Big Systems, Big Data

When considering Big Distributed Systems, it can be noted that a major concern is dealing with data, and in particular, Big Data

Have general data issues (such as latency, availability, synchronisation) allied with issues of size

Big Data

Quote from Tim O'Reilly, in What is Web 2.0, (2005) "data is the next Intel Inside."

Hal Varian quote (2009) :

"The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades."

“Moore’s Law”

From early PCs in 1980s to now

Memory has scaled as quickly or more so as CPU cycles

- processor speed has increased from MHz to GHz
- memory from MB to GB

Reduction in price

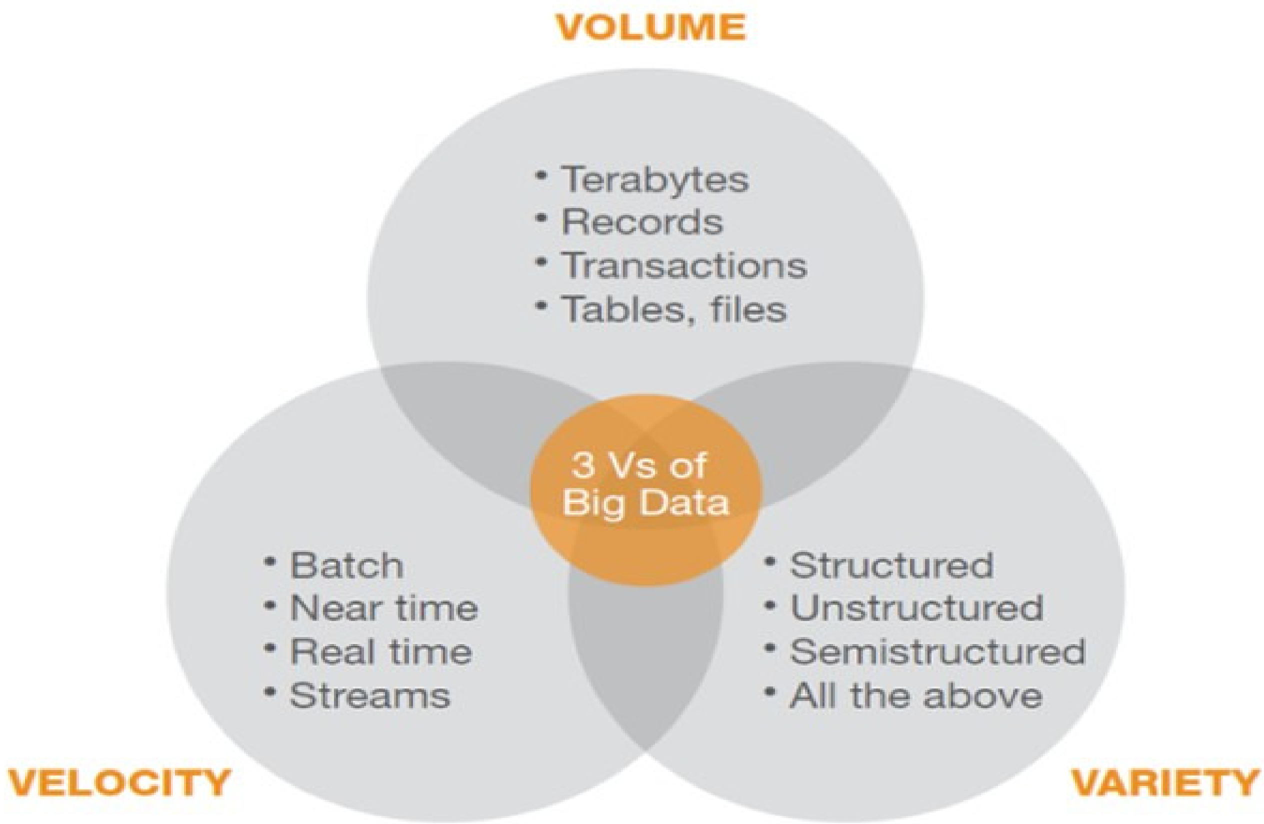
- $1,000 a MB to <$25 a GB

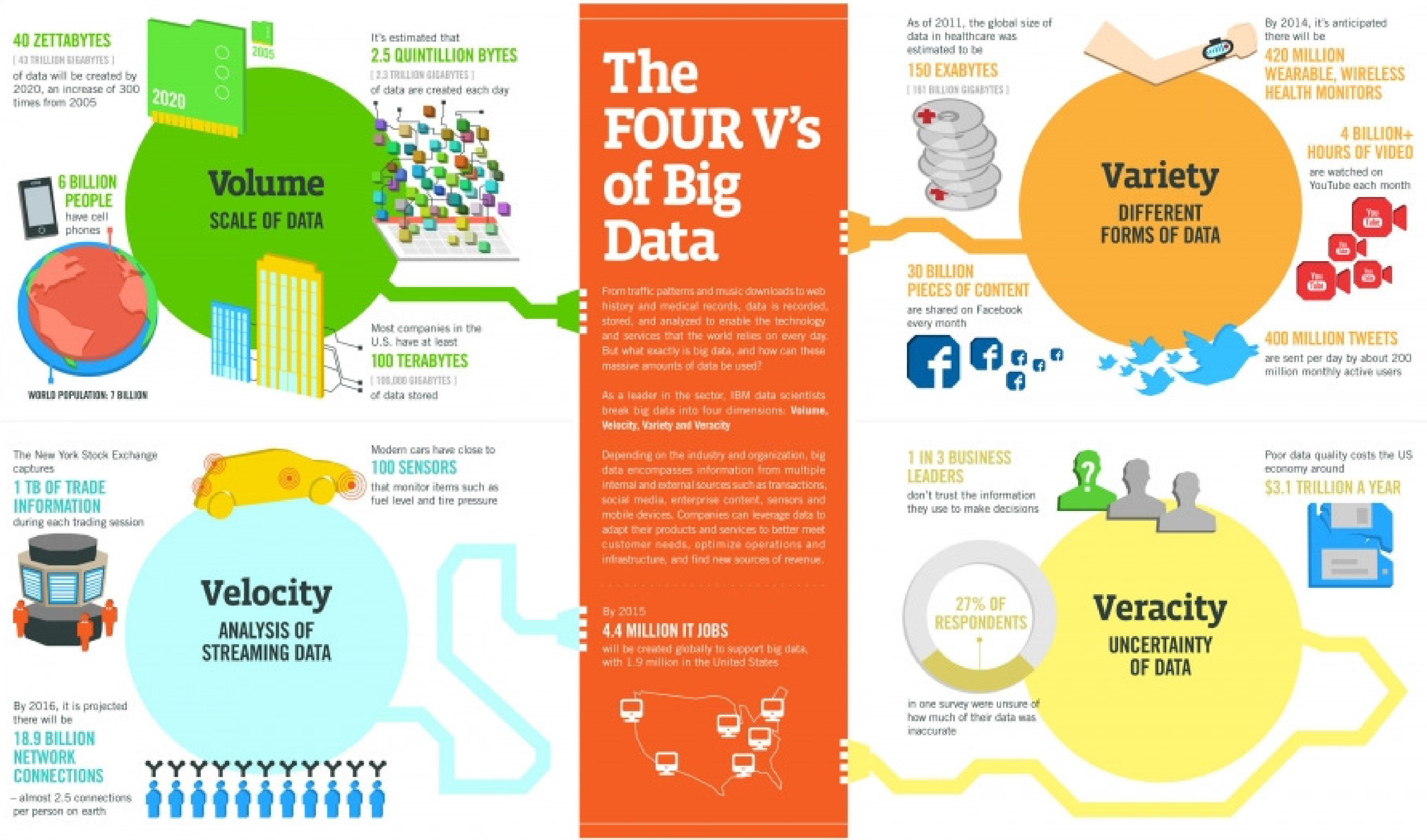
Big Data

How big is Big? When the size of the data itself becomes part of the problem.

- Importance of leveraging data
  - Google
  - Facebook
  - LinkedIn
  - Amaxon

  Google mapping trends, learning from large data sets
- Analytics – the science of analysis


VOLUME
• Terabytes
• Records
• Transactions
• Tables, files
3 Vs of Big Data
• Batch
• Near time
• Real time
• Streams
• Structured
• Unstructured
• Semistructured
• All the above
VELOCITY
VARIETY



Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

Big Data: need scalable databases

Data server I/O bottleneck
- many writes – fast changing data
- lots of unrelated reads (not in cache)
- Examples: Facebook, LinkedIn, Flickr, MySpace etc

- Scaling-up is expensive and doesn’t solve the problem

Solution:
- sharding the database
- look up using key (e.g. user id) which database to use
  - cost-effective
  - increased complexity

Common Sharding Schemes (outline by Dare Obasanjo)

Vertical Partitioning

- segment the application data by having tables related to specific features on their own server. For example,
  - user profile on one database server
  - order history on another
  - reviews and blog on another

Common Sharding Schemes

Range Based Partitioning:

- Large data set for a single feature/entity subdivided across multiple servers
- can use some value range that occurs within entity for predictable partitioning

Examples

- partition by date
- partition by post-code

Needs careful choice of value whose range is used for partitioning

- If not chosen well then leads to unbalanced servers

Common Sharding Schemes

Key or Hash Based Partitioning:

- value from each entity used as input to a hash function whose output determines which database server to use.
- commonly used for user based partitioning

Example

- Have N servers
- Have a numeric value, e.g. Student Number
- Then hashing the numeric value can be done using the Modulo N operation

Suitable when fixed number of servers

Common Sharding Schemes

Directory Based Partitioning:

- Employ a lookup scheme
- Maps each entity key to the database server it resides
- Use directory database or directory web-service for lookup

- Abstracts away partitioning scheme and actual database used
- Loose coupling offers advantages in updating, altering number of servers, changing hash function etc

Sharding Problems

Operations across multiple tables or multiple rows in the same table no longer run on the same server.

Joins and Denormalization

- often not feasible to perform joins that span database shards due to performance constraints since data has to be retrieved from multiple servers

Sharding Problems

Referential integrity

- difficult to enforce data integrity constraints such as foreign keys in a sharded database.
- applications that require referential integrity often have to enforce it in special code and run regular SQL jobs to tidy up.
- dealing with data inconsistency issues (due to lack of referential integrity and denormalization) can become a significant development cost to the service.

Denormalising

- Denormalise to avoid accessing two shards and doing a join
- Denormalise:
  - Duplicate some data so that it's available in two shards
  - Efficient to retrieve data
  - Danger of inconsistency – data updates need to be duplicates

Sharding problems

Changing sharding scheme

- e.g. due to bad choice of partitioning scheme
- need to change partitioning scheme change and relocate data to different servers
- downtime, and recode of some applications

- Directory based partitioning diminishes overhead of changing sharding but is single point of failure

Alternatives to relational databases

NoSQL databases (Non-Relational databases) (Not Only SQL)

- includes many different kinds of solution

- Use tree, graph, key-value pairs … rather than tables, fixed schema
- distributed across many nodes
- support BASE rather than ACID guarantees
- eventually consistent
- flexible schema, denormalized

Original examples: BigTable at Google, and Dynamo at Amazon,

- Cassandra used at Facebook, Twitter, Rackspace,
- HBase: part of the Apache Hadoop project

Bigdata: Structured vs unstructured data

General software engineering data model: describes classes/objects, their properties and relationships (e.g. as might be used to describe real-world entities in the UML Domain Class Diagram). More specific data models used for describing relational databases

- Structured Data conforms to a pre-defined data model or is organized according to a pre-defined structural data model

Bigdata:
Structured vs unstructured data

- Structured data is often represented by a Table, and implemented in a relational database

Employee	Department	Salary
J Jones	Sales	75000
M Smith	Sales	75000
I Walsh	Admin	60000

(have datatypes with well-defined sets of values, ranges of values)

Bigdata: Structured vs unstructured data

Unstructured Data:

- refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner.
- typical examples: natural language text, audio, images, video

Semi-structured data: some structure associated with most data, e.g. sender, time-stamp, hashtags of tweet; logfile structure

SAP example: analysing text

Analyzing Text Data: Entities and Facts

Steven Paul Jobs (born February 24, 1955) is an American business magnate and inventor. He is well known for being the co-founder and chief executive officer of Apple. Jobs also previously served as chief executive of Pixar Animation Studios; he became a member of the board of The Walt Disney Company in 2006, following the acquisition of Pixar by Disney.

Steven Paul Jobs; Jobs	Person
February 24, 1955; 2006	Date
co-founder; chief executive officer; chief executive; member of the board	Title
Apple; Pixar Animation Studios; Pixar; The Walt Disney Company; Disney	Organization_Commercial
American business magnate; inventor; acquisition	Noun_Group or Concept
Jobs also previously served as chief executive of Pixar…	Executive Job Change
Acquisition of Pixar by Disney	Merger and Acquisition

Bigdata:
Structured vs unstructured data

In natural language text analysis

- Need to often
  - Find structural elements
  - Do some keyword queries including operators
  - Map to some sophisticated concept queries e.g.,
    - find all tweets referring to *illness*

Bigdata:
Structured vs unstructured data

- Organisations like International Data Corporation (IDC) and Merril Lynch estimate that 80-90% of data is now unstructured, and that in 2014 almost 90% of data storage was for unstructured data
- Storing and analysing big-data involves both traditional relational databases (RDBs) and the newer MapReduce/Hadoop paradigm

Computing over Big Data Sets

- Google's MapReduce

- See Google tutorial

MapReduce Overview

MapReduce

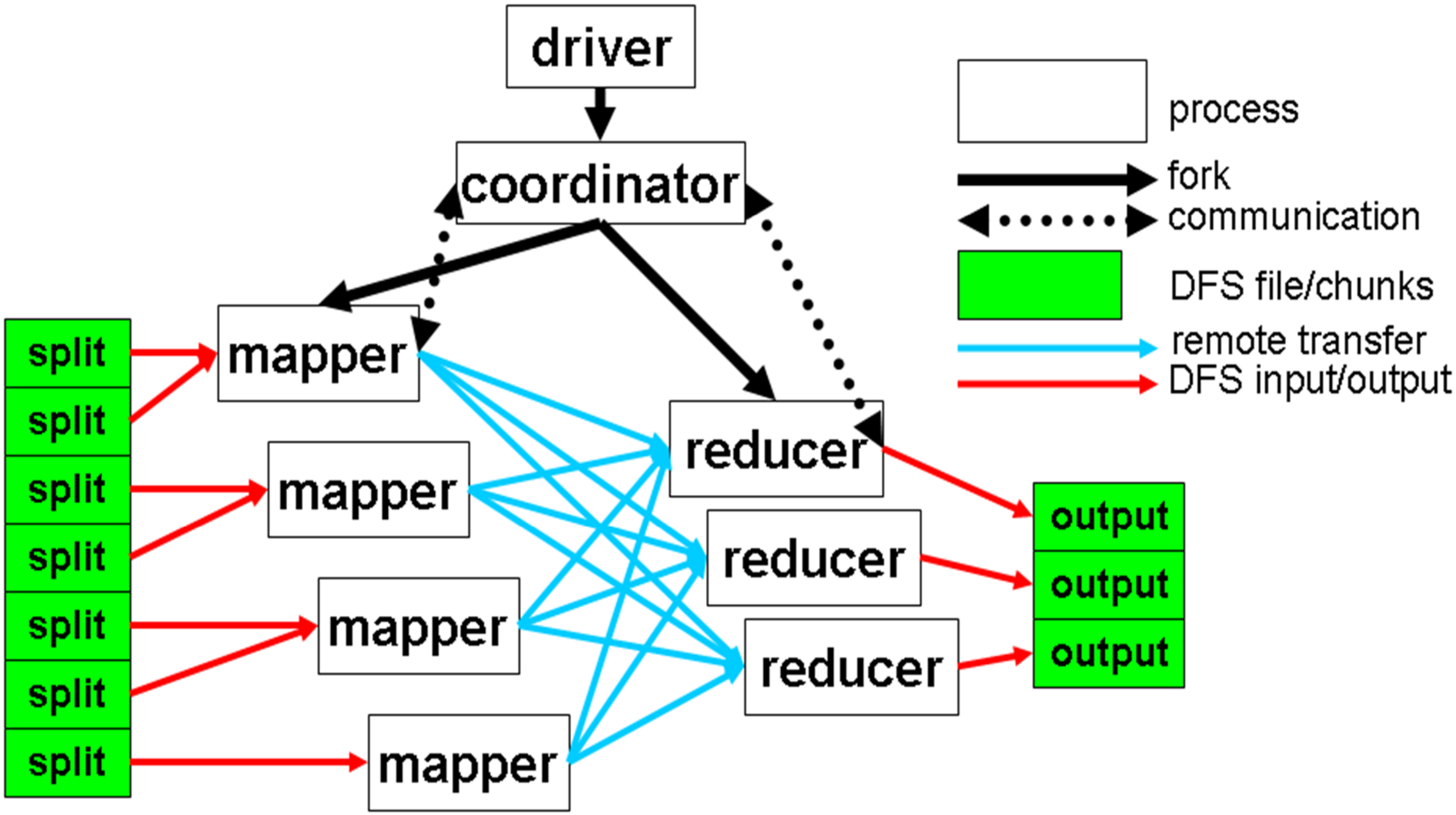
– Framework for parallel computing

– Programmers get simple API

– Framework handles

- parallelization
- data distribution
- load balancing
- fault tolerance

Works with sharding to allow parallel processing of large (terabytes and petabytes) datasets



MapReduce framework overview

Framework

- Map
  - user supplied function gets called for each chunk of input;
  - sort and partition output

- Reduce
  - user supplied function gets called to combine/aggregate the results

MapReduce Main Implementations

- Google's original MapReduce
- Apache Hadoop MapReduce
  - Standard open-source implementation
- Amazon Elastic MapReduce
  - Uses Hadoop MapReduce running on Amazon EC2

MapReduce/Hadoop Simple Example (IBM MapReduce overview)

- The simple problem is to take lots of temperature records for cities around the world (e.g. hourly samples over a year), and do some simple analysis such as maximum in each city over that extended period.
- This could be implemented with a RDB using a simple table with 2 columns … but the data and analysis are much simpler than that implemented by a RDB , and can be done more efficiently without the overhead of a RDB
- Each piece of data can be represented as a key-value pair, where city is the key and temperature is the value.

Key-value pairs model

Dublin, 17
New York, 22
Rome, 32
Toronto, 4
Rome, 33
Dublin, 15
New York, 24
Rome, 32
Toronto, 4
Rome, 30
New York, 18
Toronto, 8

Simple Example

- The MapReduce/Hadoop paradigm is optimized for this kind of key-value pair data analysis, where we can efficiently analysis these files in parallel, and then collect the intermediate results to produce the final result
- From all the data we have collected, we want to find the maximum temperature for each city across all of the data files (note that each file might have the same city represented multiple times).
- Assume we split the data into a number of files (for this example, 5), and each file contains two columns that represent a city and the corresponding temperature recorded in that city for the various measurement days.

Map tasks

- Using the MapReduce/Hadoop framework, let’s break this down into five map tasks, where each mapper works on one of the five files and the mapper task goes through the data and returns the maximum temperature for each city.

For example, the intermediate results produced from one mapper task for one file:

(Toronto, 20) (Dublin, 25) (New York, 22) (Rome, 33)

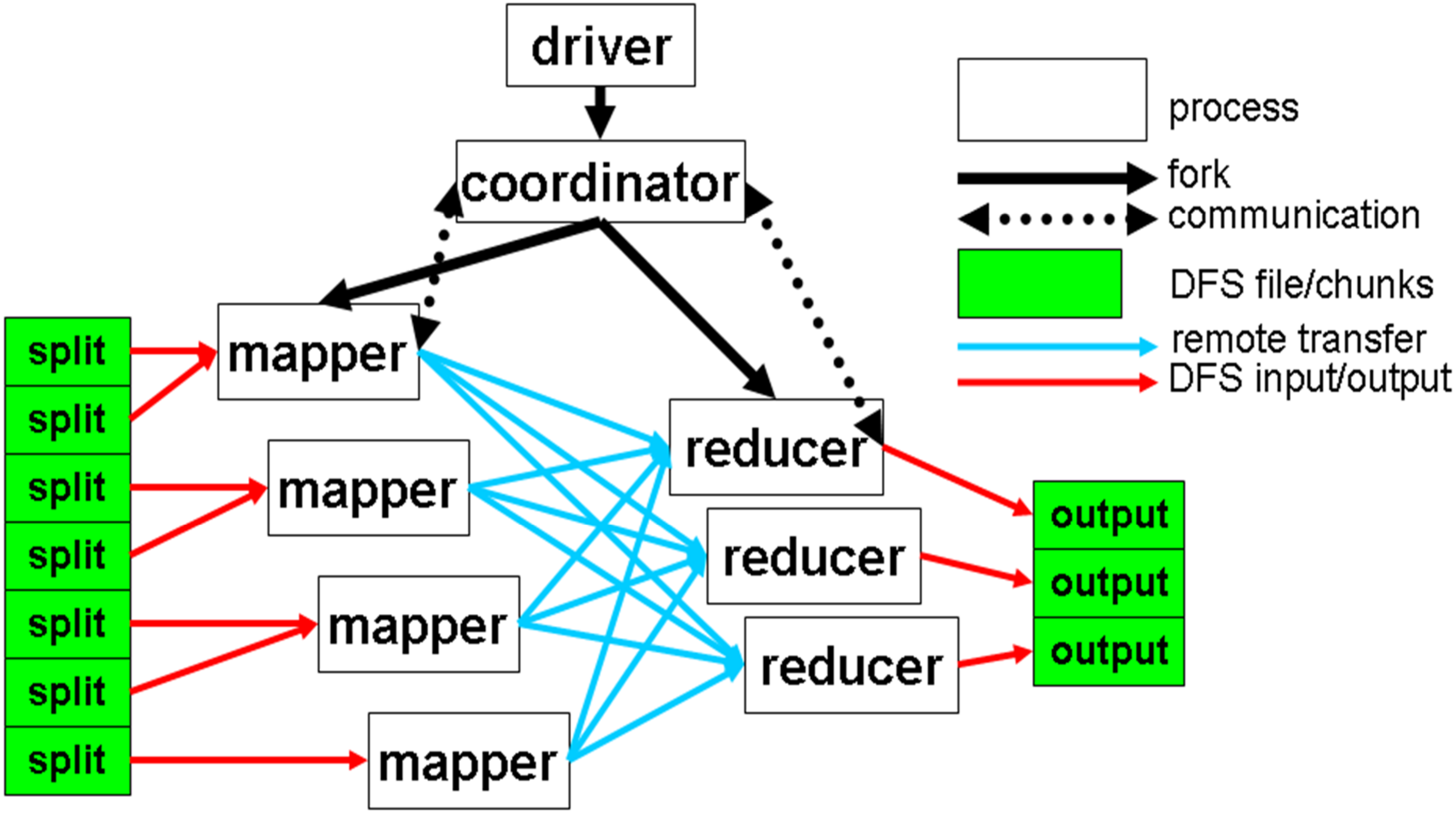
Reduce tasks

- Let's assume the five mapper tasks produce the following intermediate results:

(Toronto, 20) (Dublin, 25) (New York, 22) (Rome, 33)
(Toronto, 18) (Dublin, 27) (New York, 32) (Rome, 37)
(Toronto, 32) (Dublin, 20) (New York, 33) (Rome, 38)
(Toronto, 22) (Dublin, 19) (New York, 20) (Rome, 31)
(Toronto, 31) (Dublin, 22) (New York, 19) (Rome, 30)

- All five of these output streams would be fed into the reduce tasks, which combine the input results and output a single value for each city, producing a final result set as follows:

  (Toronto, 32) (Dublin, 27) (New York, 33) (Rome, 38)



MapReduce framework overview

Map and Reduce in Functional Programming

Map(function, set of values)

• Applies function to each value in the set

Examples

map ‘length ‘((a b) (a) (a b) (a b c)) ⇒ (2 1 2 3)

map (\x. x*2) (4 1 3 2) ⇒ (16 1 9 4)

Reduce(function, set of values)

• Combines all the values using a binary function (e.g., +)

reduce #‘+ 0 ‘(16 1 9 4)) ⇒ 30

First application is (0 + 16), the result of that then combined with next value, 1, (16 + 1), then … (17 + 9) … (26 + 4)

reduce #‘* 1 ‘(2 1 2 3)) ⇒ 12

Big data

Issues include

- Size and nature of data
- Structured/Unstructured data
- Database bottleneck
- Sharding of data/memcaching
- NoSQL alternatives to traditional relational database solutions
- More efficient computation over large data sets, for example, MapReduce

nano-	n	10^{-9} *	--
micro-	m	10^{-6} *	--
milli-	m	10^{-3} *	--
centi-	c	10^{-2} *	--
deci-	d	10^{-1} *	--
(none)	--	10^{0}	2^{0}
deka-	D	10^{1} *	--
hecto-	h	10^{2} *	--
kilo-	k or K **	10^{3}	2^{10}
mega-	M	10^{6}	2^{20}
giga-	G	10^{9}	2^{30}
tera-	T	10^{12}	2^{40}
peta-	P	10^{15}	2^{50}
exa-	E	10^{18} *	2^{60}
zetta-	Z	10^{21} *	2^{70}
yotta-	Y	10^{24} *	2^{80}

** $k = 10^{3}$ and $K = 2^{10}$

- http://wikibon.org/blog/big-data-statistics/
- http://www.idc.com/prodserv/FourPillars/bigData/index.jsp
- http://www.bigdatalandscape.com/blog/unstructured-data