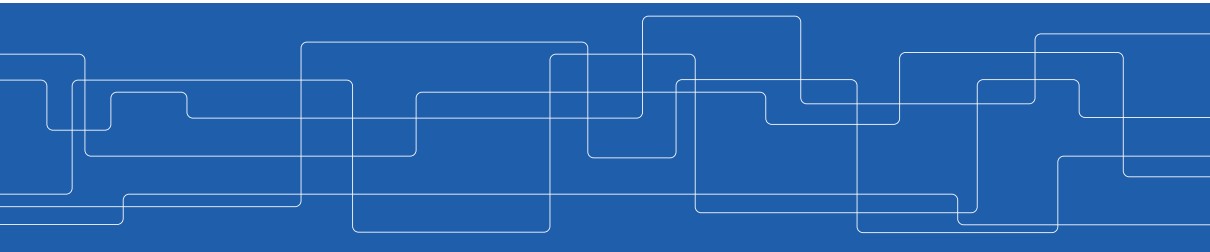
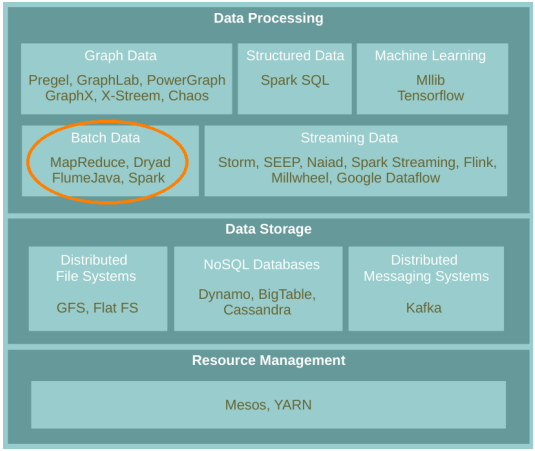
Parallel Processing - MapReduce

Amir H. Payberah
payberah@kth.se
2025-09-8



Where Are We?

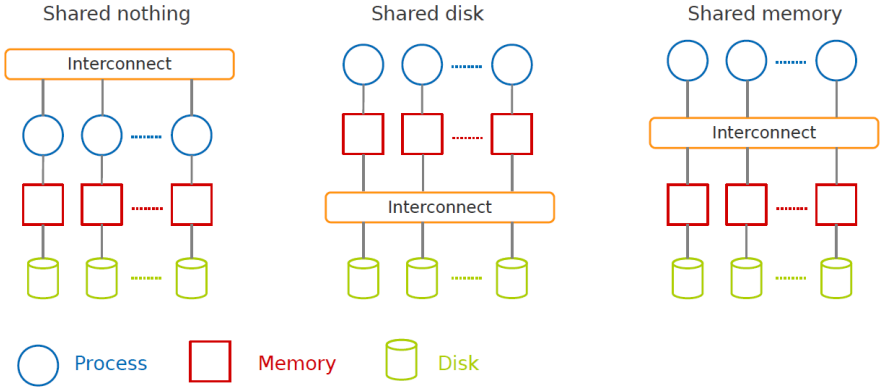

Data Processing
Graph Data
Pregel, GraphLab, PowerGraph
GraphX, X-Streem, Chaos
Structured Data
Spark SQL
Machine Learning
Mllib
Tensorflow
Batch Data
MapReduce, Dryad
FlumeJava, Spark
Streaming Data
Storm, SEEP, Naiad, Spark Streaming, Flink,
Millwheel, Google Dataflow
Data Storage
Distributed
File Systems
GFS, Flat FS
NoSQL Databases
Dynamo, BigTable,
Cassandra
Distributed
Messaging Systems
Kafka
Resource Management
Mesos, YARN

What do we do when there is too much data to process?

Scale Up vs. Scale Out

- Scale up or scale vertically: adding resources to a single node in a system.
- Scale out or scale horizontally: adding more nodes to a system.

Taxonomy of Parallel Architectures



DeWitt, D. and Gray, J. "Parallel database systems: the future of high performance database systems". ACM Communications, 35(6), 85-98, 1992.

MapReduce

- A shared nothing architecture for processing large data sets with a parallel/distributed algorithm on clusters of commodity hardware.

Challenges

Challenges

- How to distribute computation?
- How can we make it easy to write distributed programs?
- Machines failure.

Simplicity

- MapReduce takes care of parallelization, fault tolerance, and data distribution.
- Hide system-level details from programmers.

[http://www.johnlund.com/page/8358/elephant-on-a-scooter.asp]

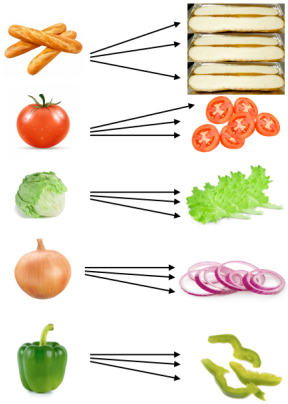
MapReduce Definition

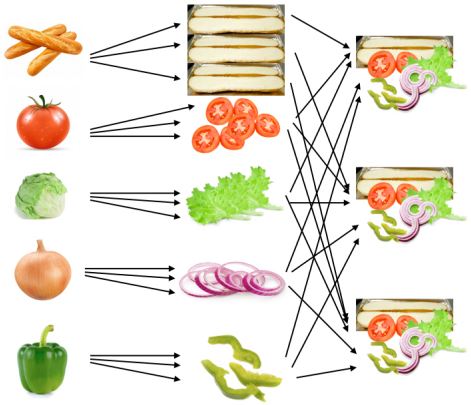
- A programming model: to batch process large data sets (inspired by functional programming).
- An execution framework: to run parallel algorithms on clusters of commodity hardware.

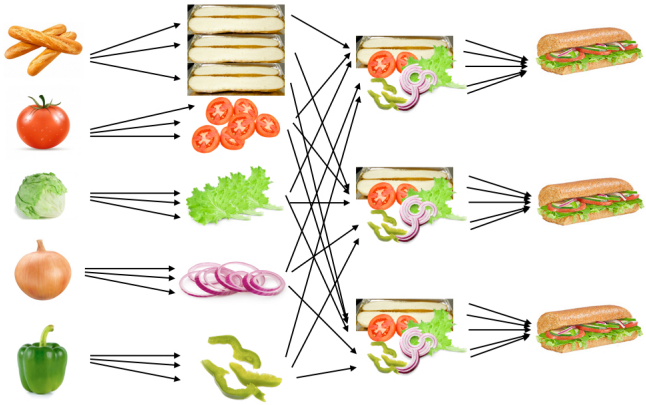
Programming Model

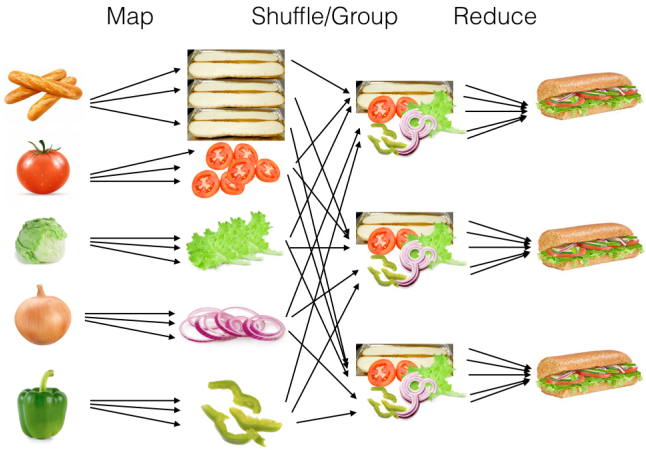
I'm glad I don't have to hunt for my food,

I don't even know where sandwiches live.





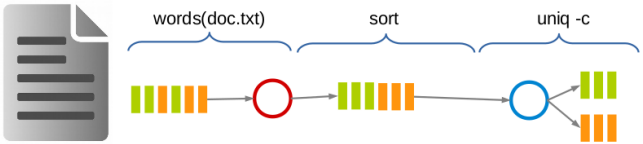



Map
Shuffle/Group
Reduce


hello world
ICANHASCHEEZBURGER.COM

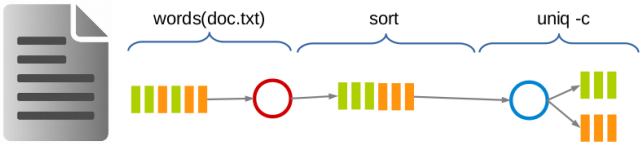
Word Count

- Count the number of times each distinct word appears in the file
- If the file fits in memory: `words(doc.txt) | sort | uniq -c`



Word Count

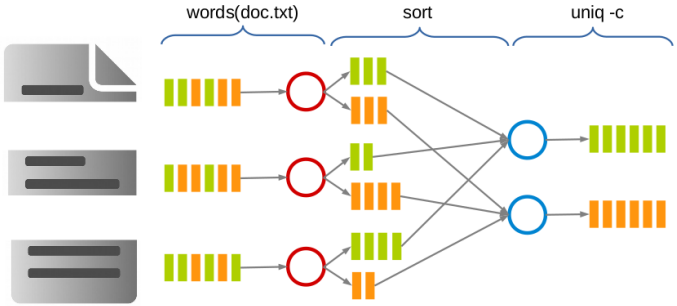
- Count the number of times each distinct word appears in the file
- If the file fits in memory: `words(doc.txt) | sort | uniq -c`



- If not?

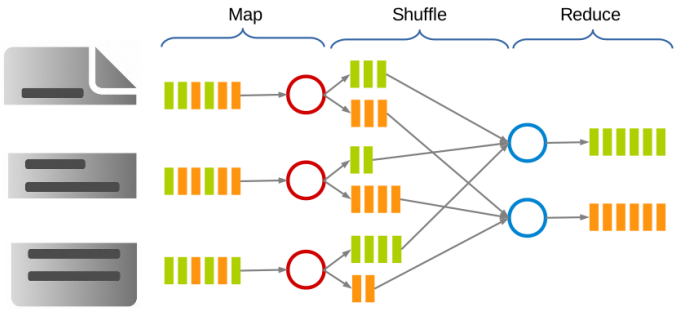
Data-Parallel Processing (1/2)

- Parallelize the data and process.



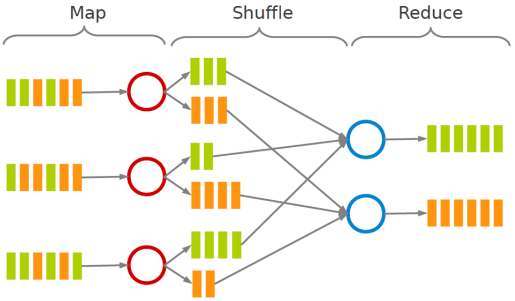
Data-Parallel Processing (2/2)

- MapReduce



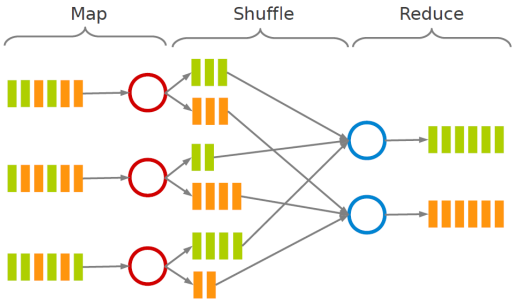
MapReduce Stages - Map

- Each Map task (typically) operates on a single HDFS block.
- Map tasks (usually) run on the node where the block is stored.
- Each Map task generates a set of intermediate key/value pairs.



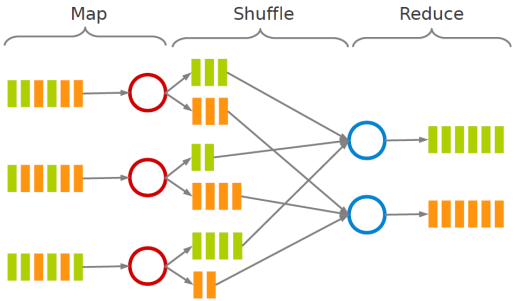
MapReduce Stages - Shuffle and Sort

- Sorts and consolidates intermediate data from all mappers.
- Happens after all Map tasks are complete and before Reduce tasks start.



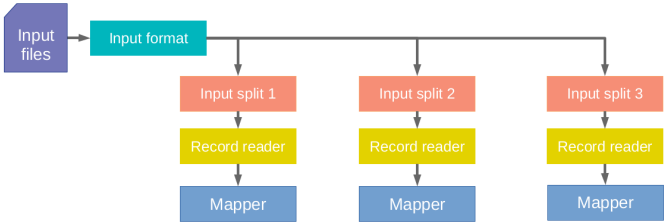
MapReduce Stages - Reduce

- Each Reduce task operates on all intermediate values associated with the same intermediate key.
- Produces the final output.

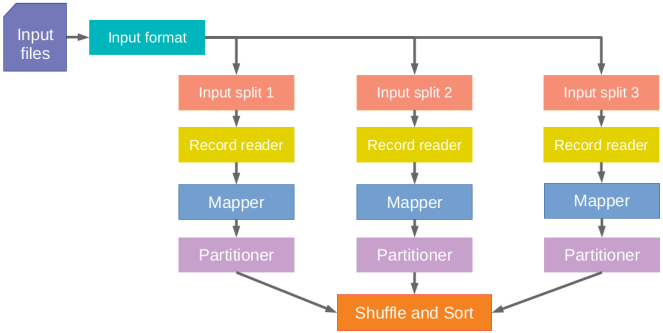


MapReduce Data Flow (1/5)

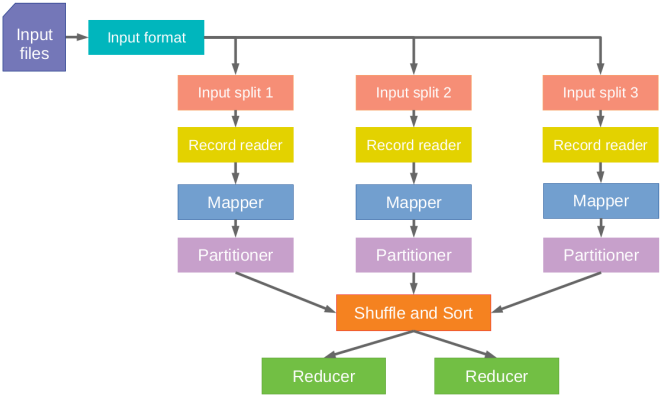
MapReduce Data Flow (2/5)



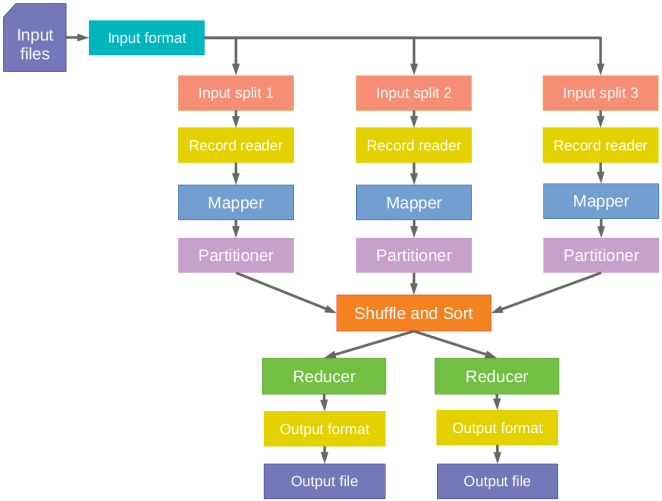
MapReduce Data Flow (3/5)



MapReduce Data Flow (4/5)

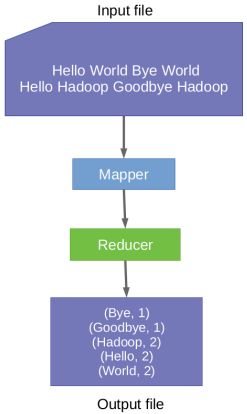


MapReduce Data Flow (5/5)

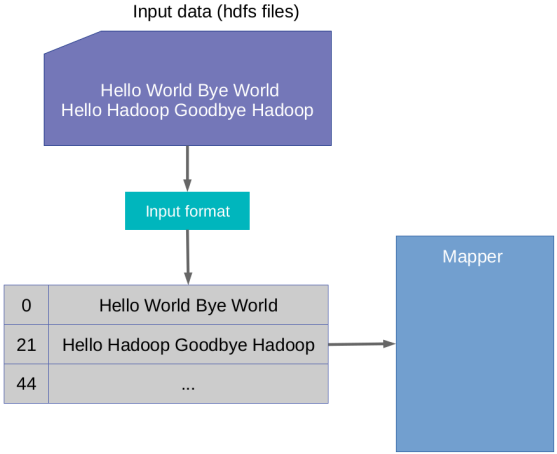


Word Count in MapReduce

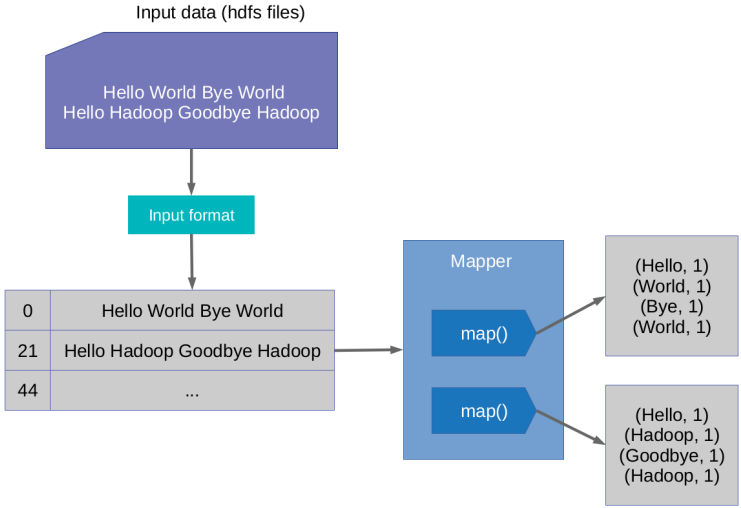
- Consider doing a word count of the following file using MapReduce



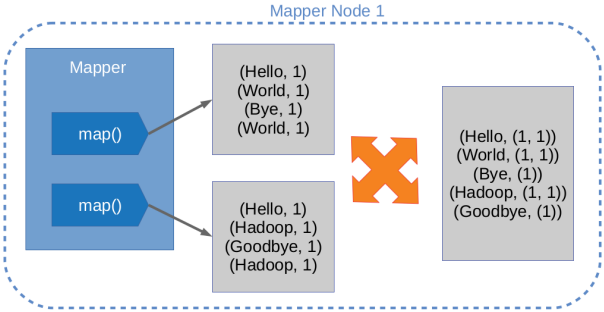
Word Count in MapReduce - Map (1/2)



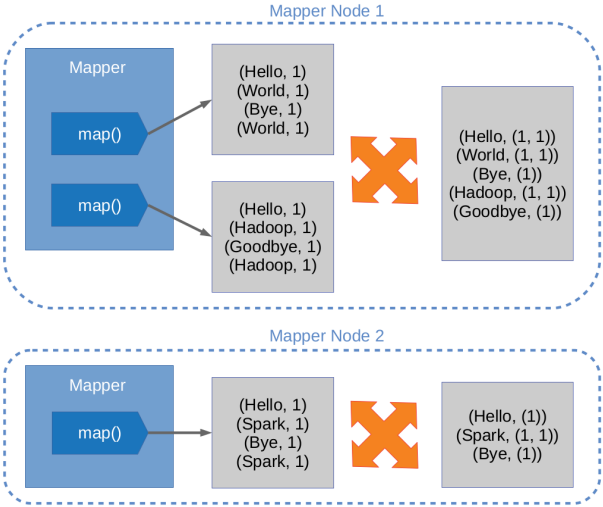
Word Count in MapReduce - Map (2/2)



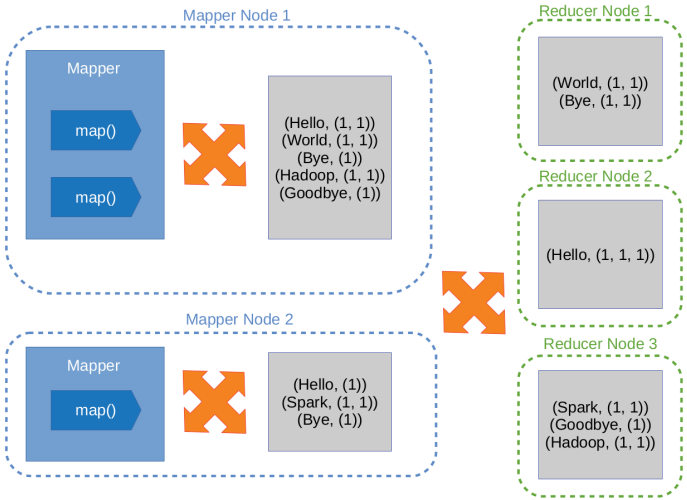
Word Count in MapReduce - Shuffle and Sort (1/3)



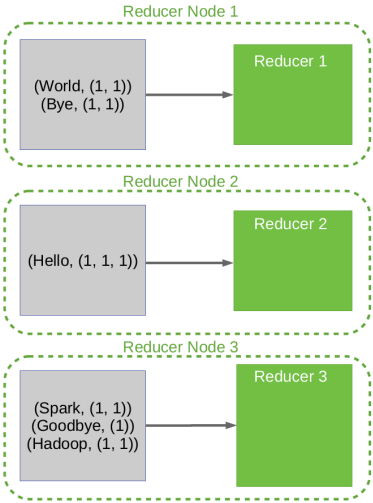
Word Count in MapReduce - Shuffle and Sort (2/3)



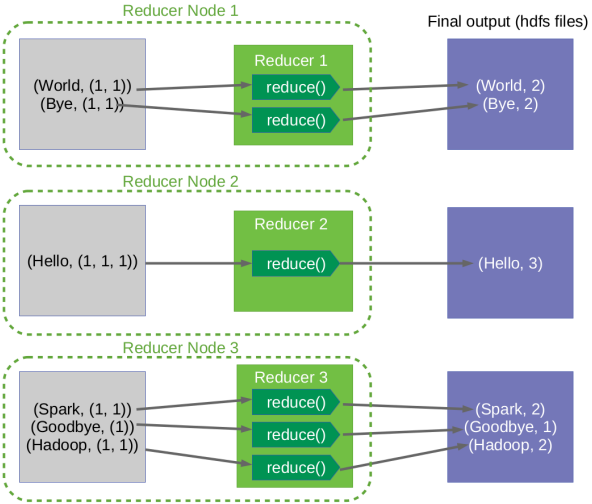
Word Count in MapReduce - Shuffle and Sort (3/3)



Word Count in MapReduce - Reduce (1/2)



Word Count in MapReduce - Reduce (2/2)



Go to www.menti.com, and use the code 4814 2124

- The most expensive task?

1. Map
2. Shuffle
3. Reduce

Example: Word Count - map

```
public static class MyMap extends Mapper<...> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);

    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

Example: Word Count - reduce

```
public static class MyReduce extends Reducer<...> {
  public void reduce(Text key, Iterator<...> values, Context context)
    throws IOException, InterruptedException {
    int sum = 0;

    while (values.hasNext())
      sum += values.next().get();

    context.write(key, new IntWritable(sum));
  }
}
```

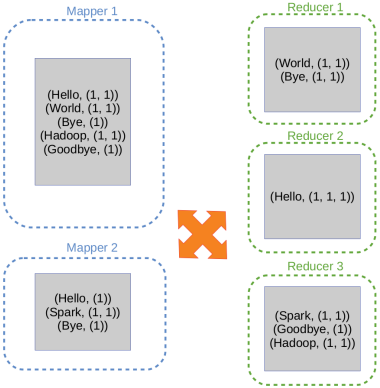
Example: Word Count - driver

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = new Job(conf, "wordcount");

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  job.setMapperClass(MyMap.class);
  job.setCombinerClass(MyReduce.class);
  job.setReducerClass(MyReduce.class);

  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.waitForCompletion(true);
}
```

Local Aggregation

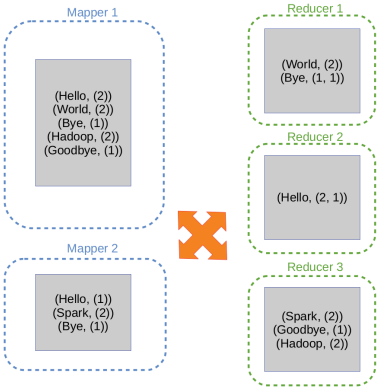
Local Aggregation - In-Map Combiner (1/2)

- In some cases, there is significant repetition in the intermediate keys produced by each map task, and the reduce function is commutative and associative.



Local Aggregation - In-Map Combiner (2/2)

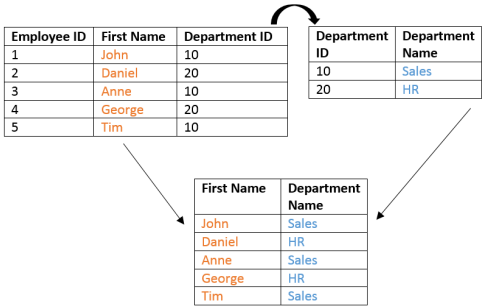
- Merge partially data before it is sent over the network to the reducer.
- Typically the same code for the combiner and the reduce function.



Joining

Joins

- Joins are relational constructs you use to combine relations together.
- In MapReduce joins are applicable in situations where you have two or more datasets you want to combine.



Employee ID	First Name	Department ID
1	John	10
2	Daniel	20
3	Anne	10
4	George	20
5	Tim	10

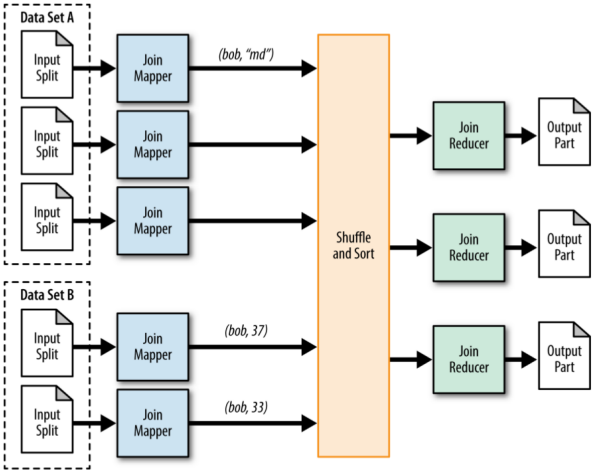
Department ID	Department Name
10	Sales
20	HR

First Name	Department Name
John	Sales
Daniel	HR
Anne	Sales
George	HR
Tim	Sales

Joins - Two Strategies

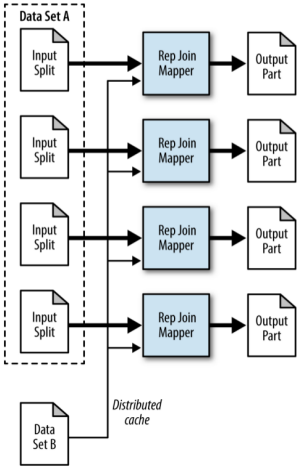
- Reduce-side join
  - Repartition join
  - When joining two or more large datasets together
- Map-side join
  - Replication join
  - When one of the datasets is small enough to cache

Joins - Reduce-Side Join



[M. Donald et al., MapReduce design patterns, O'Reilly, 2012.]

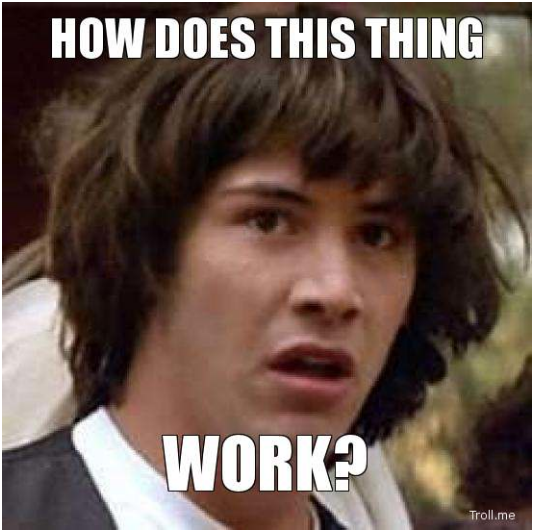
Joins - Map-Side Join



[M. Donald et al., MapReduce design patterns, O'Reilly, 2012.]

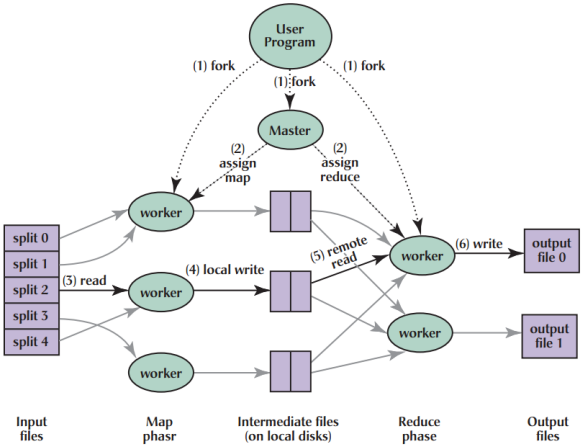
Count The Same Length Words?

https://tinyurl.com/bp6zsenu


HOW DOES THIS THING
WORK?
Troll.me

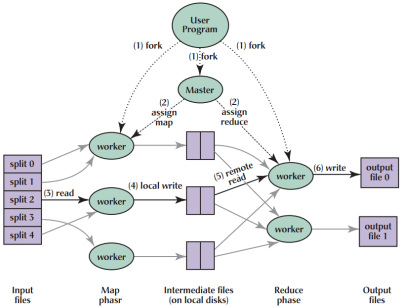
Implementation

Architecture



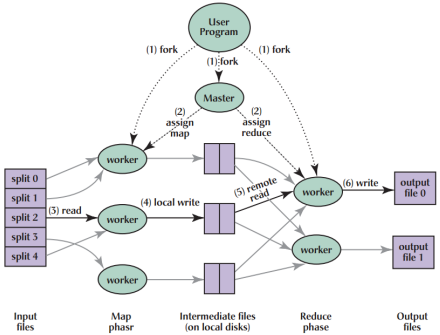
MapReduce Execution (1/7)

- The user program divides the input files into M splits.
  - A typical size of a split is the size of a HDFS block (64 MB).
  - Converts them to key/value pairs.
- It starts up many copies of the program on a cluster of machines.



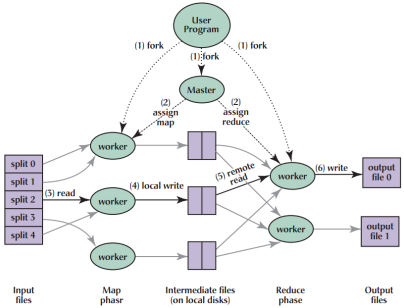
MapReduce Execution (2/7)

- One of the copies of the program is master, and the rest are workers.
- The master assigns works to the workers.
  - It picks idle workers and assigns each one a map task or a reduce task.



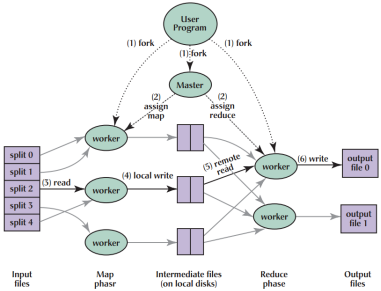
MapReduce Execution (3/7)

- A map worker reads the contents of the corresponding input splits.
- It parses key/value pairs out of the input data and passes each pair to the user defined map function.
- The key/value pairs produced by the map function are buffered in memory.



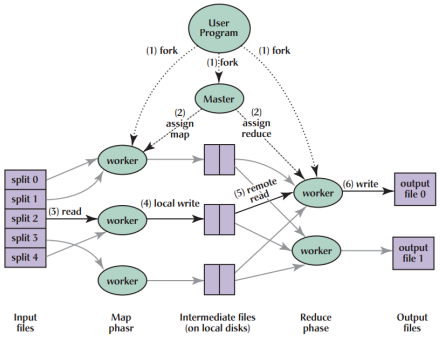
MapReduce Execution (4/7)

- The buffered pairs are periodically written to local disk.
  - They are partitioned into R regions (hash(key) mod R).
- The locations of the buffered pairs on the local disk are passed back to the master.
- The master forwards these locations to the reduce workers.



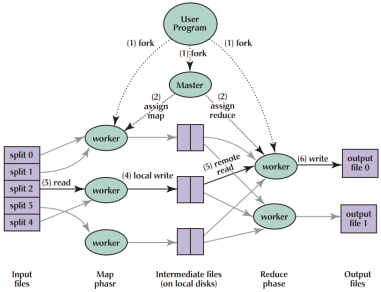
MapReduce Execution (5/7)

- A reduce worker reads the buffered data from the local disks of the map workers.
- When a reduce worker has read all intermediate data, it sorts it by the intermediate keys.



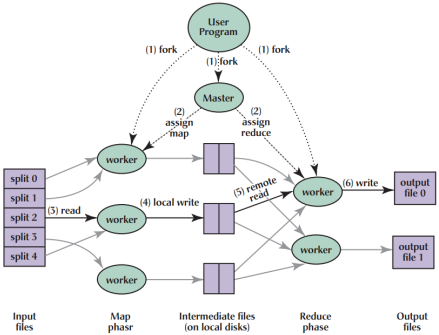
MapReduce Execution (6/7)

- The reduce worker iterates over the intermediate data.
- For each unique intermediate key, it passes the key and the corresponding set of intermediate values to the user defined reduce function.
- The output of the reduce function is appended to a final output file for this reduce partition.

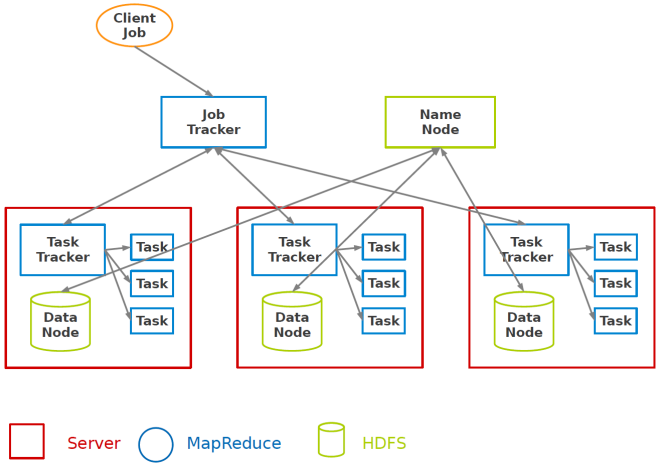


MapReduce Execution (7/7)

- When all map tasks and reduce tasks have been completed, the master wakes up the user program.



Hadoop MapReduce and HDFS



Go to www.menti.com, and use the code 4878 2868

- How to handle a failure if a map task fails after it finishes its task?

1. Re-execute the task on a different machine
2. Do not need to re-xecute, because the task is completed

Fault Tolerance - Worker

- Detect failure via periodic heartbeats.

Fault Tolerance - Worker

- Detect failure via periodic heartbeats.
- Re-execute in-progress map and reduce tasks.

Fault Tolerance - Worker

- Detect failure via periodic heartbeats.
- Re-execute in-progress map and reduce tasks.
- Re-execute completed map tasks: their output is stored on the local disk of the failed machine and is therefore inaccessible.

Fault Tolerance - Worker

- Detect failure via periodic heartbeats.
- Re-execute in-progress map and reduce tasks.
- Re-execute completed map tasks: their output is stored on the local disk of the failed machine and is therefore inaccessible.
- Completed reduce tasks do not need to be re-executed since their output is stored in a global filesystem.

Fault Tolerance - Master

- State is periodically checkpointed: a new copy of master starts from the last checkpoint state.

Questions

- What ethical limitations do you see in the MapReduce framework?

Possible Answers

- Concentration of power

Possible Answers

- Concentration of power
  - Designed for Google-scale workloads: reinforces big tech dominance.
  - Smaller communities lack access to infrastructure at this scale.

Possible Answers

- Concentration of power
  - Designed for Google-scale workloads: reinforces big tech dominance.
  - Smaller communities lack access to infrastructure at this scale.
- Bias toward data-rich groups

Possible Answers

- Concentration of power
  - Designed for Google-scale workloads: reinforces big tech dominance.
  - Smaller communities lack access to infrastructure at this scale.
- Bias toward data-rich groups
  - Skew + volume-based weighting means majority data dominates: minority patterns are systematically underrepresented.

Possible Answers

- Concentration of power
  - Designed for Google-scale workloads: reinforces big tech dominance.
  - Smaller communities lack access to infrastructure at this scale.
- Bias toward data-rich groups
  - Skew + volume-based weighting means majority data dominates: minority patterns are systematically underrepresented.
- Erasure of context

Possible Answers

- Concentration of power
  - Designed for Google-scale workloads: reinforces big tech dominance.
  - Smaller communities lack access to infrastructure at this scale.
- Bias toward data-rich groups
  - Skew + volume-based weighting means majority data dominates: minority patterns are systematically underrepresented.
- Erasure of context
  - Forcing all data into key-value pairs strips nuance from social, cultural, or relational data: certain knowledge systems become uncomputable.

Questions

- MapReduce struggles with data skew (when some keys have far more records). If one group's data is much larger than others, how might the system overrepresent or bias toward that group in the results?

Possible Answers

- One key with far more records dominates the job → data skew.

Possible Answers

- One key with far more records dominates the job → data skew.
- Overrepresentation: large groups outweigh smaller ones in results.
- Delayed tasks: hot keys slow down reducers, jobs may favor majority data.

Possible Answers

- One key with far more records dominates the job → data skew.
- Overrepresentation: large groups outweigh smaller ones in results.
- Delayed tasks: hot keys slow down reducers, jobs may favor majority data.
- Critical lens: reinforces power imbalances, assumes "more data = more importance".

Questions

- How to address them?

Possible Answers

- Reducer logic

Possible Answers

- Reducer logic
  - Apply weighted aggregation so small groups are not drowned out by majority counts.
  - Use stratified sampling to ensure minority groups are proportionally represented in results.

Possible Answers

- Reducer logic
  - Apply weighted aggregation so small groups are not drowned out by majority counts.
  - Use stratified sampling to ensure minority groups are proportionally represented in results.
- Job scheduling

Possible Answers

- Reducer logic
  - Apply weighted aggregation so small groups are not drowned out by majority counts.
  - Use stratified sampling to ensure minority groups are proportionally represented in results.
- Job scheduling
  - Split heavy keys across multiple reducers (e.g., key-splitting technique).
  - Assign dynamic load balancing so straggler reducers with hot keys don't dominate job completion.
  - Allow priority scheduling for underrepresented groups, ensuring their outputs are processed fairly.

Questions

- Possible negative impacts of skew mitigation?

Possible Answers

- Weighted aggregation

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling
  - Risk of introducing sampling bias if strata are not chosen carefully.

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling
  - Risk of introducing sampling bias if strata are not chosen carefully.
- Key splitting and dynamic load balancing

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling
  - Risk of introducing sampling bias if strata are not chosen carefully.
- Key splitting and dynamic load balancing
  - Adds system complexity, harder to debug and maintain.
  - May increase latency if jobs need more coordination across reducers.

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling
  - Risk of introducing sampling bias if strata are not chosen carefully.
- Key splitting and dynamic load balancing
  - Adds system complexity, harder to debug and maintain.
  - May increase latency if jobs need more coordination across reducers.
- Priority scheduling for small groups

Possible Answers

- Weighted aggregation
  - May distort results if the goal is to reflect true frequency (over-correcting minority data).
  - Could be misused to justify fairness while hiding actual distributions.
- Stratified sampling
  - Risk of introducing sampling bias if strata are not chosen carefully.
- Key splitting and dynamic load balancing
  - Adds system complexity, harder to debug and maintain.
  - May increase latency if jobs need more coordination across reducers.
- Priority scheduling for small groups
  - Slows down majority group processing.

Summary

Summary

- Scaling out: shared nothing architecture
- MapReduce
  - Programming model: Map and Reduce
  - Execution framework

References

- J. Dean et al., "MapReduce: simplified data processing on large clusters", Communications of the ACM, 2008.
- J. Lin et al., "Data-intensive text processing with MapReduce", Synthesis Lectures on Human Language Technologies, 2010.

Questions?