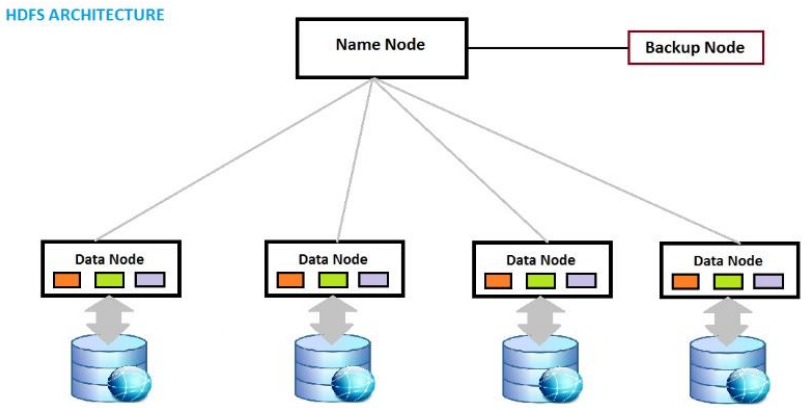
- Portability Across Heterogeneous Hardware and Software Platforms
  - HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

What is HDFS?



- HDFS is designed to reliably store very large files across machines in a large cluster.
- It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
- The blocks of a file are replicated for fault tolerance. (Typically 3 replicas are created)
- The block size and replication factor are configurable per file. An application can specify the number of replicas of a file.
- The replication factor can be specified at file creation time and can be changed later.
- Files in HDFS are write-once and have strictly one writer at any time.

What is HDFS?

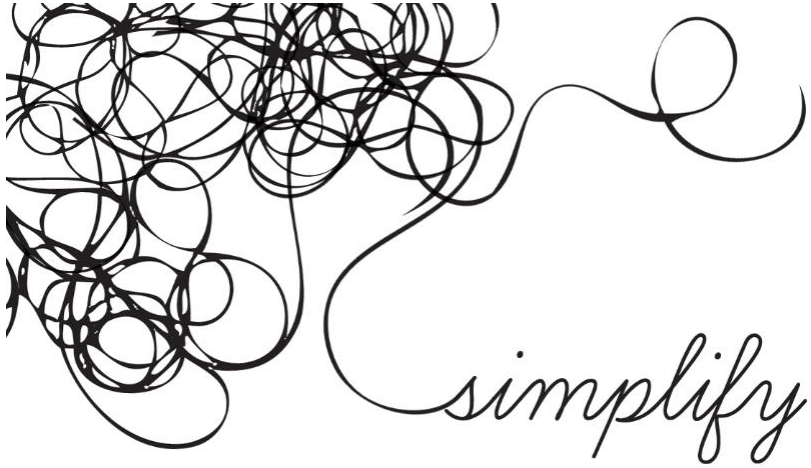


- In order to setup the HDFS environment, we need to identify which computers (servers) act as the primary repository of the data (called the NameNode).
- Additionally, we need to identify those which contain pieces of the data (called the DataNode) in the cluster.
- The NameNode and DataNode are pieces of software designed to run on commodity machines.
- A typical deployment has a dedicated machine that runs only the NameNode software.
- Each of the other machines in the cluster runs one instance of the DataNode software.

Note: These machines typically run a GNU/Linux operating system (OS).

What is HDFS?

- The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.
- The existence of a single NameNode in a cluster greatly simplifies the architecture of the system.
- The system is designed in such a way that user data never flows through the NameNode.



What is HDFS?

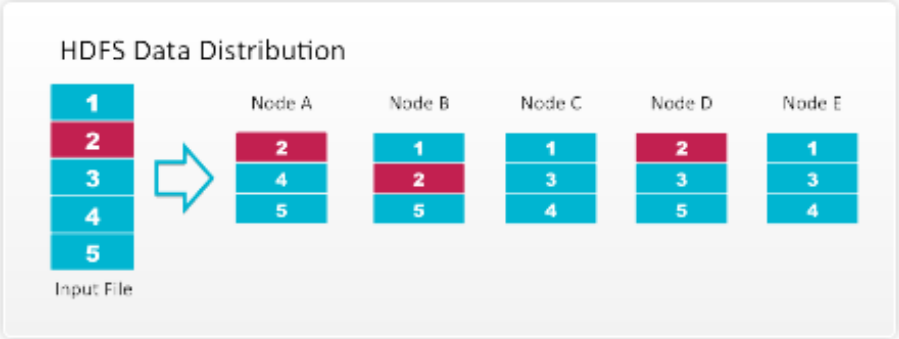
- The NameNode maintains the file system namespace and determines the mapping of blocks to DataNodes.
- Any change to the file system namespace or its properties is recorded by the NameNode.
- The NameNode makes all decisions regarding replication of blocks.
- An application can specify the number of replicas of a file that should be maintained by HDFS. This information is stored by the NameNode.

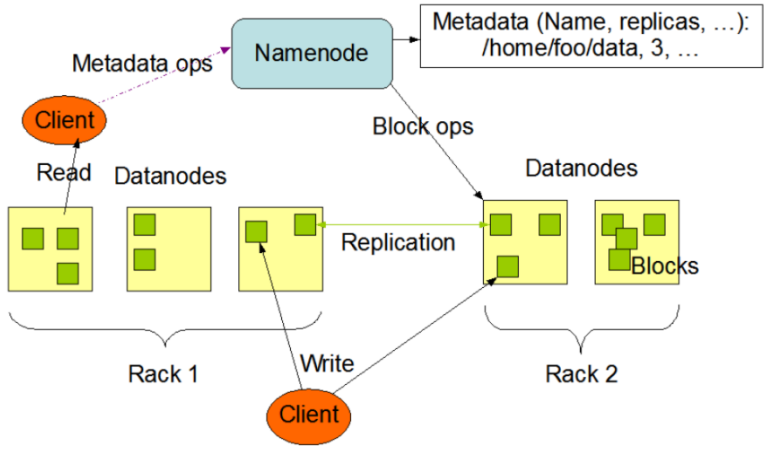
What is HDFS?

- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories.
- The NameNode periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster.
- Receipt of a Heartbeat implies that the DataNode is functioning properly.
- A Blockreport contains a list of all blocks on a DataNode.

What is HDFS?

- Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
- The DataNodes are responsible for serving read and write requests from the file system's clients.
- The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



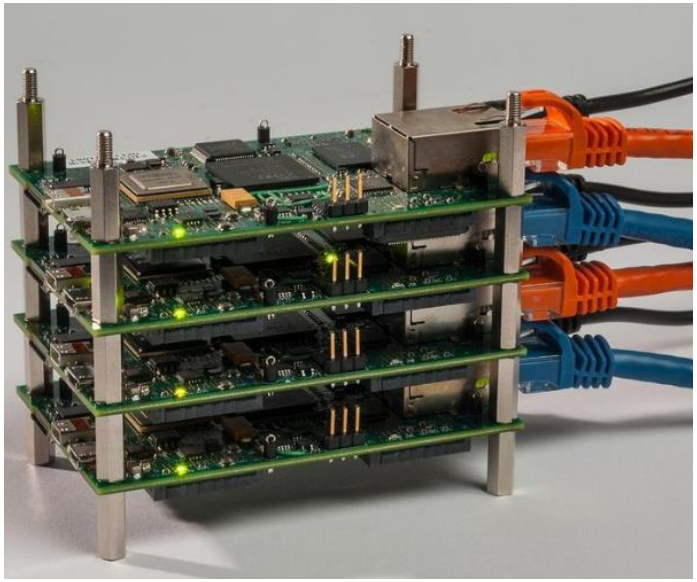


What is HDFS?

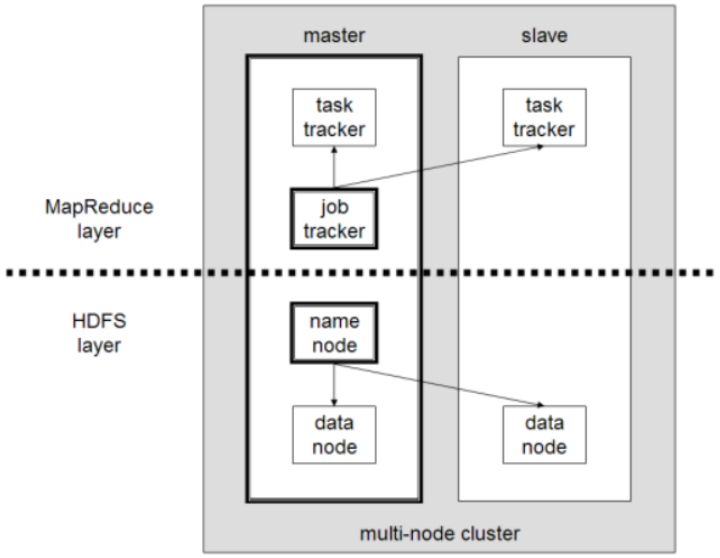
- For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the network switch) where a worker node is.
- Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic.
- HDFS uses this method when replicating data to try to keep different copies of the data on different racks.
- The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable.

Small Hadoop Clusters

- A small Hadoop cluster includes a single master and multiple worker nodes.
- The master node consists of a JobTracker, TaskTracker, NameNode, and DataNode.
- A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have data-only worker nodes and compute-only worker nodes.
- These are normally used only in nonstandard applications.



Large Hadoop Clusters



- In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index.
- There is also a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing file-system corruption and reducing loss of data.
- Similarly, a standalone JobTracker server can manage job scheduling.
- In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents.

The Data Store

- The data store running on each server in a cluster is a suitable distributed storage service such as HDFS or a compatible equivalent.
- Hadoop implementations may use a cluster where all the servers are co-located in the same datacenter.
- This is in order to minimize bandwidth use and maximize query performance.

NoSQL Data Store

- The data store in a Hadoop implementation is usually referred to as a NoSQL store.
- Although this is not technically accurate because some implementations do support a structured query language (such as SQL).
- In fact, some people prefer to use the term "Not Only SQL" for just this reason.
- The particular suitability of a given NoSQL database depends on the problem it must solve.
- There are more than 100+ NoSQL data store implementations available at the time of writing.

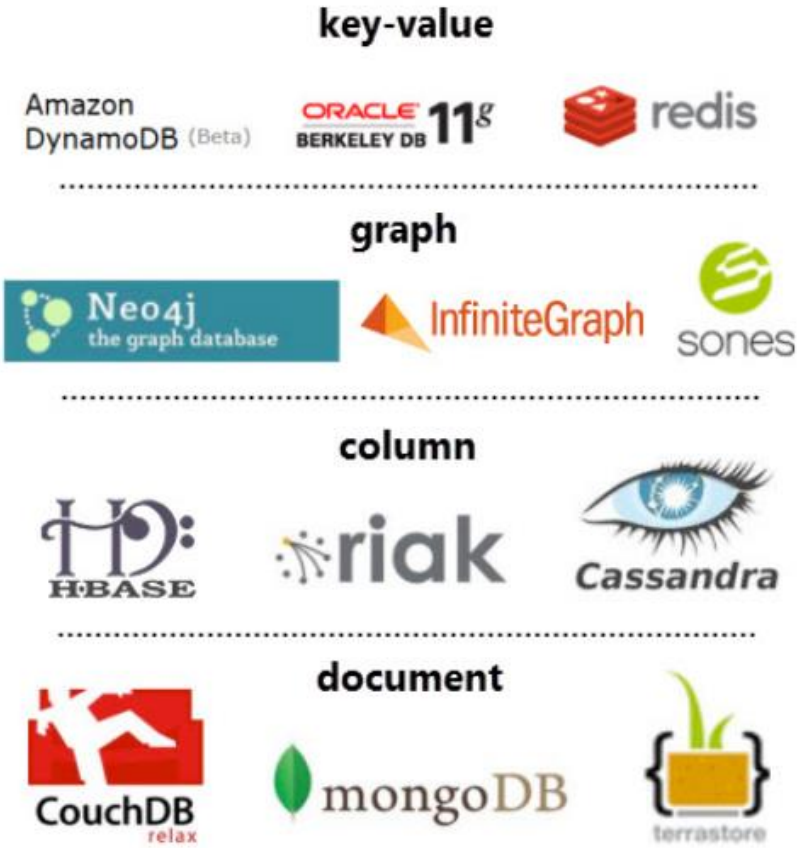
NoSQL Data Store

- They can be divided into the following basic categories:

- Key/Value Stores: These are data stores that hold data as a series of key/value pairs. There is no fixed schema for the data, and so these types of data store are ideal for unstructured data.
- Document Stores: These are data stores optimized to hold structured, semi-structured, and unstructured data items such as JSON objects, XML documents, and binary data. They are usually indexed stores.
- Column Data Stores: These are data stores use a schema, but the schema can contain families of columns rather than just single columns. They are ideally suited to storing semi-structured data, where some columns can be predefined but others are capable of storing differing elements of unstructured data.
- Graph Data Stores: These are data stores that hold the relationships between objects. They are less common than the other types of data store, many still being experimental, and they tend to have specialist uses.

NoSQL Data Store

- NoSQL storage is typically much cheaper than relational storage, and usually supports a write once capability that allows only for data to be appended.
- To update data in these stores you must drop and recreate the relevant file, or maintain delta files and implement mechanisms to conflate the data.
- This limitation maximizes throughput; storage implementations are usually measured by throughput rather than capacity because this is usually the most significant factor for both storage and query efficiency.



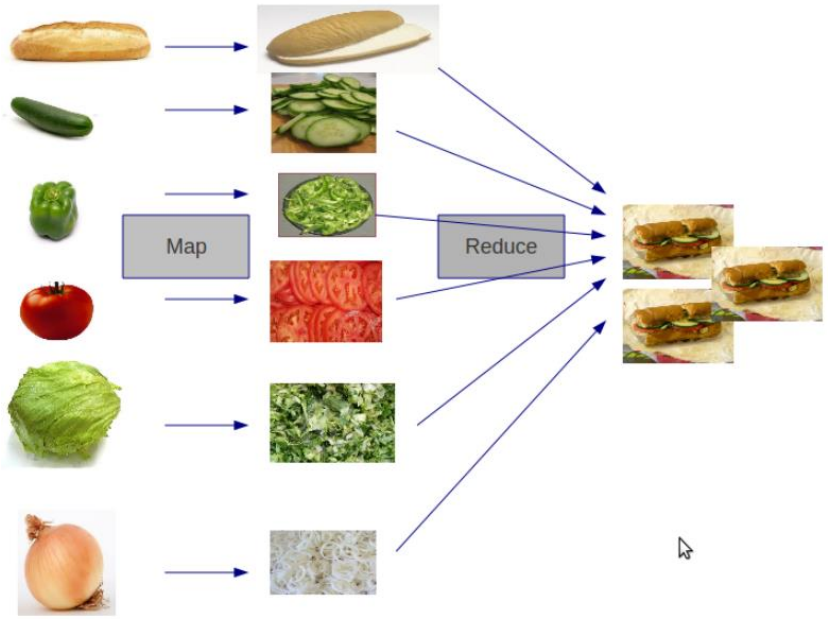
Map Reduce Fundamentals

Introduction to MapReduce

- Big Data batch processing queries are commonly based on a distributed processing mechanism called map/reduce (often written as MapReduce) that provides optimum performance across the servers in a cluster.
- Map and reduce are mathematical operations.
- A map operation applies a function to a list of data items, returning a list of transformed items.
- A reduce operation applies a combining function that takes multiple lists and recursively generates an output.

Introduction to MapReduce

- In a Big Data framework such as Hadoop, a map/reduce query uses two components, usually written in Java, which implement the algorithms that perform a two-stage data extraction and rollup process.
- The Map component runs on each data node server in the cluster extracting data that matches the query, and optionally applying some processing or transformation to the data to acquire the required result set from the files on that server.
- The Reduce component runs on one or more of the data node servers, and combines the results from all of the Map components into the final results set.



Introduction to MapReduce

Lets look at a simplified example of a map/reduce query:

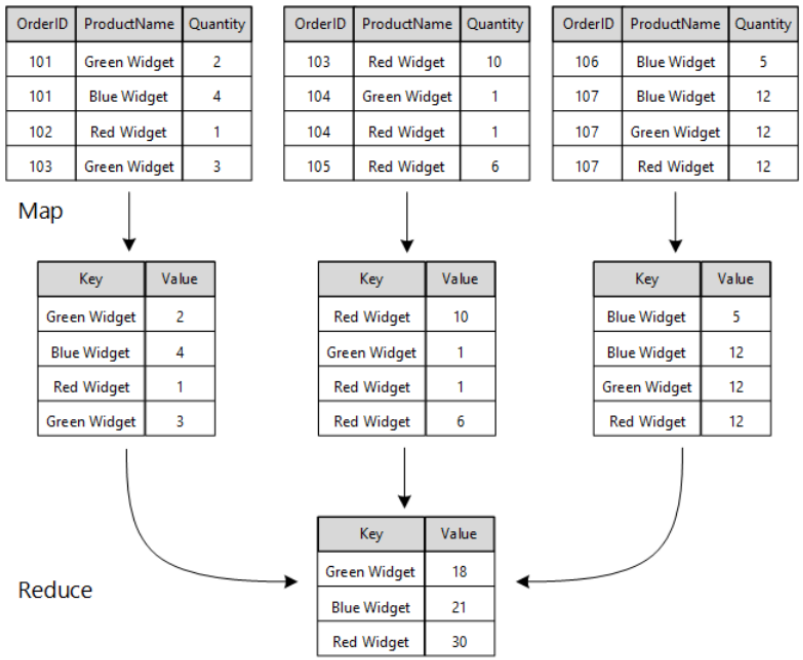
- We will assume that the input data contains a list of the detail lines from customer orders.
- Each detail line contains a reference (foreign key) that links it to the main order record, the name of the item ordered, and the quantity ordered.
- If this data was stored in a relational database using SQL, a query of the following form could be used to generate a summary of the total number of each item sold:

SQL

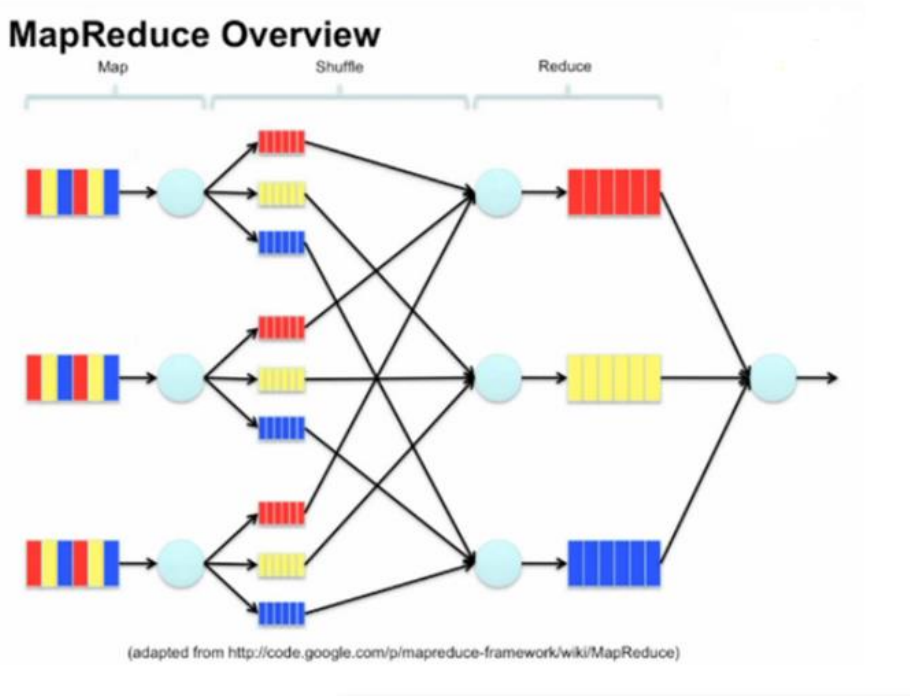
```
SELECT ProductName, SUM(Quantity) FROM OrderDetails GROUP BY ProductName
```

Introduction to MapReduce

- The equivalent using a Big Data solution requires a Map and a Reduce component.
- The Map component running on each node operates on a subset, or chunk, of the data.
- It transforms each order line into a name/value pair where the name is the product name, and the value is the quantity from that order line.
- Note that in this example the Map component does not sum the quantity for each product, it simply transforms the data into a list.



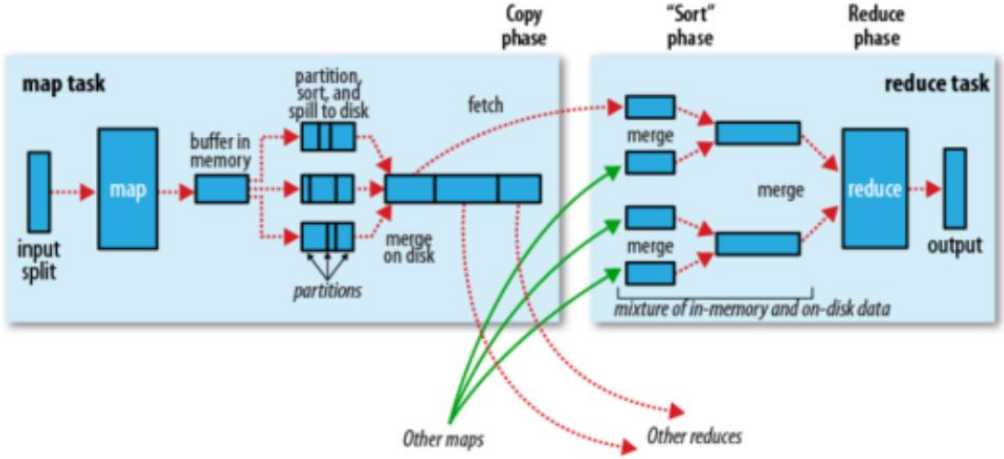
Introduction to MapReduce

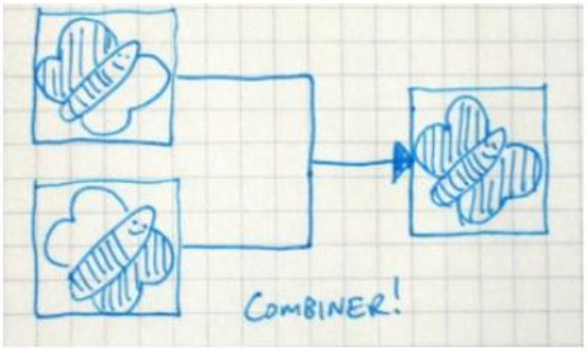


- Next, the framework shuffles and sorts all of the lists generated by the Map component instances into a single list.
- Then the routine executes the Reduce component with this list as the input.
- The Reduce component sums the totals for each product, and outputs the results.

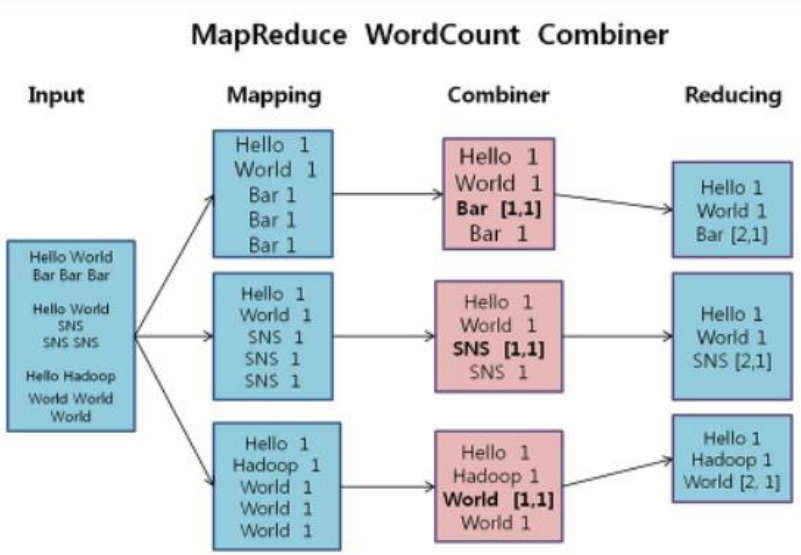
Introduction to MapReduce

- Depending on the configuration of the query job, there may be more than one Reduce component instance running.
- The output from each Map component instance is stored in a buffer on disk, and the component exits.
- The content of the buffer is then sorted, and passed to one or more Reduce component instances.
- Intermediate results are stored in the buffer until the final Reduce component instance combines them all.





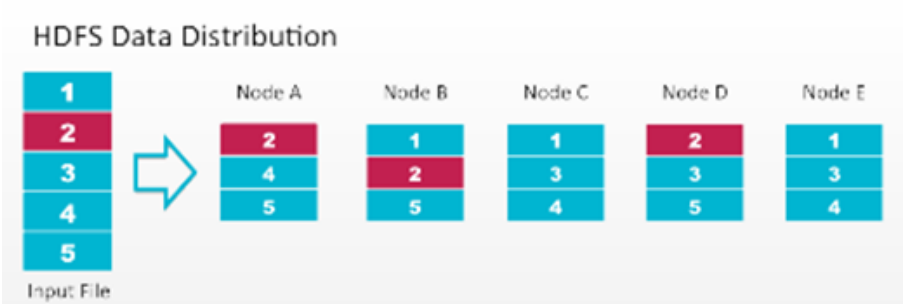
Introduction to MapReduce

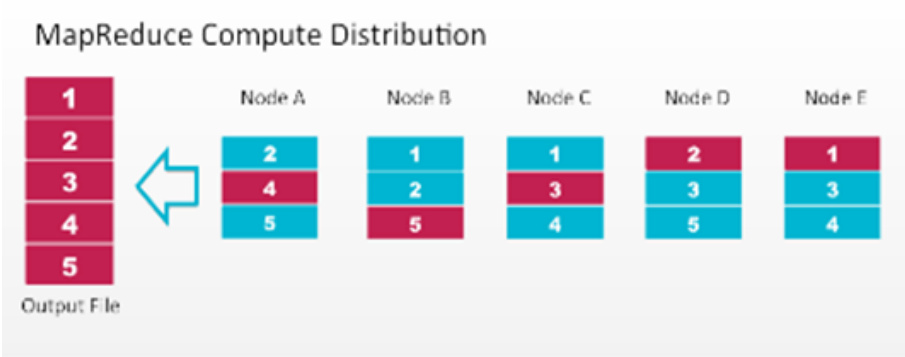


- In some cases the process might include an additional component called a Combiner that runs on each data node as part of the Map process, and performs a "reduce" type of operation on this part of the data each time the map process runs.
- It may also run as part of the reduce phase, and again when large datasets are being merged.
- In the example shown here, a Combiner could sum the values for each WordCount so that the output is smaller, which can reduce network load and memory requirements—with a subsequent increase in overall query efficiency.

Introduction to MapReduce

- Performing a map/reduce operation involves several stages such as partitioning the input data, reading and writing data, and shuffling and sorting the intermediate results.
- Some of these operations are quite complex.
- However, they are typically the same every time—irrespective of the actual data and the query.
- The great thing with a map/reduce framework such as Hadoop is that you usually need to create only the Map and Reduce components. The framework does the rest.





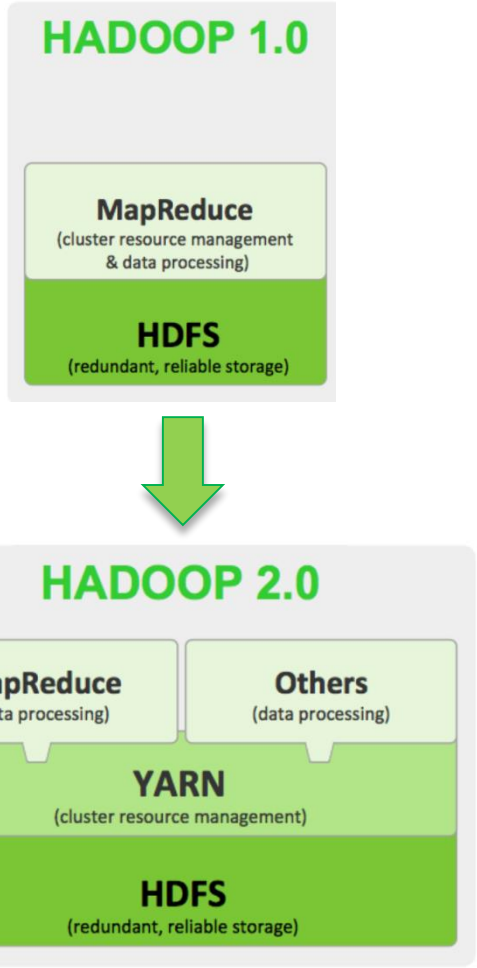
Introduction to MapReduce

- Although the core Hadoop engine requires the Map and Reduce components it executes to be written in Java, you can use other techniques to create them in the background without writing Java code.
- For example you can use tools named Hive and Pig that are included in most Big Data frameworks to write queries in a SQL-like or a high-level language.
- You can also use the Hadoop streaming API to execute components written in other languages.
- We will go through an example later in the presentation of executing MapReduce jobs from the R language.

Apache Hadoop - YARN

- The initial Hadoop framework has been refined into a new version sometimes referred to as Hadoop 2.0.
- Apache Hadoop YARN (in short for "Yet Another Resource Negotiator") is a cluster management technology and one of the key features in second-generation Hadoop.
- Originally described by Apache as a redesigned resource manager, YARN is now characterized as a large-scale, distributed operating system for Big Data applications.

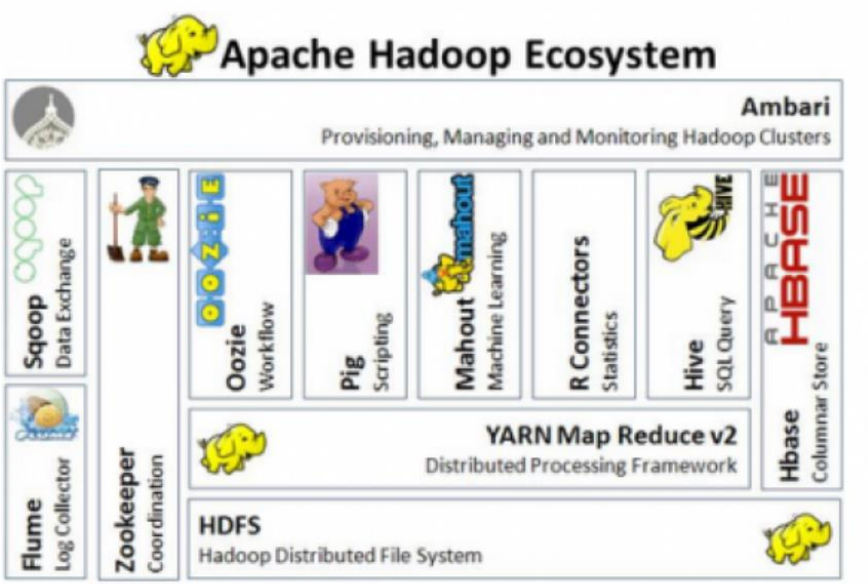
Apache Hadoop - YARN



- YARN is a software rewrite that decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications.
- For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs.
- The original incarnation of Hadoop closely paired the Hadoop Distributed File System (HDFS) with the batch-oriented MapReduce programming framework, which handles resource management and job scheduling on Hadoop systems and supports the parsing and condensing of data sets in parallel.

Apache Hadoop - YARN

- In 2012, YARN became a sub-project of the larger Apache Hadoop project.
- YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes.
- Separating HDFS from MapReduce with YARN makes the Hadoop environment more suitable for operational applications that can't wait for batch jobs to finish.

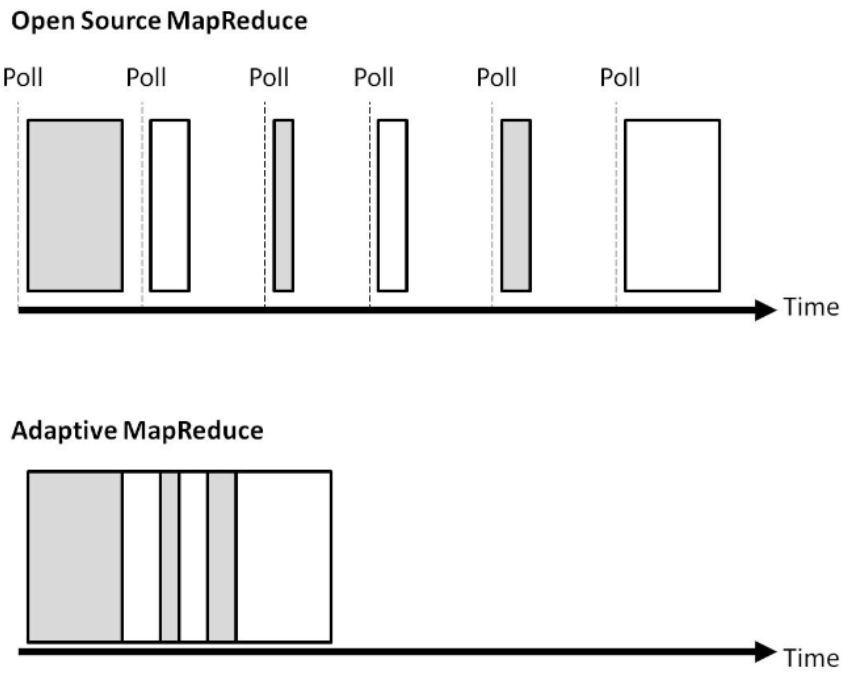


Closing Thoughts on Hadoop

- Hadoop has always been a catch-all for disparate open source initiatives that combine for a more or less unified Big Data architecture.
- Some would claim that Hadoop has always been, at its very heart, simply a distributed file system (HDFS), but the range of HDFS-alternative databases, including HBase and Cassandra, undermines that assertion.
- Until recently, Hadoop has been, down deep, a specific job-execution layer -- MapReduce -- that executes on one or more alternative, massively parallel data-persistence layers, one of which happens to be HDFS.

Closing Thoughts on Hadoop

- But the recent introduction of the next-generation execution layer for Hadoop -- YARN -- eliminates the strict dependency of Hadoop environments on MapReduce.
- Just as critical, YARN eliminates a job-execution bottleneck that has bedeviled MapReduce from the start: the fact that all MapReduce jobs (pre-YARN) have had to run as batch processes through a single daemon (JobTracker), a constraint that limits scalability and dampens processing speed.
- These MapReduce constraints have spurred many vendors to implement their own speedups, such as IBM's Adaptive MapReduce, to get around the bottleneck of native MapReduce.



Closing Thoughts on Hadoop

- All of this might make one wonder what, specifically, "Hadoop" means anymore, in terms of an identifiable "stack" distinct from other Big Data and analytics platforms and tools.
- That's a definitional quibble -- YARN is a foundational component of the evolving Big Data mosaic.
- YARN puts traditional Hadoop into a larger context of compostable, fit-to-purpose platforms for processing the full gamut of data management, analytics, and transactional computing jobs.
- Though it retains backward compatibility with the MapReduce API and continues to execute MapReduce jobs, a YARN engine is capable of executing a wide range of jobs developed in other languages.

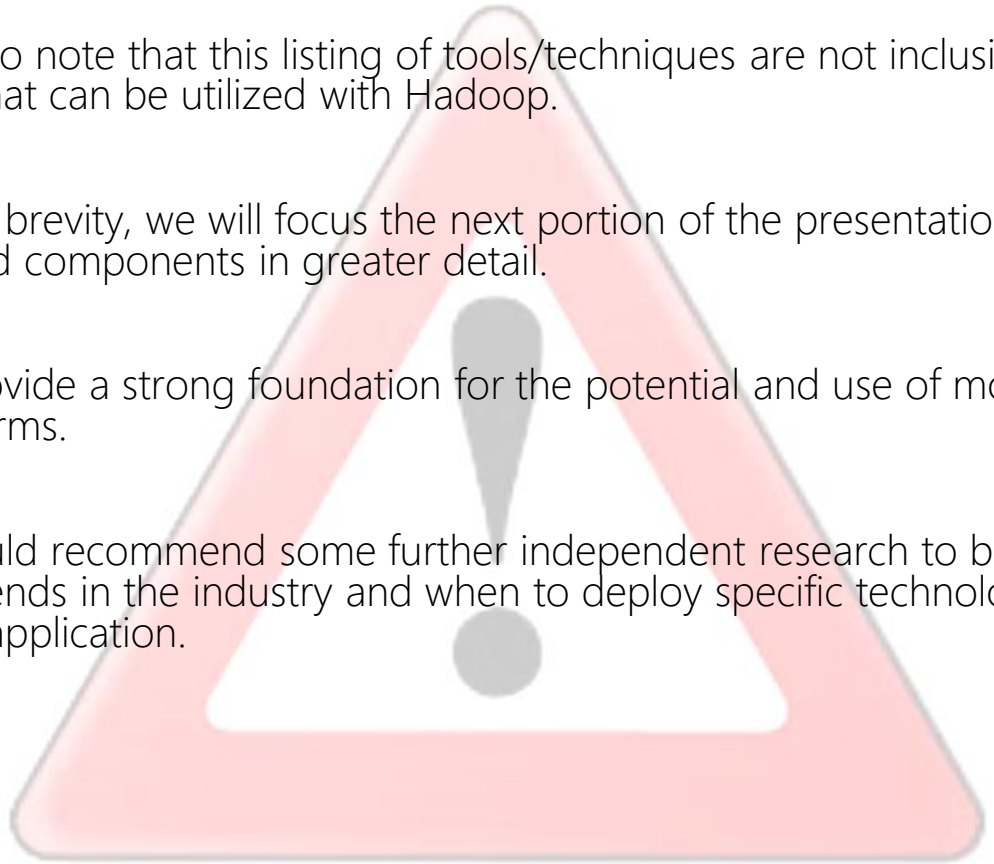
Understanding How Big Data Solutions Work (Leveraging Hadoop 2.0)

Leveraging Hadoop 2.0

- Map/Reduce is just one application that you can run on a Hadoop cluster. Several query, management, and other types of tools that can be used with Hadoop 2.0 (YARN) are available or under development.

- Examples from the Apache Project include:
  - HBase - A scalable, distributed database that supports structured data storage for large tables.
  - Hive - A data warehouse infrastructure that provides data summarization and ad hoc querying.
  - Pig - A high-level data-flow language and execution framework for parallel computation.
  - Mahout - A scalable machine learning and data mining library.
  - Kafka - A low-latency message broker platform for handling real-time data feeds.
  - ZooKeeper - A high-performance coordination service for distributed applications.
  - Storm - A distributed real-time computation system for processing fast, large streams of data.

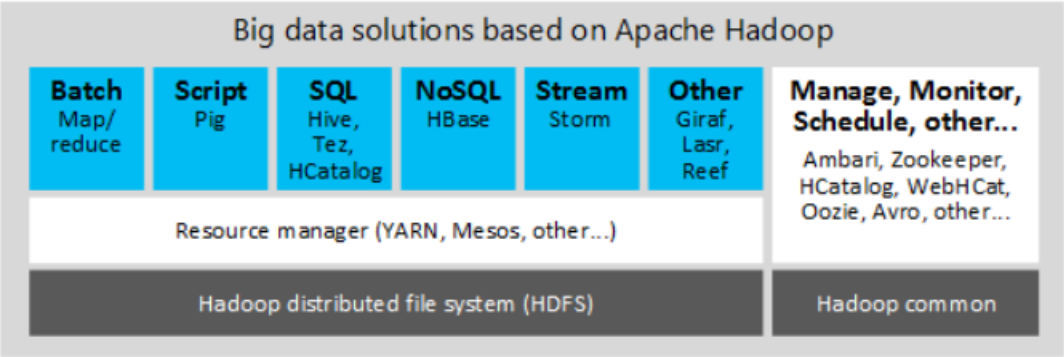
Leveraging Hadoop 2.0



- It is important to note that this listing of tools/techniques are not inclusive of all the technologies that can be utilized with Hadoop.
- For the sake of brevity, we will focus the next portion of the presentation going over the previously listed components in greater detail.
- This should provide a strong foundation for the potential and use of modern "Big Data" analytics platforms.
- However, I would recommend some further independent research to better understand the growing trends in the industry and when to deploy specific technologies that suits your Big Data application.

Leveraging Hadoop 2.0

- Here is an overview of a typical Hadoop-based Big Data mechanism.

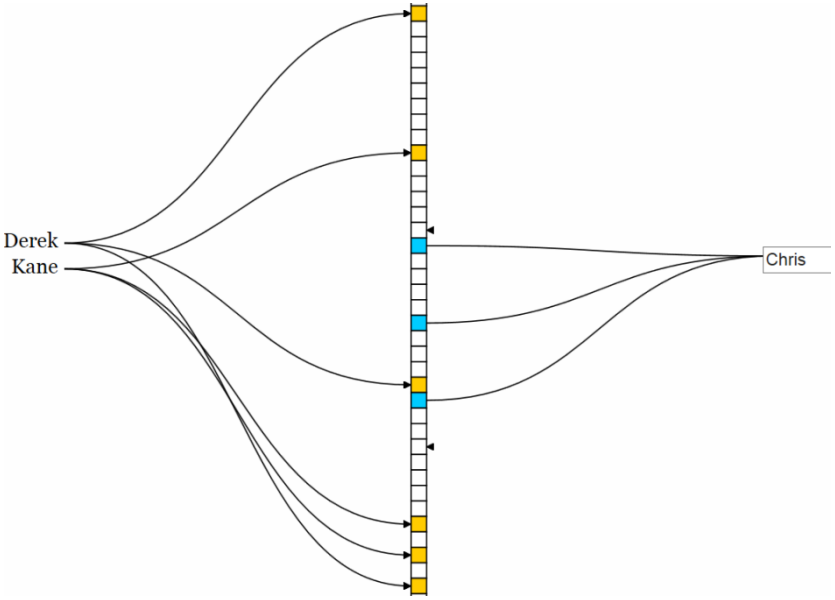


- These are the main assets of Apache Hadoop (version 2 onwards).
- We'll be exploring and using some of these components in the scenarios described in subsequent sections of this guide. For more information about all of them, see the Apache Hadoop website.

Apache HBase

- The first component of the Big Data solution we will be providing an overview on within this lecture is related to an open source, non-relational, distributed database modeled after Google's BigTable.
- This is NoSQL platform is called HBase and is developed as part of Apache Software Foundation's Apache Hadoop project.
- The HBase platform is a column oriented system which runs on top of HDFS (Hadoop Distributed Filesystem), provides real-time read/write access to those large datasets.
- With YARN as the architectural center of Apache Hadoop, multiple data access engines such as Apache HBase interact with data stored in the cluster.

Apache HBase



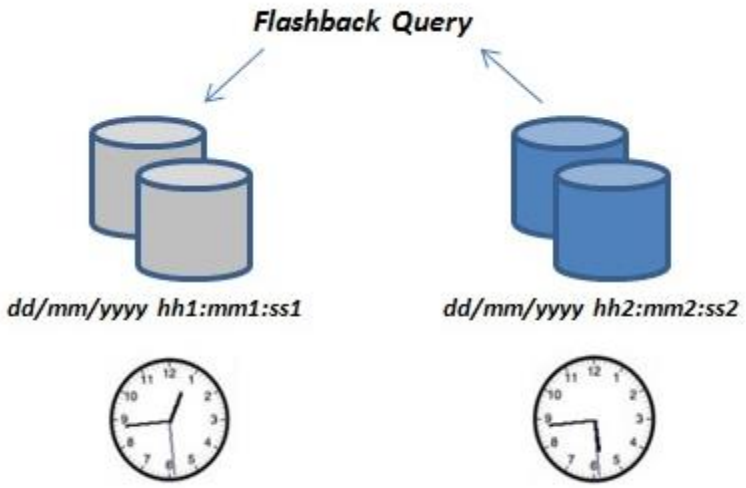
A Bloom Filter Example

- Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API but also through REST, Avro or Thrift gateway APIs.
- HBase features compression, in-memory operation, and Bloom filters on a per-column basis as outlined in the original BigTable paper.
- A Bloom filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set.
- If the element is not in the bloom filter, then we know for sure we don't need to perform the expensive lookup. On the other hand, if it is in the bloom filter, we perform the lookup, and we can expect it to fail some proportion of the time (the false positive rate).

Apache HBase

- Apache HBase provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data.
- An example of this includes finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).
- HBase is not a direct replacement for a classic SQL database, although recently its performance has improved, and it is now serving several data-driven websites, including Facebook's Messaging Platform.

Apache HBase



- Apache HBase scales linearly to handle huge data sets with billions of rows and millions of columns, and it easily combines data sources that use a wide variety of different structures and schemas.
- Apache HBase provides random, real time access to your data in Hadoop.
- It was created for hosting very large tables, making it a great choice to store multi-structured or sparse data.
- Users can query HBase for a particular point in time, making "flashback" queries possible.

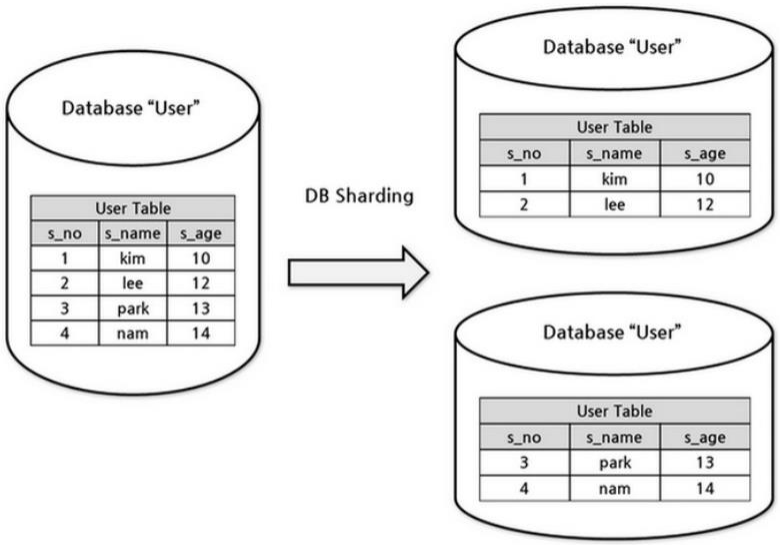
Apache HBase

- These following examples make HBase a great choice for storing semi-structured data like log data and then providing that data very quickly to users or applications integrated with HBase.
- One company that provides web security services maintains a system accepting billions of event traces and activity logs from its customer' desktops every day.
- The company's programmers can tightly integrate their security solutions with HBase (to assure that the protection they provide keeps pace with real-time changes in the threat landscape.)

Apache HBase

- Another company provides stock market ticker plant data that its users query more than thirty thousand times per second, with an SLA of only a few milliseconds.
- Apache HBase provides that super low-latency access over an enormous, rapidly changing data store.
- Enterprises use Apache HBase's low latency storage for scenarios that require real-time analysis and tabular data for end user applications.

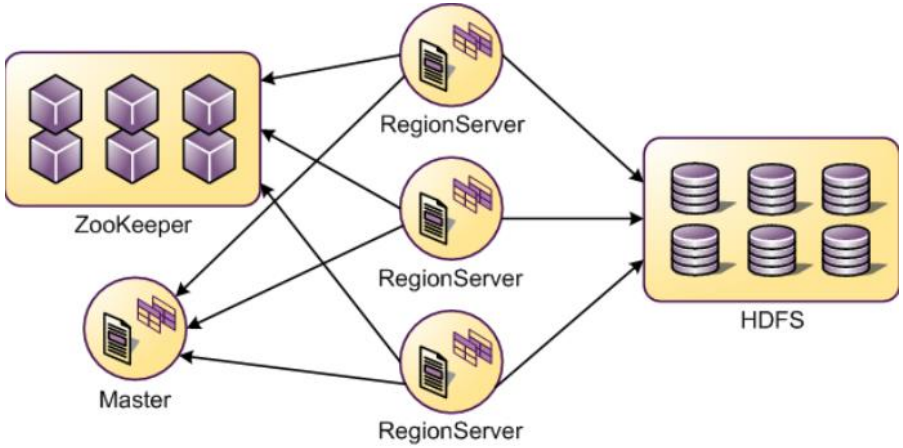
Apache HBase



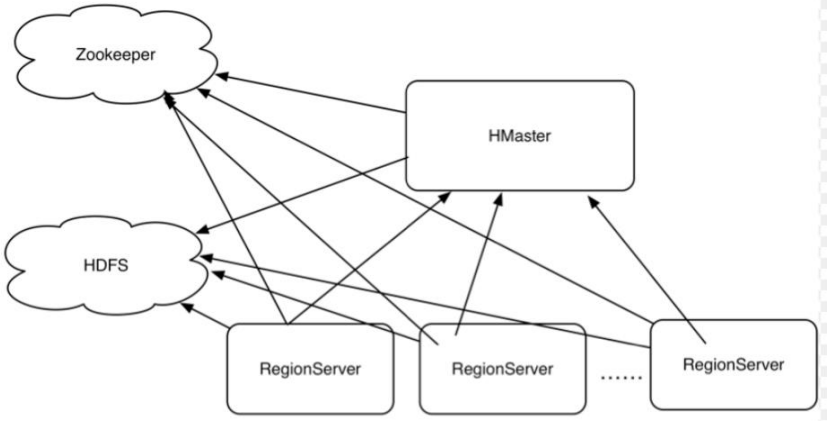
- HBase scales linearly by requiring all tables to have a primary key.
- The key space is divided into sequential blocks that are then allotted to a region.
- RegionServers own one or more regions, so the load is spread uniformly across the cluster.
- If the keys within a region are frequently accessed, HBase can further subdivide the region by splitting it automatically, so that manual data sharding is not necessary.

Apache HBase

- ZooKeeper and HMaster servers make information about the cluster topology available to clients. (More on ZooKeeper later)
- Clients connect to these and download a list of RegionServers, the regions contained within those RegionServers and the key ranges hosted by the regions.
- Clients know exactly where any piece of data is in HBase and can contact the RegionServer directly without any need for a central coordinator.
- RegionServers include a memstore to cache frequently accessed rows in memory.



Apache HBase



Apache HBase provides high data availability in several ways:

- Highly available cluster topology information through production deployments with multiple HMaster and ZooKeeper instances.
- Data distribution across many nodes means that loss of a single node only affects data stored on that node.
- HBase allows data storage, ensuring that loss of a single node does not result in loss of data availability.
- HFile file format stores data directly in HDFS. HFile can be read or written to by Apache Hive, Apache Pig, and MapReduce permitting deep analytics on HBase without data movement.

- For further research on Apache HBase concepts and techniques, please refer back to Apache Software Foundation's User Forums.

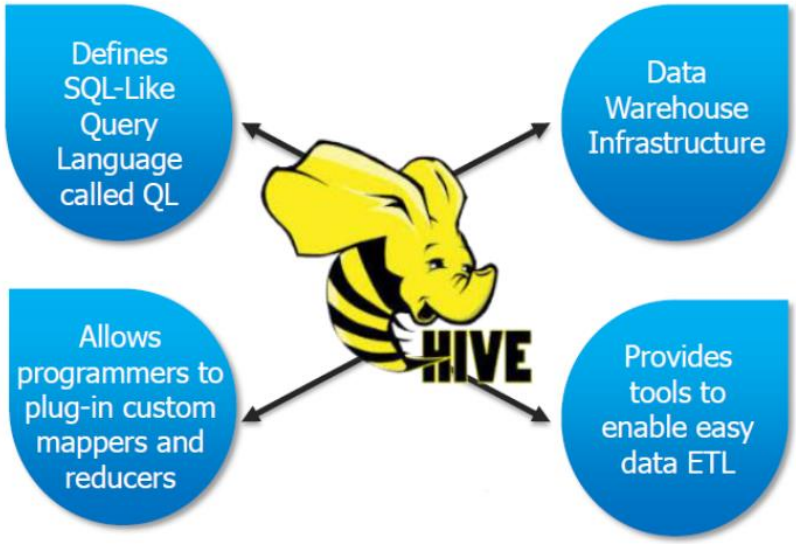
Apache Hive

- The second component of the Big Data solution we will be providing an overview on within this lecture is more related to conventional DW techniques but in a Big Data environment.

- The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage.

Built on top of Apache Hadoop, it provides:

- Tools to enable easy data extract/transform/load (ETL).
- A mechanism to impose structure on a variety of data formats.
- Access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase.
- Query execution via MapReduce.



Apache Hive


```
--Number of Accesses per Unique IP
SELECT ip, COUNT(1) FROM www_access \
  GROUP BY ip LIMIT 30;

--Unique IPs Sorted by Number of Accesses
SELECT ip, COUNT(1) AS cnt FROM www_access \
  GROUP BY ip
  ORDER BY cnt DESC LIMIT 30;

--Number of Accesses After a Certain Time
SELECT COUNT(1) FROM www_access \
  WHERE TD_TIME_RANGE(time, "2011-08-19", NULL, "PDT")

--Number of Accesses Each Day
SELECT \
  TD_TIME_FORMAT(time, "yyyy-MM-dd", "PDT") AS day, \
  COUNT(1) AS cnt \
FROM www_access \
GROUP BY TD_TIME_FORMAT(time, "yyyy-MM-dd", "PDT")
```

- Hive defines a simple SQL-like query language, called QL (HiveQL), that enables users familiar with SQL to query the data.
- At the same time, this language also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers.
- This allows the user to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language.
- QL can also be extended with custom scalar functions (UDF's), aggregations (UDAF's), and table functions (UDTF's).

Apache Hive

- Hive does not mandate read or written data be in the "Hive format" — there is no such thing.

Hive SQL Datatypes	Hive SQL Semantics
INT	SELECT, LOAD, INSERT from query
TINYINT/SMALLINT/BIGINT	Expressions in WHERE and HAVING
BOOLEAN	GROUP BY, ORDER BY, SORT BY
FLOAT	Sub-queries in FROM clause
DOUBLE	GROUP BY, ORDER BY
STRING	CLUSTER BY, DISTRIBUTE BY
TIMESTAMP	ROLLUP and CUBE
BINARY	UNION
ARRAY, MAP, STRUCT, UNION	LEFT, RIGHT and FULL INNER/OUTER JOIN
DECIMAL	CROSS JOIN, LEFT SEMI JOIN
CHAR	Windowing functions (OVER, RANK, etc.)
VARCHAR	INTERSECT, EXCEPT, UNION DISTINCT
DATE	Sub-queries in WHERE (IN/NOT IN, EXISTS/NOT EXISTS
	Sub-queries in HAVING

Hive 0.10
Hive 0.11
Future

- Hive works equally well on Thrift, control delimited, or specialized data formats.

Apache Hive

- Hive is not designed for OLTP workloads and does not offer real-time queries or row-level updates.
- It is best used for batch jobs over large sets of append-only data (like web logs).
- What Hive values most are scalability (scale out with more machines added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and UDF/UDAF/UDTF), fault-tolerance, and loose-coupling with its input formats.

Apache Hive

- There are a couple of sub components of Hive which help aid in the data processing.

- These components are called HCatalog and WebHCat.
  - Hcatalog: a table and storage management layer for Hadoop that enables users with different data processing tools — including Pig and MapReduce — to more easily read and write data on the grid.
  - WebHCat: Provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs or perform Hive metadata operations using an HTTP (REST style) interface.

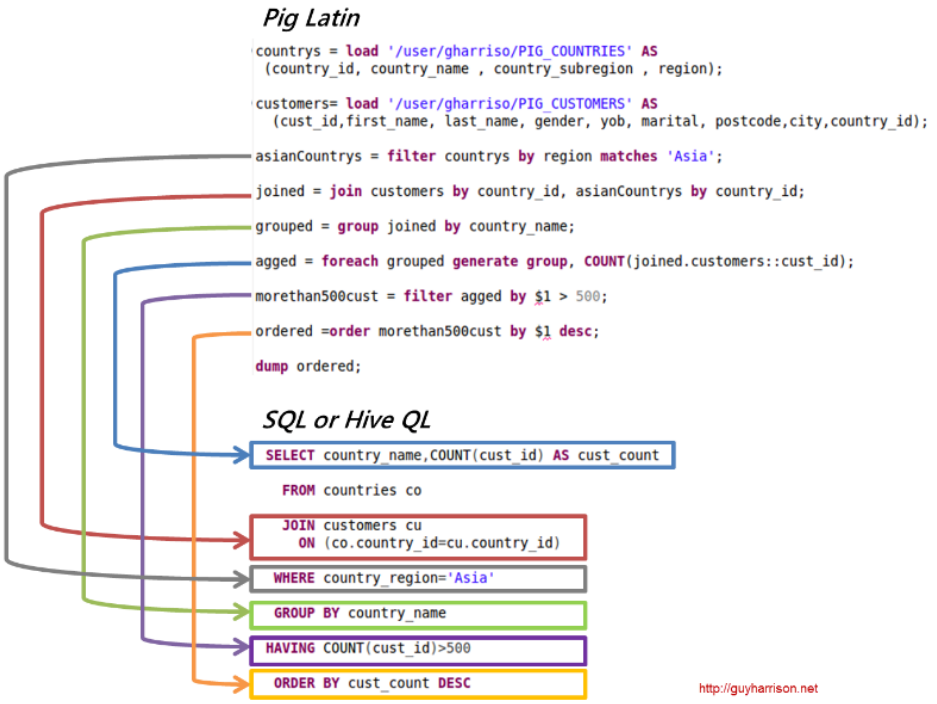
- For further research on Apache Hive concepts and techniques, please refer back to Apache Software Foundation's User Forums.

Apache Pig

- Apache Pig is a platform for analyzing large data sets that consists of a high-level scripting language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.
- At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject).

Apache Pig

- Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:
  - **Ease of programming:** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks.
  - Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
  - **Optimization opportunities:** The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
  - **Extensibility:** Users can create their own functions to do special-purpose processing.



Apache Pig

- Pig Latin is a simple query algebra that lets you express data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. Users can create their own functions to do special-purpose processing.

- For example:

```
$ pig                                          <-- Start Grunt with default MapReduce mode
grunt> cat /training/playArea/pig/a.txt
a       1                                      <-- Grunt supports file system commands
d       4
c       9                                      <-- Load contents of text files into a Bag named records
k       6
grunt> records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
grunt> dump records;                           <-- Display records bag to the screen
...
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 50% complete
2012-07-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 100% complete
...
(a,1)
(d,4)                                          <-- Results of the bag named records are printed to the screen
(c,9)
(k,6)
grunt>
```

Apache Pig

- Here is another example of a word count script:

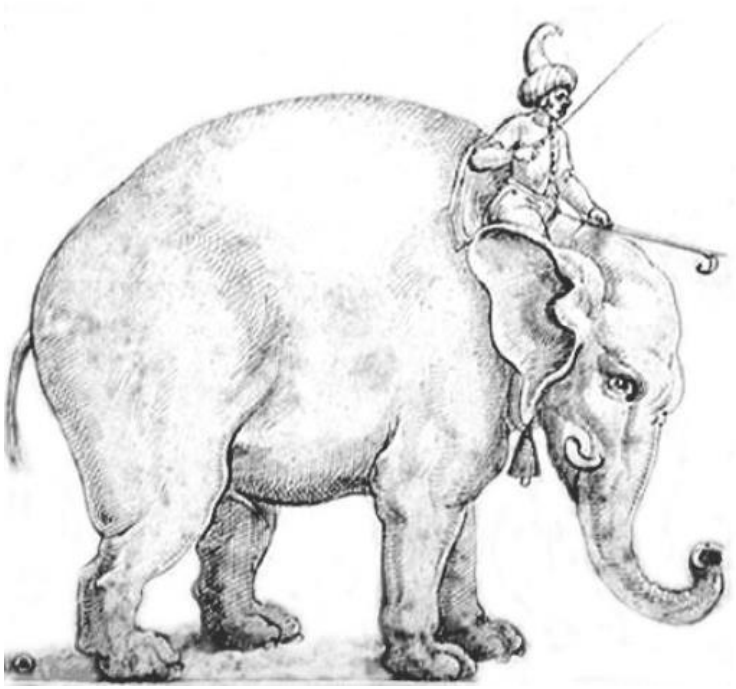
```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS
word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

NETFLIX

* http://en.wikipedia.org/wiki/Pig_(programming_tool)#Example

Apache Mahout

- The Apache Mahout project's goal is to build an environment for quickly creating scalable performance machine learning applications.
- The name Mahout was original chosen for it's association with the Apache Hadoop project.
- A Mahout is a person who drives an elephant (Hadoop's logo).
- The Apache Community wanted a name that complemented Hadoop but the project managers see the project as a good driver of Hadoop in the sense that they will be using and testing it.



Apache Mahout

- Here are some notable machine learning techniques that is currently supported in the Mahout Framework:

Classification	Clustering
Logistic Regression	k-Means Clustering
Naive Bayes	Fuzzy k-Means
Random Forest	Streaming k-Means
Hidden Markov Models	Spectral Clustering
Multilayer Perceptron	

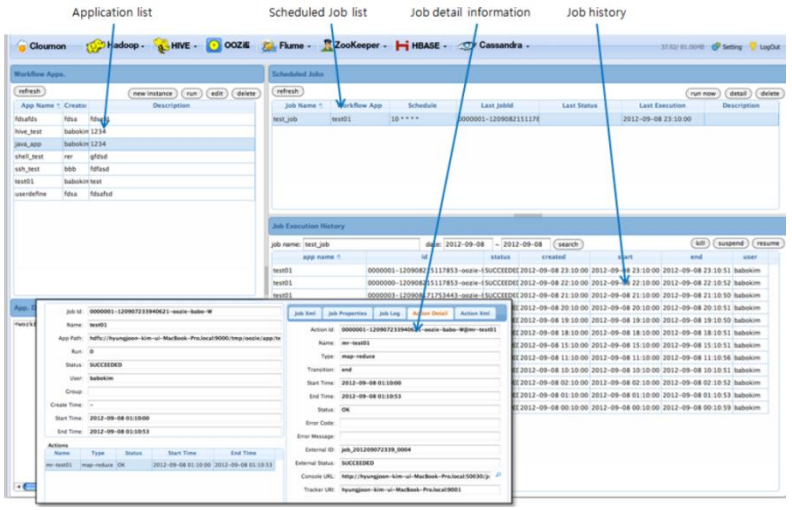
- The strength of Mahout is that it is designed to integrate seamlessly within the YARN Framework.
- However, the depth of Machine Learning and Statistical Algorithms is not on the same level as R or Python.
- For our purposes, it is important to know of the Apache Mahout project and its capabilities even though this might not be the framework that we draw from for machine learning.

Apache Mahout

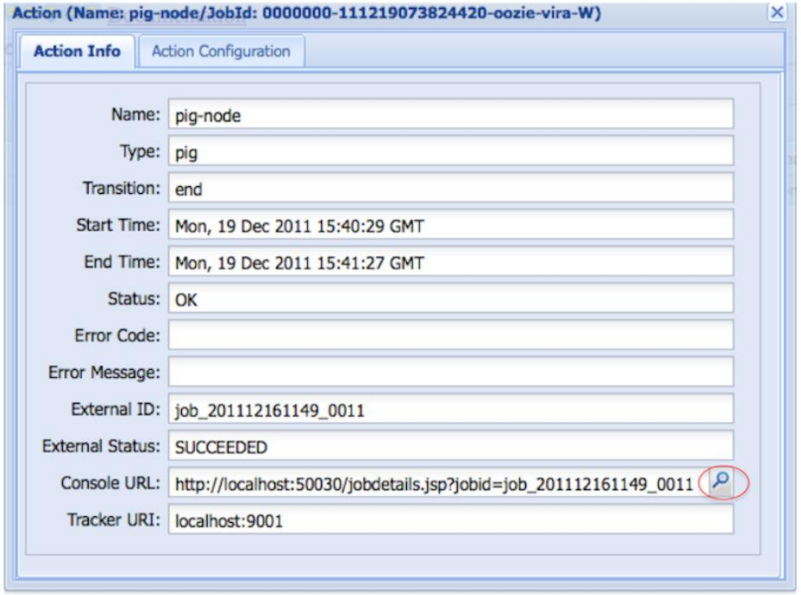
- The project's community has decided to rework Mahout to support the increasingly popular Apache Spark in-memory data-processing framework, as well as the H2O engine for running machine learning and mathematical workloads at scale.
- While data processing in Hadoop has traditionally been done using MapReduce, the batch-oriented framework has fallen out of vogue as users began demanding lower-latency processing for certain types of workloads — such as machine learning.
- However, nobody really wants to abandon Hadoop entirely because it's still great for storing lots of data and many still use MapReduce for most of their workloads.

Apache Oozie

- Oozie is a workflow scheduler system to manage Apache Hadoop jobs.
- Oozie combines multiple jobs sequentially into one logical unit of work.
- It is integrated with the Hadoop stack, with YARN as its architectural center, and supports Hadoop jobs for Apache MapReduce, Apache Pig, Apache Hive, and Apache Sqoop.
- Oozie is a scalable, reliable and extensible system.



Apache Oozie



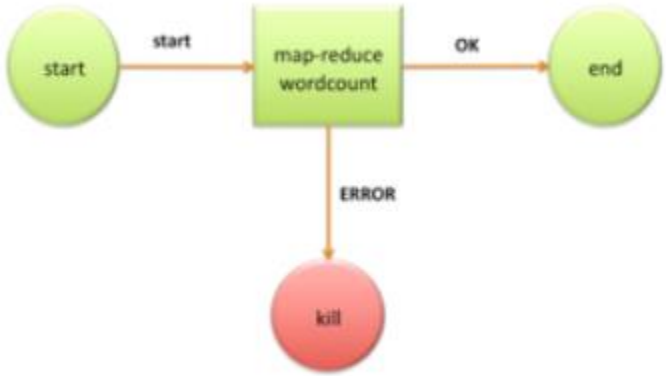
- Apache Oozie is a tool for Hadoop operations that allows cluster administrators to build complex data transformations out of multiple component tasks.
- Oozie triggers workflow actions, but Hadoop MapReduce executes them. This allows Oozie to leverage other capabilities within the Hadoop stack to balance loads and handle failures.
- This provides greater control over jobs and also makes it easier to repeat those jobs at predetermined intervals.
- At its core, Oozie helps administrators derive more value from Hadoop.

Apache Oozie

- Oozie Bundle provides a way to package multiple coordinator and workflow jobs and to manage the lifecycle of those jobs.

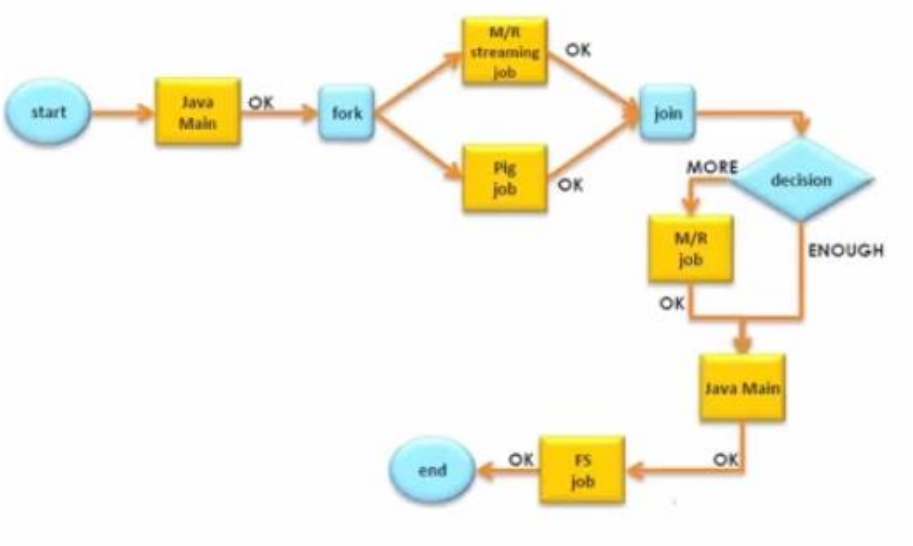
There are two basic types of Oozie jobs:

- Oozie Workflow Jobs: Directed Acyclical Graphs (DAGs), specifying a sequence of actions to execute. The Workflow job has to wait.
- Oozie Coordinator Jobs: Recurrent Oozie Workflow jobs that are triggered by time and data availability.



MapReduce Workflow DAG

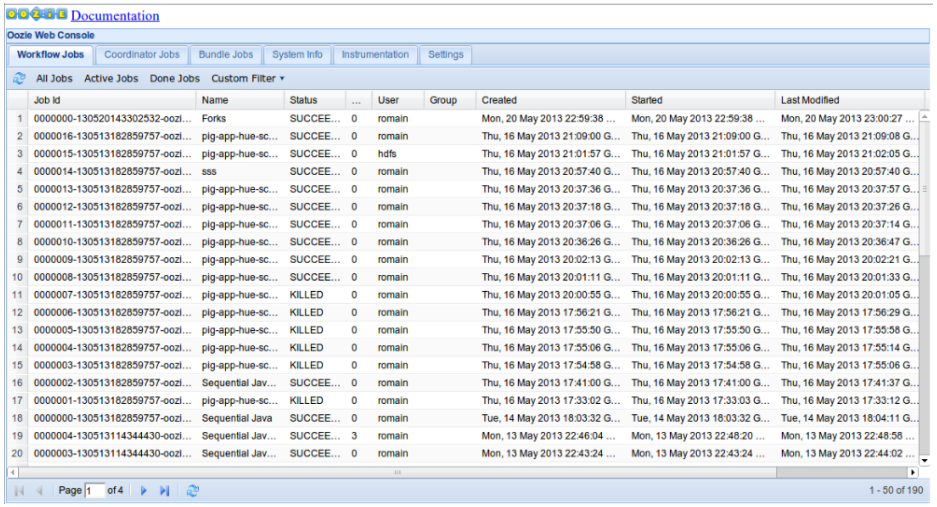
Apache Oozie



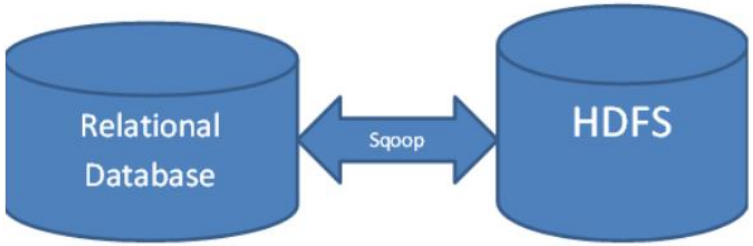
- An Oozie Workflow is a collection of actions arranged in a Directed Acyclic Graph (DAG).
- Control nodes define job chronology, setting rules for beginning and ending a workflow.
- In this way, Oozie controls the workflow execution path with decision, fork and join nodes.
- Action nodes trigger the execution of tasks.
- Oozie Coordinator can also manage multiple workflows that are dependent on the outcome of subsequent workflows.
- The outputs of subsequent workflows become the input to the next workflow. This chain is called a “data application pipeline”.

Apache Oozie

- Oozie detects completion of tasks through callback and polling.
- When Oozie starts a task, it provides a unique callback HTTP URL to the task, thereby notifying that URL when it's complete.
- If the task fails to invoke the callback URL, Oozie can poll the task for completion.
- Often it is necessary to run Oozie workflows on regular time intervals, but in coordination with unpredictable levels of data availability or events.
- In these circumstances, Oozie Coordinator allows you to model workflow execution triggers in the form of the data, time or event predicates.
- The workflow job is started after those predicates are satisfied.



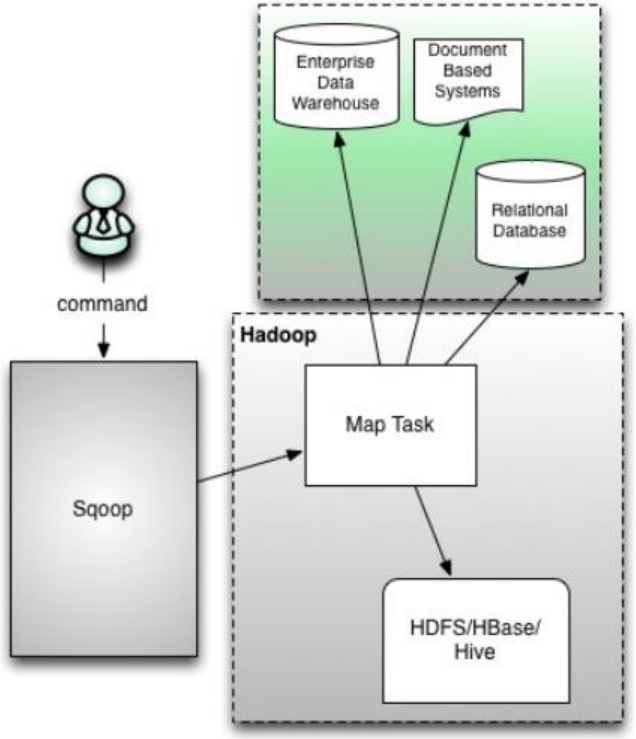
Apache Sqoop



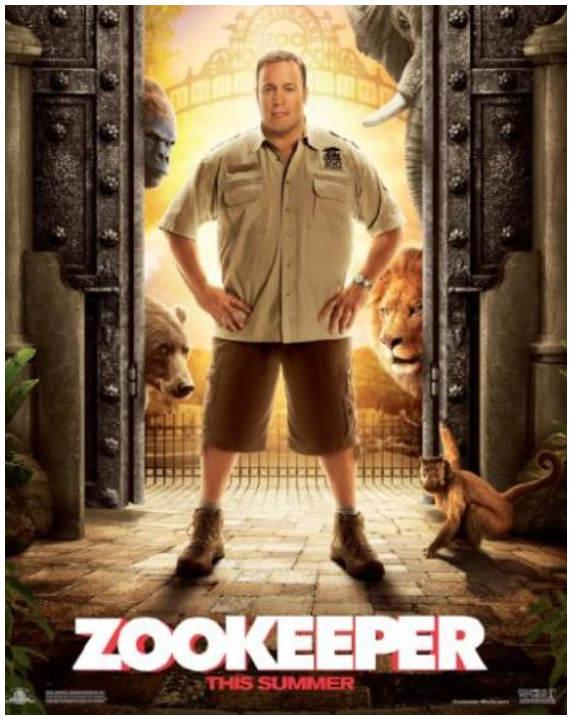
- Apache Sqoop efficiently transfers bulk data between Apache Hadoop and structured datastores such as relational databases.
- Sqoop helps offload certain tasks (such as ETL processing) from the EDW to Hadoop for efficient execution at a much lower cost.
- Sqoop can also be used to extract data from Hadoop and export it into external structured datastores.
- Sqoop works with relational databases such as Teradata, Netezza, Oracle, MySQL, Postgres, and HSQLDB.

Apache Sqoop

- Sqoop provides a pluggable mechanism for optimal connectivity to external systems.
- The Sqoop extension API provides a convenient framework for building new connectors which can be dropped into Sqoop installations to provide connectivity to various systems.
- Sqoop itself comes bundled with various connectors that can be used for popular database and data warehousing systems.
- This is a great mechanism to utilize when exploring how to bridge data analytics projects between conventional DW structures into the Hadoop/HDFS approach and vice versa.



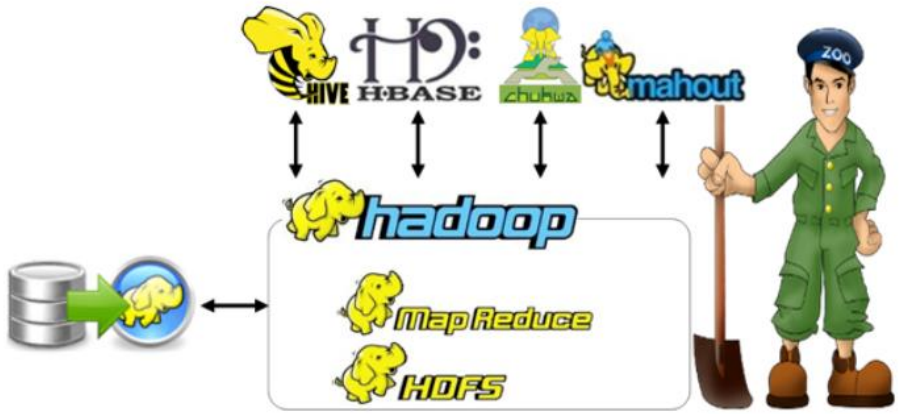
Apache ZooKeeper



- Apache ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- All of these kinds of services are used in some form or another by distributed applications.
- Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.
- Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage.

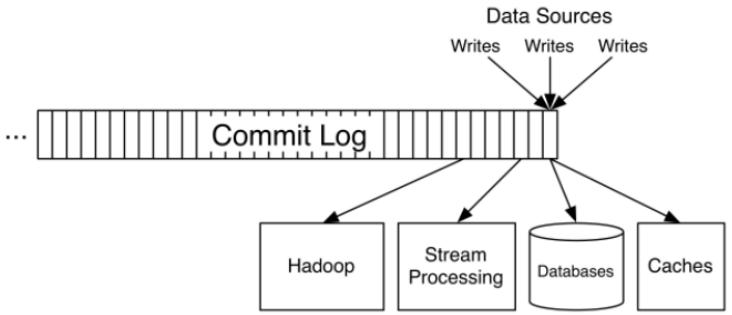
Apache ZooKeeper

- Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.
- ZooKeeper's architecture supports high availability through redundant services.
- The clients can thus ask another ZooKeeper leader if the first fails to answer.
- ZooKeeper nodes store their data in a hierarchical name space, much like a file system or a tree data structure.
- ZooKeeper is used by companies including Rackspace, Yahoo!, and eBay.

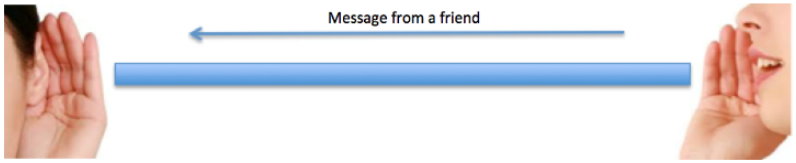


Apache Kafka

- Apache Kafka is an open-source message broker project developed by the Apache Software Foundation.
- The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.
- The design is heavily influenced by transaction logs.
- Kafka is often categorized as a messaging system, and it serves a similar role, but provides a fundamentally different abstraction.
- The key abstraction in Kafka is a structured commit log of updates.



Apache Kafka



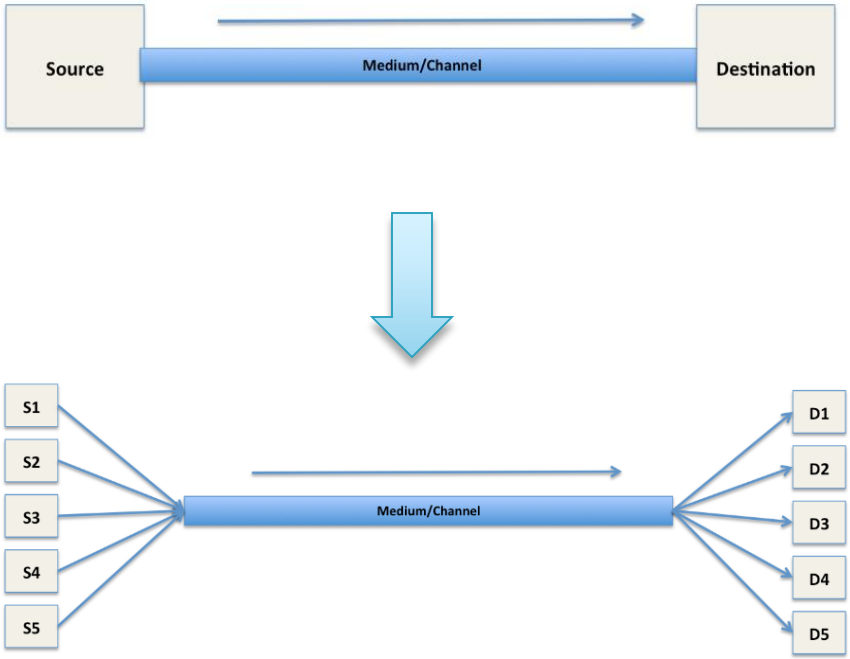
Before we dive deeper into Kafka, lets take the following example:

- A message in its literal sense would be some information that needs to go from a source to a destination.
- When this information is sent from the source and is received by the destination we state that the message has been delivered.
- For example, let us consider you and your friend use a long hollow pipe to talk to each other. When your friend speaks into it from one end, you would keep your ears to the other end to hear what he just said.
- In this way you will receive the messages he would have to convey.

Apache Kafka

- In its simplest form a message is nothing but data.
- A system that can handle transmission of data from a source to a destination is called a messaging system.
- In the computing world, the source could be machine that is generating data (for e.g. a Web Server generates large number of logs) or a human user.
- In most cases you will see the volumes generated by systems and machines to be way larger than the ones generated by human beings.

Apache Kafka



- In a messaging system you would want to ensure that the message is delivered to the target.
- Even if the target is not ready to receive it, there has to be a provision to hold it.
- To achieve this, the messaging system should provide a way of retaining the message in the medium of communication until the target is ready to receive it.
- However, in reality, the scenarios could be such that the messages may have to be delivered to more than one destination.

Apache Kafka

- With this understanding, lets give some technical labels to the different components in a messaging system.

- Publisher: The source or the different sources create messages.
- Subscriber: The destinations/targets are the one who read the message, in other words, they subscribe to the messages generated by the publishers.

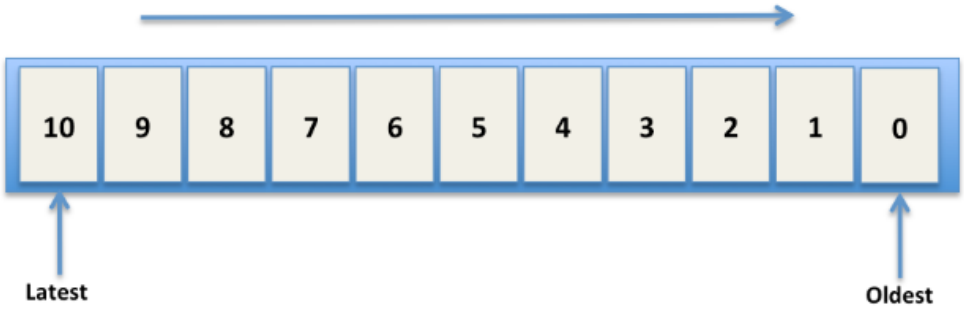
- There is one more component that we still need to mention. So far, we have been referring to it as the channel/medium to which publishers write data and from where subscribers read data.

- That is not an entirely correct way of referring to it. A more appropriate way to call it would be to refer to it as – the log.

Apache Kafka

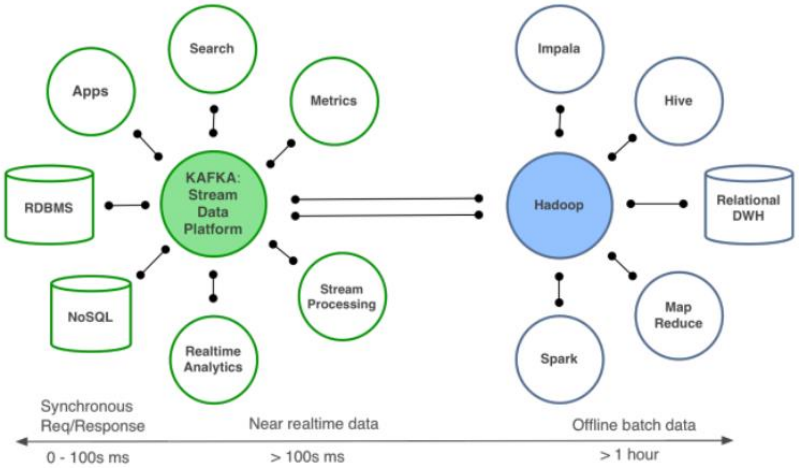
- Logs, in a general sense stand for a record of “what happened”. In our context the message can be classified as an event and the log is what stores that event.

- Publishers publish events to the log and the subscribers read these events from the log to know “what happened”. Following is a simple illustration of how a log appears.



- As you can see above, the log is a sequence of numbered messages that are append only and ordered by time. The log, in the database world, is often referred to as a commit log.

Apache Kafka

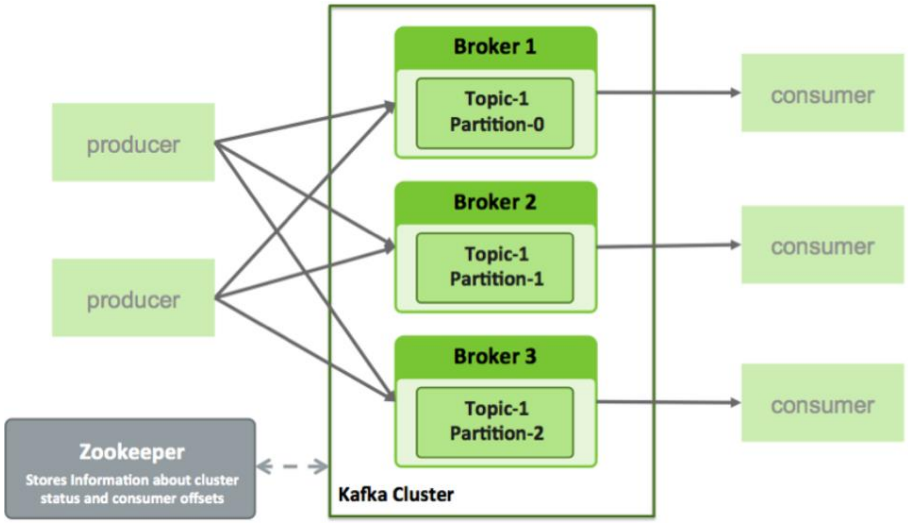


- A producer of data sends a stream of records which are appended to this log, and any number of consumers can continually stream these updates off the tail of the log with millisecond latency.
- Each of these data consumers has its own position in the log and advances independently.
- This allows a reliable, ordered stream of updates to be distributed to each consumer.

Apache Kafka

- The log can be sharded and spread over a cluster of machines, and each shard is replicated for fault-tolerance.
- This gives a model for parallel, ordered consumption which is key to Kafka's use as a change capture system for database updates (which must be delivered in order).

Apache Kafka

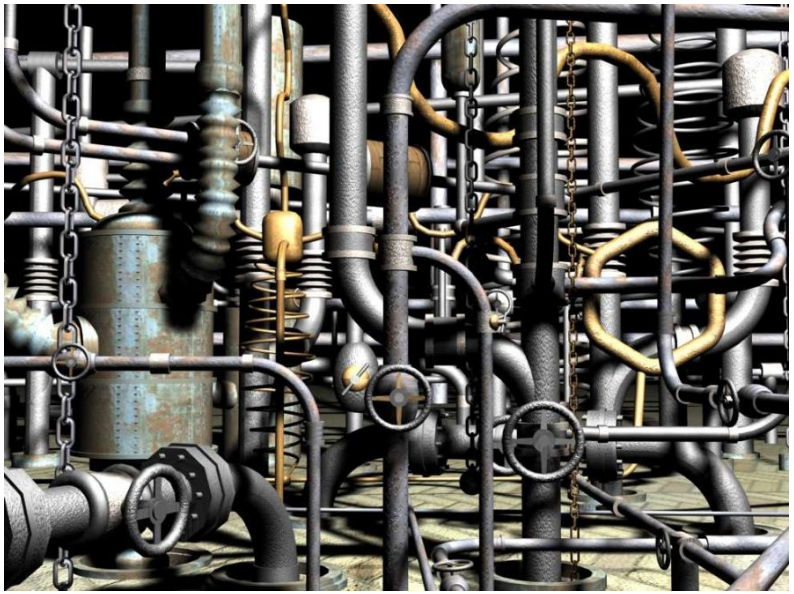


- Kafka is built as a modern distributed system.
- Data is replicated and partitioned over a cluster of machines that can grow and shrink transparently to the applications using the cluster.
- Consumers of data can be scaled out over a pool of machines as well and automatically adapt to failures in the consuming processes.

Apache Kafka

- The initial adoption is usually for a single particularly large-scale use case: Log data, feeds into Hadoop, or other data streams beyond the capabilities of their existing messaging systems or infrastructure.
- From there, though, the usage spreads.
- Though the initial use case may have been feeding a Hadoop cluster, once there is a continual feed of events available, the use cases for processing these events in real-time quickly emerge.
- Existing applications will end up tapping into the event streams to react to what is happening more intelligently, and new applications will be built to harness intelligence derived off these streams.

Apache Kafka



- At a number of Silicon Valley companies today you can see this concept in action—everything from user activity to database changes to administrative actions like restarting a process are captured in real-time streams that are subscribed to and processed in real-time.
- What is interesting about this is that what begins as simple plumbing quickly evolves into something much more.
- These data streams begin to act as a kind of central nervous system that applications organize themselves around.

Apache Storm

- Apache Storm is an Open Source distributed, reliable, fault tolerant system for real time processing of data at high velocity.
- This is a key technology towards handling the Velocity aspects of the 3 V's problem.
- Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing and volumous data.
- Storm was designed from the ground up to be usable with any programming language.

Apache Storm

- Storm is fast: a benchmark clocked it at over a million tuples processed per second per node.
- It is scalable, fault-tolerant, guarantees your data will be processed, and is fairly easy to set up and operate.
- Storm integrates with the queueing and database technologies you already use.

Apache Storm

- Apache Storm is a free and open source project licensed under the Apache License, Version 2.0.

- Storm has a large and growing ecosystem of libraries and tools to use in conjunction with Storm including everything from:
  - Spouts: These spouts integrate with queueing systems such as JMS, Kafka, Redis pub/sub, and more.
  - Storm-State: storm-state makes it easy to manage large amounts of in-memory state in your computations in a reliable by using a distributed filesystem for persistence.
  - Database Integrations: There are helper bolts for integrating with various databases, such as MongoDB, RDBMS's, Cassandra, and more.
  - Other Miscellaneous Utilities

Apache Storm

The Apache Storm framework is primarily used for:

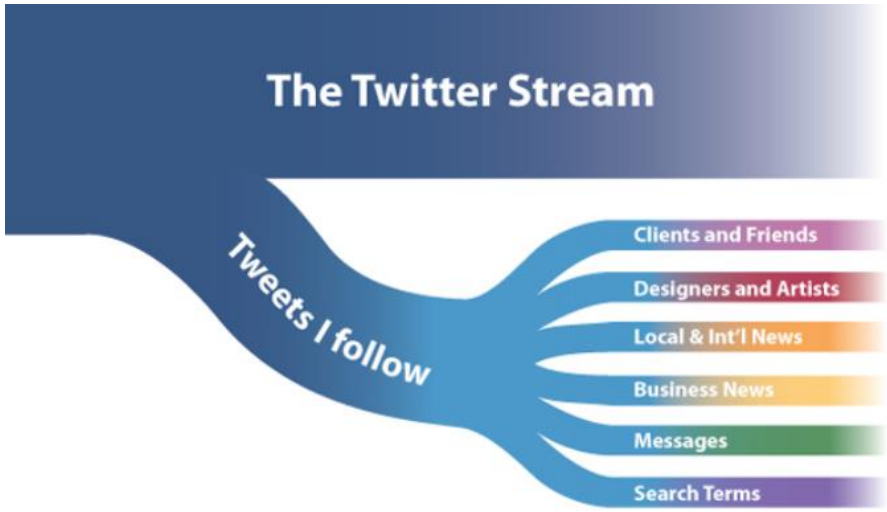
- Real time analytics
- Online machine learning
- Continuous statics computations
- Operational Analytics
- And, to enforce Extract, Transform, and Load (ETL) paradigms.

Note: The Storm documentation at the Adobe Foundation's website has links to notable Storm-related projects hosted outside of Apache.

Apache Storm

- Here is a list of some companies that are using Apache Storm:

Apache Storm

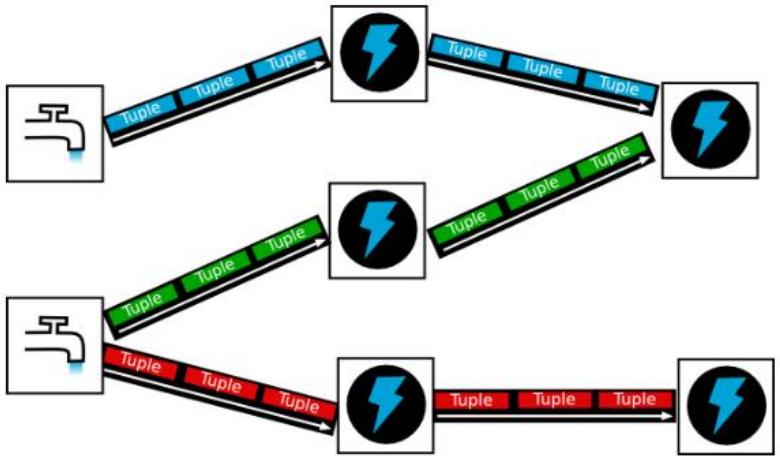


- The core abstraction in Storm is the "stream".
- A stream is an unbounded sequence of tuples.
- Storm provides the primitives for transforming a stream into a new stream in a distributed and reliable way.
- For example, you may transform a stream of tweets into a stream of trending topics.

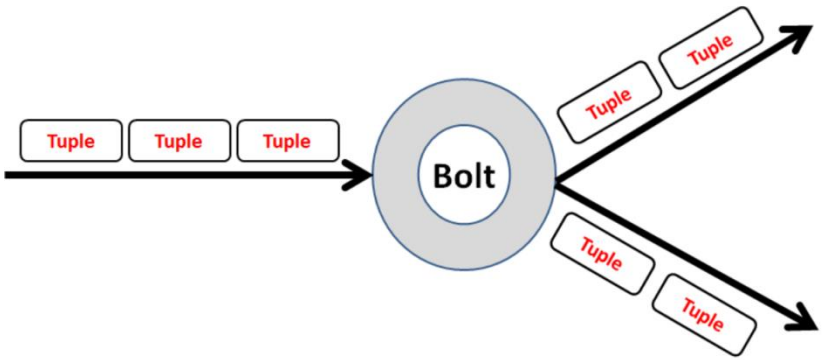
Apache Storm

There are just three abstractions in Storm which work to process streams of data:

- Spout: A source of streams in a computation. Typically a spout reads from a queueing broker such as Kafka, but a spout can also generate its own stream or read from somewhere like the Twitter streaming API.
- Bolt: Spout passes streams of data to Bolt which processes and passes it to either a data store or another Bolt. Bolts can do anything from run functions, filter tuples, do streaming aggregations, do streaming joins, talk to databases, and more.
- Topology: A topology is a network of spouts and bolts, with each edge in the network representing a bolt subscribing to the output stream of some other spout or bolt.



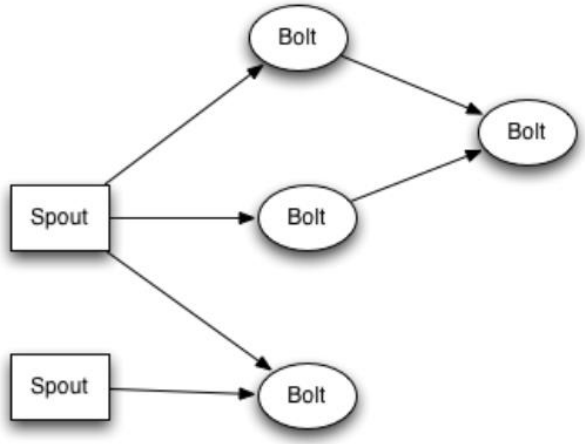
Apache Storm

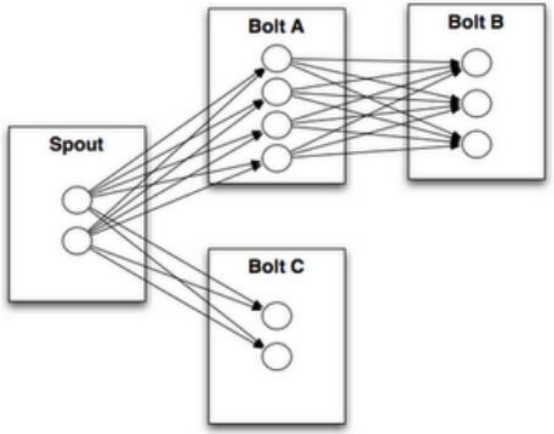


- When programming on Storm, you manipulate and transform streams of tuples, and a tuple is a named list of values.
- Tuples can contain objects of any type.
- If you want to use an object type Storm doesn't know about, it's very easy to register a serializer for that type.
- To do real-time computation on Storm, you create what are called "topologies".

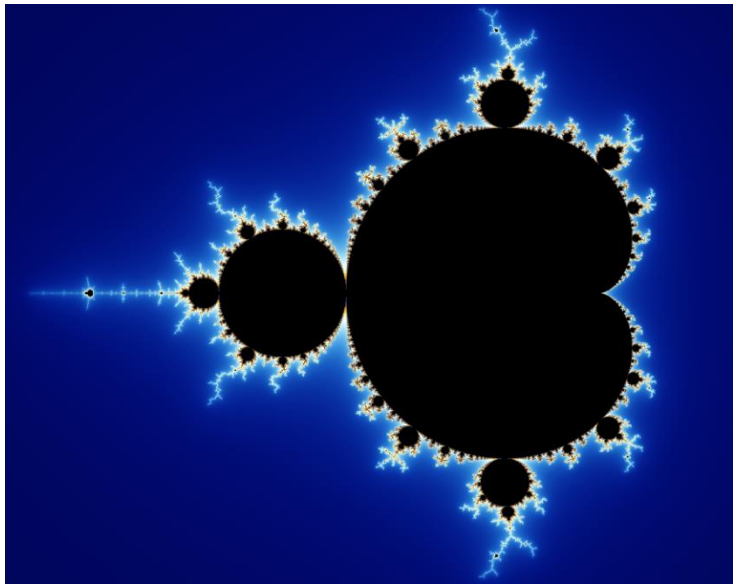
Apache Storm

- A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.
- A topology is an arbitrarily complex multi-stage stream computation. Topologies run indefinitely when deployed.
- When a spout or bolt emits a tuple to a stream, it sends the tuple to every bolt that subscribed to that stream.
- Each node in a Storm topology executes in parallel. In your topology, you can specify how much parallelism you want for each node, and then Storm will spawn that number of threads across the cluster to do the execution.





Apache Storm

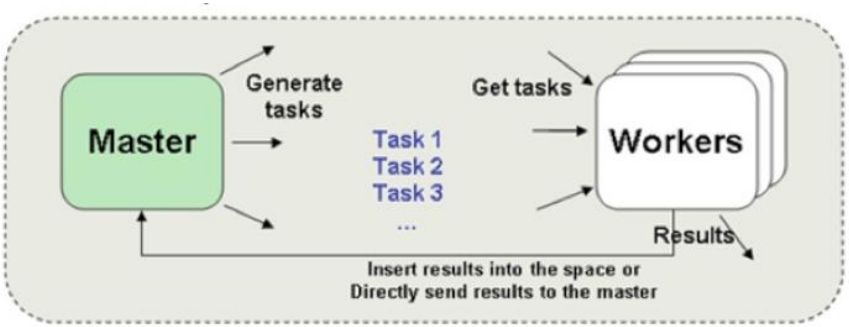


The Mandelbrot set based on infinitely recurring fractals.

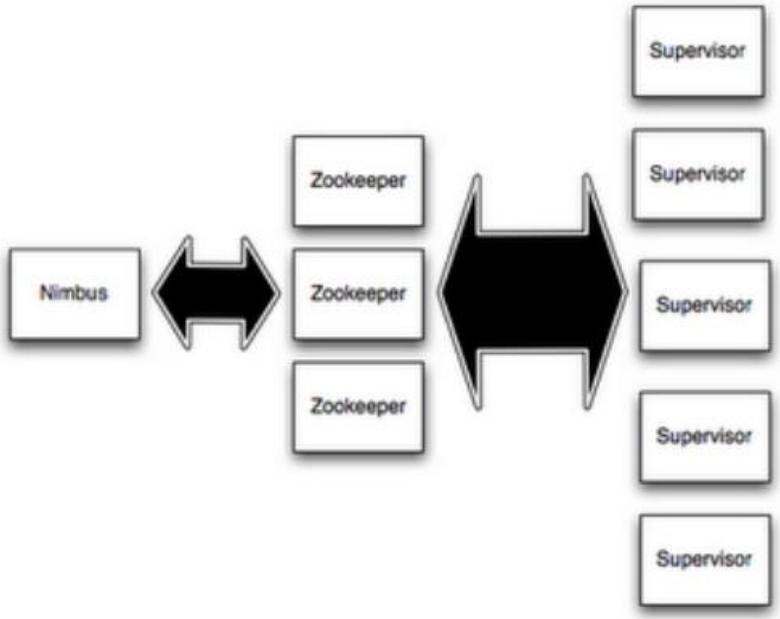
- A Storm cluster is superficially similar to a Hadoop cluster.
- Whereas on Hadoop you run "MapReduce jobs", on Storm you run "topologies".
- "Jobs" and "topologies" themselves are very different -- one key difference is that a MapReduce job eventually finishes, whereas a topology processes messages forever (or until you kill it).

Apache Storm

- There are two kinds of nodes on a Storm cluster: the master node and the worker nodes.
- The master node runs a daemon called "Nimbus" that is similar to Hadoop's "JobTracker".
- Nimbus is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.



Apache Storm



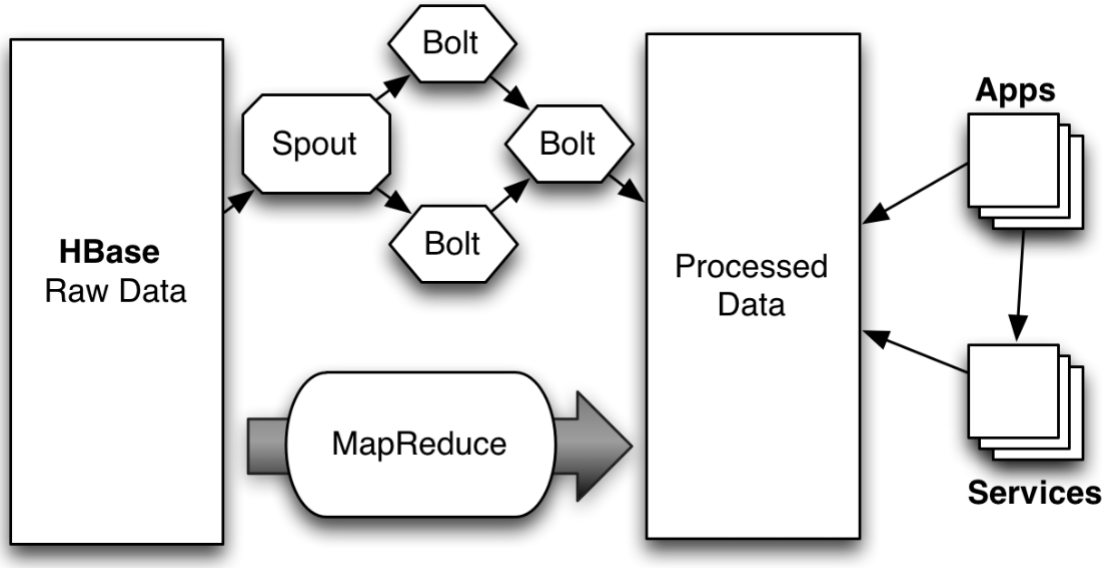
- Each worker node runs a daemon called the "Supervisor".
- The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.
- Each worker process executes a subset of a topology; a running topology consists of many worker processes spread across many machines.

Apache Storm

- All coordination between Nimbus and the Supervisors is done through a ZooKeeper cluster.
- Additionally, the Nimbus daemon and Supervisor daemons are fail-fast and stateless; all state is kept in ZooKeeper or on local disk.
- This means you can kill -9 Nimbus or the Supervisors and they'll start back up like nothing happened.
- This design leads to Storm clusters being incredibly stable.

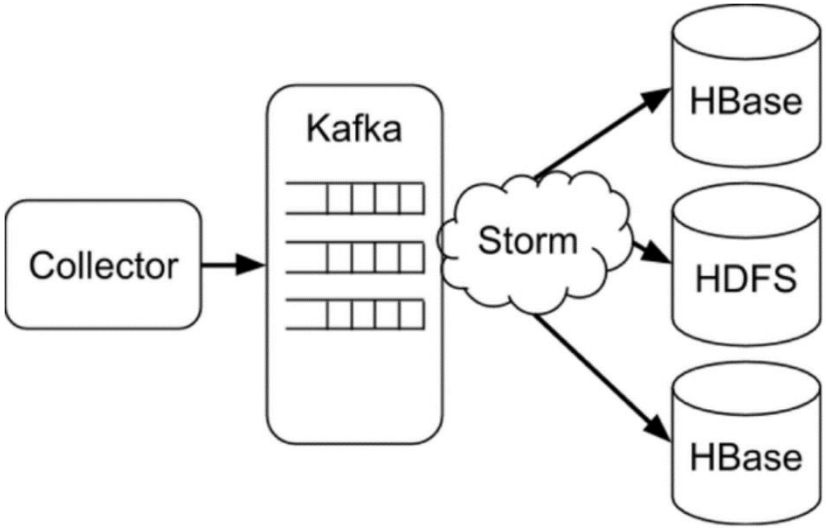
Apache Storm

- Here we can see how the Apache Storm framework can be integrated with other aspects within your Big Data environment:



Apache Storm

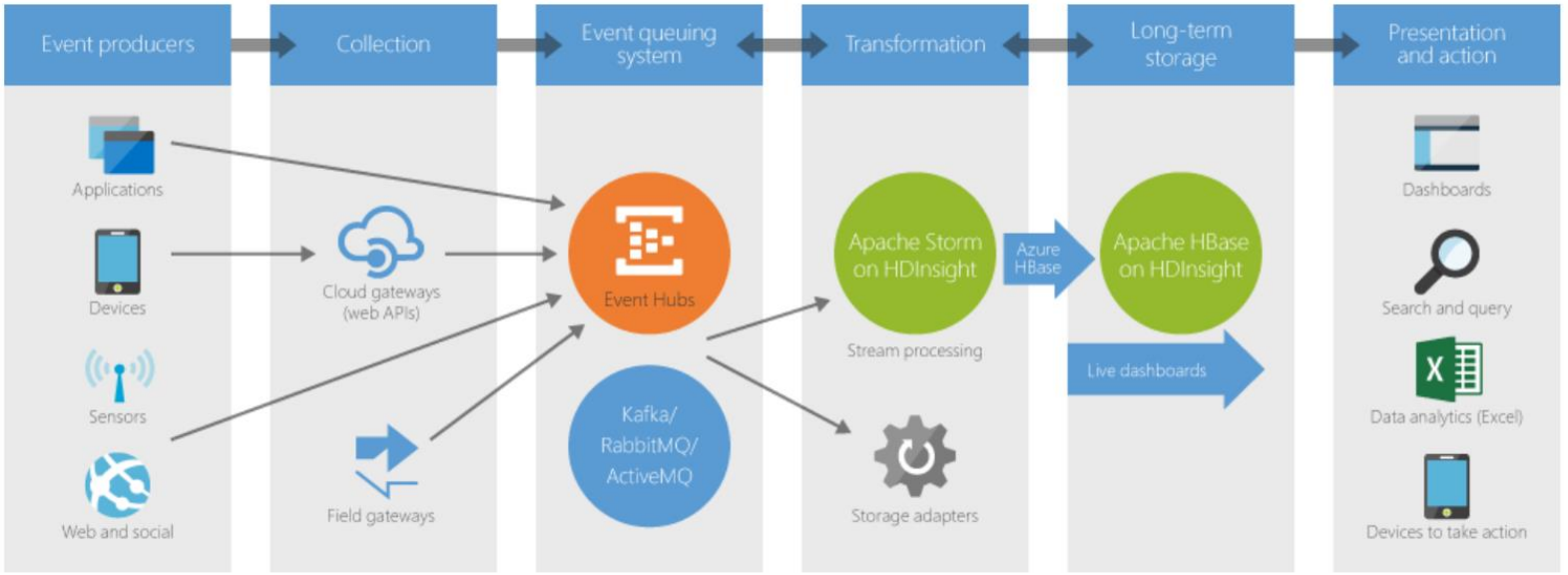
- Here is another depiction of how Apache Storm can be integrated within a Big Data environment:



Bringing It All Together

- The content of this presentation should provide us a general understanding on how to apply some of these Big Data concepts in tandem with each other.

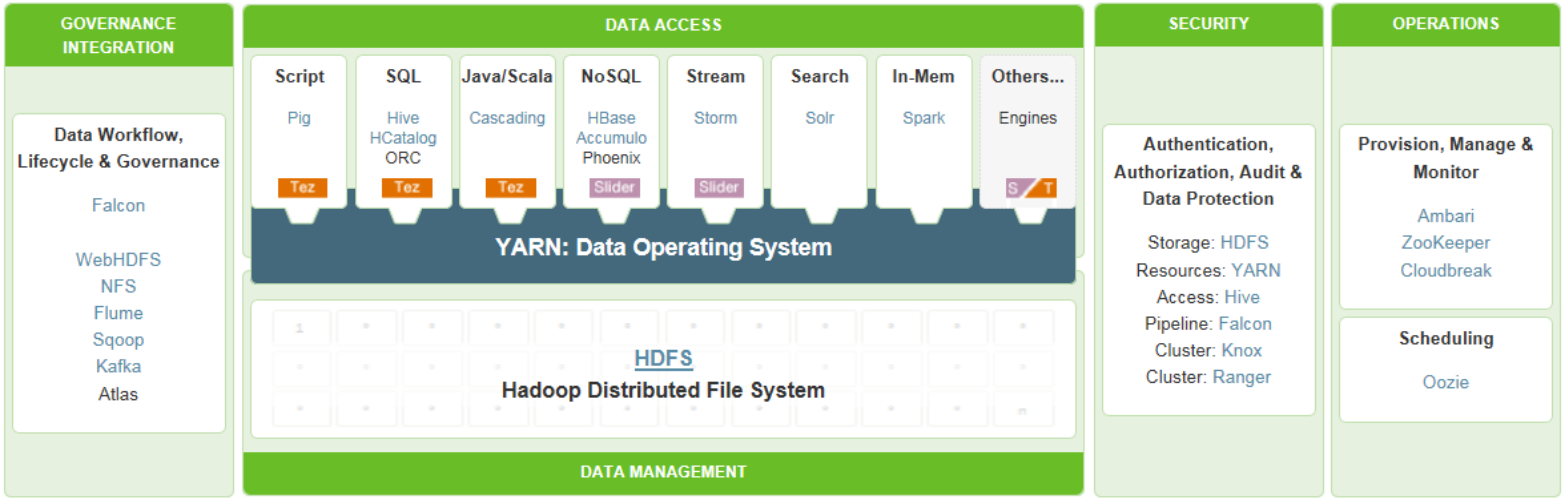
- To showcase this, lets take a look at the following chart provided by Microsoft. Hopefully, this chart makes a little more sense to us now.



Bringing It All Together

- Additionally, we should now be in a better position to evaluate different implementations of Hadoop ecosystems.

- Here is a 100% open source ecosystem provided by Hortonworks.

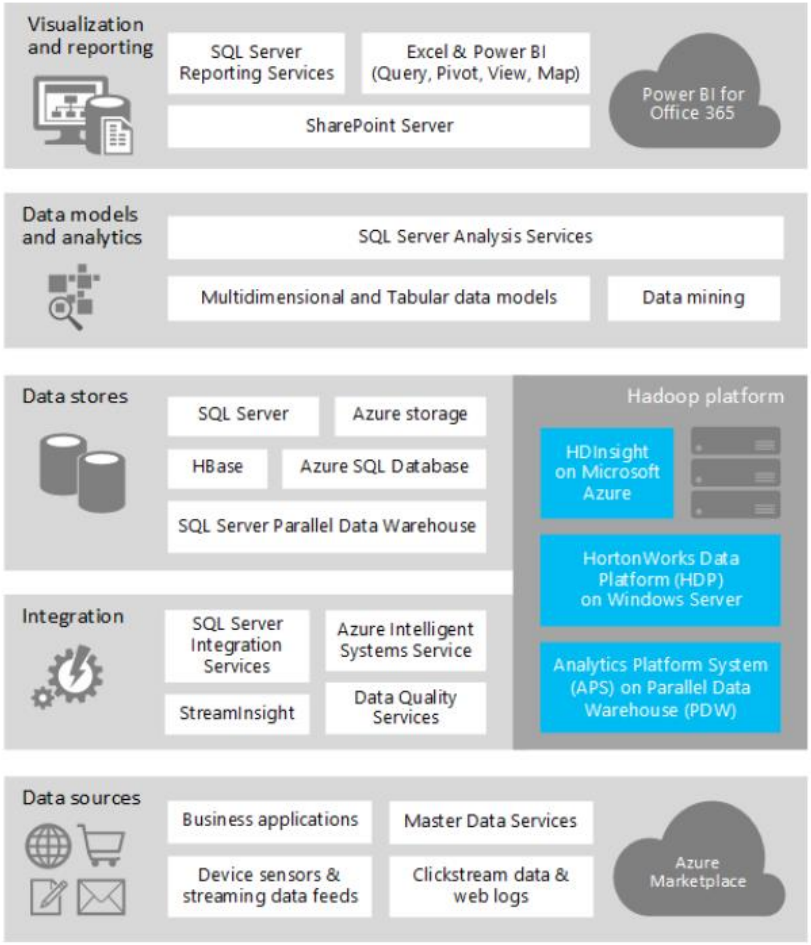


Bringing It All Together

- I personally feel that a key point when deciding what the right Big Data solution is for your organization, is that we need to remember to focus on the types of problems that you will be needing to solve.

- Do you need high storage capabilities?
  - Hadoop / HDFS
  - Pig / Hive / HBase
  - Sqoop / Oozie
- Do you need real time processing?
  - Kafka
  - Storm
- Or both?

- In any case, I feel that this tutorial should provide us with a strong enough framework for making a high level assessment for Big Data needs.



Here is a Big Data platform implementation described by Microsoft utilizing Azure.

Final Thoughts on Big Data

- The quintessential aspect of Big Data is not the data itself; it's the ability to discover useful information hidden in the data.
- Big Data is not just Hadoop—solutions may use traditional data management systems such as relational databases and other types of data store.
- It's really all about the analytics that a Big Data solution can empower.
- Therefore, the data scientists are the people who will drive the intelligence that Big Data has to offer and will continue to grow the technologies adoption.
- Good luck and keep moving forward!!!

Using R with Big Data

Leveraging Hadoop in R

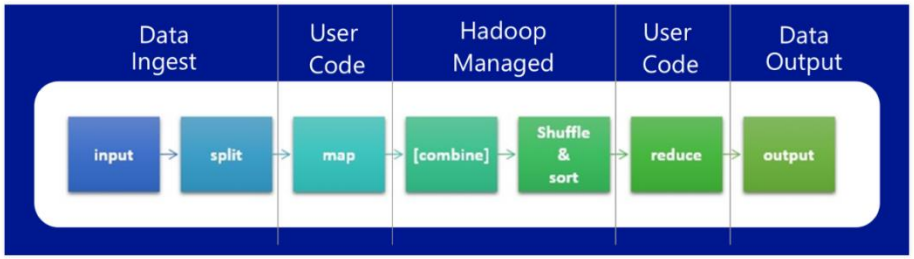
- Now that we have discussed the fundamental aspects of a modern Big Data architecture, lets now approach working with data from the R programming language.

- Here are some R packages which are useful when working with Big Data.

Big Data Concept	Type	R Package
HDFS	Distributed File System	rhdfs
Mapreduce	Task Manager	rmr2
Hbase	NoSQL Database	rhbase
Hive	SQL Like Database	RHbase

- Note: These packages may have to be uploaded from the Github repository as they are not currently accessible via CRAN.

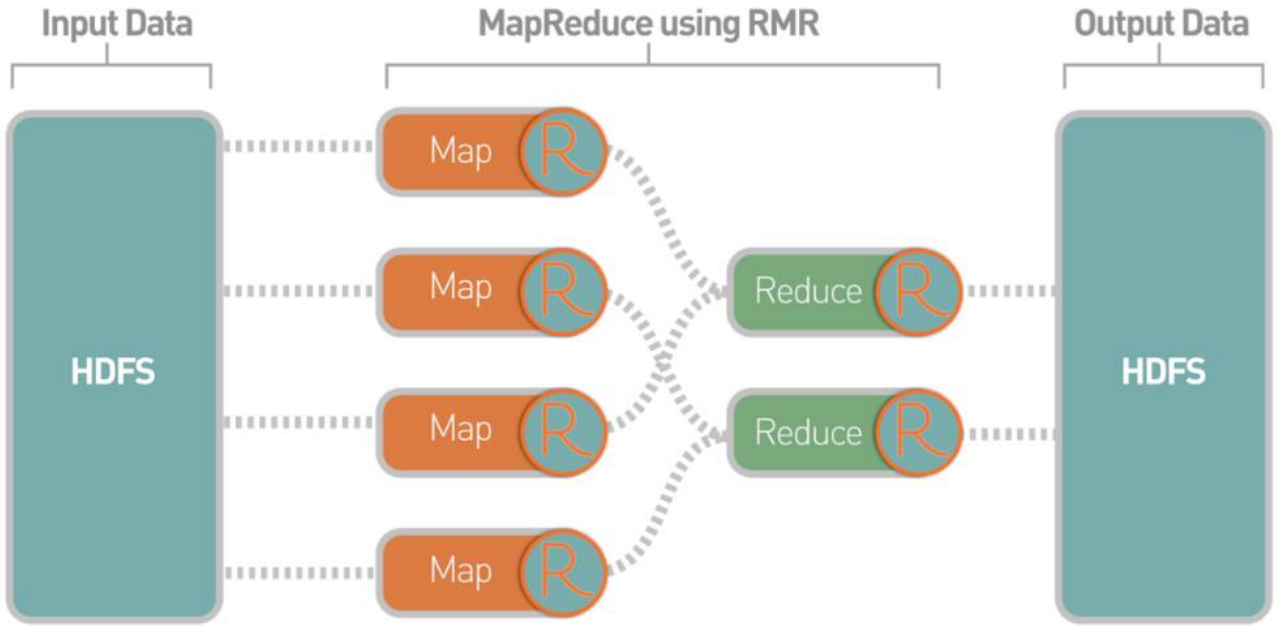
Leveraging Hadoop in R



- The first technique which we will discuss in R will be the MapReduce.
- As data scientists, we do not need to think too much about the inner mechanics of the technique as the implementation detail is hidden within the library.
- Remember that this MapReduce technique is generic and we can implement the technique to solve the problem at hand:
  - Read a large amount of data
  - MAP
  - Extract a summary from each record / block
  - Shuffle and sort
  - REDUCE
  - Aggregate, filter transform

Leveraging Hadoop in R

- The rmr2 package allows you to write R code in the mapper and reducer. This is great for those who do not know how to write code in Java.



Leveraging Hadoop in R

Here is some pseudo code in R for a MapReduce function:

- In the mapper, v' is available as data – no need for an explicit read statement:

```
mapper <- function(k, v){
    ...
    keyval(k', v')
    }
```

- In the reducer, all v' with the same k' is processed together

```
reducer <- function(k', v'){
    ...
    keyval(k'', v'')
    }
```

- Pass these functions to mapreduce():

```
mapreduce(input,
          map = mapper,
          reduce = reducer,
          ...)
```

Leveraging Hadoop in R

- A nice feature of the rmr2 package is that we can test our MapReduce jobs on a local PC rather than through the HDFS architecture.
- This allows us to be able to test code efficiently and then move into the Big Data platform.

Local Backend

```
rmr.options(backend = "local")
```

- The rmr2 package has a "local" back end, completely implemented in R.
- Useful for development, testing and debugging.
- Since computation runs entirely in memory, on small data, it's fast!
- This allows easy (and fast) testing before scaling to the "hadoop" backend.

Hadoop Backend

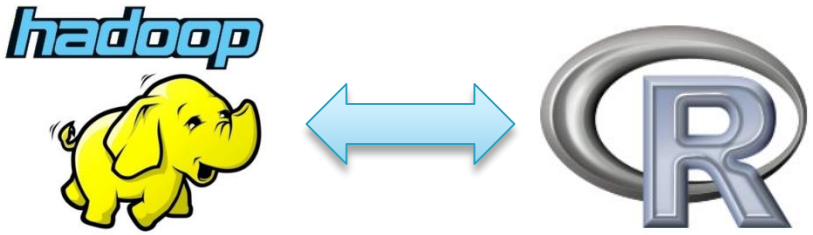
```
rmr.options(backend = "hadoop")
```

- Computation distributed to hdfs and mapreduce
- Hadoop computation overhead.

Leveraging Hadoop in R

- In a production Hadoop context, your data will be ingested into data frames (dfs) by dedicated tools. Ex. Sqoop.
- For easy testing and development, the rmr2 package has two convenient functions that allow you to import / export a big.data.object into the R session.
- Here are the functions:

```
to.dfs()
from.dfs()
```



Leveraging Hadoop in R

- Lets now take a look at an example on how to apply the MapReduce technique and get the data to work with:

Here is what we need to do in R:

- Specify the input, map and reduce functions.
- Optionally, specify output, to persist result in hdfs.
- If output = NULL, then mapreduce() returns a temporary big.data.object.
- A big.data.object is a pointer to a temporary file in dfs.
- If you know that the resulting object is small, use to.dfs() to return data from dfs into your R session.

```
m <- mapreduce(input,
               input.format,
               map,
               reduce,
               output = NULL, ...)
```

```
m()   # returns the file location of the big.data.object
from.dfs(m) # available in R session
```

Leveraging Hadoop in R

- We need to remember that everything in hadoop uses a key - value pair.
- Use keyval() to create the key-value pair inside your mapper and reducer.

```
mapper <- function(k, v){
    ...
    keyval(k', v')
}
reducer <- function(k, v){
    ...
    keyval(k', v')
}
```

- Use the helper functions keys() and values() to separate components from the big.data.object.

```
m <- mapreduce(input, map, reduce, ...)
x <- from.dfs(m) # available in R session

keys(x)
values(x)
```

Leveraging Hadoop in R

Here is a cheat sheet for a MapReduce flow using rmr2:

- Optional: get a sample of data to dfs:

```
hdp.file <- to.dfs(...)
```

- MapReduce:

```
map.fun <- function(k, v){...; keyval(k', v')}
reduce.fun <- function(k, v){...; keyval(k', v')}

m <- mapreduce(input,
               map = map.fun,
               reduce = reduce.fun,
               ...
)
```

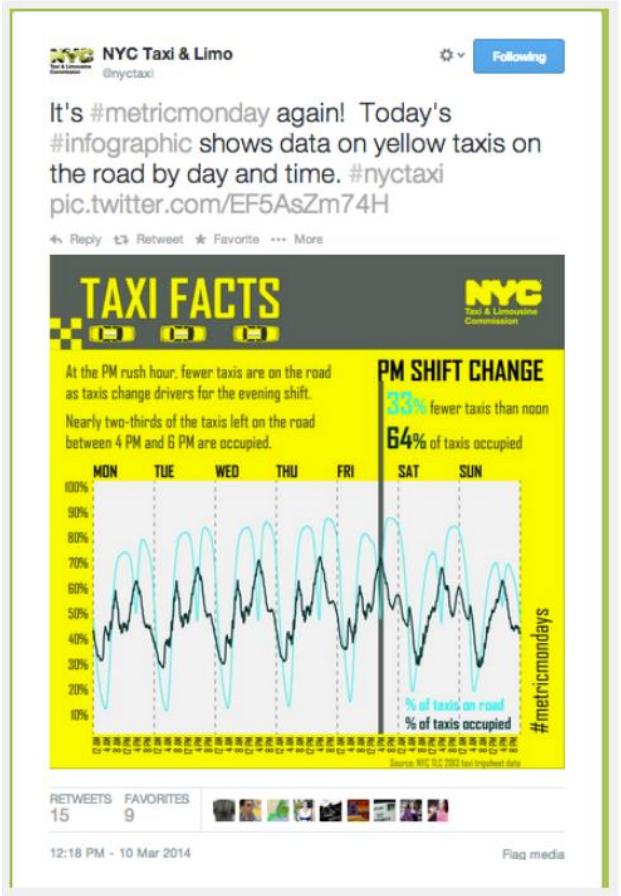
- Inspect Results:

```
x <- from.dfs(m)

keys(x)
values(x)
```

Practical Example – Big Data in R

New York Taxi Data

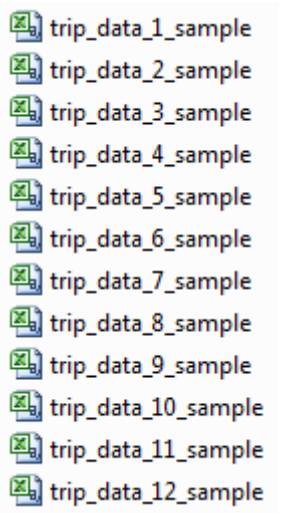
- The following Big Data example was presented at the useR! conference in Aalborg, Denmark in July 2015.
- The basis of the work was related to information made publicly available related to New York Taxi services.
- This dataset is quite large (~200GB uncompressed CSV) and stored in HDFS.
- The dataset contains information about every single taxi trip in New York over a 4-year period.
- Goal: To bring in the data from HDFS into R and run a simple analysis using Big Data techniques.



Understanding the Data

- The data is at: http://publish.illinois.edu/dbwork/open-data/
- For demonstration purposes, we will be utilizing a sample of the full dataset.

2013000153	2013000153	VTS	1		1/1/2013 0:01	1/1/2013 0:05	1	240	0.59	-74.004517	40.721241	-73.997459	40.719055
2013001117	2013001114	VTS	1		1/1/2013 0:06	1/1/2013 0:22	1	960	3.97	-73.995415	40.759789	-73.991066	40.72361
2013002494	2013002491	VTS	1		1/1/2013 0:10	1/1/2013 0:19	4	540	2.39	-73.988754	40.722275	-73.984291	40.733356
2013000562	2013000560	CMT	1	N	1/1/2013 0:12	1/1/2013 0:34	2	1336	5.2	-73.952782	40.791897	-73.989883	40.732738
2013000118	2013000118	VTS	2		1/1/2013 0:14	1/1/2013 0:50	1	2160	17.66	-73.995392	40.754562	-73.785934	40.638897
2013001783	2013001780	VTS	1		1/1/2013 0:17	1/1/2013 0:20	3	180	0.83	-74.002182	40.726425	-74.00058	40.720837
2013005459	2013005456	VTS	1		1/1/2013 0:18	1/1/2013 0:38	1	1200	3.33	-73.985664	40.763676	-73.993309	40.744827
2013001304	2013001301	CMT	1	N	1/1/2013 0:20	1/1/2013 0:26	2	329	0.8	-74.006737	40.73164	-74.010773	40.723591
2013006603	2013006599	VTS	1		1/1/2013 0:22	1/1/2013 0:30	1	480	1.79	-74.003532	40.734226	-73.998878	40.755116
2013003205	2013003202	CMT	1	N	1/1/2013 0:24	1/1/2013 0:30	3	323	1.7	-73.961975	40.763859	-73.944893	40.78339
2013002411	2013002408	CMT	1	N	1/1/2013 0:26	1/1/2013 0:34	1	496	2.2	-73.972183	40.749954	-73.999191	40.738483
2013007943	2013007939	VTS	1		1/1/2013 0:28	1/1/2013 0:34	1	360	1.14	-73.993263	40.73584	-73.995796	40.725163
2013008173	2013008169	VTS	1		1/1/2013 0:30	1/1/2013 0:42	5	720	1.68	-73.994347	40.760849	-73.98278	40.777222
2013003010	2013003007	VTS	1		1/1/2013 0:31	1/1/2013 0:44	2	780	2.6	-73.959175	40.714935	-73.931381	40.6968



Previous analysis published at:

- Brian Donovan and Daniel B. Work. "Using coarse GPS data to quantify city-scale transportation system resilience to extreme events." to appear, Transportation Research Board 94th Annual Meeting, August 2014.

Understanding the Data

- Here is the data definition of the various fields of the New York Taxi Data.

Field	Description
medallion	a permit to operate a yellow taxi cab in New York City, it is effectively a (randomly assigned) car ID. See also medallions.
hack_license	a license to drive the vehicle, it is effectively a (randomly assigned) driver ID. See also hack license.
vendor_id	e.g., Verifone Transportation Systems (VTS), or Mobile Knowledge Systems Inc (CMT), implemented as part of the Technology Passenger Enhancements
rate_code	taximeter rate, see NYCT&L description.
store_and_fwd_flag	unknown attribute.
pickup_datetime	start time of the trip, mm-dd-yyyy hh24:mm:ss EDT.
dropoff_datetime	end time of the trip, mm-dd-yyyy hh24:mm:ss EDT.
passenger_count	number of passengers on the trip, default value is one.
trip_time_in_secs	trip time measured by the taximeter in seconds.
trip_distance	trip distance measured by the taximeter in miles.
pickup_longitude	GPS coordinates at the start of the trip.
pickup_latitude	GPS coordinates at the start of the trip.
dropoff_longitude	GPS coordinates at the end of the trip.
dropoff_latitude	GPS coordinates at the end of the trip.

- **Key Point:** The columns in the individual .csv files do not have descriptive headers which will be addressed later.

Taxi Data in R

- We will first setup a local instance within the rmr2 package:

```
library(rmr2)
rmr.options(backend = "local")
```

```
NULL
```

- The function make.input.format() allows you to specify the attributes of your input data.
- The argument format = "csv" specifies a csv file. This is a wrapper around read.table().

```
taxi.format <- make.input.format("csv", sep = ",",
                                 colClasses = "character",
                                 stringsAsFactors = FALSE
)
```

Taxi Data in R

- Use from.dfs() to get a (small) file from dfs to local memory:

```
taxi.hdp <- "data/trip_data_1_sample.csv"
x <- from.dfs(taxi.hdp, format = taxi.format)
str(x)
```

```
List of 2
 $ key: NULL
 $ val:'data.frame':    14777 obs. of  14 variables:
  ..$ V1 : chr [1:14777] "2013000153" "2013001117" "2013002494" "2013000562" ...
  ..$ V2 : chr [1:14777] "2013000153" "2013001114" "2013002491" "2013000560" ...
  ..$ V3 : chr [1:14777] "VTS" "VTS" "VTS" "CMT" ...
  ..$ V4 : chr [1:14777] "1" "1" "1" "1" ...
  ..$ V5 : chr [1:14777] "" "" "" "N" ...
  ..$ V6 : chr [1:14777] "2013-01-01 00:01:00" "2013-01-01 00:06:00" "2013-01-01
00:10:00" "2013-01-01 00:12:06" ...
  ..$ V7 : chr [1:14777] "2013-01-01 00:05:00" "2013-01-01 00:22:00" "2013-01-01
00:19:00" "2013-01-01 00:34:22" ...
  ..$ V8 : chr [1:14777] "1" "1" "4" "2" ...
  ..$ V9 : chr [1:14777] "240" "960" "540" "1336" ...
  ..$ V10: chr [1:14777] ".59" "3.97" "2.39" "5.20" ...
  ..$ V11: chr [1:14777] "-74.004517" "-73.995415" "-73.988754" "-73.952782" ...
  ..$ V12: chr [1:14777] "40.721241" "40.759789" "40.722275" "40.791897" ...
  ..$ V13: chr [1:14777] "-73.997459" "-73.991066" "-73.984291" "-73.989883" ...
  ..$ V14: chr [1:14777] "40.719055" "40.72361" "40.733356" "40.732738" ...
```

Taxi Data in R

- We will use keys() and values() to extract the components utilizing the from.dfs() command:

```
x <- from.dfs(taxi.hdp, format = taxi.format)
head(
    values(x)
)
```

```
         V1         V2  V3 V4 V5                  V6                  V7
1 2013000153 2013000153 VTS  1    2013-01-01 00:01:00 2013-01-01 00:05:00
2 2013001117 2013001114 VTS  1    2013-01-01 00:06:00 2013-01-01 00:22:00
3 2013002494 2013002491 VTS  1    2013-01-01 00:10:00 2013-01-01 00:19:00
4 2013000562 2013000560 CMT  1  N 2013-01-01 00:12:06 2013-01-01 00:34:22
5 2013000118 2013000118 VTS  2    2013-01-01 00:14:00 2013-01-01 00:50:00
6 2013001783 2013001780 VTS  1    2013-01-01 00:17:00 2013-01-01 00:20:00
  V8   V9   V10        V11       V12        V13       V14
1  1  240   .59 -74.004517 40.721241 -73.997459 40.719055
2  1  960  3.97 -73.995415 40.759789 -73.991066  40.72361
3  4  540  2.39 -73.988754 40.722275 -73.984291 40.733356
4  2 1336  5.20 -73.952782 40.791897 -73.989883 40.732738
5  1 2160 17.66 -73.995392 40.754562 -73.785934 40.638897
6  3  180   .83 -74.002182 40.726425  -74.00058 40.720837
```

Taxi Data in R

- Why do the columns not have any labels?
- Remember: hdfs splits the individual files across nodes.
- Implications:
  - The chunk that's available to your mapper may not have a header file.
  - Therefore csv files should not have a header at all!
- The Solution:
  - Make a custom input format, specifying the column names.
  - (in the same way you specify col.names to read.table()).

Taxi Data in R

- One of the files we have to work with contains the data dictionary and we will need to use this to make an input format:

```
headerInfo <- read.csv("data/dictionary_trip_data.csv", stringsAsFactors =
FALSE)
headerInfo
```

```
  medallion hack_license vendor_id rate_code store_and_fwd_flag
1   integer      integer character   integer          character
  pickup_datetime dropoff_datetime passenger_count trip_time_in_secs
1       character        character         integer           integer
  trip_distance pickup_longitude pickup_latitude dropoff_longitude
1       numeric          numeric         numeric           numeric
  dropoff_latitude
1          numeric
```

```
colClasses <- as.character(as.vector(headerInfo[1, ]))
names(headerInfo)
```

```
 [1] "medallion"          "hack_license"       "vendor_id"
 [4] "rate_code"          "store_and_fwd_flag" "pickup_datetime"
 [7] "dropoff_datetime"   "passenger_count"    "trip_time_in_secs"
[10] "trip_distance"      "pickup_longitude"   "pickup_latitude"
[13] "dropoff_longitude"  "dropoff_latitude"
```

Taxi Data in R

- Use the data dictionary information to configure the input format:

```
taxi.format <- make.input.format(format = "csv", sep = ",",
                                 col.names = names(headerInfo),
                                 colClasses = colClasses,
                                 stringsAsFactors = FALSE
)

x <- from.dfs(taxi.hdp, format = taxi.format)
str(values(x))
```

```
'data.frame':   14777 obs. of  14 variables:
 $ medallion         : int  2013000153 2013001117 2013002494 2013000562
2013000118 2013001783 2013005459 2013001304 2013006603 2013003205 ...
 $ hack_license      : int  2013000153 2013001114 2013002491 2013000560
2013000118 2013001780 2013005456 2013001301 2013006599 2013003202 ...
 $ vendor_id         : chr  "VTS" "VTS" "VTS" "CMT" ...
 $ rate_code         : int  1 1 1 1 2 1 1 1 1 1 ...
 $ store_and_fwd_flag: chr  "" "" "" "N" ...
 $ pickup_datetime   : chr  "2013-01-01 00:01:00" "2013-01-01 00:06:00" "2013-
01-01 00:10:00" "2013-01-01 00:12:06" ...
 $ dropoff_datetime  : chr  "2013-01-01 00:05:00" "2013-01-01 00:22:00" "2013-
01-01 00:19:00" "2013-01-01 00:34:22" ...
 $ passenger_count   : int  1 1 4 2 1 3 1 2 1 3 ...
 $ trip_time_in_secs : int  240 960 540 1336 2160 180 1200 329 480 323 ...
 $ trip_distance     : num  0.59 3.97 2.39 5.2 17.66 ...
 $ pickup_longitude  : num  -74 -74 -74 -74 -74 ...
 $ pickup_latitude   : num  40.7 40.8 40.7 40.8 40.8 ...
```

Taxi Data in R

- In order to showcase the Big Data techniques in R, we will start off by performing a simple MapReduce job and then add more complexity later.

- Execute a basic MapReduce job without any transformation:

```
m <- mapreduce(taxi.hdp, input.format = taxi.format)
m
```

```
function ()
{
    fname
}
<bytecode: 0x3be8f88>
<environment: 0x3be7148>
```

```
m()
```

```
[1] "/tmp/Rtmp3wachy/file98a73d6623ac"
```

Taxi Data in R

- Lets take a look at some of the values of the dataframe:

```
m <- mapreduce(taxi.hdp, input.format = taxi.format)
head(
    values(from.dfs(m))
)
```

```
  medallion hack_license vendor_id rate_code store_and_fwd_flag
1 2013000153   2013000153       VTS         1
2 2013001117   2013001114       VTS         1
3 2013002494   2013002491       VTS         1
4 2013000562   2013000560       CMT         1                  N
5 2013000118   2013000118       VTS         2
6 2013001783   2013001780       VTS         1
     pickup_datetime    dropoff_datetime passenger_count
1 2013-01-01 00:01:00 2013-01-01 00:05:00               1
2 2013-01-01 00:06:00 2013-01-01 00:22:00               1
3 2013-01-01 00:10:00 2013-01-01 00:19:00               4
4 2013-01-01 00:12:06 2013-01-01 00:34:22               2
5 2013-01-01 00:14:00 2013-01-01 00:50:00               1
6 2013-01-01 00:17:00 2013-01-01 00:20:00               3
  trip_time_in_secs trip_distance pickup_longitude pickup_latitude
1               240          0.59        -74.00452         40.72124
2               960          3.97        -73.99541         40.75979
3               540          2.39        -73.98875         40.72228
4              1336          5.20        -73.95278         40.79190
5              2160         17.66        -73.99539         40.75456
6               180          0.83        -74.00218         40.72642
  dropoff_longitude dropoff_latitude
1         -73.99746         40.71905
```

Taxi Data in R

- Lets expand on this job and build in simple query parameter.
- This technique is mapping in the 6th column of our dataframe, which is the "pickup_datetime".

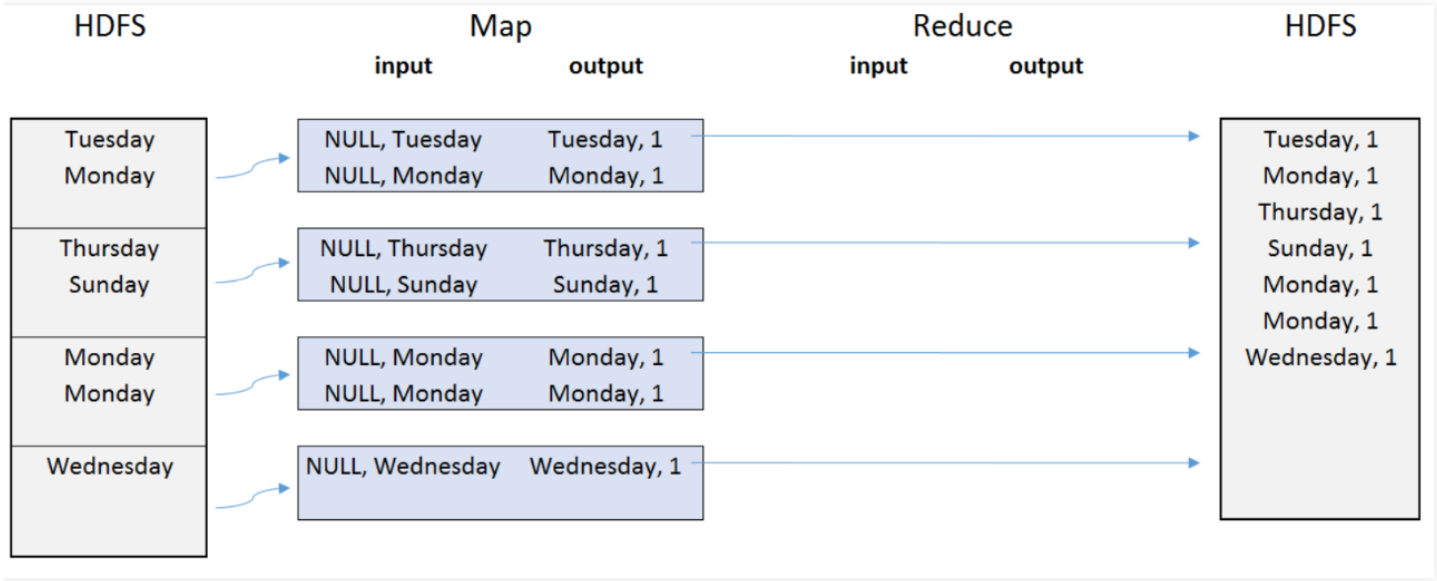
```
taxi.map <- function(k, v){
    original <- v[[6]]
    original
}
m <- mapreduce(taxi.hdp, input.format = taxi.format,
               map = taxi.map
)
head(
    values(from.dfs(m))
)
```

```
[1] "2013-01-01 00:01:00" "2013-01-01 00:06:00" "2013-01-01 00:10:00"
[4] "2013-01-01 00:12:06" "2013-01-01 00:14:00" "2013-01-01 00:17:00"
```

- The upcoming slide will graphically show what we had just performed.

Taxi Data in R

- Here is a basic diagram showcasing the technique we had just applied:



Taxi Data in R

- Here is a technique which incorporates the reducer function:

```
taxi.map <- function(k, v){
    original <- v[[6]]
    date <- as.Date(original, origin = "1970-01-01")
    wkday <- weekdays(date)
    keyval(wkday, 1)
}
taxi.reduce <- function(k, v){
    keyval(k, sum(v))
}
m <- mapreduce(taxi.hdp, input.format = taxi.format,
               map = taxi.map,
               reduce = taxi.reduce
)
head(
    keys(from.dfs(m))
)
```

```
[1] "Tuesday"   "Wednesday" "Thursday"  "Friday"    "Saturday"  "Sunday"
```

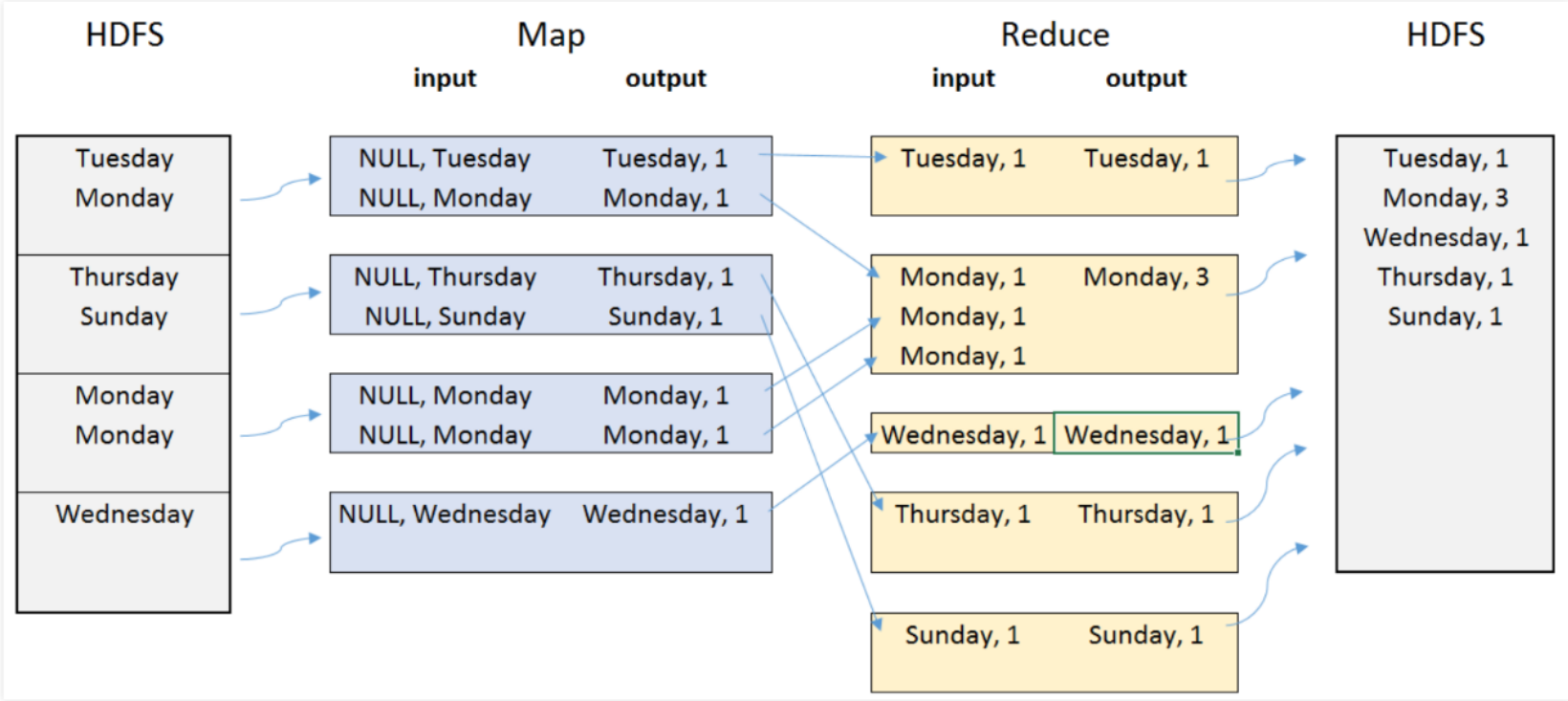
- Notice that the taxi.reduce component contains a sum(v) which is a sum of the count of keyval for wkday.

```
head(
    values(from.dfs(m))
)
```

```
[1] 2323 2354 2506 2080 2049 1776
```

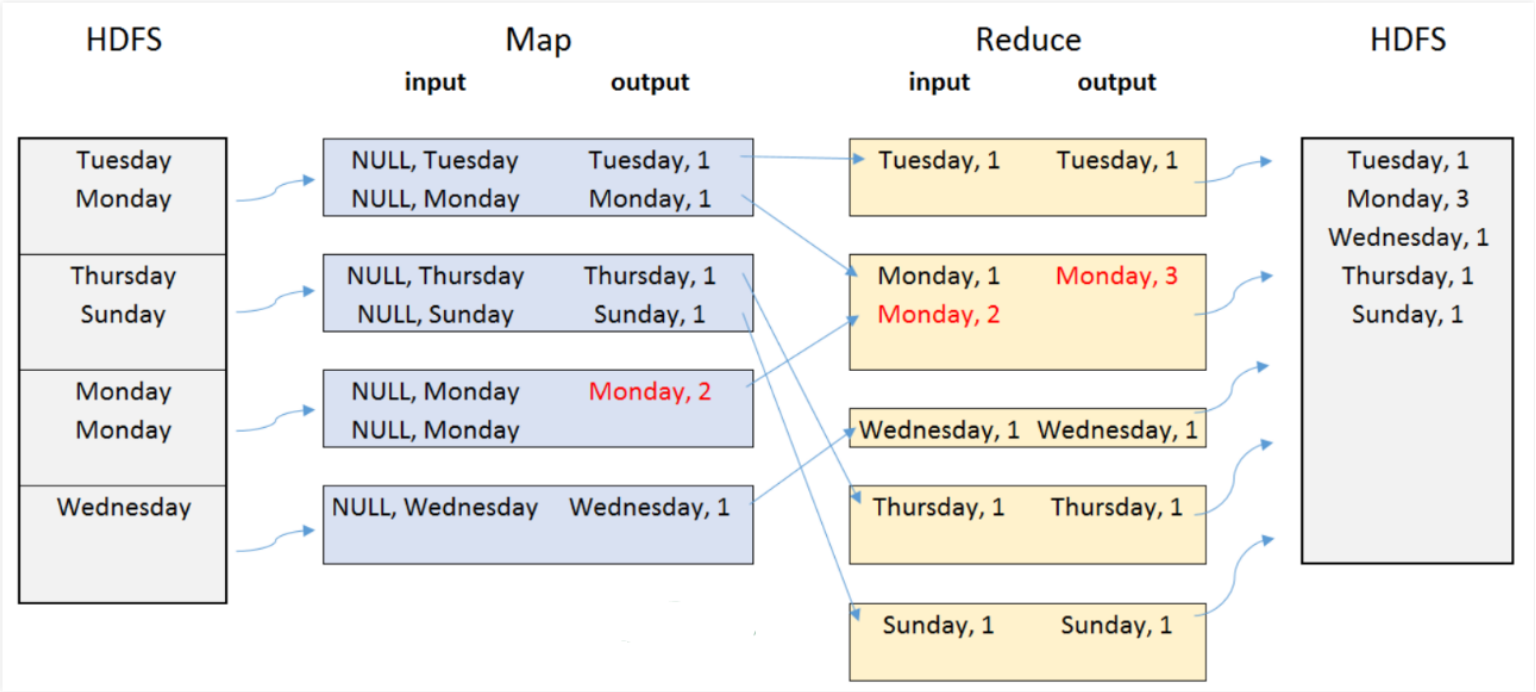
Taxi Data in R

- Here is a technique which incorporates the reducer function:



Taxi Data in R

- Notice the aggregation of the Monday count in the mapping & reducing output step.



- Question: Can we write a better mapper and reducer that is more efficient in the processing?

Taxi Data in R

- Here is how we execute that technique with a more effective mapper:

```
taxi.map <- function(k, v){
    original <- v[[6]]
    date <- as.Date(original, origin = "1970-01-01")
    wkday <- weekdays(date)
    dat <- data.frame(date, wkday)
    z <- aggregate(date ~ wkday, dat, FUN = length)
    keyval(z[[1]], z[[2]])
}
m <- mapreduce(taxi.hdp, input.format = taxi.format,
               map = taxi.map
)
keys(from.dfs(m))
```

Notice that the aggregation is being applied within the mapper stage using the aggregate function.

```
 [1] Friday    Friday    Friday    Friday    Monday    Monday    Monday
 [8] Monday    Saturday  Saturday  Saturday  Saturday  Sunday    Sunday
[15] Sunday    Sunday    Thursday  Thursday  Thursday  Thursday  Thursday
[22] Tuesday   Tuesday   Tuesday   Tuesday   Wednesday Wednesday Wednesday
[29] Wednesday Wednesday
Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
```

```
values(from.dfs(m))
```

```
 [1] 478 521 544 537 408 444 386 451 474 508 510 557 403 443 459 471 441
[18] 488 513 533 531 854 497 495 477 406 563 558 438 389
```

Taxi Data in R

- Lets put it all together now:

```
taxi.map <- function(k, v){
    original <- v[[6]]
    date <- as.Date(original, origin = "1970-01-01")
    wkday <- weekdays(date)
    dat <- data.frame(date, wkday)
    z <- aggregate(date ~ wkday, dat, FUN = length)
    keyval(z[[1]], z[[2]])
}
taxi.reduce <- function(k, v){
    data.frame(weekday = k, trips = sum(v), row.names = k)
}
m <- mapreduce(taxi.hdp, input.format = taxi.format,
               map = taxi.map,
               reduce = taxi.reduce
)
keys(from.dfs(m))
```

```
values(from.dfs(m))
```

	weekday	trips
Friday	Friday	2080
Monday	Monday	1689
Saturday	Saturday	2049
Sunday	Sunday	1776
Thursday	Thursday	2506

Taxi Data in R

- Lets now analyze the taxi data by the hour:

```
taxi.map <- function(k, v){
    original <- v[[6]]
    date <- as.Date(original, origin = "1970-01-01")
    wkday <- weekdays(date)
    hour <- format(as.POSIXct(original), "%H")
    dat <- data.frame(date, hour)
    z <- aggregate(date ~ hour, dat, FUN = length)
    keyval(z[[1]], z[[2]])
}

taxi.reduce <- function(k, v){
    data.frame(hour = k, trips = sum(v), row.names = k)
}
```

- Then we run the MapReduce job:

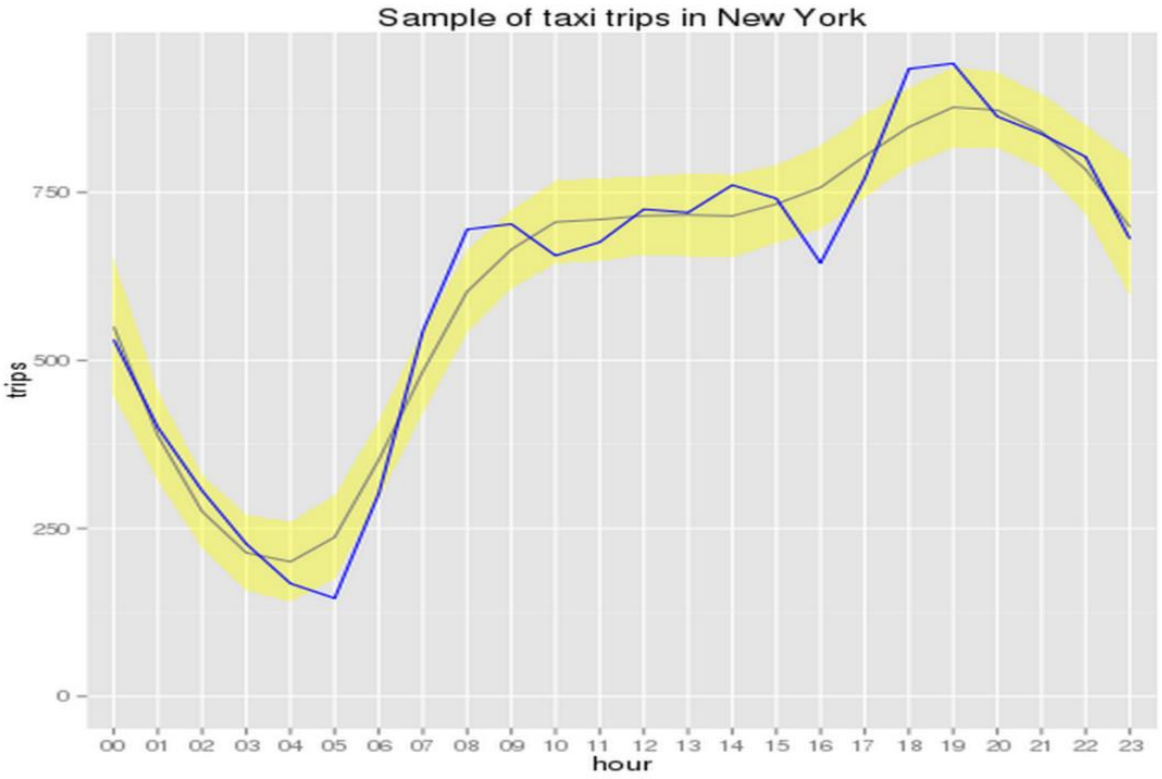
```
m <- mapreduce(taxi.hdp, input.format = taxi.format,
               map = taxi.map,
               reduce = taxi.reduce
)
keys(from.dfs(m))
```

```
dat <- values(from.dfs(m))
dat
```

	hour	trips
00	00	531
01	01	400
02	02	306
03	03	227
04	04	168
05	05	146
06	06	302
07	07	544
08	08	695
09	09	703
10	10	656
11	11	676
12	12	725
13	13	720
14	14	761

Taxi Data in R

- Lets now see the results in ggplot2:




```
library("ggplot2")
p <- ggplot(dat, aes(x = hour, y = trips, group = 1)) +
    geom_smooth(method = loess, span = 0.5,
                col = "grey50", fill = "yellow") +
    geom_line(col = "blue") +
    expand_limits(y = 0) +
    ggtitle("Sample of taxi trips in New York")
```

Taxi Data in R

- This section will show you how to get the output results back into the HDFS file structure utilizing the rhdfs package in R.

- Here is an overview of different functions in the rhdfs package:

- Initialize
  - `hdfs.init()`
  - `hdfs.defaults()`
- File and directory manipulation
  - `hdfs.ls()`
  - `hdfs.delete()`
  - `hdfs.mkdir()`
  - `hdfs.exists()`
- Copy and move from local <-> HDFS
  - `hdfs.put()`
  - `hdfs.get()`
- Manipulate files within HDFS
  - `hdfs.copy()`
  - `hdfs.move()`
  - `hdfs.rename()`
- Reading files directly from HDFS
  - `hdfs.file()`
  - `hdfs.read()`
  - `hdfs.write()`
  - `hdfs.flush()`
  - `hdfs.seek()`
  - `hdfs.tell(con)`
  - `hdfs.close()`
  - `hdfs.line.reader()`
  - `hdfs.read.text.file()`

Taxi Data in R

- Here is how we can put the data from our local file system into HDFS through the rhdfs package:

```
library(rhdfs)
hdfs.init()

localFiles <- dir("data", pattern = "_sample.csv", full.names = TRUE)
localFiles
hdfs.mkdir("taxi")
hdfs.put(localFiles, "taxi")
hdfs.ls("taxi")

hdfs.ls("taxi")$file
```

Taxi Data in R

- Once you've tested your script in local context, it is generally very easy to deploy on all your data.

```
rmr.options(backend = "hadoop")
```

- This is what the hadoop output will show when the MapReduce job is executed:

```
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-
streaming-2.4.0.2.1.5.0-695.jar] /tmp/streamjob4577673905010649130.jar
tmpDir=null
15/06/23 14:18:01 INFO impl.TimelineClientImpl: Timeline service address:
http://0.0.0.0:8188/ws/v1/timeline/
15/06/23 14:18:01 INFO client.RMProxy: Connecting to ResourceManager at ra-ldn-
cluster-master-02.cloudapp.net/172.16.0.5:8050
15/06/23 14:18:02 INFO impl.TimelineClientImpl: Timeline service address:
http://0.0.0.0:8188/ws/v1/timeline/
....
15/06/23 14:18:17 INFO mapreduce.Job: Job job_1434365936669_0041 running in uber
mode : false
15/06/23 14:18:17 INFO mapreduce.Job:  map 0% reduce 0%
15/06/23 14:18:31 INFO mapreduce.Job:  map 1% reduce 0%
15/06/23 14:18:34 INFO mapreduce.Job:  map 17% reduce 0%
15/06/23 14:18:40 INFO mapreduce.Job:  map 19% reduce 0%
15/06/23 14:18:41 INFO mapreduce.Job:  map 21% reduce 0%
15/06/23 14:18:47 INFO mapreduce.Job:  map 37% reduce 0%
15/06/23 14:18:48 INFO mapreduce.Job:  map 92% reduce 0%
15/06/23 14:18:49 INFO mapreduce.Job:  map 92% reduce 19%
15/06/23 14:18:50 INFO mapreduce.Job:  map 100% reduce 19%
15/06/23 14:18:52 INFO mapreduce.Job:  map 100% reduce 100%
15/06/23 14:19:00 INFO mapreduce.Job: Job job_1434365936669_0041 completed
successfully
...
  rmr
        reduce calls=24
15/06/23 14:19:01 INFO streaming.StreamJob: Output
directory: /tmp/file55a346591243
```

Taxi Data in R

- The final aspect we will address in this tutorial will pertain to utilizing the Hive query language (HiveQL) within R.

Here are some key considerations when working with R & Hive:

- Hive allows us to define group-by queries writing SQL-like code.
- HiveQL translates into MapReduce jobs and underlying code is not displayed.
- R tools for string manipulation allow for us to easily build complex queries.
- The package RHive allows us to run Hive queries directly from R.
- For more complex queries, we still need to use other tools like rmr2.

Specifically, we will focus on:

- Building queries using string manipulation tools like sprintf and paste.
- Running queries using RHive.

Taxi Data in R

- There is a syntax difference that we need to remember when writing queries in Hive versus writing Hive queries within the R interface.

- Hive: semi-colon in the end.

```
SELECT COUNT(*) FROM table_data;
```

- RHive: no semi-colon in the end.

```
SELECT COUNT(*) FROM table_data
```

Taxi Data in R

- Using Hive, we can replicate the rmr2 example counting the number of taxi runs by hour.

The steps are as follows:

- Importing the data
- Querying the data

- Starting from our raw data, we can create a Hive table. Since the data is already in CSV format, the easiest option is creating an external table.

We need to specify:

- The data location
- The data format
- The field formats

Taxi Data in R

- Here is the full Hive query to create an external table:

```
CREATE EXTERNAL TABLE taxi_sample(medallion STRING, hack_license STRING,
vendor_id STRING, rate_code INT, store_and_fwd_flag STRING, pickup_datetime
STRING, dropoff_datetime STRING, passenger_count INT, trip_time_in_secs INT,
trip_distance FLOAT, pickup_longitude FLOAT, pickup_latitude FLOAT,
dropoff_longitude FLOAT, dropoff_latitude FLOAT) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/share/taxi/sample';
```

- Notice the similarities to the SQL language. If you are comfortable writing queries in SQL (R Package: sqldf), the learning curve for Hive will be lessened.

- We will break this query down into its constituents in the upcoming slide.

Taxi Data in R

- The first part of the query defines a table called taxi_sample and the format of its fields.

```
CREATE EXTERNAL TABLE taxi_sample(medallion STRING, hack_license STRING,
vendor_id STRING, rate_code INT, store_and_fwd_flag STRING, pickup_datetime
STRING, dropoff_datetime STRING, passenger_count INT, trip_time_in_secs INT,
trip_distance FLOAT, pickup_longitude FLOAT, pickup_latitude FLOAT,
dropoff_longitude FLOAT, dropoff_latitude FLOAT) ROW FORMAT
```

- This part of the query defines the data format.

```
DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
```

- This part of the query defines the HDFS path to the data folder.

```
LOCATION '/user/share/taxi/sample';
```

Taxi Data in R

- Now that we reviewed the basics of a Hive query, lets setup RHive.

```
# install.packages("RHive")
library(RHive)
dirHive <- "/user/hive"
Sys.setenv(HIVE_HOME = "/usr/lib/hive")
rhive.init()
rhive.connect(host = "127.0.0.1", hiveServer2 = TRUE)
```

- Here is a simple Hive query in R to count the number of records.

```
query_count <- "SELECT COUNT(*) FROM taxi_sample"
```

```
table_count <- rhive.query(query_count)
head(table_count)
```

```
    X_c0
1 173187
```

Taxi Data in R

- Here is a string manipulation technique for writing the same query utilizing sprintf.

```
name_table <- "taxi_sample"
query_count <- sprintf("SELECT COUNT(*) FROM %s",
                       name_table)
cat(query_count)
```

```
SELECT COUNT(*) FROM taxi_sample
```

```
table_count <- rhive.query(query_count)
head(table count)
```

```
     X_c0
 1 173187
```

Taxi Data in R

- Now that we have an understanding of the basic mechanics of a Hive query within R, lets recreate the count example we originally built using MapReduce.

```
query_hour <- "
SELECT pickup_datetime, substring(pickup_datetime, 12, 2) AS hour
  FROM taxi_sample LIMIT 100"
```

- Here is how we define the hour in sprintf:

```
field_time <- "pickup_datetime"
field_hour <- sprintf("substring(%s, 12, 2)",
                      field_time)
query_hour <- sprintf(
  "SELECT %s, %s AS hour
  FROM %s LIMIT 100",
  field_time, field_hour, name_table)
cat(query_hour)
```

- RHive query syntax:

```
SELECT pickup_datetime, substring(pickup_datetime, 12, 2) AS hour
  FROM taxi_sample LIMIT 100
```

Taxi Data in R

- Now that we have the query syntax developed, we can execute the Hive query:

```
head(rhive.query(query_hour))
```

```
      pickup_datetime hour
1 2013-10-01 00:02:10   00
2 2013-10-01 00:05:00   00
3 2013-10-01 00:13:13   00
4 2013-10-01 00:15:00   00
5 2013-10-01 00:24:56   00
6 2013-10-01 00:29:36   00
```

- The results of this Hive query gives us the hour associated with each pickup_datetime entry.

Taxi Data in R

- We can now build off of the query in order to create an aggregate count by hour.

```
query_count <- "
SELECT substring(pickup_datetime, 12, 2) AS hour, COUNT(*) AS count
FROM taxi_sample
GROUP BY substring(pickup_datetime, 12, 2)"
```

- Here is how we define the hour in sprintf:

```
query_count <- sprintf(
  "SELECT %s AS hour, COUNT(*) AS count
FROM %s
GROUP BY %s",
  field_hour, name_table, field_hour)
cat(query count)
```

- RHive query syntax:

```
SELECT substring(pickup_datetime, 12, 2) AS hour, COUNT(*) AS count
FROM taxi_sample
GROUP BY substring(pickup_datetime, 12, 2)
```

Taxi Data in R

- Finally, we have the query syntax developed and we can execute the Hive query:

```
head(rhive.query(query_count))
```

```
  hour count
1   00  6921
2   01  5101
3   02  3769
4   03  2784
5   04  2025
6   05  1742
```

- The results of this Hive query gives us the aggregate count of each hour associated with each pickup_datetime entry.

Taxi Data in R

In Big Data example on NY Taxi Cab data, we:

- Developed scripts in R with the rmr2 package.
- Constructed mappers.
- Constructed reducers.
- Deployed mapreduce() functions.

Then we:

- Returned the data to our local R session and plotted results with ggplot2.
- Uploaded files into hdfs using rhdfs.
- Ran the script directly in Hadoop, using the Hadoop context.

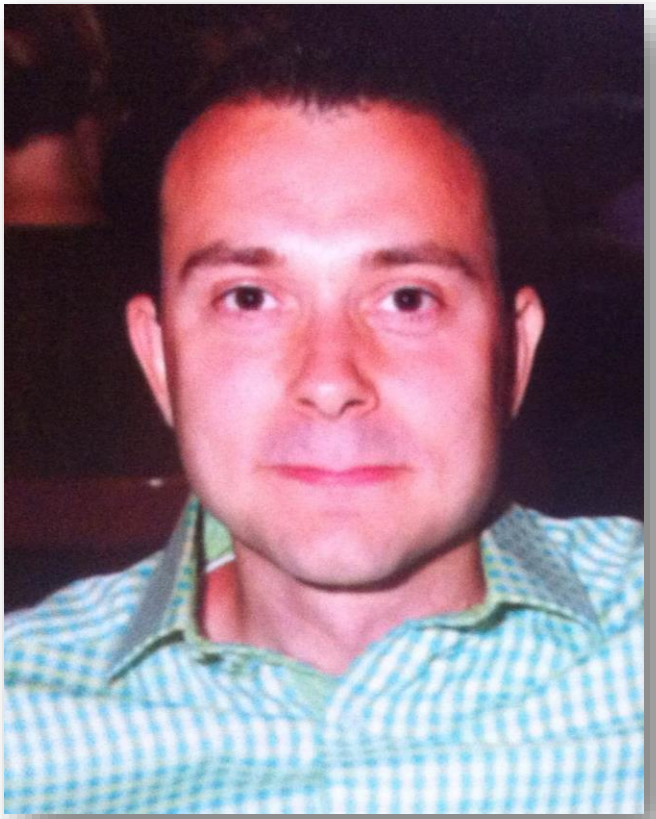
Finally we:

- Constructed the same example as before utilizing Hive through the Rhive package.

- This practical example showcases how we can integrate our data science techniques in R within a Big Data ecosystem.

About Me

- Reside in Wayne, Illinois
- Active Semi-Professional Classical Musician (Bassoon).
- Married my wife on 10/10/10 and been together for 10 years.
- Pet Yorkshire Terrier / Toy Poodle named Brunzie.
- Pet Maine Coons' named Maximus Power and Nemesis Gul du Cat.
- Enjoy Cooking, Hiking, Cycling, Kayaking, and Astronomy.
- Self proclaimed Data Nerd and Technology Lover.



Acknowledgements

- https://msdn.microsoft.com/en-us/library/dn749804.aspx
- https://msdn.microsoft.com/en-us/library/dn749868.aspx
- https://msdn.microsoft.com/en-us/library/dn749785.aspx
- https://msdn.microsoft.com/en-us/library/dn749852.aspx
- http://hortonworks.com/hdp/
- http://htmlpreview.github.io/?https://github.com/andrie/RHadoop-tutorial/blob/master/1-Using-R-with-Hadoop.html
- http://htmlpreview.github.io/?https://github.com/andrie/RHadoop-tutorial/blob/master/2-Taxi-analysis-with-RHadoop.html#/1
- http://htmlpreview.github.io/?https://github.com/andrie/RHadoop-tutorial/blob/master/4-Computing-on-distributed-matrices.html#/
- http://htmlpreview.github.io/?https://github.com/andrie/RHadoop-tutorial/blob/master/5-hive.html#/
- http://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator
- http://www.infoworld.com/article/2609513/big-data/yarn-unwinds-mapreduce-s-grip-on-hadoop.html
- http://chriswhong.com/open-data/foil_nyc_taxi/
- https://storm.apache.org/
- https://storm.apache.org/documentation/Tutorial.html

Acknowledgements

- https://en.wikipedia.org/wiki/Apache_Kafka
- https://en.wikipedia.org/wiki/Apache_HBase
- http://hortonworks.com/hadoop/HBase/
- https://mahout.apache.org/general/faq.html
- https://hive.apache.org/
- https://cwiki.apache.org/confluence/display/Hive/Home
- https://pig.apache.org/
- http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- http://hadoop.apache.org/
- https://en.wikipedia.org/wiki/Apache_Hadoop
- http://dataconomy.com/wp-content/uploads/2014/06/Understanding-Big-Data-The-Ecosystem.png
- http://kafka.apache.org/documentation.html#introduction
- http://mohamednabeel.blogspot.com/2011/03/starting-sub-sandwitch-business.html
- https://www.jasondavies.com/bloomfilter/
- http://www.rohitmenon.com/

Acknowledgements

- https://gigaom.com/2014/03/27/apache-mahout-hadoops-original-machine-learning-project-is-moving-on-from-mapreduce/
- http://oozie.apache.org/
- http://hortonworks.com/hadoop/oozie/
- http://kafka.apache.org/documentation.html#uses
- http://www.confluent.io/blog/stream-data-platform-1/

Fine