Map Reduce
Group Meeting

Yasmine Badr

10/07/2014

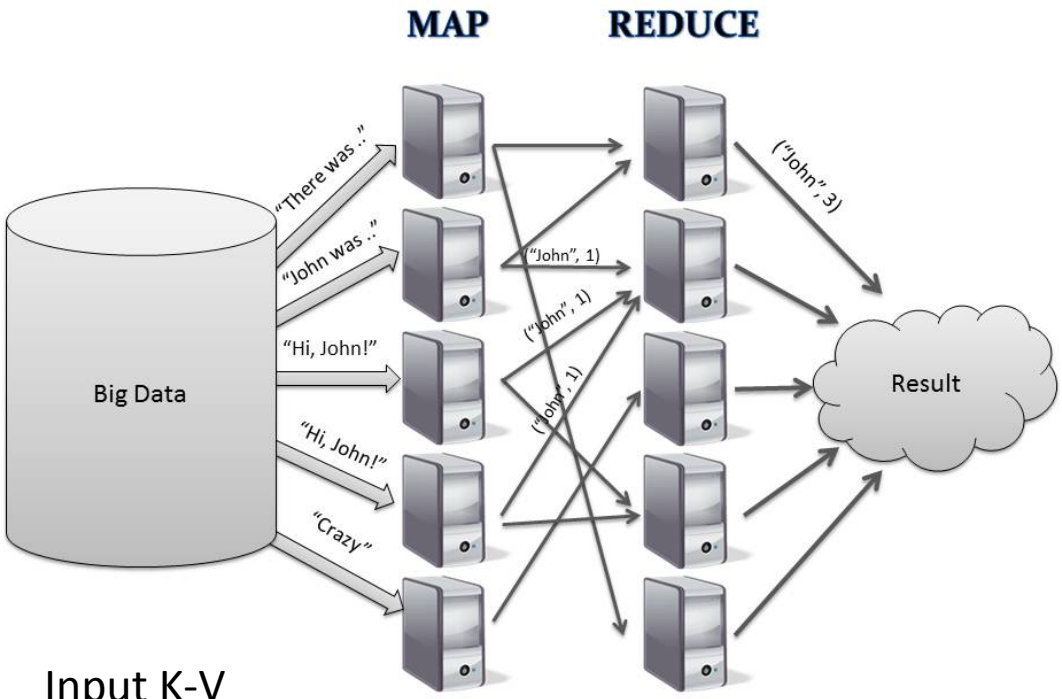
A lot of material in this presentation has been adopted from the original MapReduce paper in OSDI 2004

What is Map Reduce?

- Programming paradigm/model for processing of large data sets in parallel on large distributed clusters
- The framework or runtime system takes care of:
  - partitioning input data,
  - scheduling,
  - fault tolerance and
  - communication
- Programs are written in functional style
- Basically an abstraction

Programming Model of MapReduce

- Input & output are sets of key-value pairs
- Programmer expresses computation as 2 functions: Map and Reduce



Programming Model of MapReduce (cont'd)

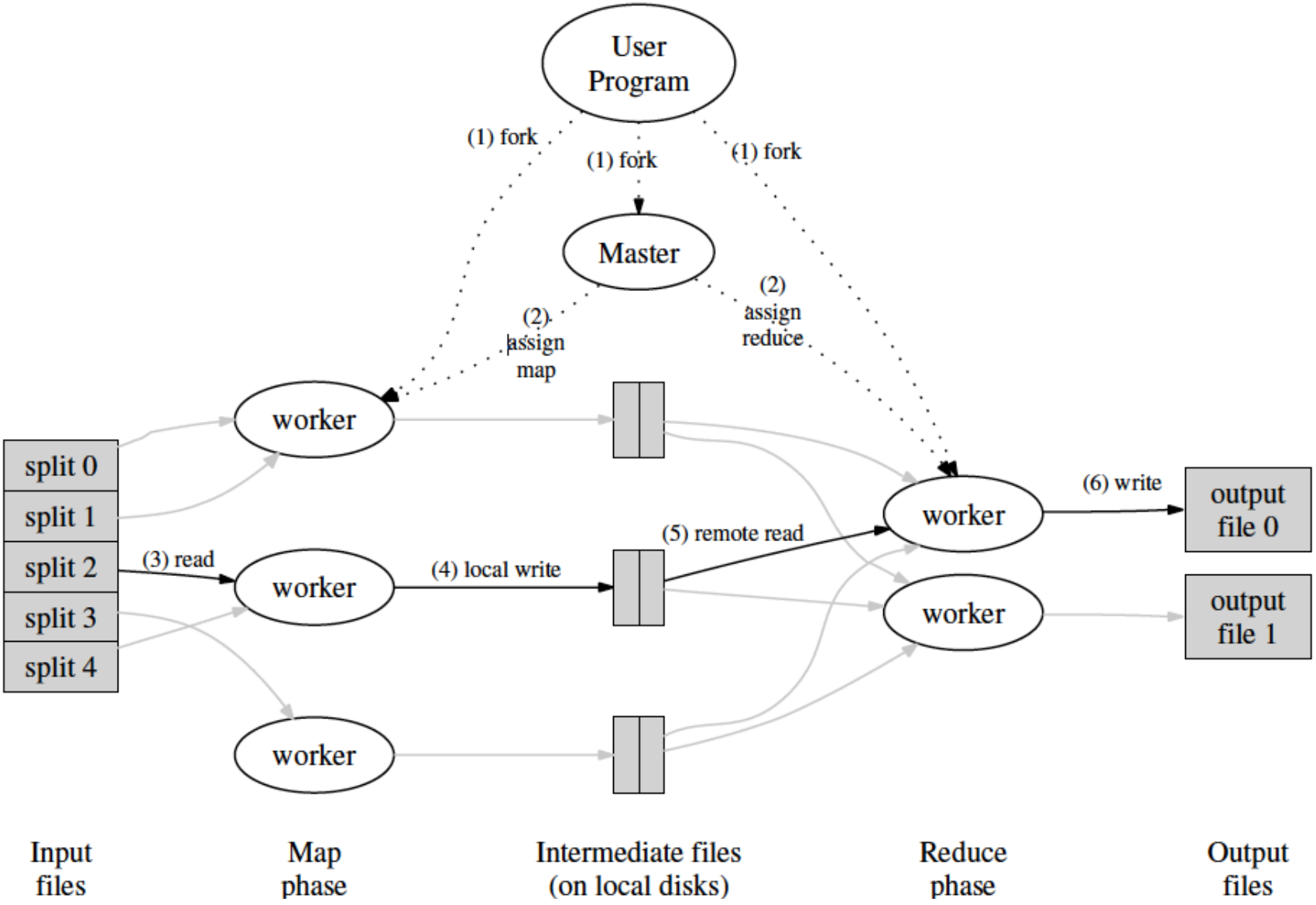
- **Map** function:
  - Takes input K-V pair and produces a set of intermediate K-V pairs
- **MapReduce library**
  - groups together all intermediate values associated with same intermediate key & passes them to Reduce function
- **Reduce** function:
  - accepts an intermediate key and a set of values for that key.
  - merges together these values to form a possibly smaller set of values.
  - Typically just zero or one output value is produced per Reduce

Example: Pseudo code to count number of occurrences of words [O.P.]

```
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));
```

Execution Overview [O.P]



Execution Overview (cont'd)

1. **Map Reduce library** splits input files into M pieces. Then it starts up multiple **workers**. One of them is **master**.
2. Master picks idle workers and assigns each one a map or reduce task.
   - M MAP tasks and R reduce tasks in total
3. A **MAP worker**, when assigned **a MAP task**
   - Parses its share of key-value pairs and passes each pair to the **Map function**.
   - Output of Map is intermediate K-V pairs
     - buffered in memory.
     - periodically partitioned into **R** regions on local disk (uses hashing function)
     - Locations on the disk are passed to master, which forwards it to the **reduce workers**
4. When a **REDUCE** worker is notified by **master**, it
   - Uses remote procedure calls to read data from local disks of MAP workers
   - Sorts data by intermediate keys, so all occurrences of same key are grouped together
   - Iterates on sorted data and for each key calls the **Reduce function**.
   - Appends output of **Reduce function** to final output file for this reduce partition
5. When all MAP and REDUCE workers complete, master wakes up user program→ back to user code

Fault Tolerance

- To tolerate failing machines:
  - Worker failure:
    - Master pings worker periodically
    - No response➔ marked as failed
      ➔map tasks completed or in progress are eligible for re-scheduling
      ➔Workers executing reduce tasks are notified of the re-execution
    - In case of large-scale failures (like network maintenance on a cluster) master re-executes tasks done by unreachable workers
  - Master failure:
    - Unlikely failure but can be handled by writing periodic checkpoints and starting another master

Map Reduce libraries

- HADOOP
  - Open Source by Apache
  - In Java
  - Can be used with C++, Java, Python
  - Uses Hadoop Distributed File System (HDFS)
  - Interfaces on top of it:
    - Amazon Elastic Map Reduce to use Amazon cloud compute
    - Hive
    - Cloudera
  - Most popular
- MARS
  - On GPUs
  - In CUDA
- Others including open and closed source

Other Examples

- **Distributed Grep:**
  - Map function: emits a line if it matches pattern.
  - Reduce function: just copies the supplied intermediate data to the output
- **Distributed Sort:**
  - **Map** function: extracts the key from each record, and produces a (key, record) pair.
  - **Reduce** function: emits all pairs unchanged.
  - It depends on the partition and order facilities in the execution overview
  - Incl. startup overhead, performed similar to best reported result for TeraSort benchmark at that time

Google used it for

- Large scale Machine Learning problems
- Indexing
- Clustering problems for google news
- Large-scale graph computations

Task Granularity: M & R

- Map phase is M pieces
- Reduce phase is R pieces
- M, R>> number of machines
  - Improves dynamic load balancing
  - Faster recovery when a worker fails
- **Practical bounds:**
  - Master takes O(M+R) scheduling decisions
  - Master keeps O(M*R) state in memory (~byte each)
  - R usually constrained by users because output of each reduce task ends up in separate file
- In practice (2004): they choose M so that each individual task ~16 to 64MB of input data so that locality optimization is most effective (M=200K, R=5K on 2000 workers)

Map Reduce…

- Proposed by Google in 2003
- Later Open sourced
- According to Data Center Knowledge article in June 2014, Google abanoned it lately and is using Cloud Dataflow because MapReduce didn't work well when the data size reached few petabytes!! ☹

http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/,

References

[O.P.] Dean, Jeffrey, Sanjay Ghemawat. “MapReduce simplified data processing on large clusters." *OSDI 2004*

[G.V.] Cluster Computing and Map Reduce course by Google

https://www.youtube.com/watch?v=-vD6PUdf3Js&list=PLD20C9DE1E63E1617&index=2

http://www.slideshare.net/tugrulh/distributed-computing-seminar-lecture-2-mapreduce-theory-and-implementation?related=1

[J.E.G]examples

http://www.java2s.com/Code/Jar/h/hadoop-mapreduce.htm

BACKUP

Locality

- We conserve network bandwidth by taking advantage of the fact that the input data (managed by GFS) is stored on the local disks of the machines that make up our cluster. GFS divides each file into 64 MB blocks, and stores several copies of each block (typically 3 copies) on different machines.
- Master takes location information of input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data. Failing that, it attempts to schedule map task near a replica of that task's input data (e.g., on worker machine on same network switch as machine containing the data).

Functional Programming

- MapReduce is Inspired from Functional programming languages like LISP
  - Functional programming vs imperative languages
    - Functional:
      - do not modify data, create new ones. Oder of operations doesn't matter, each operation is creating a copy
      - Declarative programming(programming with expressions)

How is it like?

- Most of computations consist of applying a map operation to each logical record. in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately.

Why?

- Large input data
- Computations are often straightforward but need to be distributed
- MapReduce is an abstraction that allows programmers to express the simple computations but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library

Even though the previous pseudo-code is written in terms of string inputs and outputs, conceptually the map and reduce functions supplied by the user have associated types:

map	`(k1,v1)`	$\rightarrow$ `list(k2,v2)`
reduce	`(k2,list(v2))`	$\rightarrow$ `list(v2)`

I.e., the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values.

Our C++ implementation passes strings to and from the user-defined functions and leaves it to the user code to convert between strings and appropriate types.