Big Data 2.0 Processing Systems: Technologies, Challenges and Opportunities

Sherif Sakr

2nd International Forum on Research and Technologies for Society and Industry

(RTSI 2016)
Bologna, Italy
7-9 September 2016

Today's Agenda

- **Big Data Phenomena**
- **Big Data 1.0 Systems**
    - Hadoop
    - Hadoop Extensions
- **Big Data 2.0 Systems**
    - General-Purpose Systems
    - Big SQL Systems
    - Big Stream Processing Systems
    - Big Graph Processing Systems
- **Open Challenges**

Part I

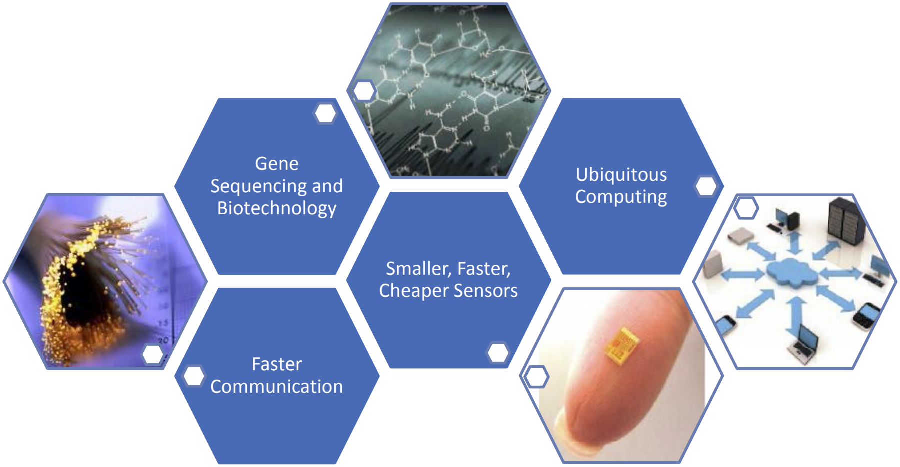
Big Data Phenomena

Big Data

- Data is key resource in the modern world.
- According to IBM, we are currently creating 2.5 quintillion bytes of data everyday.
- IDC predicts that the world wide volume of data will reach 40 zettabytes by 2020.
- The radical expansion and integration of computation, networking, digital devices and data storage has provided a robust platform for the explosion in big data.

"Data will be the most crucial asset for enterprises."
YAN LIDA
President, Huawei Enterprise BG
#HCC2015
HUAWEI

On the Verge of A Disruptive Century: Breakthroughs


Gene Sequencing and Biotechnology
Ubiquitous Computing
Smaller, Faster, Cheaper Sensors
Faster Communication

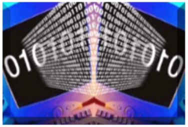
Big Data Applications are Everywhere

Smarter Healthcare

Multi-channel sales

Finance

Log Analysis



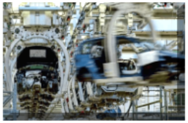
Homeland Security

Traffic Control

Telecom

Search Quality

Manufacturing



Trading Analytics

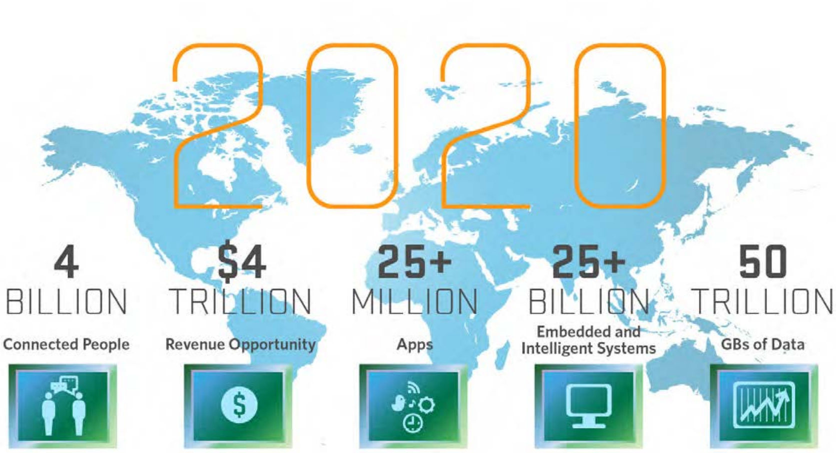
Fraud and Risk

Retail: Churn, NBO

Big Data

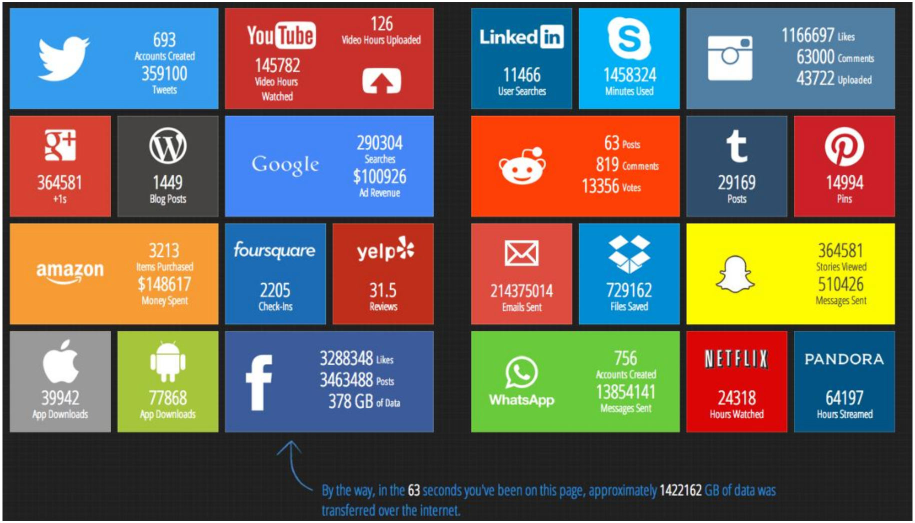
- Data generation and consumption is becoming a main part of people's daily life especially with the pervasive availability and usage of Internet technology and applications.

Big Data



Source: Mario Morales, IDC

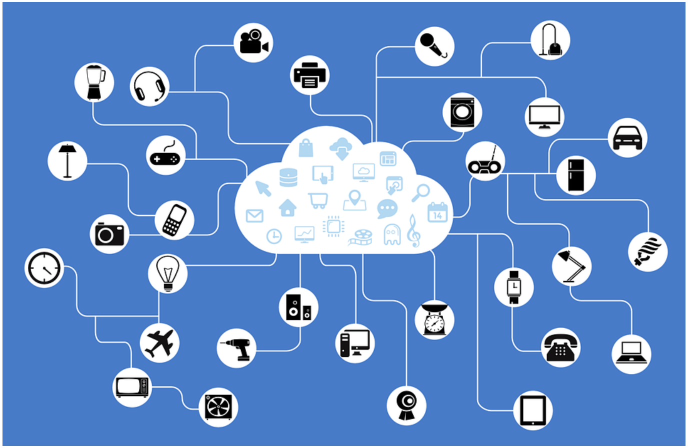
Big Data: What Happens in the Internet in a Minute?



Your Smart Phone is now Very smart

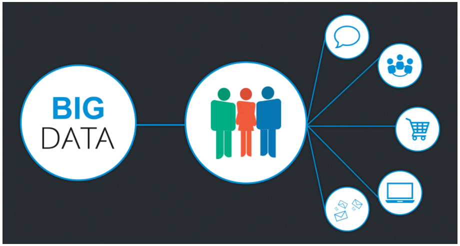
Big Data: Internet of Things

- We are witnessing radical expansion and integration of digital devices, networking, data storage and computation systems.
- We now have smart TVs that are able to collect and process data, we have smart watches, smart fridges, and smart alarms.



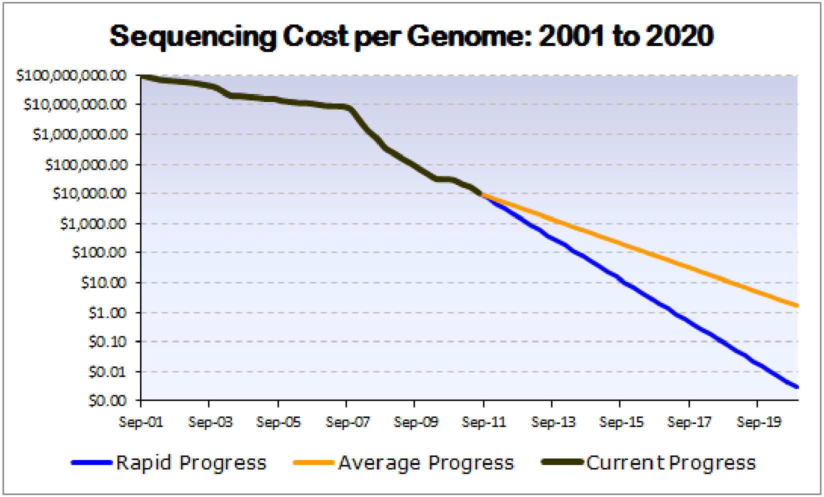
Big Data: Activity Data

- Simple activities like listening to music or reading a book are now generating data.
- Digital music players and eBooks collect data on our activities.
- Your smart phone collects data on how you use it and your web browser collects information on what you are searching for.
- Your credit card company collects data on where you shop and your shop collects data on what you buy.
- **It is hard to imagine any activity that does not generate data.**



Big Data

- The cost of sequencing one human genome has fallen from $100 million in 2001 to $1K in 2015



New Types of Data

Sentiment

understand how customers feel about your brand and products -right wow

Clickstream

Capture and analyze website visitors' data trails and optimize your website

Sensors

Discover patterns in data streaming automatically from remote sensors and machines

Geographic

Analyze location based data to manage operations where they occur

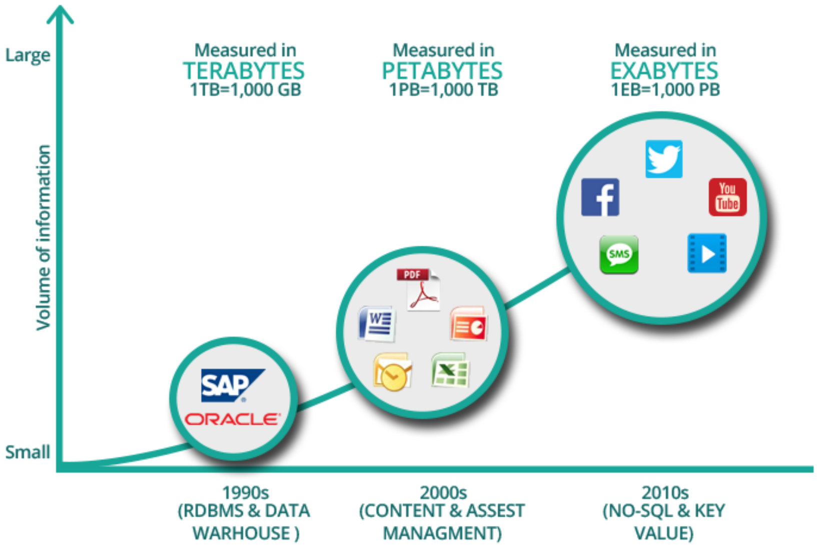
Server Logs

Research logs to diagnose process failures and prevent security breaches

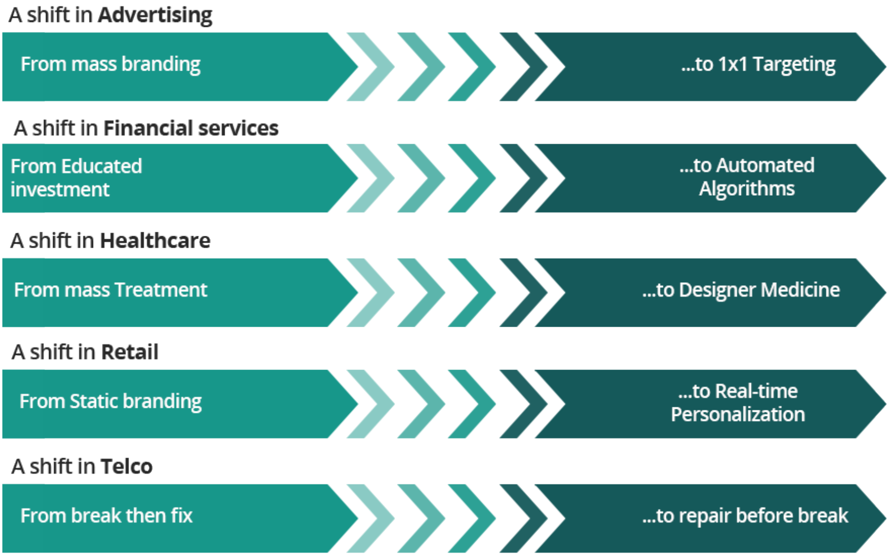
Unstructured

Understand patterns in files across millions of web pages, emails, and documents

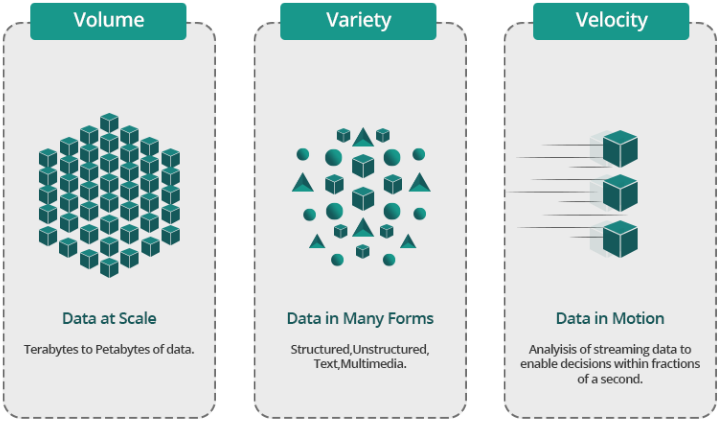
The Data Structure Evolution Over the Years


Large
Small
Volume of information
Measured in TERABYTES 1TB=1,000 GB
Measured in PETABYTES 1PB=1,000 TB
Measured in EXABYTES 1EB=1,000 PB
SAP
ORACLE
PDF
SMS
You Tube
1990s (RDBMS & DATA WARHOUSE)
2000s (CONTENT & ASSEST MANAGMENT)
2010s (NO-SQL & KEY VALUE)

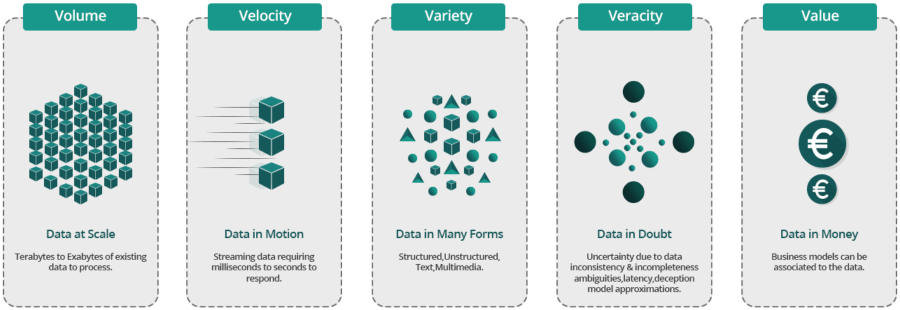
Shift in Application Requirements



Big Data (3V)


Volume
Data at Scale
Terabytes to Petabytes of data.
Variety
Data in Many Forms
Structured,Unstructured, Text,Multimedia.
Velocity
Data in Motion
Analyisis of streaming data to enable decisions within fractions of a second.

Big Data (5V)



Volume

Data at Scale

Terabytes to Exabytes of existing data to process.

Velocity

Data in Motion

Streaming data requiring milliseconds to seconds to respond.

Variety

Data in Many Forms

Structured,Unstructured, Text,Multimedia.

Veracity

Data in Doubt

Uncertainty due to data inconsistency & incompleteness ambiguities,latency,deception model approximations.

Value

Data in Money

Business models can be associated to the data.

Big Data

From the dawn of civilization until 2003, humankind generated five exabytes of data. Now we produce five exabytes every two days…and the pace is accelerating.
Eric Schmidt,
Executive Chairman, Google

Big Data Definition

- McKinsey global report described big data as *the next frontier for innovation and competition*.
- The report defined big data as "*Data whose scale, distribution, diversity, and/or timeliness require the use of new technical architectures and analytics to enable insights that unlock the new sources of business value*"

Research Report

Big Data: The Next Frontier for Innovation

McKinsey & Company

Big Data Revolution

ROB THOMAS & PATRICK McSHARRY

BIG DATA REVOLUTION

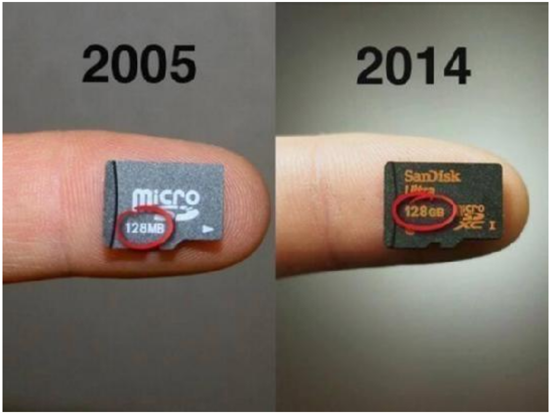
WHAT FARMERS, DOCTORS AND INSURANCE AGENTS TEACH US ABOUT DISCOVERING BIG DATA PATTERNS

WILEY

IBM 5MB Hard Disk ;-)

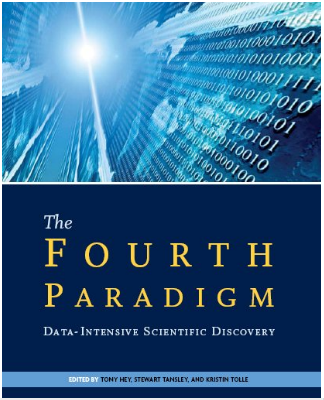
Big Data

- **Moore's Law**: The information density on silicon integrated circuits double every 18 to 24 months
- Users expect more sophisticated information



Fourth Paradigm

- Jim Gray, a database pioneer, described the big data phenomena as the **Fourth Paradigm** and called for a paradigm shift in the computing architecture and large scale data processing mechanisms.
- The first three paradigms were *experimental*, *theoretical* and, more recently, *computational science*


The
FOURTH
PARADIGM
DATA-INTENSIVE SCIENTIFIC DISCOVERY
EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE

Computing Clusters

- Many racks of computers, thousands of machines per cluster.
- Limited bisection bandwidth between racks.

Data Centers

Part II

Big Data 1.0 System: The Hadoop Decade

A Little History: Two Seminal contributions

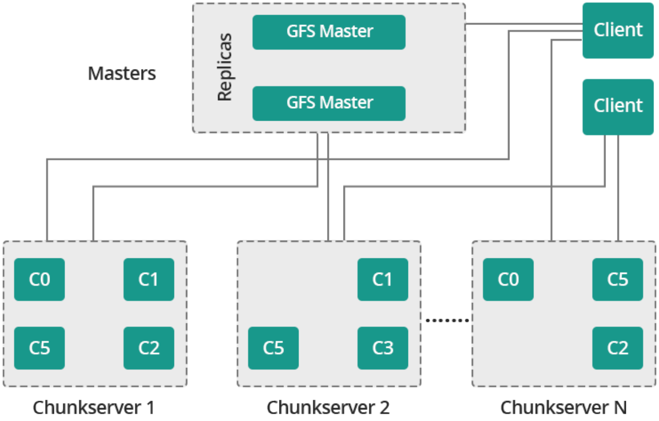
- "**The Google File System**"[1]
    - Describes a scalable, distributed, fault-tolerant file system tailored for data-intensive applications, running on inexpensive commodity hardware, delivers high aggregate performance
- "**MapReduce: Simplified Data Processing on Large Clusters**"[2]
    - Describes a simple programming model and an implementation for processing large data sets on computing clusters.

[1]S. Ghemawat, H. Gobioff, S. Leung. *The Google file system*. SOSP 2003

[2]J. Dean, S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI 2004

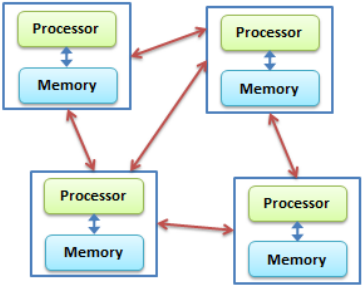
The Architecture of Google File System

- Master manages metadata
- Files broken into chunks (typically 64MB)
- Chunks are replicated across three machinery for fault-tolerance
- Data transfer happens directly between the clients and chunkserves.



What is MapReduce?

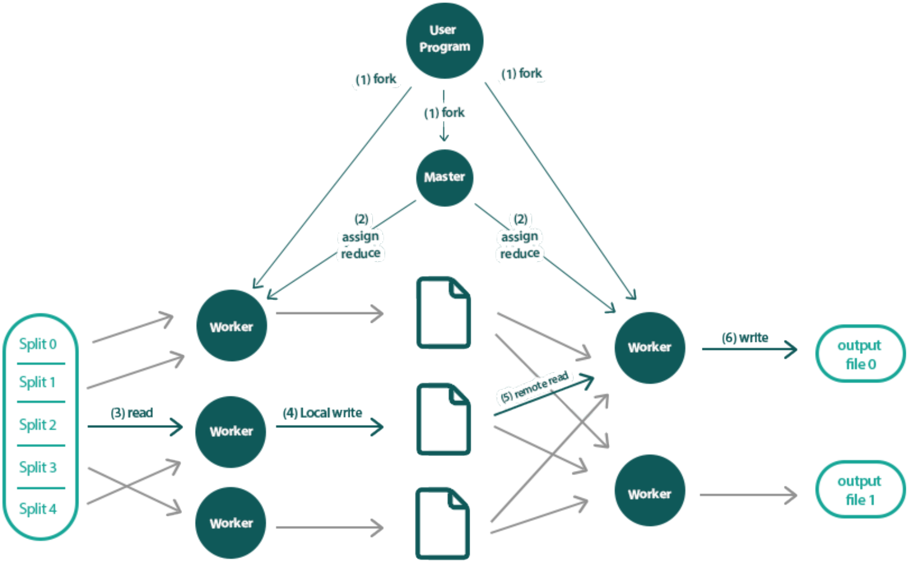
- A simple and powerful programming model that enables easy development of scalable parallel applications to process vast amounts of data on large clusters of commodity machines
- Hide messy details in distributed programming:
  - Automatic parallelization
  - Load balancing
  - Network and disk transfer optimization
  - Handling of machine failures



MapReduce's Programming Model

- The computation takes a set of key/value pairs input and produces a set of key/value pairs as output.
- The computations are expressed using two functions: **Map** and **Reduce**.
- The **Map** function takes an input pair and produces a set of intermediate key/value pairs.
- The MapReduce framework groups together all intermediate values associated with the same intermediate key **I** and passes them to the **Reduce** function.
- The **Reduce** function receives an intermediate key **I** with its set of values and merges them together.

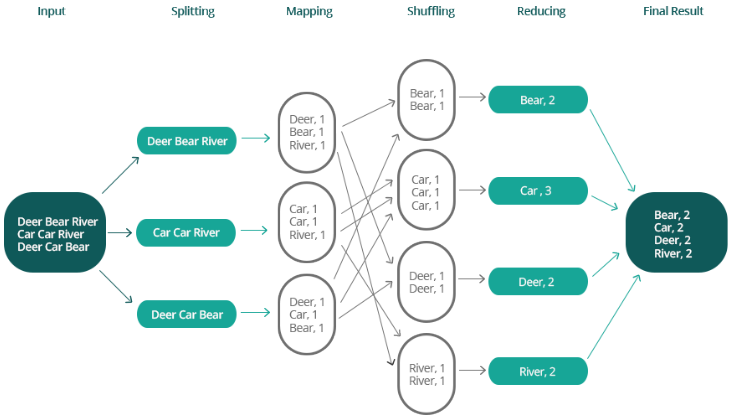
MapReduce's Execution Architecture



MapReduce's Programming Example

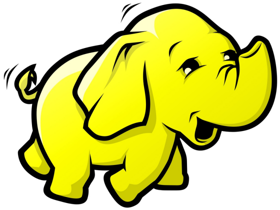
```
map( String key, String value ):
// key: document name
// value: document contents
for each word w in value:
        EmitIntermediate( w, "1" );
```

```
reduce( String key, Iterator values ):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
        result += ParseInt( v );
Emit( AsString(result ));
```



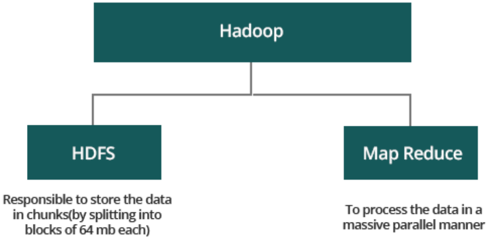
Hadoop[3]: A Star is Born

- Hadoop is an **open-source** software framework that supports data-intensive distributed applications and **clones** the Google's MapReduce framework.
- It is designed to process very large amount of unstructured and complex data.
- It is designed to run on a large number of machines that don't share any memory or disks.
- It is designed to run on a cluster of machines which can put together in relatively lower cost and easier maintenance.

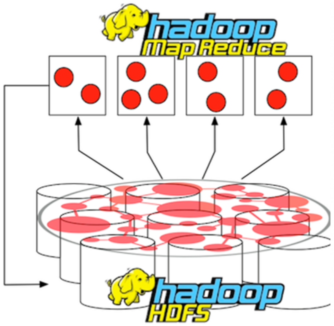


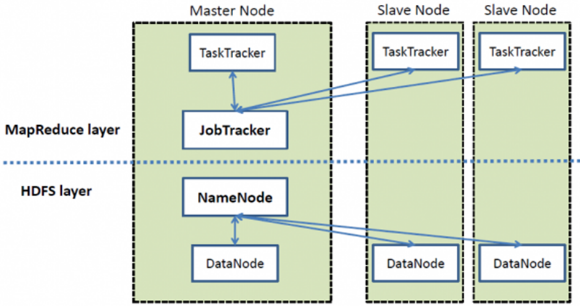
[3]http://hadoop.apache.org/

Hadoop = HDFS + MapReduce



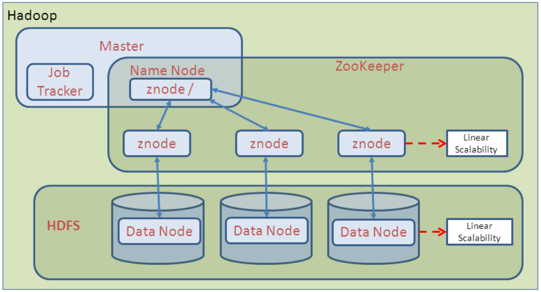
Hadoop
HDFS
Map Reduce
Responsible to store the data in chunks(by splitting into blocks of 64 mb each)
To process the data in a massive parallel manner





Zookeeper[4]

- An Open source, High Performance coordination service for distributed applications
- Centralized service for
    - Configuration Management
    - Locks and Synchronization for providing coordination between distributed systems
    - Naming service (Registry)
    - Group Membership



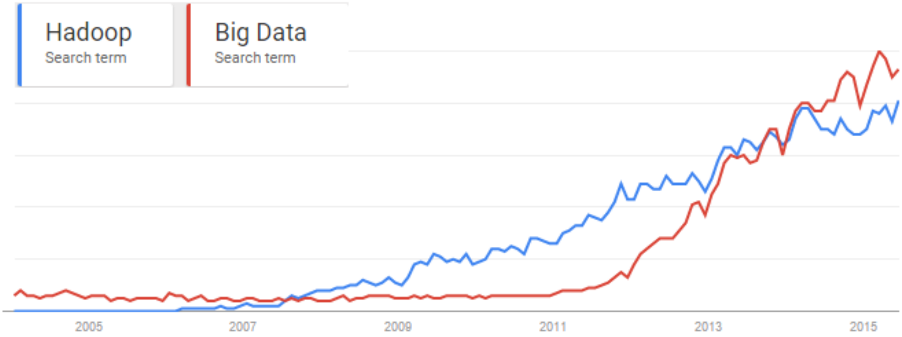
[4] http://zookeeper.apache.org/

Hadoop's Success

Big Data 1.0 = Hadoop

Hadoop's Success[5]

Big Data 1.0 = Hadoop



[5] https://www.google.com/trends/

Hadoop's Enhancments[6]

- The basic architecture of MapReduce/Haddop framework suffered from some limitations.
- Several research efforts that have been conducted in order to deal with these limitations by providing various enhancements.
    - Processing Join Operations
    - Supporting Iterative Processing
    - Data and Process Sharing
    - Data Indices
    - Effective Data Placement
    - Query Optimization

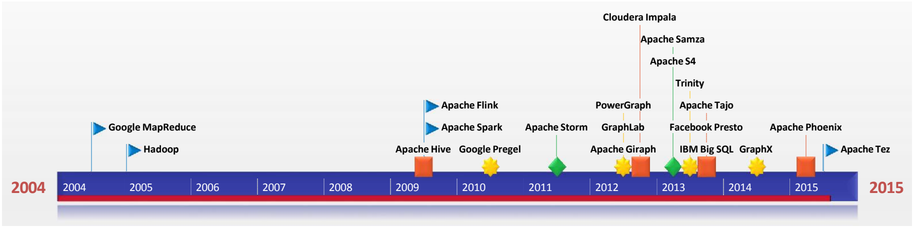
[6] S. Sakr, A. Liu, A. Fayoumi. *The family of mapreduce and large-scale data processing systems*. ACM Comput. Surv, 2013.

The Always Dilemma: Does One Size Fit All?!

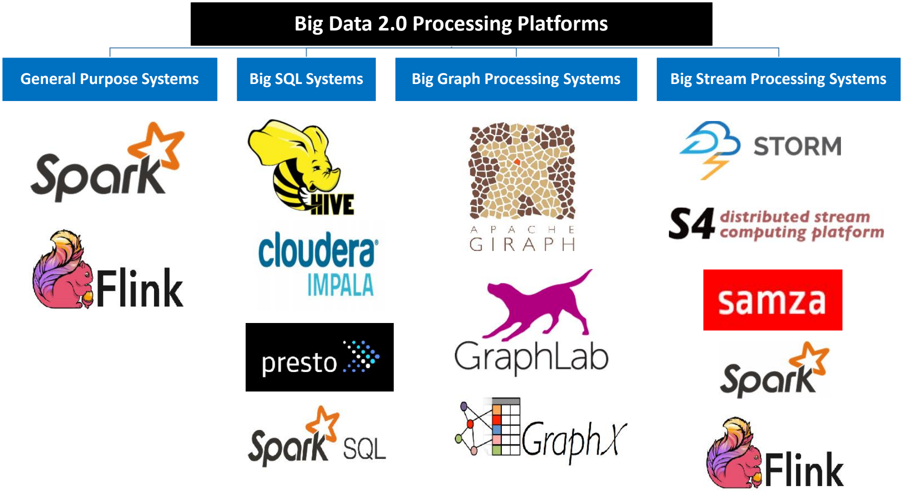
Big Data 2.0 Processing Systems

Big Data 2.0 != Hadoop

Domain-specific, optimized and vertically focused systems



An Overview of Big 2.0 Processing Systems

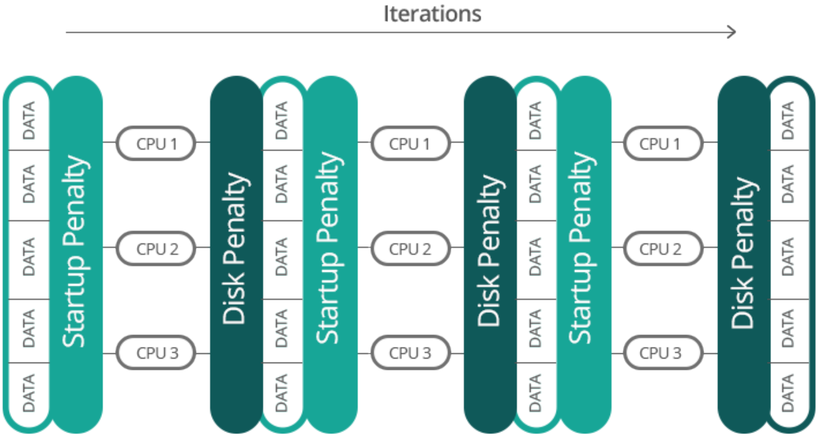

Big Data 2.0 Processing Platforms
General Purpose Systems
Big SQL Systems
Big Graph Processing Systems
Big Stream Processing Systems
Spark
Flink
HIVE
cloudera IMPALA
presto
Spark SQL
APACHE GIRAPH
GraphLab
GraphX
STORM
S4 distributed stream computing platform
samza
Spark
Flink

Part III

Big Data 2.0 Processing Systems: General-Purpose Processing Engines

MapReduce for Iterative Operations

MapReduce is **not optimized** for iterative operations



Spark[9]

- Apache Spark is a fast, general engine for large scale data processing on a computing cluster (new engine for Hadoop)[7]
- Developed initially at UC Berkeley, in 2009, in Scala, and is currently supported by Databricks[8]
- One of the most active and fastest growing Apache projects
- Committers from Cloudera, Yahoo, Databricks, UC Berkeley, Intel, Groupon and others.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: Cluster Computing with Working Sets*. HotCloud, 2010.

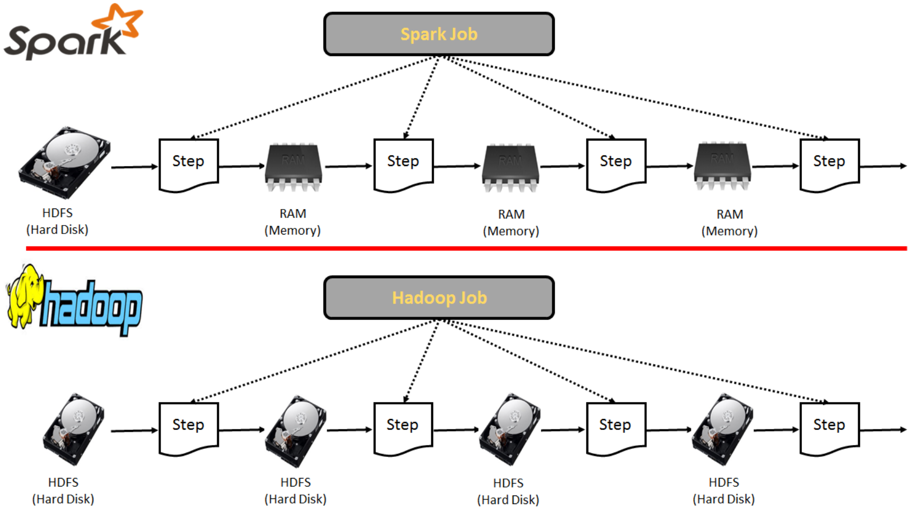
[8] `https://databricks.com/`

[9] `http://spark.apache.org/`

Spark

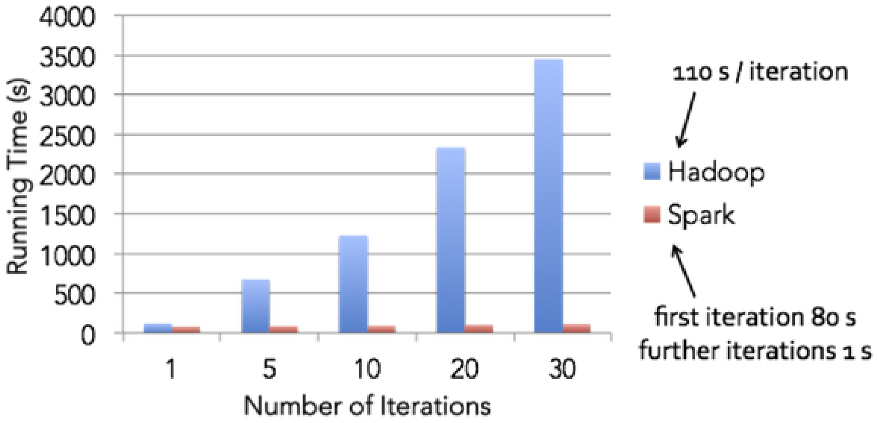
- **RDD (Resilient Distributed Dataset)**, an in-memory data abstraction, is the fundamental unit of data in Spark
- **Resilient**: if data in memory is lost, it can be recreated
- **Distributed**: stored in memory across the cluster
- **Dataset**: data can come from a file or be created programmatically
- Spark programming consists of performing operations (e.g., Map, Filter) on RDDs

Spark VS Hadoop



Spark VS Hadoop

- **Spark takes the concepts and performance of MapReduce to the next level**



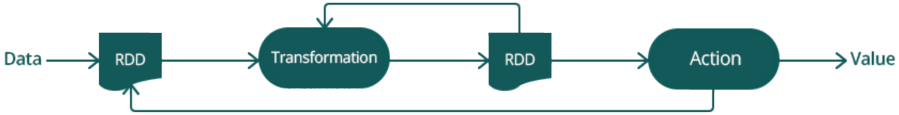
Spark VS Hadoop

- **Spark code is much more compact**

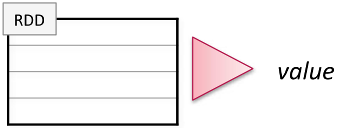
```
sc.textFile(file) \
  .flatMap(lambda s: s.split()) \
  .map(lambda w: (w,1)) \
  .reduceByKey(lambda v1,v2: v1+v2) \
  .saveAsTextFile(output)
```

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

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

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

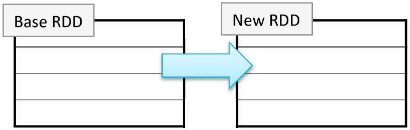
Flow of RDD Operations in Spark



– Actions – return values

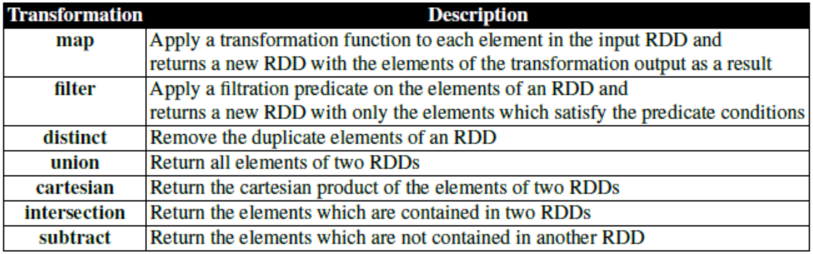


– Transformations – define a new RDD based on the current one(s)



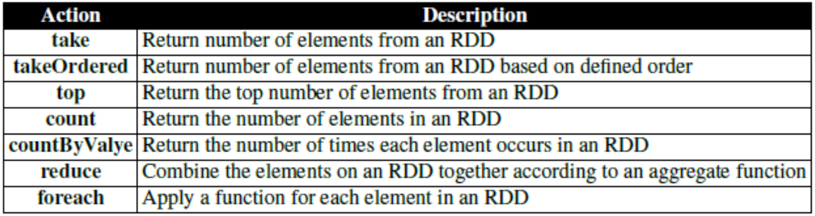
Spark's Transformations and Actions

Transformations



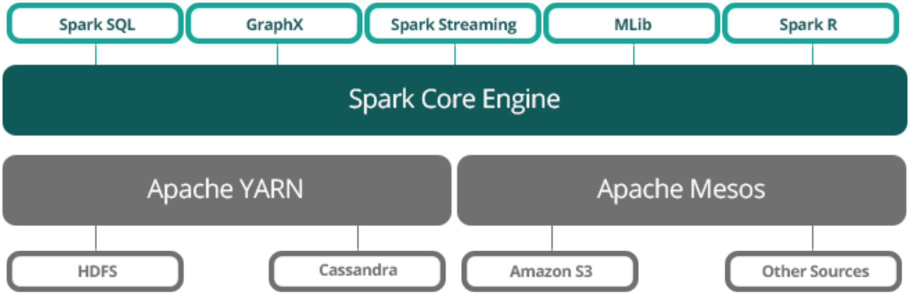
Transformation	Description
map	Apply a transformation function to each element in the input RDD and returns a new RDD with the elements of the transformation output as a result
filter	Apply a filtration predicate on the elements of an RDD and returns a new RDD with only the elements which satisfy the predicate conditions
distinct	Remove the duplicate elements of an RDD
union	Return all elements of two RDDs
cartesian	Return the cartesian product of the elements of two RDDs
intersection	Return the elements which are contained in two RDDs
subtract	Return the elements which are not contained in another RDD

Actions



Action	Description
take	Return number of elements from an RDD
takeOrdered	Return number of elements from an RDD based on defined order
top	Return the top number of elements from an RDD
count	Return the number of elements in an RDD
countByValye	Return the number of times each element occurs in an RDD
reduce	Combine the elements on an RDD together according to an aggregate function
foreach	Apply a function for each element in an RDD

Spark's Stack



Flink[12]

- Apache Flink[10] is a distributed in-memory data processing framework which represents a exible alternative for the MapReduce framework that supports both of batch and realtime processing.
- Flink has originated from the *Stratosphere* research project[11] that was started at the Technical University of Berlin in 2009 before joining the Apache's incubator in 2014
- Instead of the *map* and *reduce* abstractions, Flink uses a directed graph approach that leverages in-memory storage for improving the performance of the runtime execution.

[10]Flink is a German word that means "quick" or "nimble"

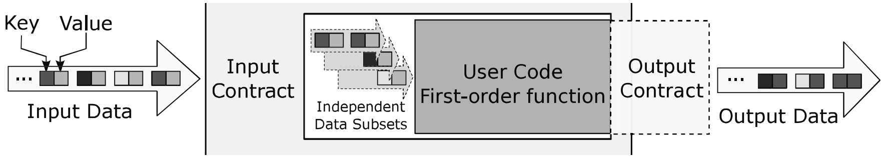
[11]`http://stratosphere.eu/`

[12]`https://flink.apache.org/`

Flink's Features

- **True streaming capabilities**: Execute everything as streams
- **Native iterative execution**: Allow some cyclic dataflows
- **Cost-Based Optimizer**: for both batch and stream processing
- **DataSet API for Static Data**: Java, Scala, and Python
- **DataStream API for Unbounded Real-Time Streams**: Java and Scala
- **Table API for Relational Queries**: Scala and Java

PACT Programming Model



- A PACT consists of exactly one second-order function which is called *Input Contract* and an optional *Output Contract*.
- An Input Contract (e.g., Cross, CoGroup, Match) takes a first-order function with task-specific user code and one or more data sets as input parameters and invokes its associated first-order function with independent subsets of its input data in a data-parallel fashion.
- An Output Contract (e.g., Same-Key, Super-Key, Unique-Key, Partitioned-by-Key) is an optional component of a PACT and gives guarantees about the data that is generated by the assigned user function.

PACT's Input Contracts

- The **Cross** contract which operates on multiple inputs and builds a distributed Cartesian product over its input sets.
- The **CoGroup** contract partitions each of its multiple inputs along the key. Independent subsets are built by combining equal keys of all inputs.
- The **Match** contract operates on multiple inputs. It matches key/value pairs from all input data sets with the same key (equivalent to the inner join operation).

PACT's Output Contracts

- The **Same-Key** contract where each key/value pair that is generated by the function has the same key as the key/value pair(s) from which it was generated. This means the function will preserve any partitioning and order property on the keys.
- The **Super-Key** where each key/value pair that is generated by the function has a superkey of the key/value pair(s) from which it was generated. This means the function will preserve a partitioning and partial order on the keys.
- The **Unique-Key** where each key/value pair that is produced has a unique key. The key must be unique across all parallel instances. Any produced data is therefore partitioned and grouped by the key.
- The **Partitioned-by-Key** where key/value pairs are partitioned by key. This contract has similar implications as the Super-Key contract, specifically that a partitioning by the keys is given, but there is no order inside the partitions.

Flink Examples

DataSet API (batch): WordCount

```
val env = ExecutionEnvironment.getExecutionEnvironment()
val lines: DataSet[String] = env.readTextFile(...)
lines.flatMap {line => line.split(" ")
                                  .map(word => Word(word,1))}
      .groupBy("word") sum("frequency")
      .print()
env.execute()
```

DataStream API (streaming): Window WordCount

```
val env = StreamExecutionEnvironment.getExecutionEnvironment()
val lines: DataStream[String] = env.fromSocketStream(...)
lines.flatMap {line => line.split(" ")
                                  .map(word => Word(word,1))}
      .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
      .groupBy("word").sum("frequency")
      .print()
env.execute()
```

Flink Examples

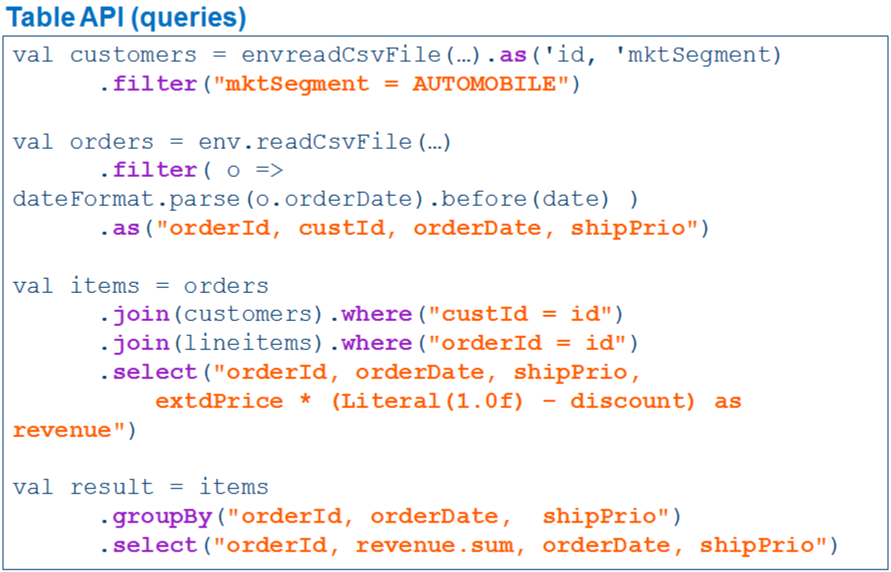


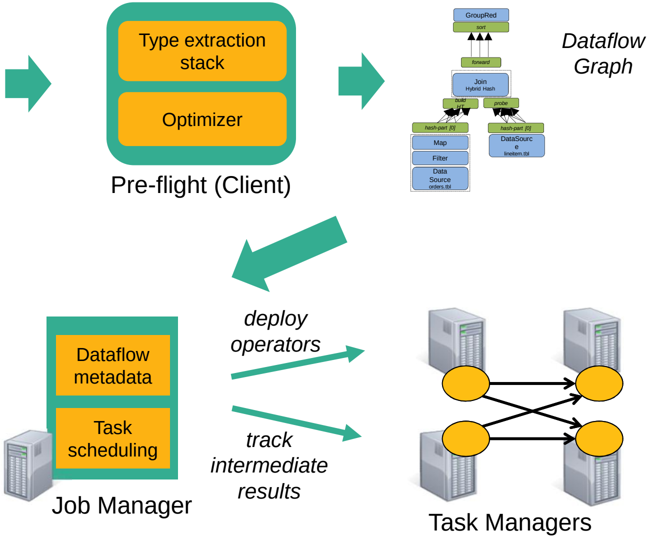
Table API (queries)

```
val customers = envreadCsvFile(…).as('id, 'mktSegment)
      .filter("mktSegment = AUTOMOBILE")

val orders = env.readCsvFile(…)
      .filter( o =>
dateFormat.parse(o.orderDate).before(date) )
      .as("orderId, custId, orderDate, shipPrio")

val items = orders
      .join(customers).where("custId = id")
      .join(lineitems).where("orderId = id")
      .select("orderId, orderDate, shipPrio,
         extdPrice * (Literal(1.0f) - discount) as
revenue")

val result = items
      .groupBy("orderId, orderDate,  shipPrio")
      .select("orderId, revenue.sum, orderDate, shipPrio")
```

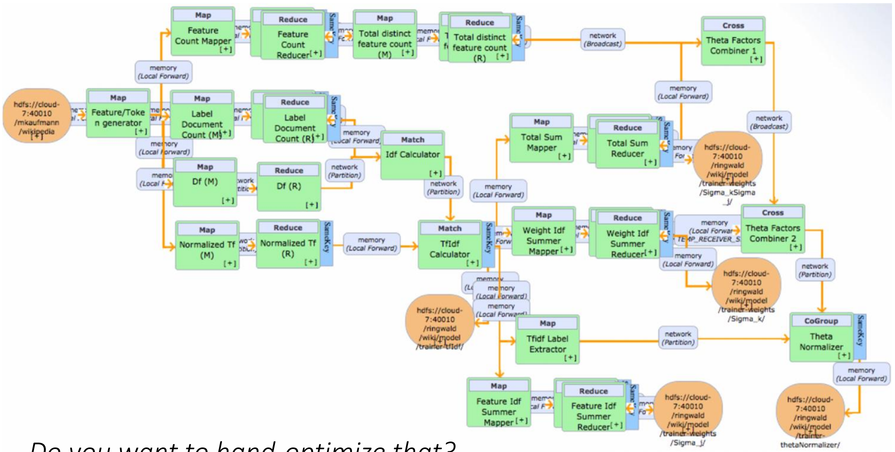
The Life Cycle of Flink's Program

```
case class Path (from: Long, to:
Long)
val tc = edges.iterate(10) {
  paths: DataSet[Path] =>
    val next = paths
      .join(edges)
      .where("to")
      .equalTo("from") {
        (path, edge) =>
          Path(path.from, edge.to)
      }
      .union(paths)
      .distinct()
    next
  }
```

Program



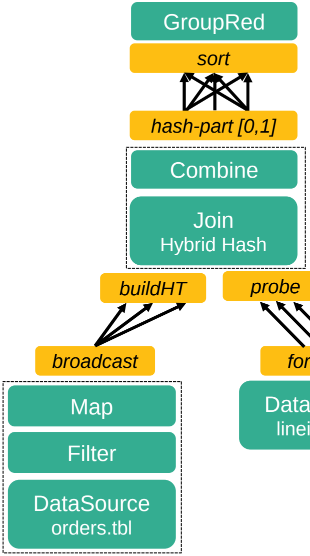
Automatic Program Execution is Important and Needed


hdfs://cloud-7:40010/mkaufmann/wikipedia
Map Feature/Token generator
Map Feature Count Mapper
Reduce Feature Count Reducer
Map Total distinct feature count (M)
Reduce Total distinct feature count (R)
network (Broadcast)
Cross Theta Factors Combiner 1
Map Label Document Count (M)
Reduce Label Document Count (R)
Match Idf Calculator
Map Df (M)
Reduce Df (R)
Map Normalized Tf (M)
Reduce Normalized Tf (R)
Match TfIdf Calculator
Map Total Sum Mapper
Reduce Total Sum Reducer
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_kSigma_j/
Map Weight Idf Summer Mapper
Reduce Weight Idf Summer Reducer
Cross Theta Factors Combiner 2
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_k/
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-tfidf/
Map Tfidf Label Extractor
CoGroup Theta Normalizer
Map Feature Idf Summer Mapper
Reduce Feature Idf Summer Reducer
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_j/
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-thetaNormalizer/
memory (Local Forward)
network (Partition)
SameKey

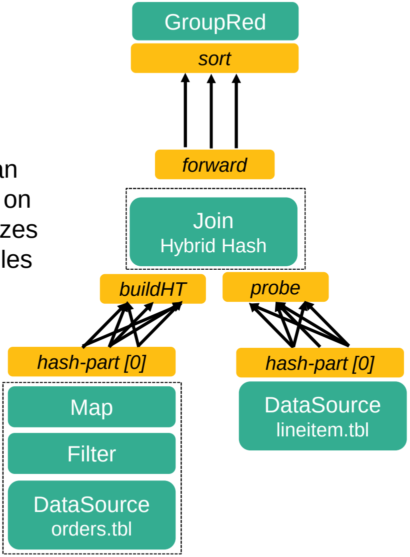
Do you want to hand-optimize that?

Automatic Program Execution is Important and Needed

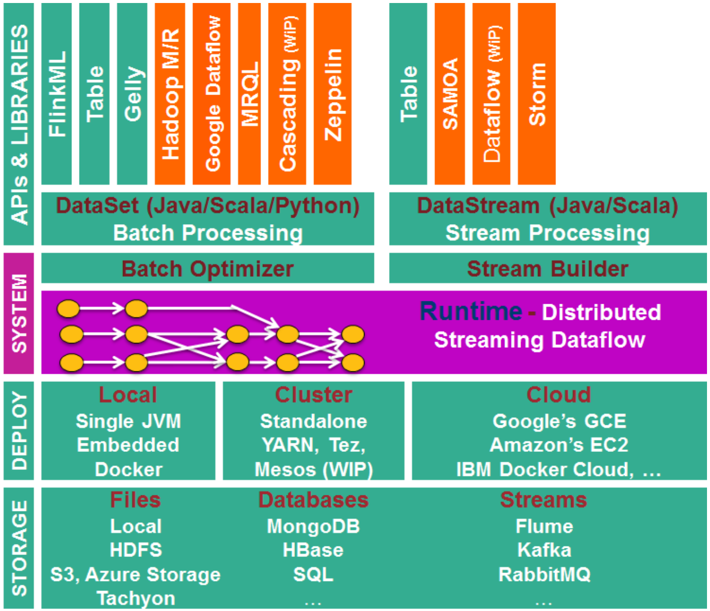
Automatic Program Execution is Important and Needed



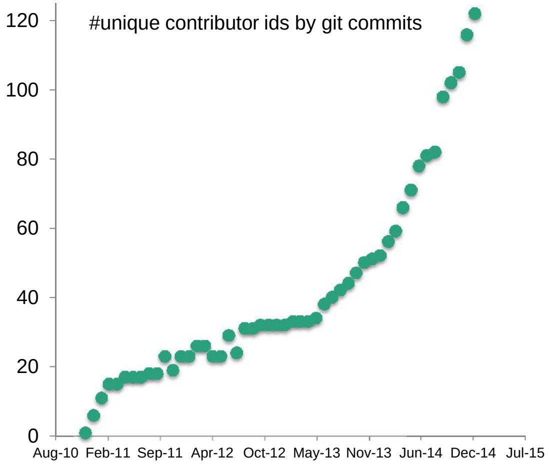
Best plan depends on relative sizes of input files



Flink's Stack



Flink Community



In top 5 of Apache's big data projects after one year in the Apache Software Foundation

Hadoop VS Spark VS Flink

	Hadoop	Spark	Flink
Year of Origin	2005	2009	2009
Place of Origin	MapReduce (Google) Hadoop (Yahoo)	UC Berkely	TU Berlin
Programming Model	Map and Reduce function over key/value pairs	RDD	PACT
Data Storage	HDFS	HDFS, Cassandra and others	HDFS, S3 and others
SQL Support	Hive, Impala, Tajo	Spark SQL	NA
Graph Support	NA	GraphX	Gelly
Streaming Support	NA	Spark Streaming	Flink Streaming

Part IV

Big Data 2.0 Processing Systems: SQL-On-Hadoop

Why SQL-On-Hadoop?

- Hadoop's one-input data format (key/value pairs) and two-stage data flow is extremely rigid. As we have previously discussed, to perform tasks that have a differen data flow (e.g. joins or *n* stages) would require the need to devise inelegant workarounds.
- Custom code has to be written for even the most common operations (e.g. projection and filtering).
- In practice, many programmers would prefer to use **SQL** as a high level declarative language to express their task while leaving all of the execution optimization details to the backend engine.
- High level language abstractions enable the underlying system to perform automatic optimization.
- Several studies[13] have reported that Hadoop is the wrong choice for interactive queries on large scale structured data with target response time of a few seconds or milliseconds.

[13] A. Pavlo et al. *A comparison of approaches to large scale data analysis*. SIGMOD 2009

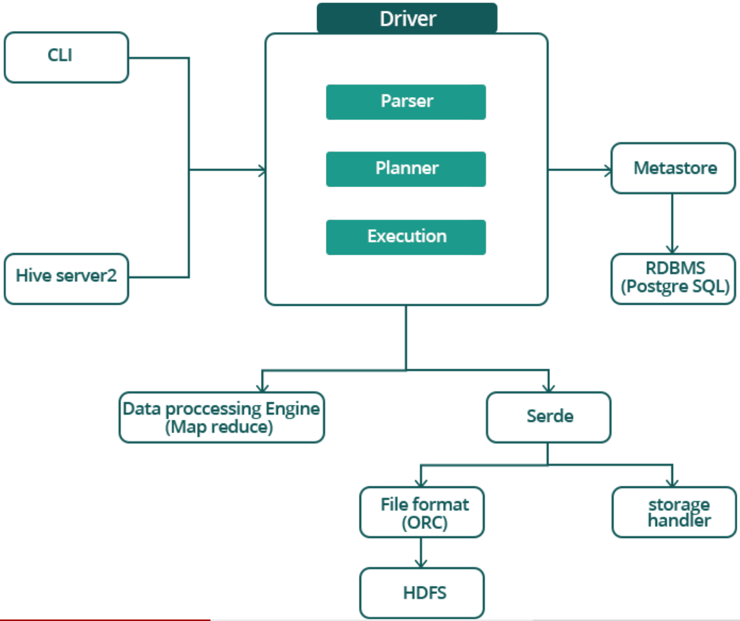
Apache Hive[15]

- The first system that has been introduced to support SQL-on-Hadoop with familiar relational database concepts such as tables and columns[14].
- Hive has been widely used in many organizations to manage and process large volumes of data, such as Facebook, eBay, LinkedIn and Yahoo!
- It supports an SQL-like declarative language, HiveQL, which represents a subset of SQL92 and therefore can be easily understood by anyone who is familiar with SQL.
- Hive queries automatically compile into MapReduce jobs that are run by using Hadoop.

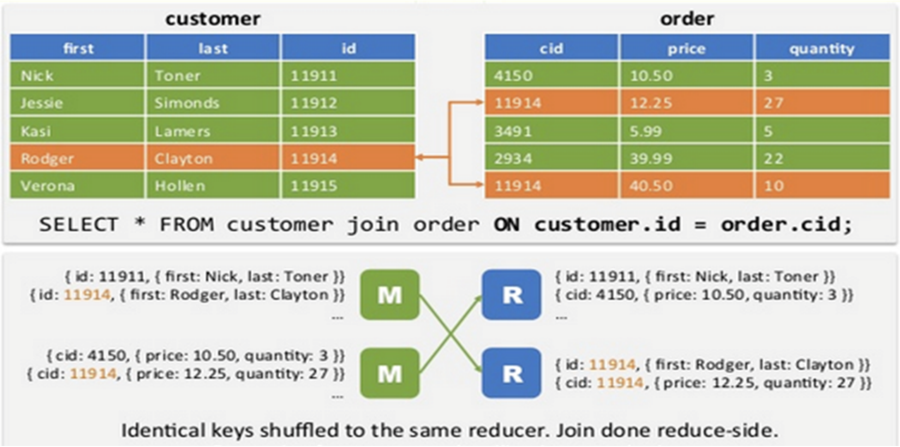
[14] A. Thusoo et al. *Data warehousing and analytics infrastructure at facebook*. SIGMOD 2010

[15] `https://hive.apache.org/`

Hive Architecture



Example: Joins in Hive



customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

```
SELECT * FROM customer join order ON customer.id = order.cid;
```

```
{ id: 11911, { first: Nick, last: Toner }}
{ id: 11914, { first: Rodger, last: Clayton }}
...

{ cid: 4150, { price: 10.50, quantity: 3 }}
{ cid: 11914, { price: 12.25, quantity: 27 }}
...
```

```
{ id: 11911, { first: Nick, last: Toner }}
{ cid: 4150, { price: 10.50, quantity: 3 }}
...

{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 }}
```

Identical keys shuffled to the same reducer. Join done reduce-side.

Impala[17]

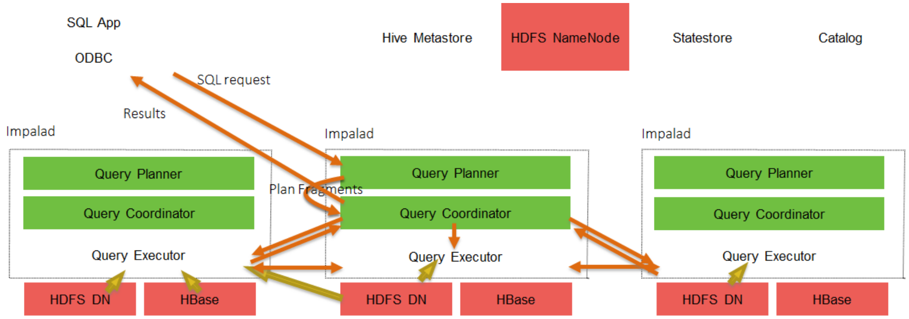
- Open source project, built by Cloudera, to provide a massively parallel processing SQL query engine that runs natively in Apache Hadoop[16].
- By using Impala, the user can query data which is stored in Hadoop Distributed File System (HDFS).
- It uses the same metadata, SQL syntax (HiveQL) of Apache Hive.
- Impala does not use the Hadoop execution engine to run the queries. Instead, it relies on its own set of *daemons* which are installed alongside the data nodes and are tuned to optimize the local processing to avoid bottlenecks.

[16] M. Kornacker et al. *Impala: A Modern, Open-Source SQL Engine for Hadoop*. CIDR 2015

[17] `http://impala.io/`

Impala Architecture

- The **Impala daemon (impalad)** that accepts queries from client processes and orchestrates their execution across the cluster.
- The **Statestore daemon (statestored)** is a meta-data publish-subscribe component which disseminates cluster-wide metadata to all Impala processes.
- **The Catalog daemon (catalogd)** serves as a catalog and metadata access repository and is responsible for broadcasting any changes to the system catalog as well.

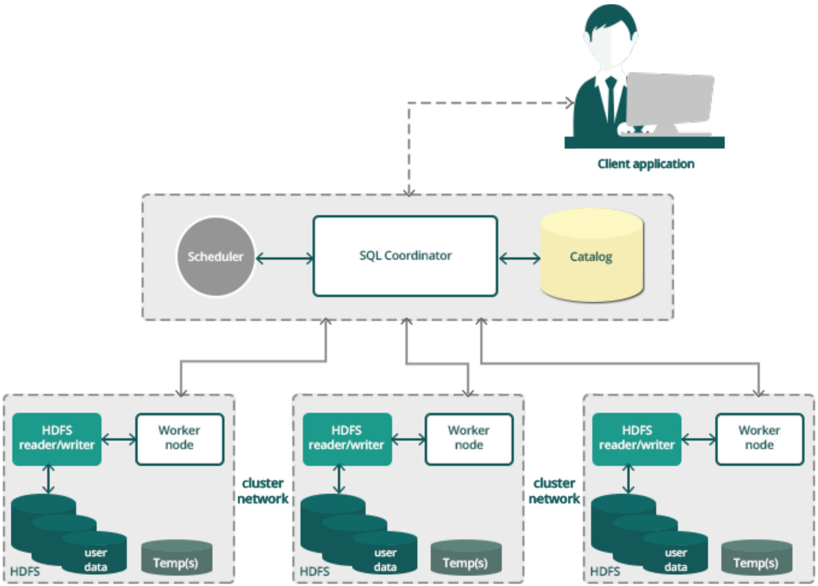


IBM Big SQL[18]

- The SQL interface for the IBM big data processing platform, InfoSphere BigInsights.
- Big SQL relies on a built-in query optimizer that rewrites the input query as a local job to help minimize latencies by using Hadoop dynamic scheduling mechanisms.
- The query optimizer also takes care of traditional query optimization such as optimal order, in which tables are accessed in the order where the most efficient join strategy is implemented for the query.
- It uses a massively parallel processing SQL engine that is deployed directly on the physical Hadoop Distributed File System (HDFS).

[18] `http://www-01.ibm.com/software/data/infosphere/hadoop/big-sql.html`

Big SQL Architecture

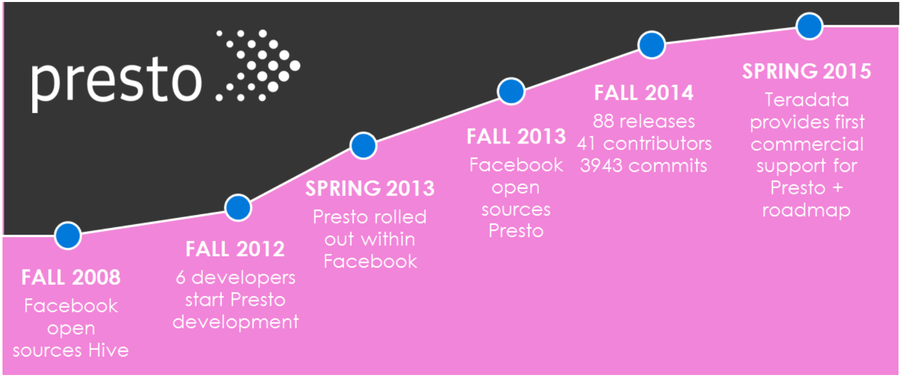


Presto[19]

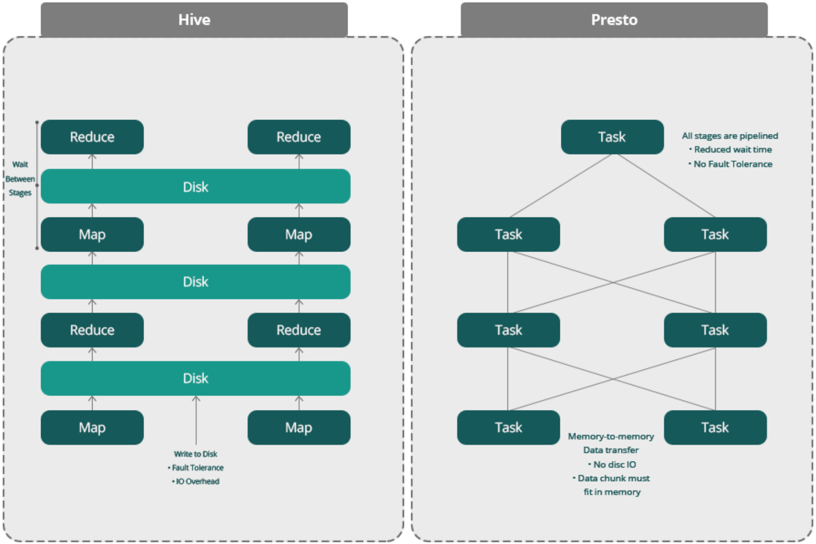
- Open source distributed SQL query engine, built by Facebook, for running interactive analytic queries against large scale structured data sources of sizes of gigabytes up to petabytes.
- Presto allows querying data where it lives, including Hive, NoSQL databases (e.g., Cassandra, HBase), relational databases or even proprietary data stores.
- A single Presto query can combine data from multiple sources.
- Presto has been recently adopted by big companies and application such as Netflix and Airbnb.

[19] `http://prestodb.io/`

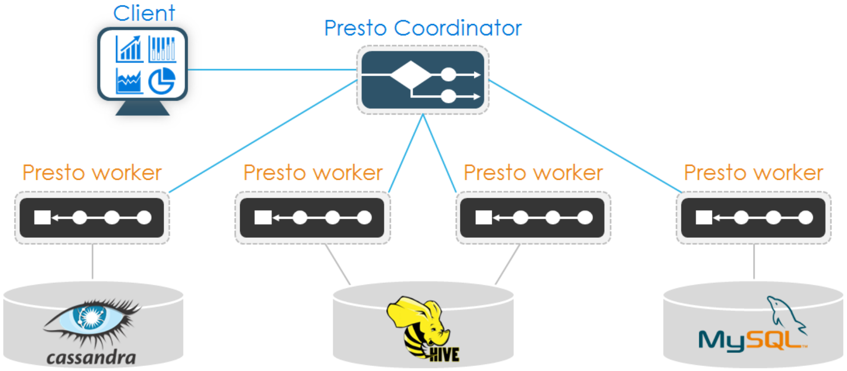
Presto's Timeline


presto
FALL 2008
Facebook open sources Hive
FALL 2012
6 developers start Presto development
SPRING 2013
Presto rolled out within Facebook
FALL 2013
Facebook open sources Presto
FALL 2014
88 releases
41 contributors
3943 commits
SPRING 2015
Teradata provides first commercial support for Presto + roadmap

Presto Architecture



Presto Architecture



Spark SQL[20]

- An alternative interface for Spark that integrates relational processing with Spark's functional programming API.
- SparkSQL bridges the gap between the two models by providing a *DataFrame* API that can execute relational operations on both external data sources and Spark's built-in distributed collections.
- DataFrames are collections of structured records that can be manipulated using Spark's procedural API, or using new relational APIs that allow richer optimizations.

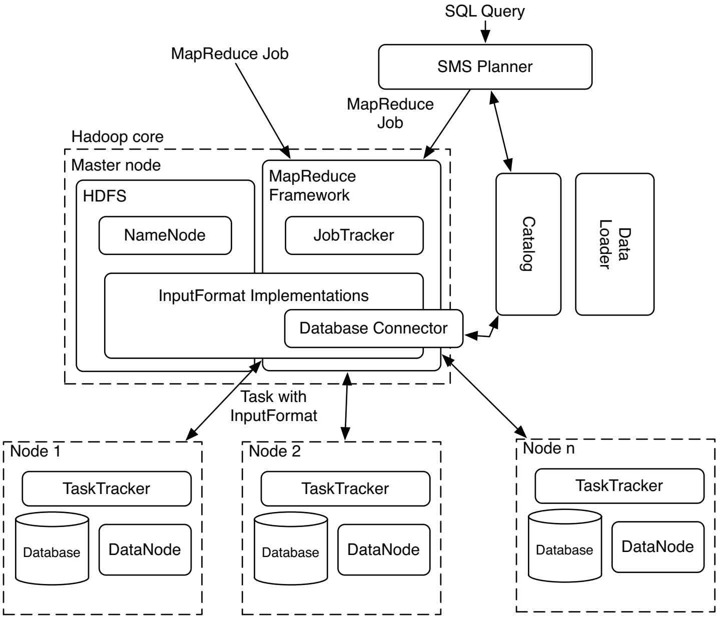
[20]M, Armbrust et al. Spark SQL: Relational Data Processing in Spark. In SIGMOD, 2015.

HadoopDB[21]

- *HadoopDB* is a hybrid system which is designed to attempt combining the scalability advantages of Hadoop framework with the performance and efficiency merits of parallel databases, Acquired by Teradata.
- HadoopDB clusters multiple single node database systems (PostgreSQL) using Hadoop as the task coordinator and network communication layer.
- Queries are expressed in SQL but their execution are parallelized across nodes using the MapReduce framework and as much as possible is pushed inside of the corresponding node databases.
- HadoopDB achieves fault tolerance and the ability to operate in heterogeneous environments by inheriting the scheduling and job tracking implementation from Hadoop. Parallelly, it tries to achieve the performance of parallel databases by doing most of the query processing inside the database engine.

[21] `http://db.cs.yale.edu/hadoopdb/hadoopdb.html`

HadoopDB Architecture



Other SQL-On-Hadoop Systems

- Apache Phoenix[22]
- Apache Drill[23]
- Actian Vortex[24]
- HP Vertica
- Pivotal HAWQ

[22] https://phoenix.apache.org/

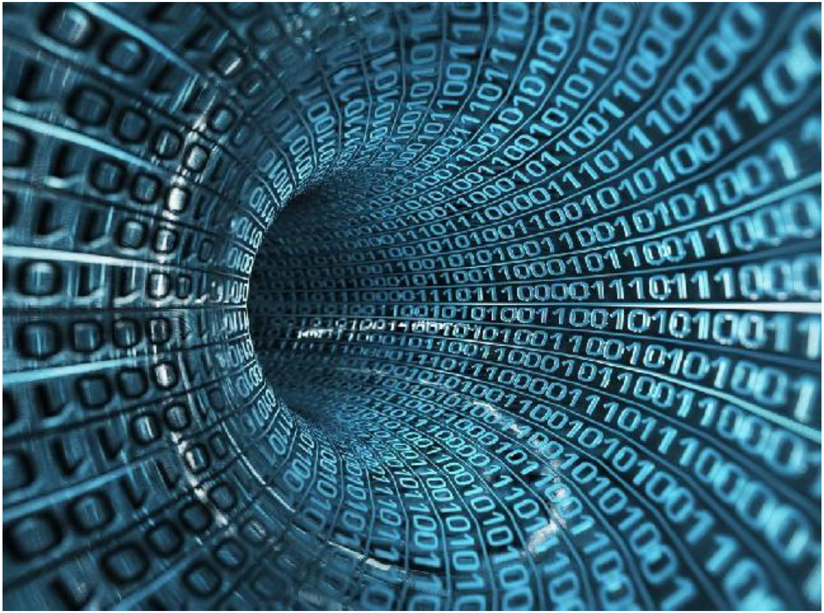
[23] https://drill.apache.org/

[24] http://www.actian.com/products/analytics-platform/vortex-sql-hadoop-analytics/

Part V

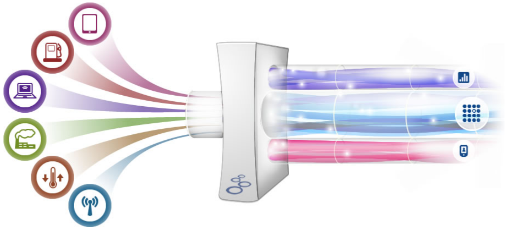
Big Data 2.0 Processing Systems: Big Stream Processing Systems

Big Streams

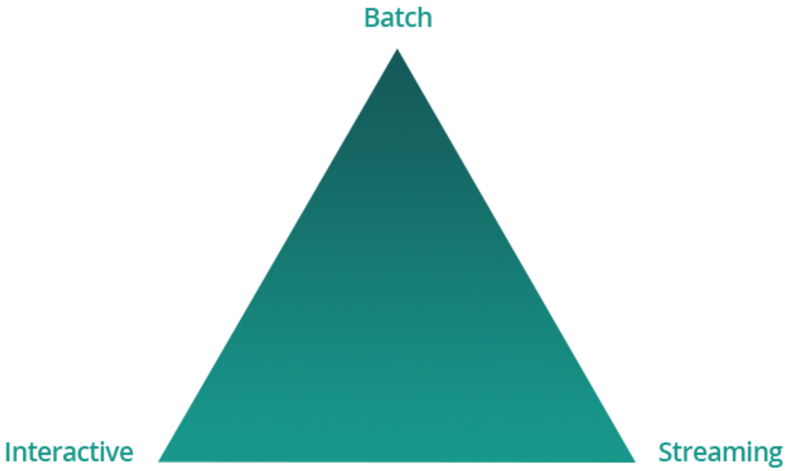


Big Streams

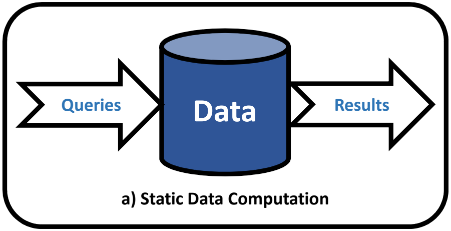
- In 2010, Walmartreported that it was handling more than 1 million customer transaction every hour.
- The New York Stock Exchange (NYSE) reported trading more than 800 million shares on a typical day in October 2012.
- By the end of 2011, there were about 30 billion Radio-Frequency Identification (RFID) tags.
- In all of these applications and domains, there is a crucial requirement to collect, process and analyse big streams of data in real time fashion.

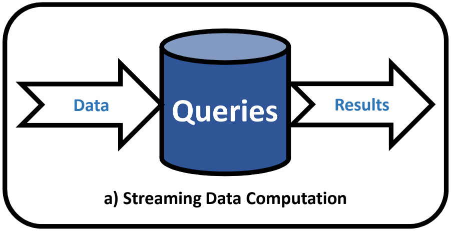


The Triad of Big Data Processing



Static Data Computation VS Streaming Data Computation





- Today, in several applications data is continuously produced (e.g., user activity logs, web logs, sensors, database transactions, ...).
- The traditional approaches to analyze such data are:
  - Record data stream to stable storage (DBMS, HDFS,...)
  - Periodically analyze data with batch processing engine (DBMS, MapReduce, ...)
- **Streaming processing engines analyze data while it arrives**
- The main goal of stream processing is to decrease the overall latency to obtain results

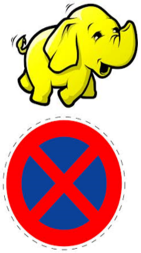
Stream Processing Vs Batch Processing

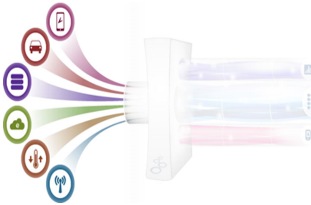
Factors		Stream Processing	Batch Processing
Data	**Freshness**	Real-time (usually < 15 min)	Historical – usually more than 15 min old
	Location	Primarily memory (moved to disk after processing)	Primarily in disk moved to memory for processing
Processing	**Speed**	Sub second to few seconds	Few seconds to hours
	Frequency	Always running	Sporadic to periodic
Clients	**Who?**	Automated systems only	Human & automated systems
	Type	Primarily operational systems	Primarily analytical applications

Hadoop for Big Streams?!

- From the stream-processing point of view, the main limitation of Hadoop is that it was designed so that the entire output of each map and reduce task is **materialized** into a local file before it can be consumed by the next stage.

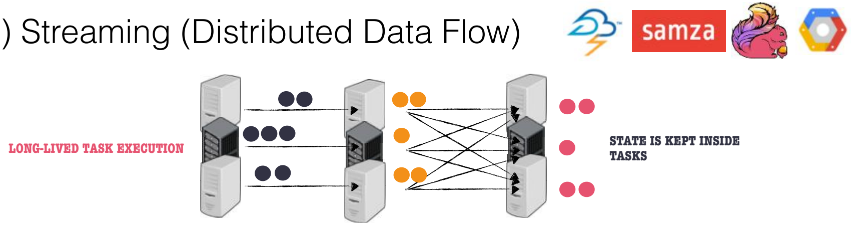
- This materialization step enables the implementation of a simple and elegant checkpoint/restart fault-tolerance mechanism. But it causes significant delay for jobs with real-time processing requirements.



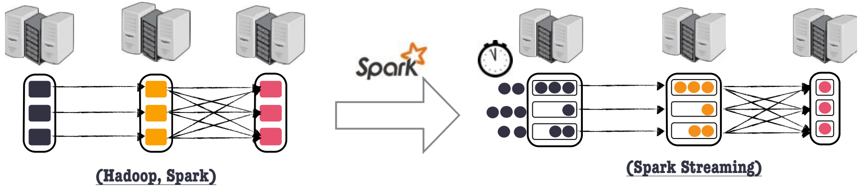


Types of Streaming Architectures

1) Streaming (Distributed Data Flow)



2) Micro-Batch

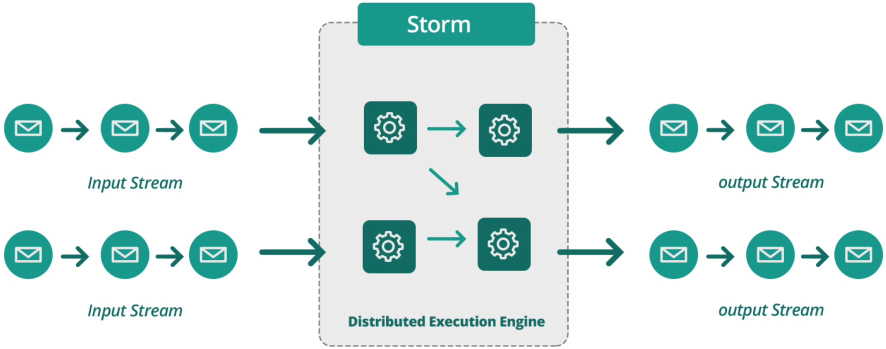


Apache Storm[25]

- Storm is a real-time distributed computing framework for reliably processing unbounded data streams.
- Storm is a project which is created by Nathan Marz and his team at BackType, and released as open source in 2011 after BackType is acquired by Twitter.
- Part of Apache Incubator since September 2013.
- Provides general primitives to do real time computations.

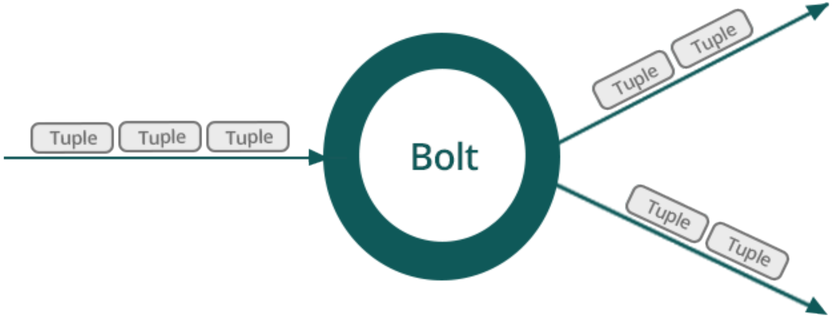
[25] https://storm.apache.org/

Storm


Storm
Input Stream
Input Stream
output Stream
output Stream
Distributed Execution Engine

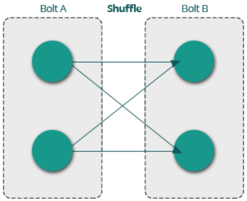
Storm Concepts

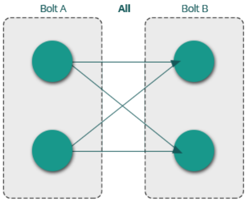
- **Bolt** processes any number of input streams and produces output streams.

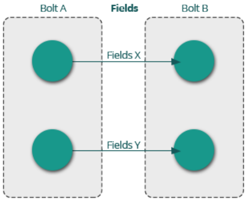


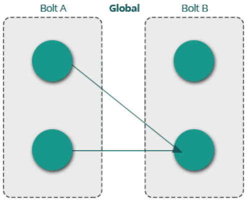
Storm Concepts

- **Field Grouping** provides various ways (e.g., shuffle, fields, global, direct, all, custom) to control tuple routing to bolts.







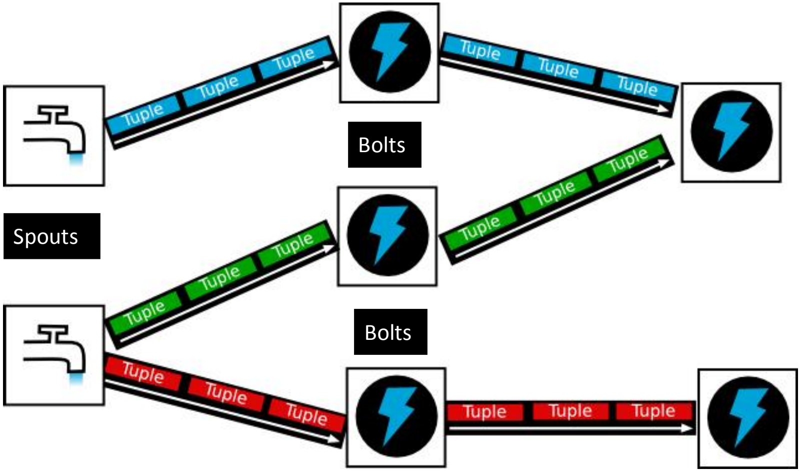


Storm Concepts

Grouping type	What it does
Shuffle Grouping	Sends tuple to a bolt in random round robin sequence
Fields Grouping	Sends tuples to a bolt based on or more field's in the tuple
All grouping	Sends a single copy of each bolt to all instances of a receiving bolt
Custom grouping	Implement your own field grouping so tuples are routed based on custom logic
Direct grouping	Source decides which bolt will receive tuple
Global grouping	Global Grouping sends tuples generated by all instances of the source to a single target instance (specifically, the task with lowest ID)

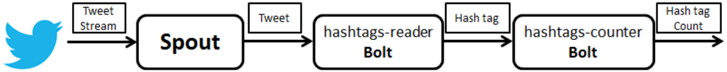
Storm Concepts

- **Topology** represents a network of **Spouts** and **Bolts** which run indefinitely when is deployed.



Storm Example

- **Counting the number of occurrences of each hash tag in an input stream of tweets**



- **Topology Creation**.

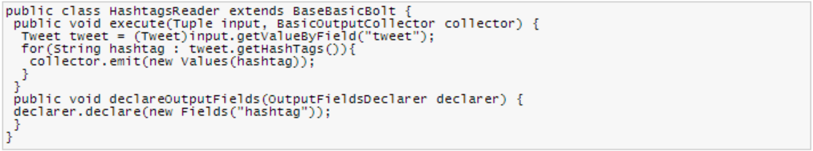
```
public static Topology createTopology() {
 TopologyBuilder builder = new TopologyBuilder();
 builder.setSpout("tweets-stream", new ApiStreamingSpout(), 1);
 builder.setBolt("hashtags-reader", new HashTagsReader(), 2)
 .shuffleGrouping("tweets- stream");

 builder.setBolt("hashtags-counter", new HashtagsCounterBolt(), 2)
 .fieldsGrouping("hashtags-reader ", new Fields("hashtags"));
 return builder.createTopology();
}
```

- **Spout Creation**.

```
public class ApiStreamingSpout extends BaseRichSpout {
 SpoutOutputCollector collector;
 TweetReader reader;
 public void nextTuple() {
  Tweet tweet = reader.getNextTweet();
  if(tweet != null)
   collector.emit(new Values(tweet));
  }
 public void open(Map conf, TopologyContext context,
 SpoutOutputCollector collector) {
  reader = new TwitterReader(conf.get("server"), conf.get("user"), conf.get("pass"));
  this.collector = collector;
 }
 public void declareOutputFields(OutputFieldsDeclarer declarer) {
 declarer.declare(new Fields("tweet"));
 }
}
```

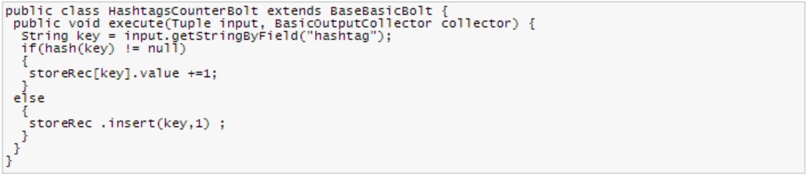
Storm Example

- *HashtagsReader* **bolt**



```
public class HashtagsReader extends BaseBasicBolt {
 public void execute(Tuple input, BasicOutputCollector collector) {
  Tweet tweet = (Tweet)input.getValueByField("tweet");
  for(String hashtag : tweet.getHashTags()){
   collector.emit(new Values(hashtag));
  }
 }
 public void declareOutputFields(OutputFieldsDeclarer declarer) {
 declarer.declare(new Fields("hashtag"));
 }
}
```

- *HashtagsCounter* **bolt**



```
public class HashtagsCounterBolt extends BaseBasicBolt {
 public void execute(Tuple input, BasicOutputCollector collector) {
  String key = input.getStringByField("hashtag");
  if(hash(key) != null)
  {
   storeRec[key].value +=1;
  }
 else
  {
   storeRec .insert(key,1) ;
  }
 }
}
```

Hadoop VS Storm

Hadoop	Storm
Batch Processing	Real-Time Processing
Scalable	Scalable
Fault-Tolerant	Fault-Tolerant
Jobs Run to Completion	Topologies Runs Forever
Job Tracker is SPOF	No Single Point of Failure
Stateful Nodes	Stateless Nodes

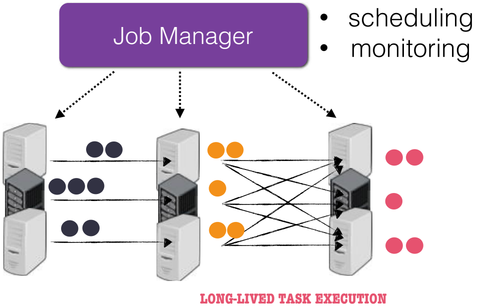
Storm Trident[26]

- Provides a high level API abstraction (DSL) for Storm operations.
- Process a group of tuples as a *batch* rather than processing tuple at a time.
- Trident has joins, aggregations, grouping, functions, and filters.

[26]https://storm.apache.org/documentation/Trident-tutorial.html

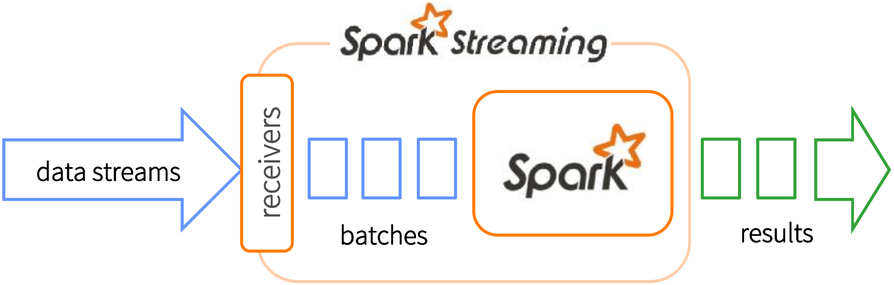
Flink Streaming

- **Tasks** run operator logic in a pipelined fashion
- They are scheduled among workers
- **State** is kept within tasks



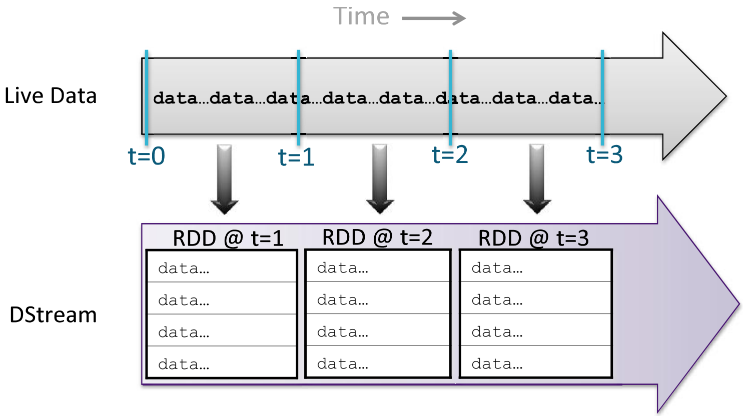
Spark Streaming

- Sprak's extension for stream processing.
- Micro batches of RDD's.
- Receives data streams and chop them up into batches to get processed and pushes out the result.

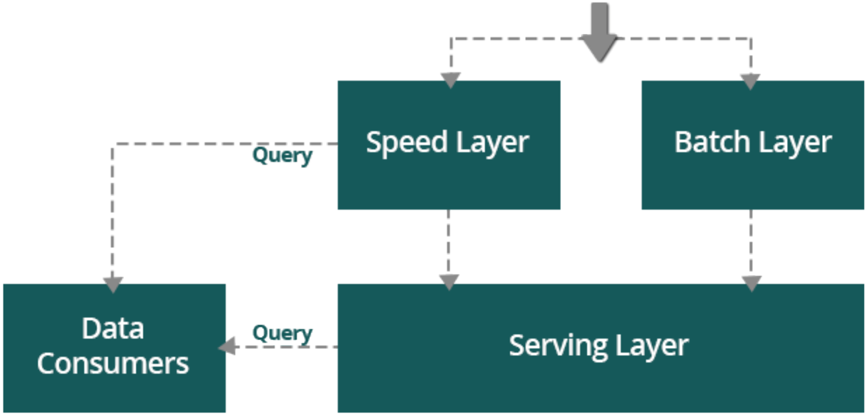


Spark DStreams (Discretized Streams)

- A **DStream** is a sequence of RDDs representing a data stream
- Divide up data stream into batches of *n* seconds
- Process each batch in Spark as an RDD
- Return results of RDD operations in batches



The Lambada Patterns



Part VI

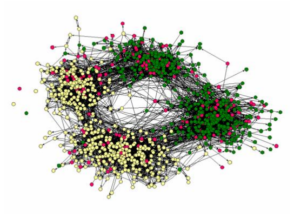
Big Data 2.0 Processing Systems: Big Graph Processing Systems

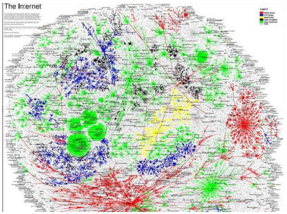
Why Big Graph Processing?

People, devices, processes and other entities have been more connected than at any other point in history!

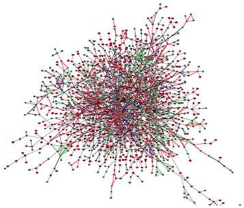
Why Big Graph Processing?

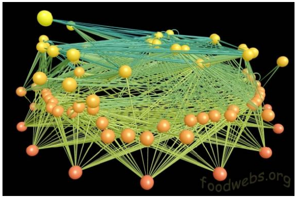
Graphs Are Everywhere!


Social Network



Web Graph


Protein Interactions

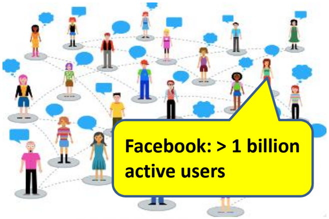


Food Web

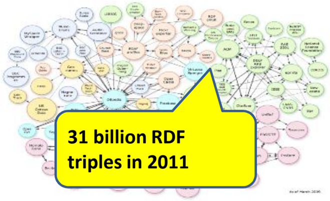
Graph Applications are Exploding

Google: > 1 trillion indexed pages
powered by TouchGraph

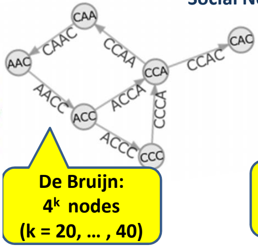
Web Graph



Social Network



Information Network



Biological Network

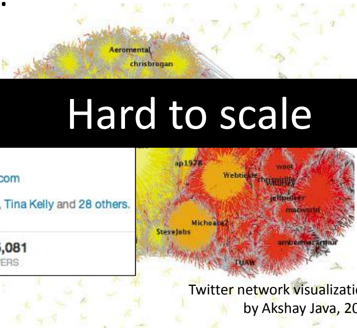
Graphs in Machine Learning

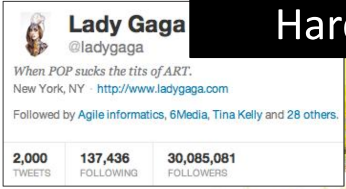
Big Data != Big Graph

Data size: facebook 140 billion connections ≈ 1 TB

Not a problem!

Computation:



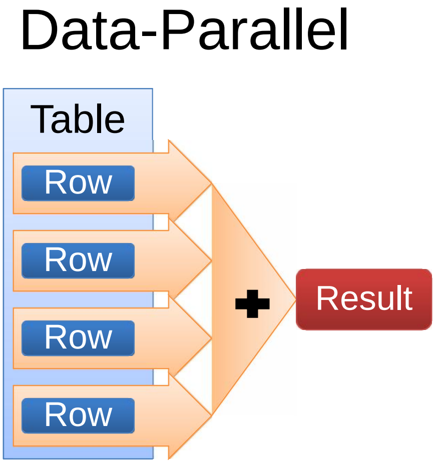

Lady Gaga
@ladygaga
When POP sucks the tits of ART.
New York, NY · http://www.ladygaga.com
Followed by Agile informatics, 6Media, Tina Kelly and 28 others.

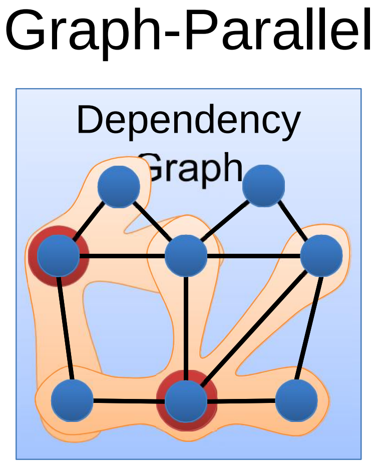
2,000	137,436	30,085,081
TWEETS	FOLLOWING	FOLLOWERS

Hard to scale

Twitter network visualization, by Akshay Java, 2009

Parallel Data Processing vs Parallel Graph Processing





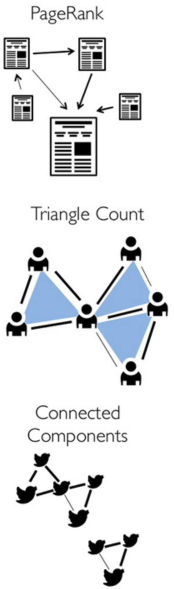
Parallel Data Processing vs Parallel Graph Processing

Data-Intensive **VS**

Complex Computation-Intensive

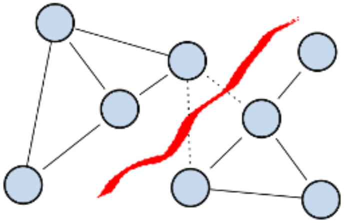
Examples of Graph Processing Algorithms

- PageRank
- Triangle Counting
- Connected Components
- Shortest Distance
- Random Walk
- Graph Coarsening
- Graph Coloring
- Minimum Spanning Forest
- Community Detection
- Collaborative Filtering
- Belief Propagation
- Named Entity Recognition
- ... **And Many Others**

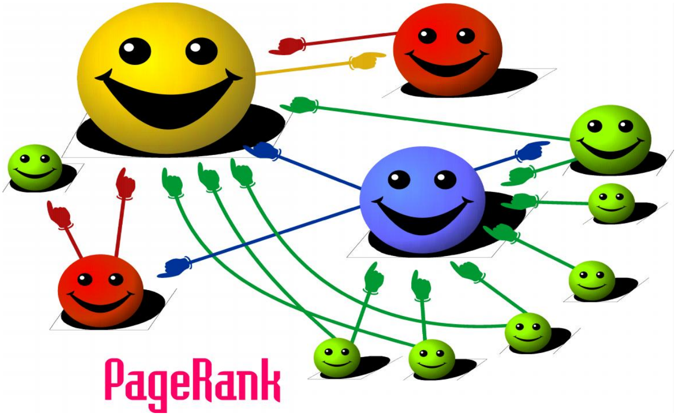


Main Challenges of Graph Processing

- **Data is dynamic** $--->$ No way of doing "schema on write"
- **Structure driven computation** $--->$ Poor Memory Locality and Data Transfer Issues
- Algorithms are **explorative** and **iterative** $--->$ I/O intensive
- **Combinatorial explosion of datasets** $--->$ Relationships Grow Exponentially and Limited Scalability
- **Irregular Structure** $--->$ Challenging Graph Partitioning and Limited Parallelism



PageRank



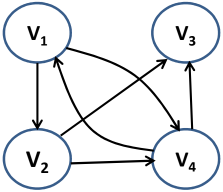
PageRank works by counting the number and quality of edges (links) to a node (web page) to determine a rough estimate of how important the node is

MapReduce for PageRank

- Multiple MapReduce iterations
- **Each Page Rank Iteration:**
  - ❑ **Input:**
    - ($\mathbf{id_1}$, [$PR_t(1)$, out_{11}, out_{12}, …]),
    - ($\mathbf{id_2}$, [$PR_t(2)$, out_{21}, out_{22}, …]),

    …
  - ❑ **Output:**
    - ($\mathbf{id_1}$, [$PR_{t+1}(1)$, out_{11}, out_{12}, …]),
    - ($\mathbf{id_2}$, [$PR_{t+1}(2)$, out_{21}, out_{22}, …]),

    …
- Iterate until convergence → another MapReduce instance



Input:

V_1, [0.25, V_2, V_3, V_4]
V_2, [0.25, V_3, V_4]
V_3, [0.25, V_1]
V_4,[0.25, V_1, V_3]

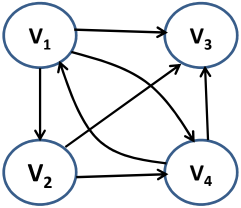
One MapReduce Iteration

Output:

V_1, [0.37, V_2, V_3, V_4]
V_2, [0.08, V_3, V_4]
V_3, [0.33, V_1]
V_4 ,[0.20, V_1, V_3]

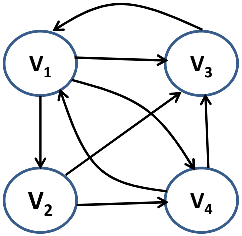
MapReduce for PageRank

- **Map**
  - ❑ **Input: (V_1**, [0.25, V_2, V_3, V_4]**)**; **(V_2**, [0.25, V_3, V_4]**)**; **(V_3**, [0.25, V_1]**)**; **(V_4**,[0.25, V_1, V_3]**)**
  - ❑ **Output: (V_2**, 0.25/3**)**, **(V_3**, 0.25/3**)**, **(V_4**, 0.25/3**)**, ……, **(V_1**, 0.25/2**)**, **(V_3**, 0.25/2**)**; **(V_1**, [V_2, V_3, V_4]**)**, **(V_2**, [V_3, V_4]**)**, **(V_3**, [V_1]**)**, **(V_4**, [V_1, V_3]**)**

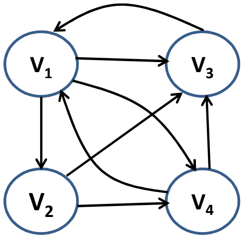


MapReduce for PageRank

- **Map**
  - **Input:** **(V_1,** [0.25, V_2, V_3, V_4]**)**; **(V_2,** [0.25, V_3, V_4]**)**; **(V_3,** [0.25, V_1]**)**; **(V_4,**[0.25, V_1, V_3]**)**
  - **Output:** **(V_2,** 0.25/3**)**, **(V_3,** 0.25/3**)**, **(V_4,** 0.25/3**)**, ..., **(V_1,** 0.25/2**)**, **(V_3,** 0.25/2**)**; **(V_1,** [V_2, V_3, V_4]**)**, **(V_2,** [V_3, V_4]**)**, **(V_3,** [V_1]**)**, **(V_4,** [V_1, V_3]**)**



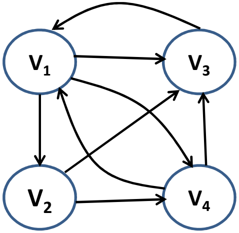
MapReduce for PageRank



- **Map**
  - ❑ **Input:** **(V_1**, [0.25, V_2, V_3, V_4]**)**; **(V_2**, [0.25, V_3, V_4]**)**; **(V_3**, [0.25, V_1]**)**; **(V_4**,[0.25, V_1, V_3]**)**
  - ❑ **Output:** **(V_2**, 0.25/3**)**, **(V_3**, 0.25/3**)**, **(V_4**, 0.25/3**)**, ……, **(V_1**, 0.25/2**)**, **(V_3**, 0.25/2**)**; **(V_1**, [V_2, V_3, V_4]**)**, **(V_2**, [V_3, V_4]**)**, **(V_3**, [V_1]**)**, **(V_4**, [V_1, V_3]**)**
- **Shuffle**
  - ❑ **Output:** **(V_1**, 0.25/1**)**, **(V_1**, 0.25/2**)**, **(V_1**, [V_2, V_3, V_4]**)**; ……. ; **(V_4**, 0.25/3**)**, **(V_4**, 0.25/2**)**, **(V_4**, [V_1, V_3]**)**

MapReduce for PageRank

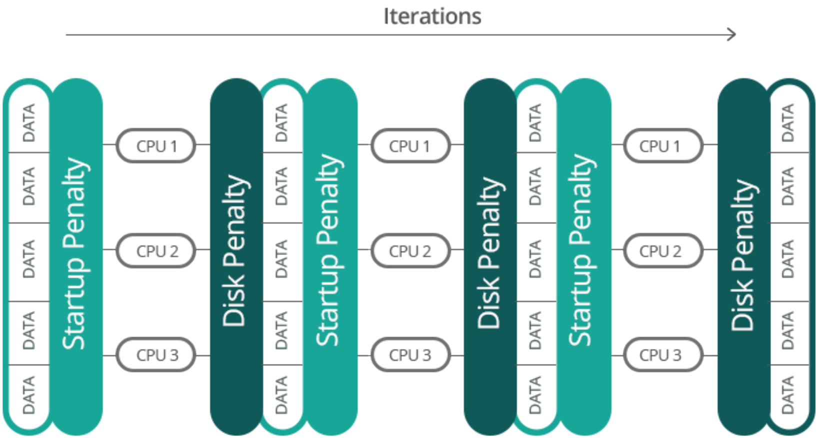
- **Map**
  - ❑ **Input: (V_1**, [0.25, V_2, V_3, V_4]**)**; **(V_2**, [0.25, V_3, V_4]**)**; **(V_3**, [0.25, V_1]**)**; **(V_4**,[0.25, V_1, V_3]**)**
  - ❑ **Output: (V_2**, 0.25/3**)**, **(V_3**, 0.25/3**)**, **(V_4**, 0.25/3**)**, ……, **(V_1**, 0.25/2**)**, **(V_3**, 0.25/2**)**; **(V_1**, [V_2, V_3, V_4]**)**, **(V_2**, [V_3, V_4]**)**, **(V_3**, [V_1]**)**, **(V_4**, [V_1, V_3]**)**



- **Shuffle**
  - ❑ **Output: (V_1**, 0.25/1**)**, **(V_1**, 0.25/2**)**, **(V_1**, [V_2, V_3, V_4]**)**; ……. ; **(V_4**, 0.25/3**)**, **(V_4**, 0.25/2**)**, **(V_4**, [V_1, V_3]**)**
- **Reduce**
  - ❑ **Output: (V_1**, [0.37, V_2, V_3, V_4]**)**; **(V_2**, [0.08, V_3, V_4]**)**; **(V_3**, [0.33, V_1]**)**; **(V_4**,[0.20, V_1, V_3]**)**

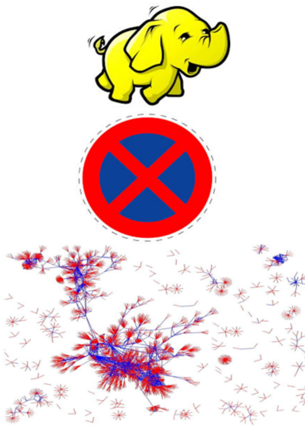
MapReduce for Iterative Operations

MapReduce is **not optimized** for iterative operations

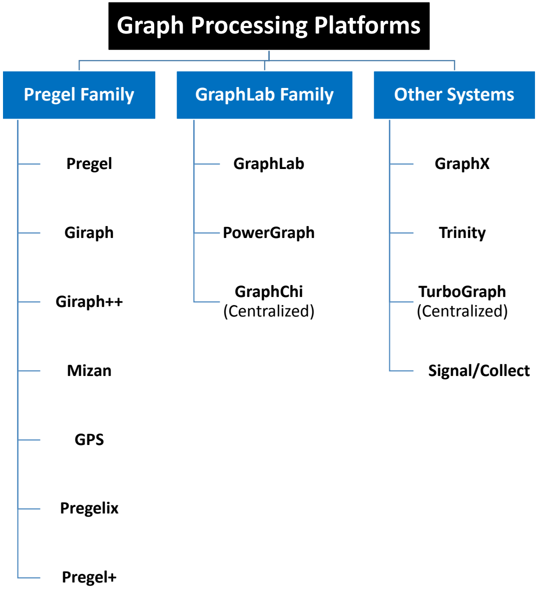


Hadoop for Big Graphs?!

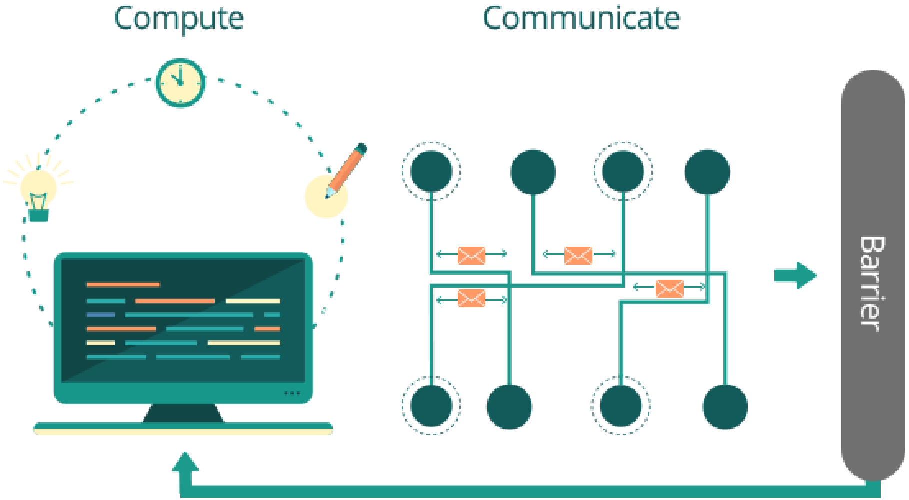
- MapReduce does not directly support iterative algorithms.
- Invariant graph-topology-data re-loaded and re-processed at each iteration $-- >$ wasting I/O, network bandwidth, and CPU
- Materializations of intermediate results at every MapReduce iteration harm performance
- Extra MapReduce job on each iteration for detecting if a fixpoint has been reached



An Overview of Big Graph Processing Systems

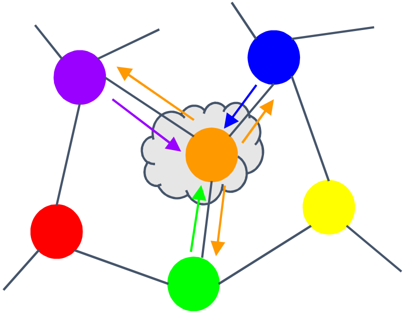


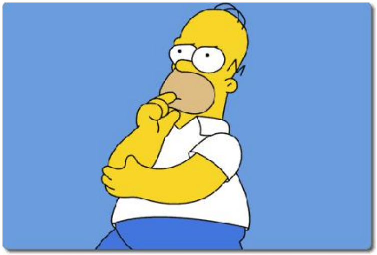
Bulk Synchronous Parallel (BSP) Programming Model[27]



[27]L. G. Valiant. *A Bridging Model for Parallel Computation*. Commun. ACM, 1990

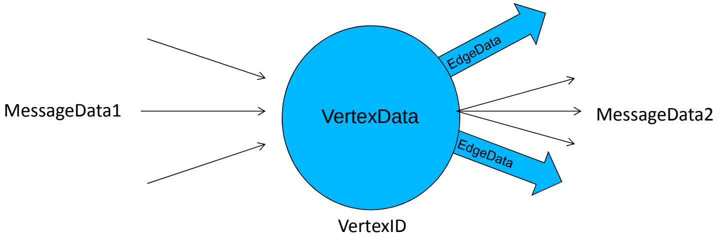
Vertex View

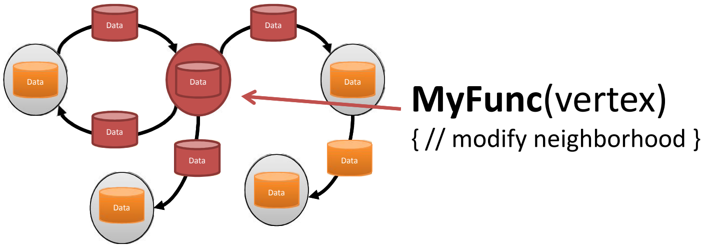




- **Think Like a Vertex**
    - Receive Messages
    - Update States
    - Send Messages

Vertex-centric Programming



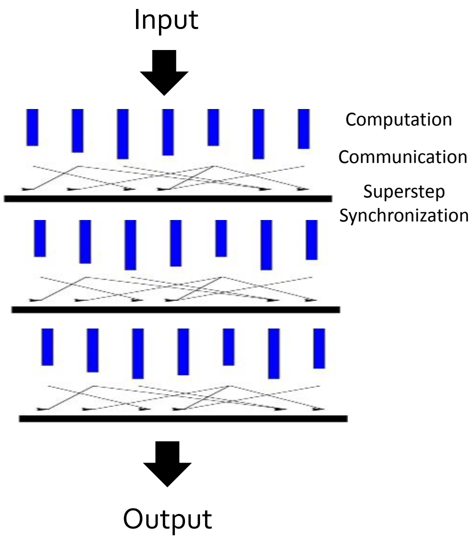


Google Pregel[28]

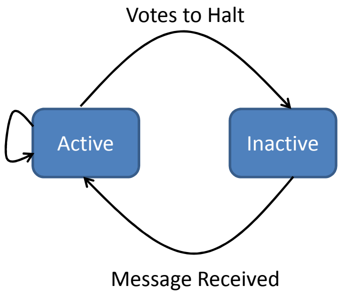
- The first BSP-based implementation for graph processing
- Communication through message passing (usually sent along the outgoing edges from each vertex) + Shared-Nothing
- Vertex-centric computation, each vertex:
  - Receives messages sent in the previous superstep
  - Executes the same user-defined function
  - Modifies its value
  - If active, sends messages to other vertices (next superstep)
  - Votes to halt if it has no further work to do $--->$ becomes inactive
- Terminate when all vertices are inactive and no messages in transmit
- **Advantages**:
  - **No locks** $--->$ message-based communication
  - **No semaphores** $--->$ global synchronization
  - **Iteration isolation** $--->$ massively parallelizable

[28] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. *Pregel: a system for large-scale graph processing*. SIGMOD, 2010.

Pregel



PREGEL Computation Model



State Machine for a Vertex in PREGEL

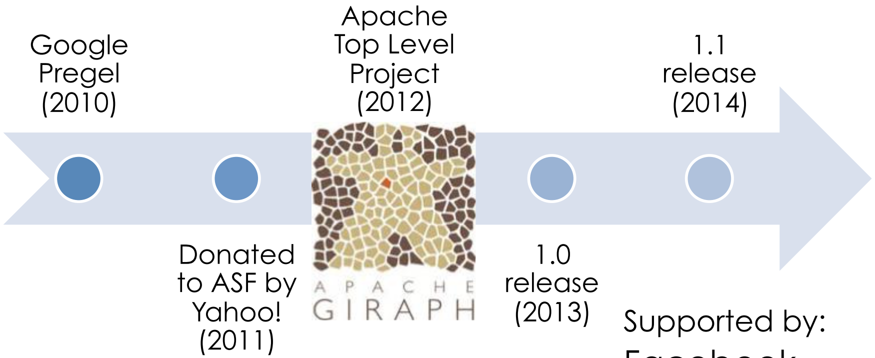
Apache Giraph[29]

Google MapReduce → hadoop

Google Pregel → APACHE GIRAPH

[29] https://giraph.apache.org/

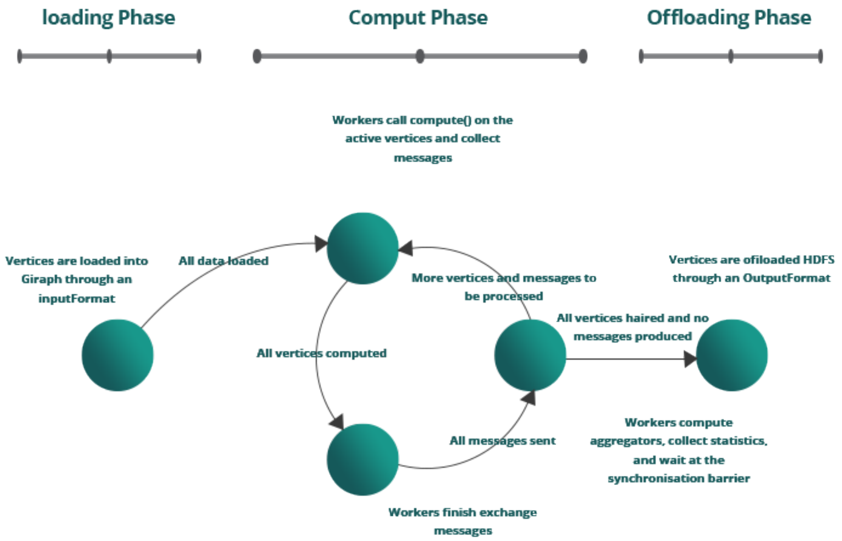
Giraph's Timeline



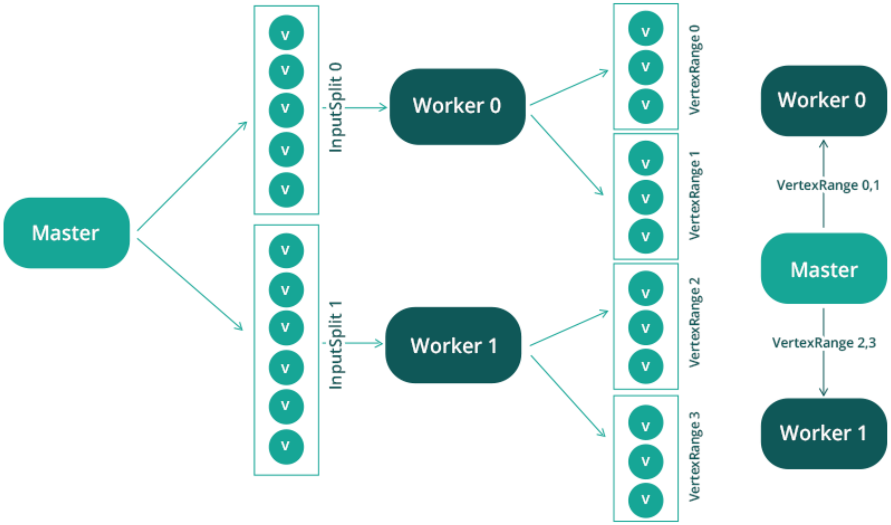
Supported by:

Facebook
Yahoo!
LinkedIn

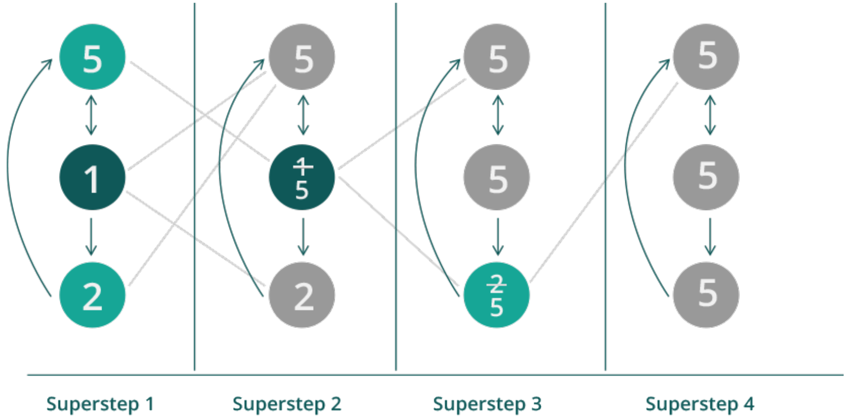
Giraph Execution Phases


loading Phase
Comput Phase
Offloading Phase
Workers call compute() on the active vertices and collect messages
Vertices are loaded into Giraph through an inputFormat
All data loaded
More vertices and messages to be processed
Vertices are ofiloaded HDFS through an OutputFormat
All vertices haired and no messages produced
All vertices computed
All messages sent
Workers compute aggregators, collect statistics, and wait at the synchronisation barrier
Workers finish exchange messages

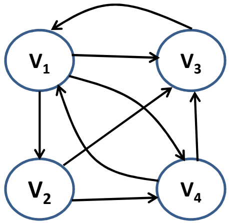
Giraph Master-Slave System Architecture



BSP Example - Max Value



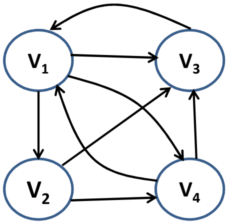
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

- PR(u): Page Rank of node u
- F_u: Out-neighbors of node u
- B_u: In-neighbors of node u

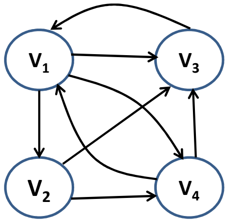
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0
PR(V_1)	0.25
PR(V_2)	0.25
PR(V_3)	0.25
PR(V_4)	0.25

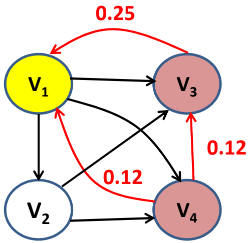
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	?
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

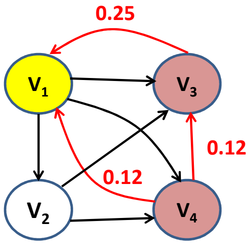
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	?
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

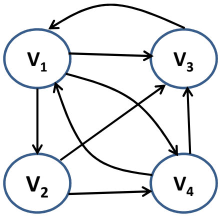
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	0.37
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

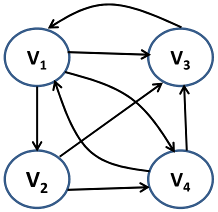
BSP Example - PageRank



$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	0.37
PR(V_2)	0.25	0.08
PR(V_3)	0.25	0.33
PR(V_4)	0.25	0.20

BSP Example - PageRank

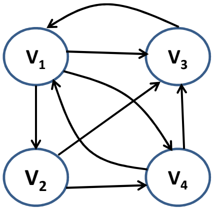


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2
PR(V_1)	0.25	0.37	0.43
PR(V_2)	0.25	0.08	0.12
PR(V_3)	0.25	0.33	0.27
PR(V_4)	0.25	0.20	0.16

BSP Example - PageRank

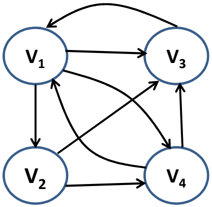


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3
PR(V_1)	0.25	0.37	0.43	0.35
PR(V_2)	0.25	0.08	0.12	0.14
PR(V_3)	0.25	0.33	0.27	0.29
PR(V_4)	0.25	0.20	0.16	0.20

BSP Example - PageRank

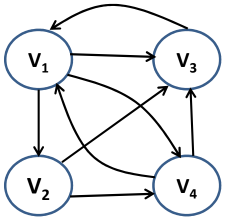


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4
PR(V_1)	0.25	0.37	0.43	0.35	0.39
PR(V_2)	0.25	0.08	0.12	0.14	0.11
PR(V_3)	0.25	0.33	0.27	0.29	0.29
PR(V_4)	0.25	0.20	0.16	0.20	0.19

BSP Example - PageRank

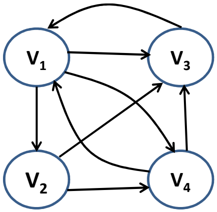


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4	K=5
PR(V_1)	0.25	0.37	0.43	0.35	0.39	0.39
PR(V_2)	0.25	0.08	0.12	0.14	0.11	0.13
PR(V_3)	0.25	0.33	0.27	0.29	0.29	0.28
PR(V_4)	0.25	0.20	0.16	0.20	0.19	0.19

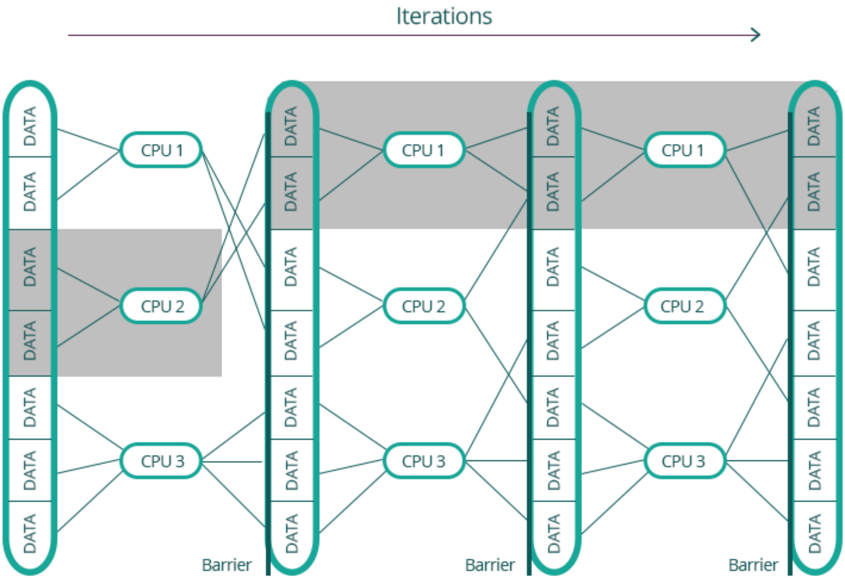
BSP Example - PageRank



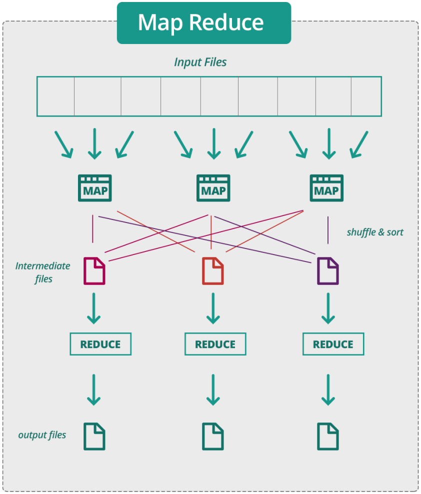
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

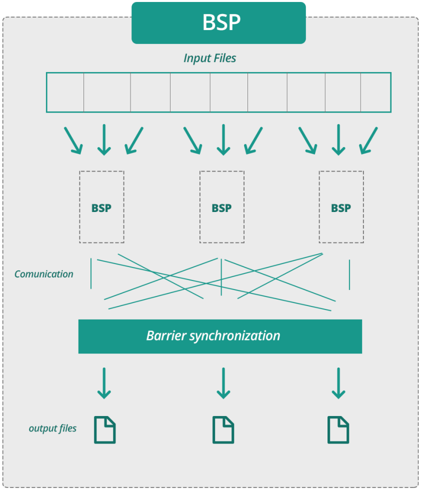
	K=0	K=1	K=2	K=3	K=4	K=5	K=6
PR(V_1)	0.25	0.37	0.43	0.35	0.39	0.39	0.38
PR(V_2)	0.25	0.08	0.12	0.14	0.11	0.13	0.13
PR(V_3)	0.25	0.33	0.27	0.29	0.29	0.28	0.28
PR(V_4)	0.25	0.20	0.16	0.20	0.19	0.19	0.19

BSP Communication - Pregel Execution



MapReduce VS Pregel





MapReduce VS Pregel

MapReduce

- Requires passing of entire graph topology from one iteration to the next
- Intermediate results after every iteration is stored at disk and then read again from the disk
- Programmer needs to write a driver program to support iterations; another MapReduce program to check for fixpoint

PREGEL

- Each node sends its state only to its neighbors. Graph topology information is not passed across iterations
- Main memory based
- Usage of supersteps and master-client architecture makes programming easy

Limitations of Pregel

- In Bulk Synchronous Parallel (BSP) model, performance is limited by the slowest machine
    - ❑ Real-world graphs have power-law degree distribution, which may lead to a few highly-loaded servers
- Does not utilize the already computed partial results from the same iteration
    - ❑ Several machine learning algorithms (e.g., belief propagation, expectation maximization, stochastic optimization) have higher accuracy and efficiency with asynchronous updates

Potential Optimizations

- Partition the graph – (1) balance server workloads
  (2) minimize communication across servers

Mizan[30]

- An open-source project developed in C++ by KAUST, in collaboration with IBM Research.
- Focuses on efficient load balancing across workers in a cluster and minimizing the variations across workers by identifying which vertices to migrate and where to migrate them to.
- Mizan (Arabic): a double-pan scale

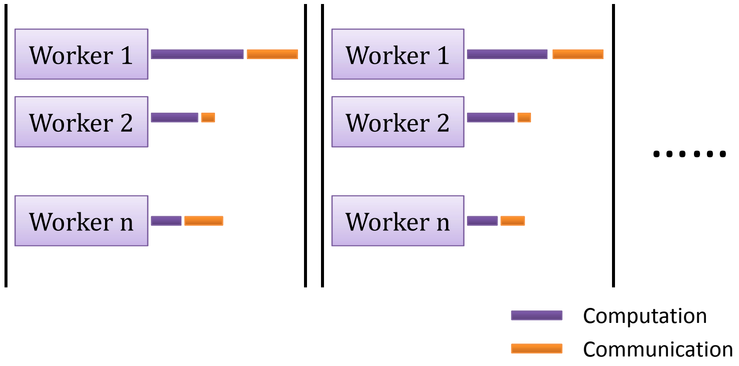
[30]Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. *Mizan: a system for dynamic load balancing in large-scale graph processing*. EuroSys, 2013.

Mizan

- Monitoring:
  - ❑ Outgoing Messages
  - ❑ Incoming Messages
  - ❑ Response Time
- Migration Planning:
  - ❑ Identify the source of imbalance
  - ❑ Select the migration objective
  - ❑ Pair over-utilized workers with under-utilized ones
  - ❑ Select vertices to migrate
  - ❑ Migrate vertices

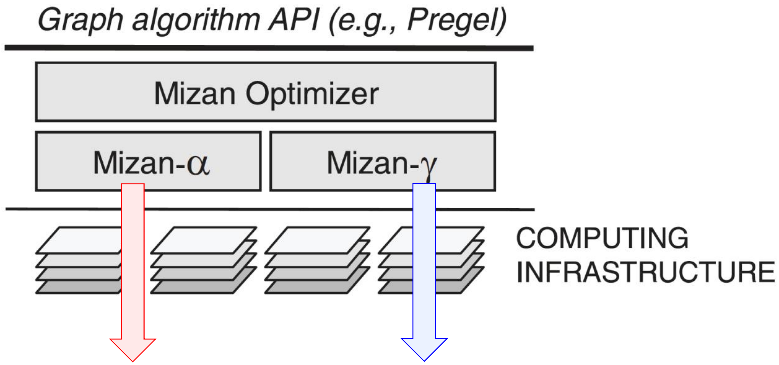
Mizan: Dynamic Re-Partition

- Dynamic Load Balancing across supersteps in PREGEL



- Adaptive re-partitioning
- Agnostic to the graph structure
- Requires no apriori knowledge of algorithm behavior

Mizan



- Min-cut partitioning of input graph
- Point-to-point message passing
- Good for power-law graphs

- Random partitioning of input
- Ring overlay message passing
- Good for non-power-law graphs

Challenges of Mizan

- Monitoring:
  - ❑ Outgoing Messages
  - ❑ Incoming Messages
  - ❑ Response Time

- Does workload in the current iteration an indication of workload in the next iteration?
- Overhead due to migration?

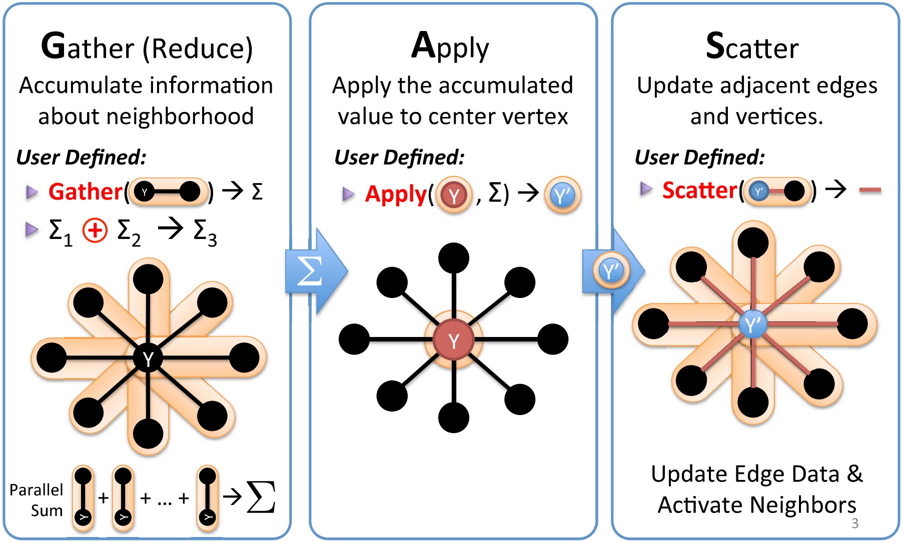
GraphLab[32]

- GraphLab is an open-source large scale graph processing project, implemented in C++, which started at CMU and is currently supported by Dato Inc[31].
- Unlike Pregel, GraphLab relies on the shared memory abstraction and the **GAS (Gather, Apply, Scatter)** processing model which is similar to but also fundamentally different from the BSP model that is employed by Pregel.
- The GraphLab abstraction consists of three main parts: the *data graph*, the *update function*, and the *sync operation*.

[31] `https://dato.com/`

[32] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. *Distributed GraphLab: A Framework for Machine Learning in the Cloud*. PVLDB, 2012.

GAS Model



Gather (Reduce)

Accumulate information about neighborhood

User Defined:

- **Gather**( ) $\rightarrow \Sigma$
- $\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3$

Parallel Sum + + … + $\rightarrow \Sigma$

Apply

Apply the accumulated value to center vertex

User Defined:

- **Apply**(Y , Σ) $\rightarrow$ Y’

Scatter

Update adjacent edges and vertices.

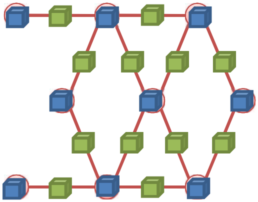
User Defined:

- **Scatter**( ) $\rightarrow$

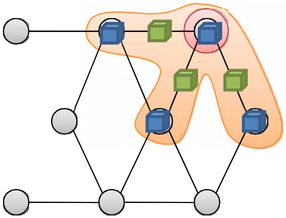
Update Edge Data & Activate Neighbors

The GraphLab Framework

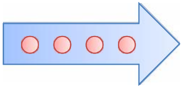
Graph Based
Data Representation



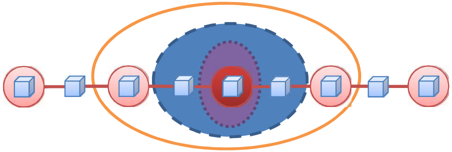
Update Functions
User Computation



Scheduler

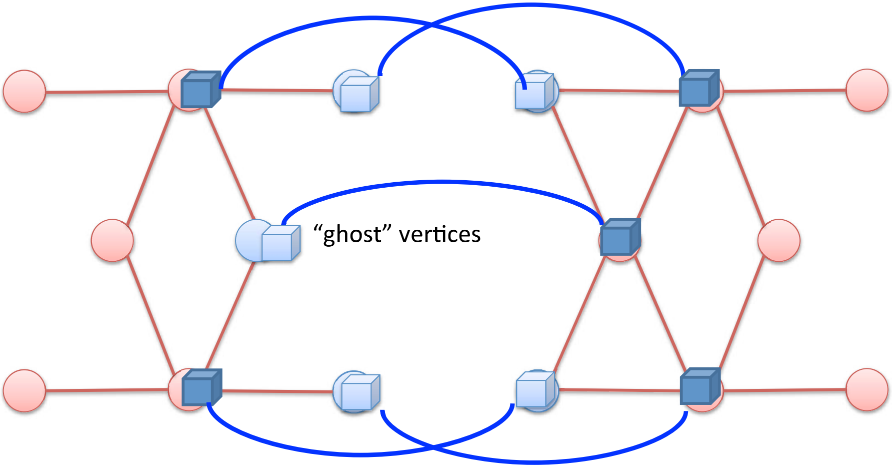


Consistency Model



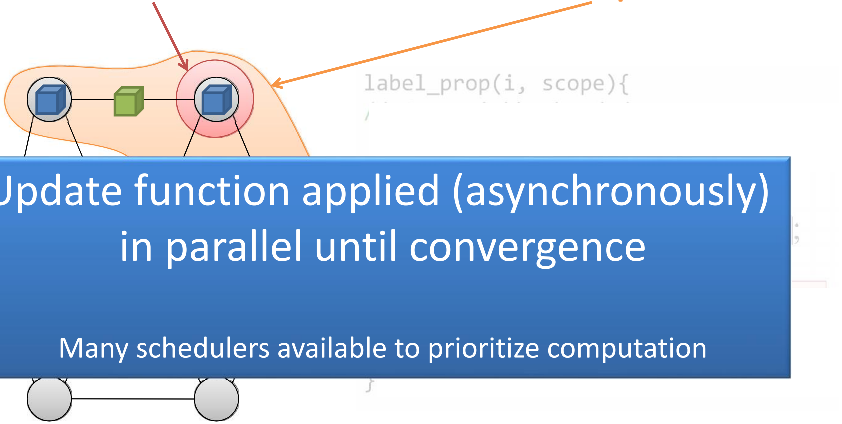
GraphLab: Ghost Vertices

Ghost vertices maintain adjacency structure and replicate remote data



GraphLab Update Function

An **update function** is a user defined program which when applied to a **vertex** transforms the data in the **scope**of the vertex



Update function applied (asynchronously) in parallel until convergence

Many schedulers available to prioritize computation

PREGEL VS GraphLab

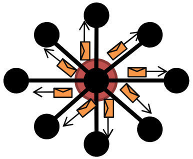
PREGEL

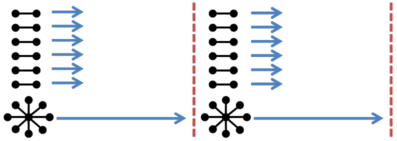
- Synchronous System
- No concurrency control, no worry of consistency
- Easy fault-tolerance, check point at each barrier
- Bad when waiting for stragglers or load-imbalance

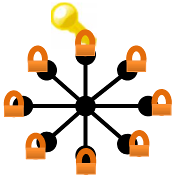
GraphLab

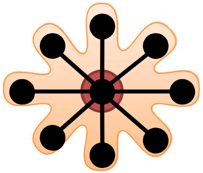
- Asynchronous System
- Consistency of updates harder (edge, vertex, sequential)
- Fault-tolerance harder (need a snapshot with consistency)
- Asynchronous model can make faster progress
- Can load balance in scheduling to deal with load skew

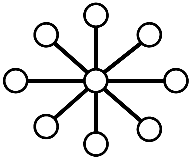
Difficulties with Power Law Graphs


Sends many messages (Pregel)


Synchronous Execution
prone to stragglers (Pregel)


Asynchronous Execution
requires heavy locking (GraphLab)


Touches a large fraction of
graph (GraphLab)


Edge meta-data
too large for single machine

PowerGraph[33]

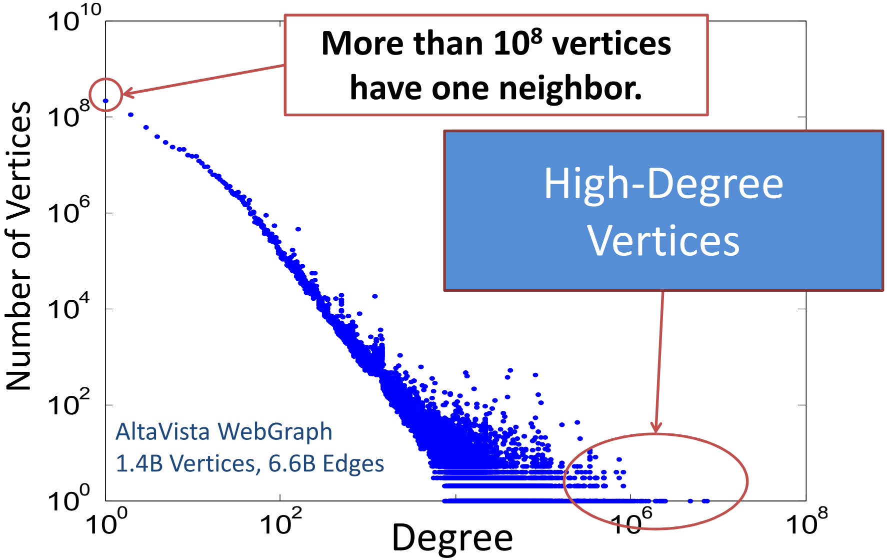
- A member of the GraphLab family of systems that have been introduced to avoid the imbalanced workload caused by high degree vertices in power-law graphs.

- PowerGraph introduced a partitioning scheme that cuts the vertex set in a way such that the edges of a high-degree vertex are handled by multiple workers.

- As a tradeoff, vertices are replicated across workers, and communication among workers are required to guarantee that the vertex value on each replica remains consistent.

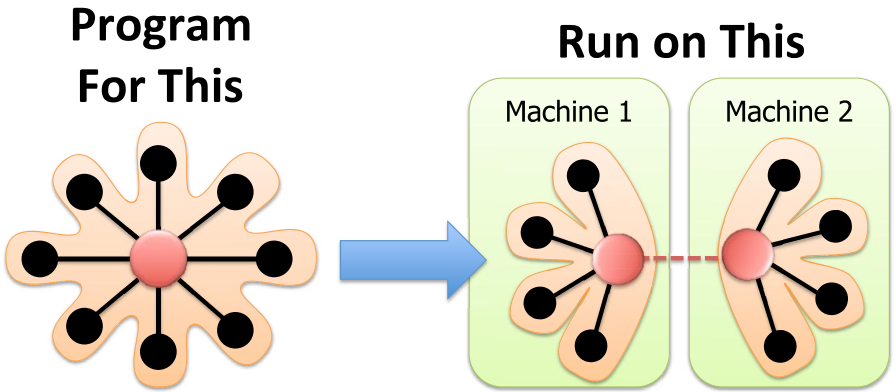
[33] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs*. In OSDI, 2012.

PowerGraph: Motivation

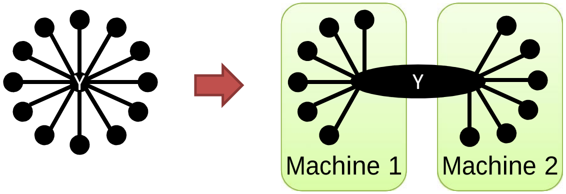


The PowerGraph Framework

Split **High-Degree** vertices



Vertex-Cut instead of Edge-Cut

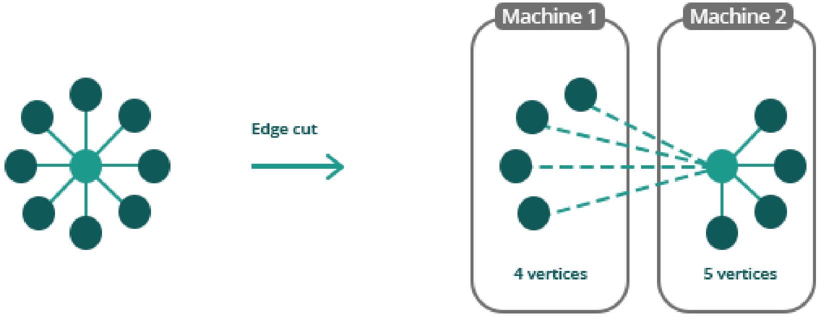


Vertex Cut (GraphLab)

- Power-Law graphs have good vertex cuts. [Albert et al., Nature ‘00]
- Communication is linear in the number of machines each vertex spans
- A vertex-cut minimizes machines each vertex spans
- Edges are evenly distributed over machines → improved work balance

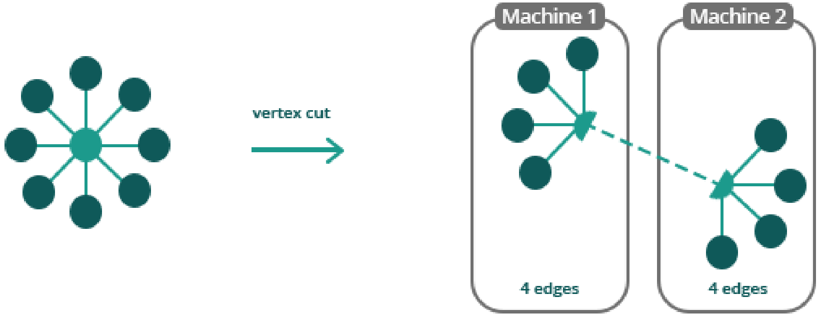
Edge-Cut

- Used by Pregel and GraphLab abstractions
- Evenly assign vertices to machines



Vertex-Cut

- Used by PowerGraph abstraction
- Evenly assign edges to machines

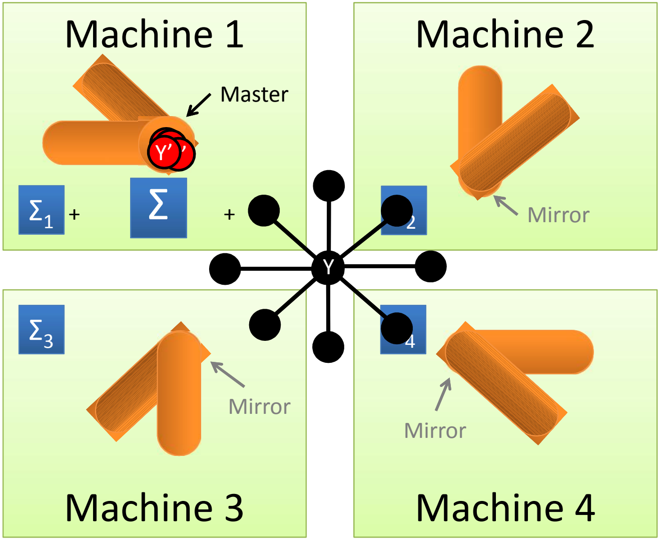


The PowerGraph Framework

Gather

Apply

Scatter



Minimizing Communication in PowerGraph

Communication is linear in the number of machines each vertex spans

A **vertex-cut** minimizes machines each vertex spans

Percolation theory suggests that power law graphs have ***good vertex cuts.*** *[Albert et al. 2000]*

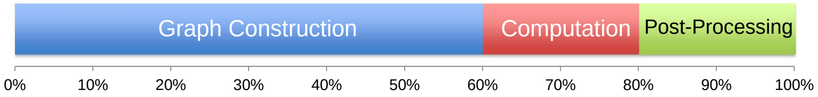
GraphX[34]

- A distributed graph engine built on top of **Spark**.
- GraphX extends Sparks Resilient Distributed Dataset (RDD) abstraction to introduce the **Resilient Distributed Graph (RDG)**, which associates records with vertices and edges in a graph and provides a collection of expressive computational primitives.
- The GraphX RDG leverages advances in distributed graph representation and exploits the graph structure to minimize communication and storage overhead.
- GraphX relies on a flexible vertex cut partitioning to encode graphs as horizontally partitioned collections.

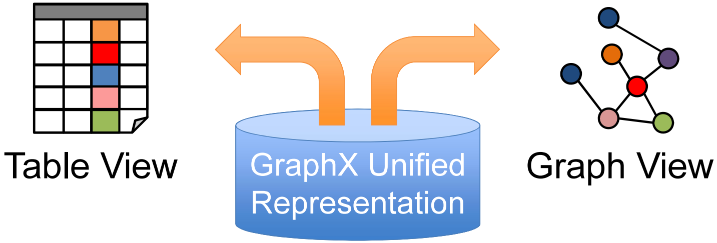
[34] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. *GraphX: Graph Processing in a Distributed Dataflow Framework*. OSDI, 2014.

GraphX

- One system for the entire graph pipeline. Unlike other graph processing systems, the GraphX API enables the composition of graphs.

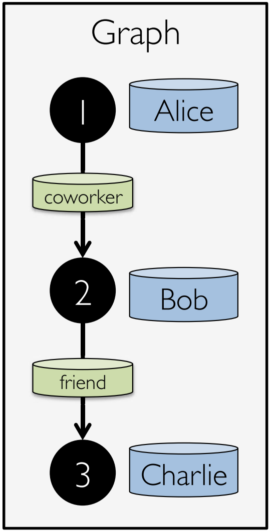


- Tables and graphs are **views** of the same physical data.
- Each view has its own operators that exploit the semantics of the view to achieve efficient execution.



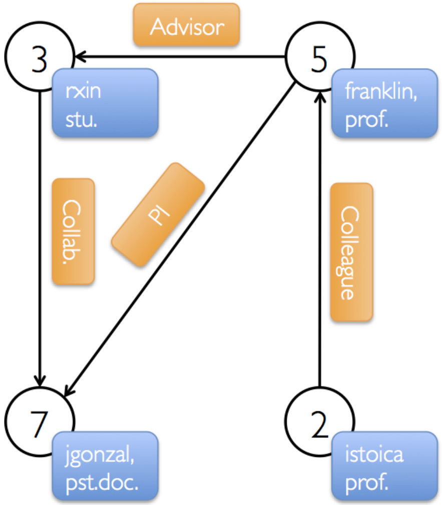
GraphX Representation

```
type VertexId = Long

val vertices: RDD[(VertexId, String)] =
  sc.parallelize(List(
    (1L, “Alice”),
    (2L, “Bob”),
    (3L, “Charlie”)))

class Edge[ED](
  val srcId: VertexId,
  val dstId: VertexId,
  val attr: ED)

val edges: RDD[Edge[String]] =
  sc.parallelize(List(
    Edge(1L, 2L, “coworker”),
    Edge(2L, 3L, “friend”)))

val graph = Graph(vertices, edges)
```



GraphX Representation

Property Graph



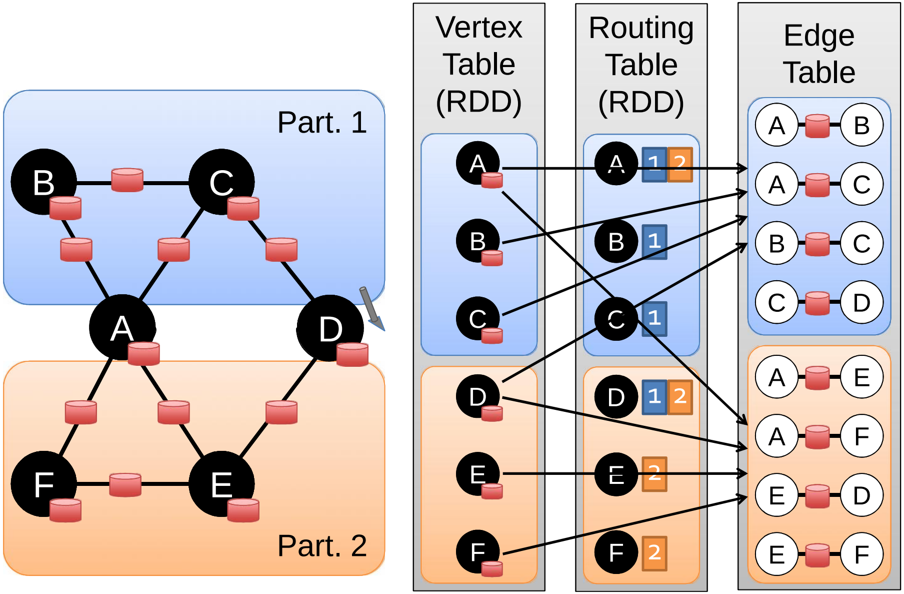
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Distributed Graphs as Tables (RDDs)



GraphX Graph Operations

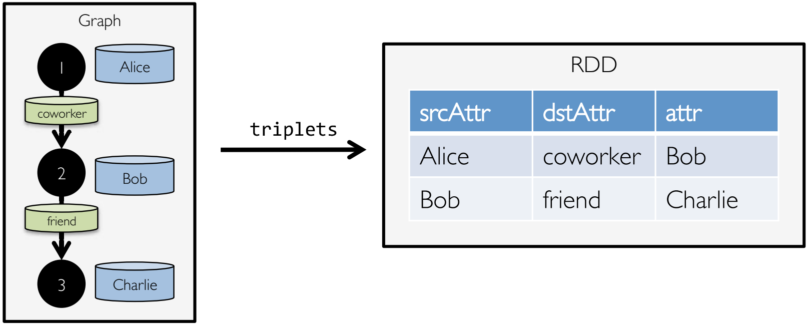
- **Basic information** (`numEdges`, `numVertices`, `inDegrees`, ...)
- **Views** (vertices, edges, triplets)
- **Caching** (persist, cache, ...)
- **Transformation** (`mapVertices`, `mapEdges`, ...)
- **Structure modification** (`reverse`, `subgraph`, ...)
- **Neighbour aggregation** (collectNeighbours, aggregations, ...)
- **Graph builders** (various I/O operations)
- ...

GraphX Graph Operations

```
class Graph[VD, ED] {
    // Table Views -----------------------
    def vertices: RDD[(VertexId, VD)]
    def edges: RDD[Edge[ED]]
    def triplets: RDD[EdgeTriplet[VD, ED]]
    // Transformations ---------------------------------------
    def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
    def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD2, ED]
    def reverse: Graph[VD, ED]
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
    // Joins --------------------------------------
    def outerJoinVertices[U, VD2]
        (tbl: RDD[(VertexId, U)])
        (f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
    // Computation -----------------------------------
    def mapReduceTriplets[A](
        sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]
```

GraphX Triplet View

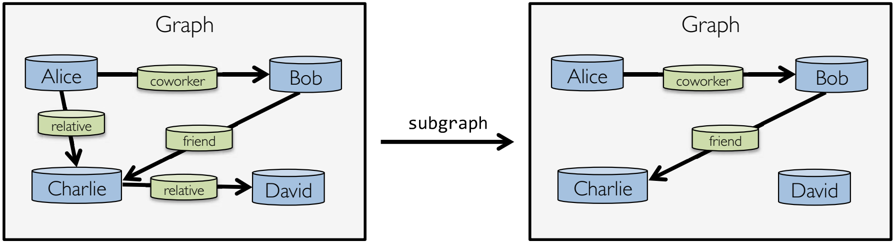
```
class Graph[VD, ED] {
    def triplets: RDD[EdgeTriplet[VD, ED]]
}

class EdgeTriplet[VD, ED](
  val srcId: VertexId, val dstId: VertexId, val attr: ED,
  val srcAttr: VD, val dstAttr: VD)
```



srcAttr	dstAttr	attr
Alice	coworker	Bob
Bob	friend	Charlie

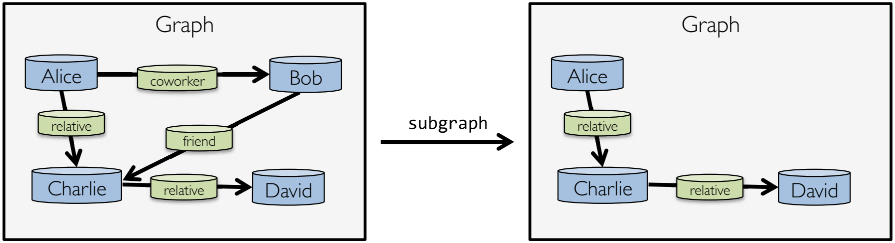
GraphX Subgraph Transformation

```
class Graph[VD, ED] {
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(epred = (edge) => edge.attr != “relative”)
```

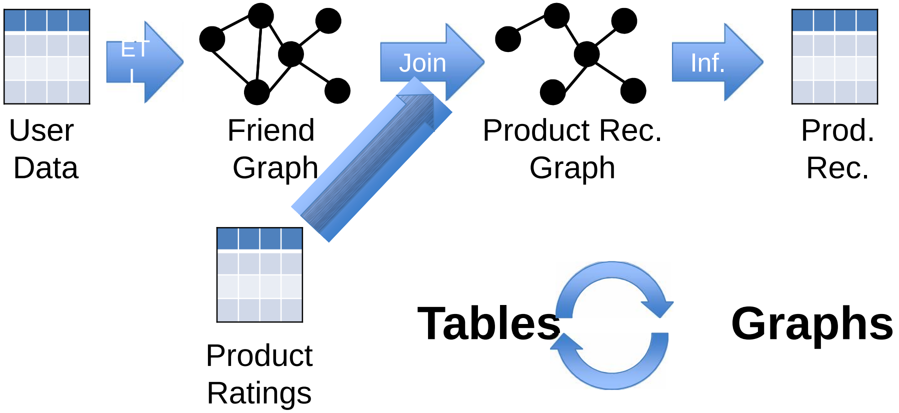


GraphX Subgraph Transformation

```
class Graph[VD, ED] {
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(vpred = (id, name) => name != “Bob”)
```

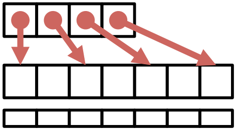


GraphX Enables Joining Tables and Graphs

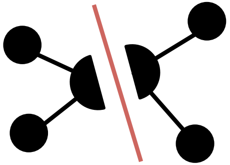


GraphX System Optimization

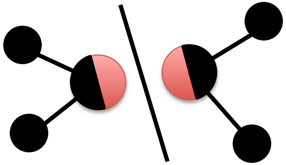
Specialized Data-Structures



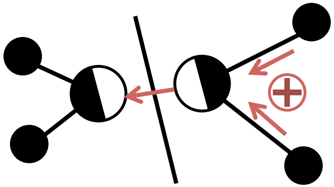
Vertex-Cuts Partitioning



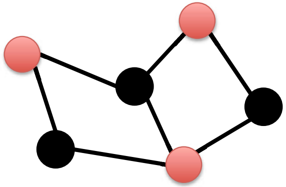
Remote Caching / Mirroring



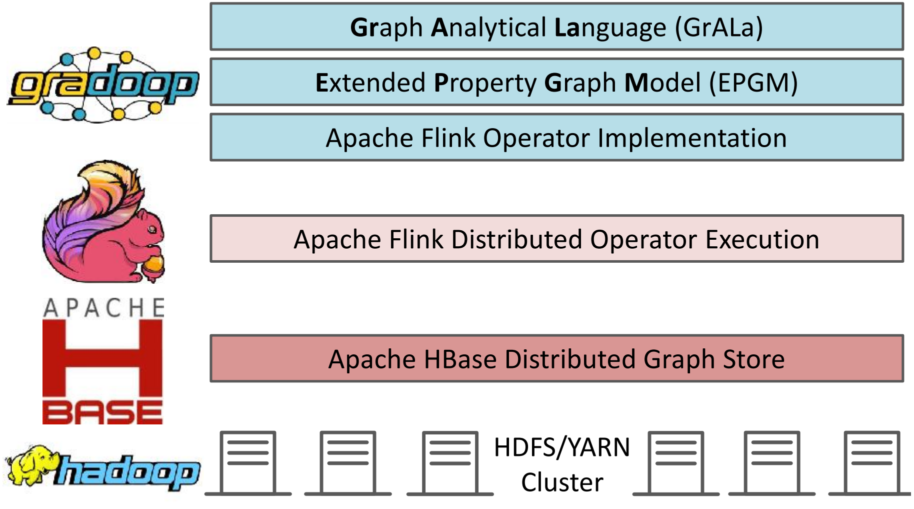
Message Combiners



Active Set Tracking



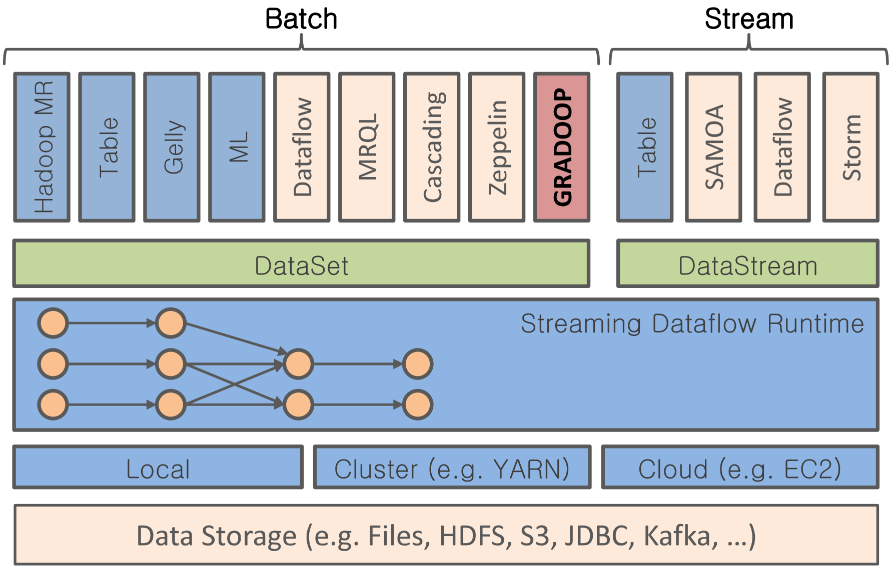
Gradoop[35]



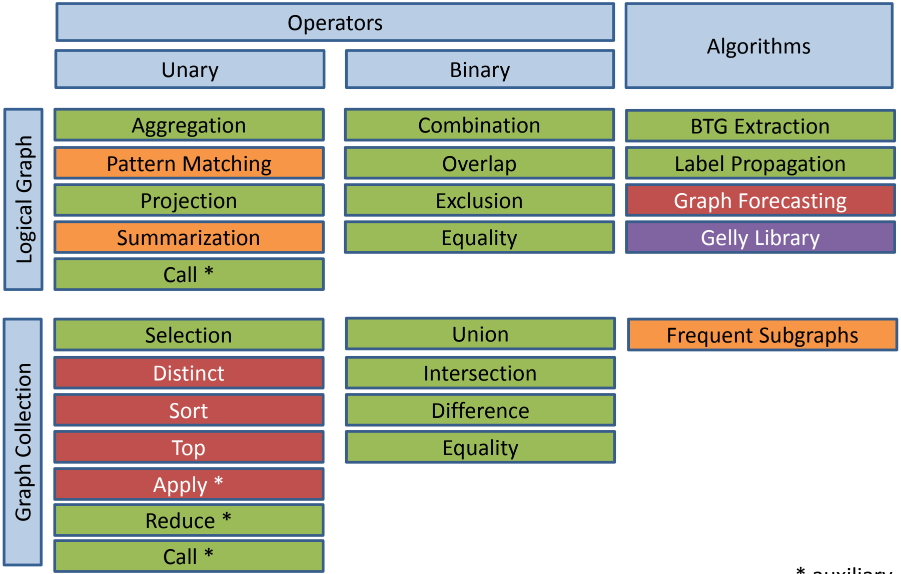
Graph **A**nalytical **La**nguage (GrALa)
Extended **P**roperty **G**raph **M**odel (EPGM)
Apache Flink Operator Implementation
Apache Flink Distributed Operator Execution
Apache HBase Distributed Graph Store
HDFS/YARN Cluster

[35] http://dbs.uni-leipzig.de/en/research/projects/gradoop

Gradoop: Apache Flink Third-party library



Gradoop Operators



	Operators: Unary	Operators: Binary	Algorithms
Logical Graph	🟩 Aggregation	🟩 Combination	🟩 BTG Extraction
	🟧 Pattern Matching	🟩 Overlap	🟩 Label Propagation
	🟩 Projection	🟩 Exclusion	🟥 Graph Forecasting
	🟧 Summarization	🟩 Equality	🟪 Gelly Library
	🟩 Call *		
Graph Collection	🟩 Selection	🟩 Union	🟧 Frequent Subgraphs
	🟥 Distinct	🟩 Intersection	
	🟥 Sort	🟩 Difference	
	🟥 Top	🟩 Equality	
	🟥 Apply *		
	🟩 Reduce *		
	🟩 Call *		

* auxiliary

Part VII

Big Machine Learning Libraries

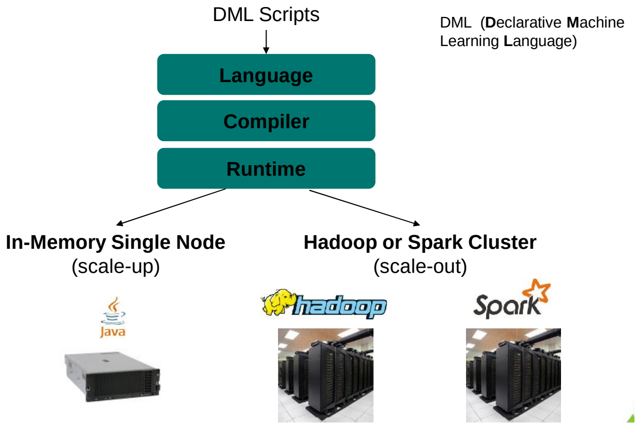
Mahout[36]

- Mahout is a Java library that implements Machine Learning techniques (e.g. classification, clustering, recommendation) on top of the Hadoop framework.
- **Example use cases**:
  - **Recommendation**: Takes users' behavior and tries to find items users might like.
  - **Clustering**: takes e.g. text documents and groups them into groups of topically related documents.

[36] `http://mahout.apache.org/`

SystemML[37]

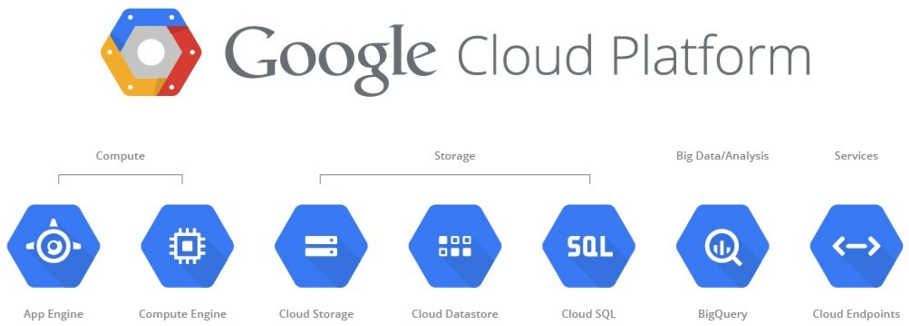
- SystemML provides declarative large-scale machine learning (ML) that aims at flexible specification of ML algorithms and automatic generation of hybrid runtime plans ranging from single node, in-memory computations, to distributed computations such as Apache Hadoop MapReduce and Apache Spark.



[37] http://incubator.apache.org/projects/systemml.html

Google Cloud Machine Learning[38]

- Google Cloud Machine Learning is a managed platform that enables its users to easily build machine learning models, that work on any type of data, of any size.



[38] `https://cloud.google.com/ml/`

Other Big Machine Learning Tools

- Microsoft Azure Machine Learning[39]
- BigML[40]
- Hunk[41]
- RHadoop[42]

[39] https://azure.microsoft.com/en-us/services/machine-learning/
[40] https://bigml.com/
[41] http://www.splunk.com/en_us/products/hunk.html
[42] https://github.com/RevolutionAnalytics/RHadoop/wiki

Part VIII

Open Challenges

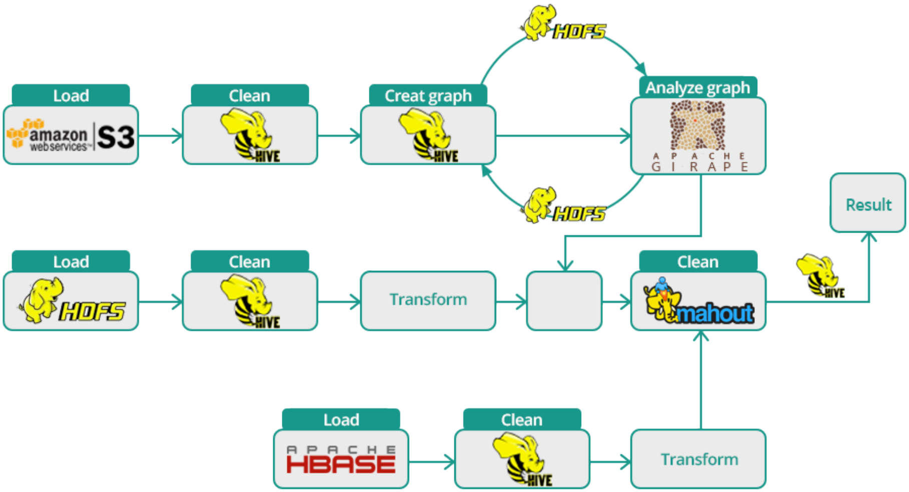
Open Challenge: Pipelining Various Big Data Jobs

- In practice, one of the possible scenarios is that users need to execute a computation that combines various analytics jobs.
- Existing systems do not address the challenges of data construction, transformation and post processing which are often just as problematic as the subsequent computation (computation pipelines)
- New trend of integrated systems: Spark and Flink.
- Emerging pipelining systems: Apache Tez[43] and Apache MRQL[44].

[43]https://tez.apache.org/

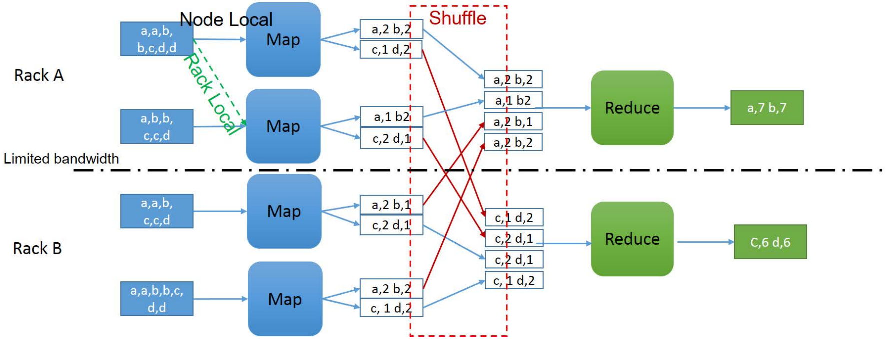
[44]https://mrql.incubator.apache.org/

Open Challenge: Pipelining Various Big Data Jobs



Open Challenge: Data Analytics across Multiple Clusters

- As data are increasingly distributed across different departments, clusters and organizations which may hold different features for the same entities due to the different products they have, there are increasing requirements to apply data analytics across clusters in order to learn comprehensive pattern and knowledge from end users.
- Current big data processing frameworks not natively designed for processing datasets distributed on infrastructures with heterogeneous connectivity.



Open Challenge: Lack of Declarative Interfaces

- In the early days of the Hadoop framework, the lack of declarative languages to express the big data processing tasks has limited its practicality and the wide acceptance and the usage of the framework.
- Several systems (e.g., Pig, Hive, Impala) have been designed to provide high-level languages for expressing big data tasks on top of Hadoop.
- Currently, the systems/stacks of large scale graph and stream processing platforms are suffering from the same challenge.
- **High level language abstractions for expressing big data processing jobs and enable the underlying systems/stack to perform automatic optimization are crucially required**.

$$\begin{tikzcd} \text{“Describing”} \arrow[r] & \textbf{Declarative} \arrow[r] & \textbf{Procedural} & \text{“Doing”} \arrow[l] \end{tikzcd}$$

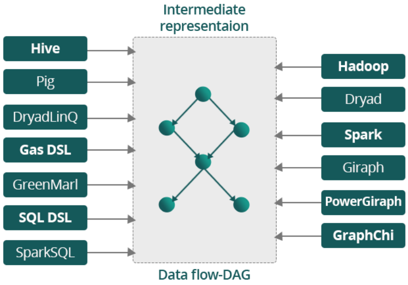
Open Challenge: Benchmarking Challenges[45]

- Designing a good benchmark is a challenging task due to the many aspects that should be considered which can influence the adoption and the usage scenarios of the benchmark.
- **Variety** on algorithms, systems, big dataets characteristics and application domains.
- There is not enough standard benchmarks that can be effectively employed in this domain.

[45] O. Batarfi, R. Elshawi, A. Fayoumi, R. Nouri, S. Beheshti, A. Barnawi, S. Sakr , *Large Scale Graph Processing Systems: Survey and An Experimental Evaluation*. Cluster Computing journal, 2015

Platform-Independent Analytics

- In general, more alternatives usually mean harder decisions for choice.
- Porting the data and the data analytics jobs between different systems is a tedious, time consuming and costly task.
- **Musketeer**[46] and **Rheem**[47] systems proposed a new direction to map the front-end of the big data jobs (e.g., Hive, SparkSQL) to a broad range of back-end execution engines (e.g., Hadoop, Spark, PowerGraph).
- More work is still required to tackle the challenge of providing a platform independence and multi-platform task execution platforms.



[46] I. Gog et al. Musketeer: all for one, one for all in data processing systems. EuroSys, 2015.
[47] D. Agrawal et al. Rheem: Enabling Multi-Platform Task Execution. SIGMOD, 2016.

Part IX

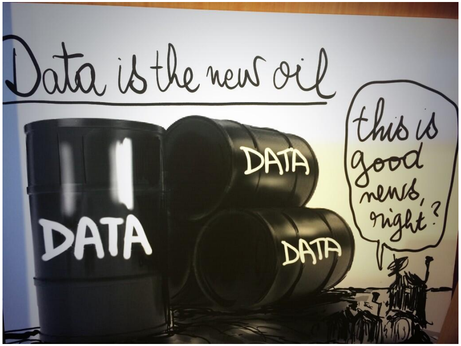
Conclusions

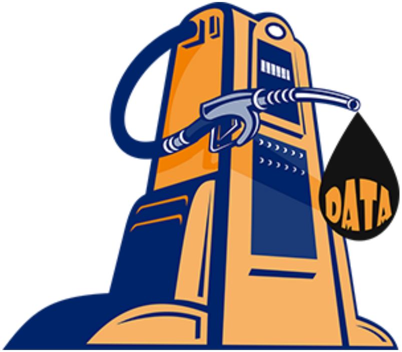
Conclusions

Big Data is the New Oil

and

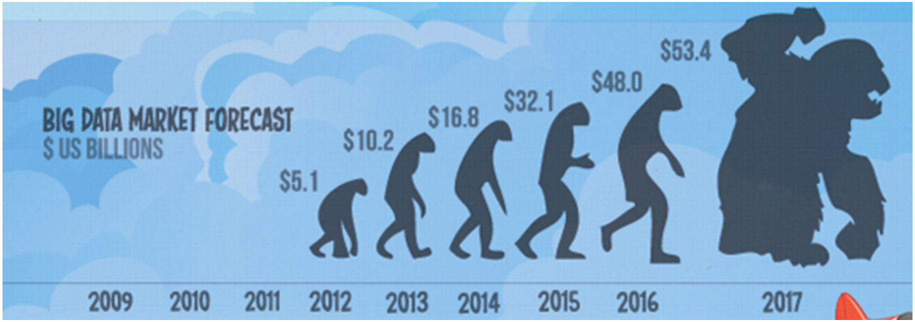
Big Data Processing Systems is the Machinery





Conclusions

- Big Data has a growing market size[48]

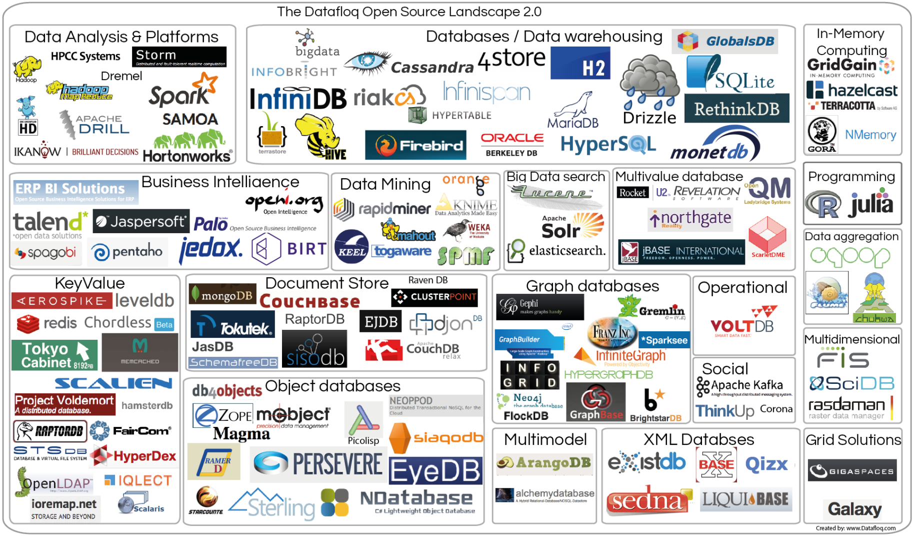


[48]Wikibon Taming Big Data

Conclusions

- Scalable Big Data processing involves various unique challenges
- In the last few years, several distributed data processing systems have been introduced with various design decisions
- Open challenges include declarative interfaces, pipelining and integrating various big data systems, and benchmarking

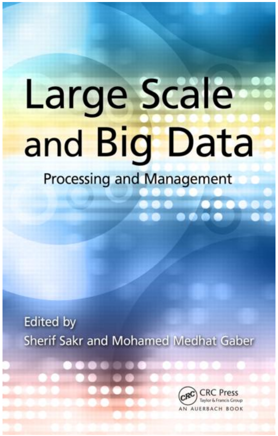
Conclusions



Conclusions

Our Big Data 1.0 Book

Sherif Sakr and Mohamed Gaber. "*Large Scale and Big Data: Processing and Management*", CRC Press, 2014


Large Scale and Big Data

Processing and Management

Edited by

Sherif Sakr and Mohamed Medhat Gaber

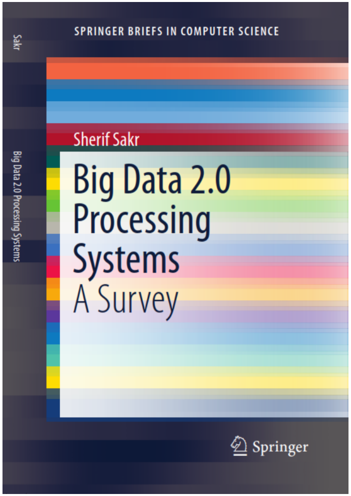
CRC Press

Taylor & Francis Group

AN AUERBACH BOOK

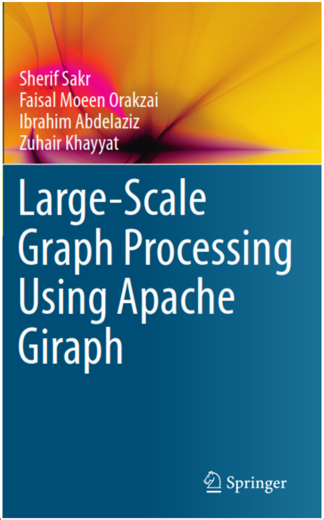
Our Big Data 2.0 Book

Sherif Sakr. "*Big Data 2.0 Processing Systems*", Springer, 2016


SPRINGER BRIEFS IN COMPUTER SCIENCE
Sakr
Big Data 2.0 Processing Systems
Sherif Sakr
Big Data 2.0 Processing Systems
A Survey
Springer

Our Giraph Book

Sherif Sakr et al. "*Large Scale Graph Processing Using Apache Giraph*", Springer, 2016


Sherif Sakr
Faisal Moeen Orakzai
Ibrahim Abdelaziz
Zuhair Khayyat

Large-Scale Graph Processing Using Apache Giraph

Springer

Our Handbook on Big Data

Albert Zomaya and Sherif Sakr. "*Handbook of Big Data Technologies*", Springer, 2017

The End

Thank You
Mahalo
Kiitos
Tack
Toda
Thanks
Grazie
Obrigado
Takk
Gracias
Merci

?
?
?