MapReduce-II

Knowledge objectives

1. Enumerate the different kind of processes in the MapReduce framework
2. Explain the information kept in the master
3. Explain how to decide the number of mappers and reducers
4. Explain the fault tolerance mechanisms in place in case of
   a) Worker failure
   b) Master failure
5. Explain the main problems of the MapReduce framework
6. Explain six potential improvement to the MapReduce framework
7. Name two kinds of generic tasks that can easily fit in a MapReduce framework

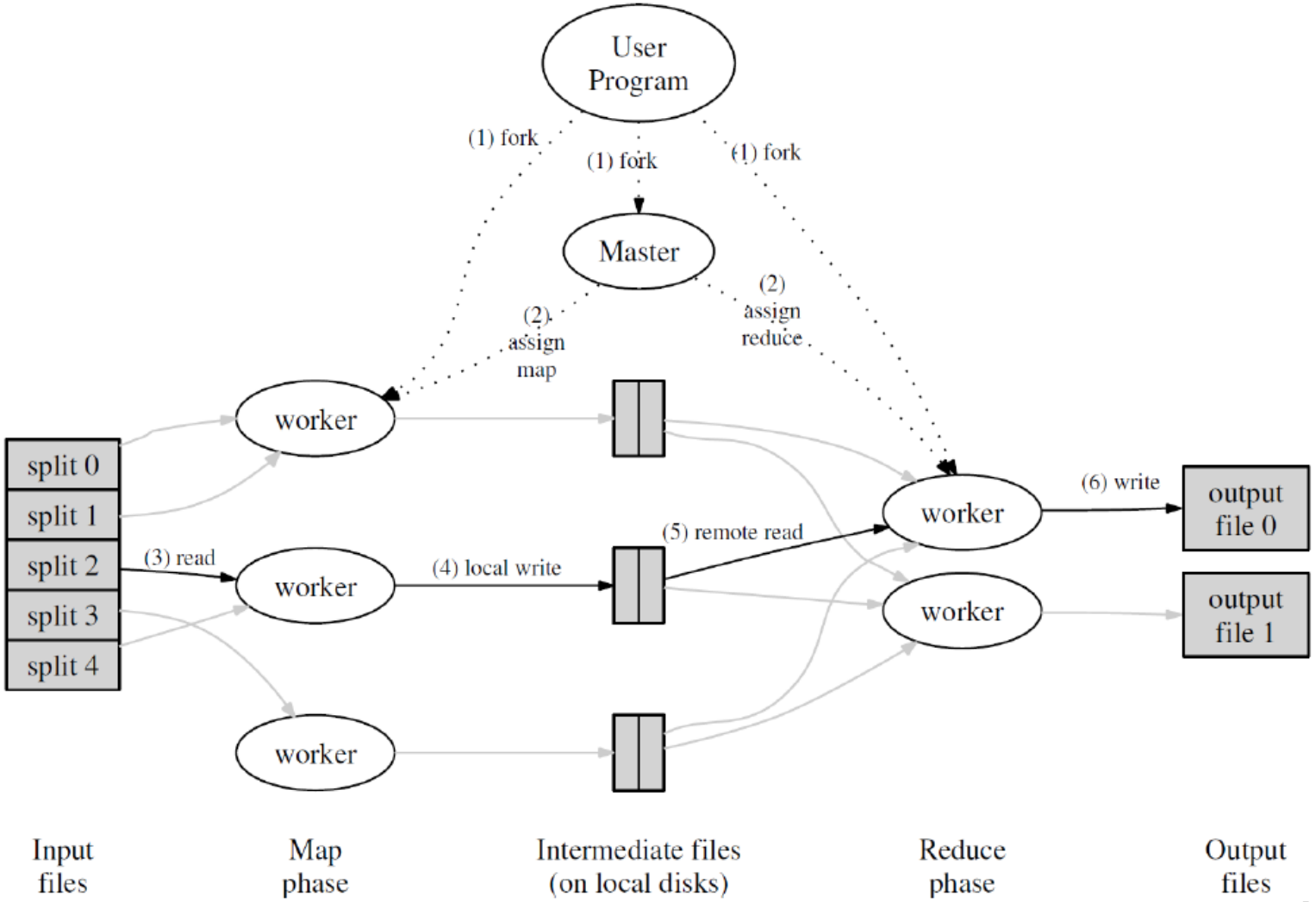
Application Objectives

1. Provide the pseudo-code of map, reduce and combine functions for a simple problem

Activity: MapReduce

- *Objective: Understand the algorithm underneath MapReduce*
- *Tasks:*
  1. *(5') Individually read the problem statement*
  2. *(30') Reproduce step by step the MapReduce execution*
  3. *Hand in the details of the execution*
  4. *(5')* Class sharing of the results

Processes



J. Dean et al.

Architectural decisions

- Users submit jobs to a master scheduling system
  - There is one master and many workers
  - Jobs are decomposed into a set of tasks
  - Tasks are assigned to available workers within the cluster/cloud by the master
    - O(M + R) scheduling decisions
    - Try to benefit from locality
    - As computation comes close to completion, master assigns the same task to multiple workers
- The master keeps all relevant information
  a) Map and Reduce tasks
     - Worker state (i.e., idle, in-progress, or completed)
     - Identity of the worker machine
  b) Intermediate file regions
     - Location and size of each intermediate file region produced by each map task
       - Stores O(M * R) states in memory

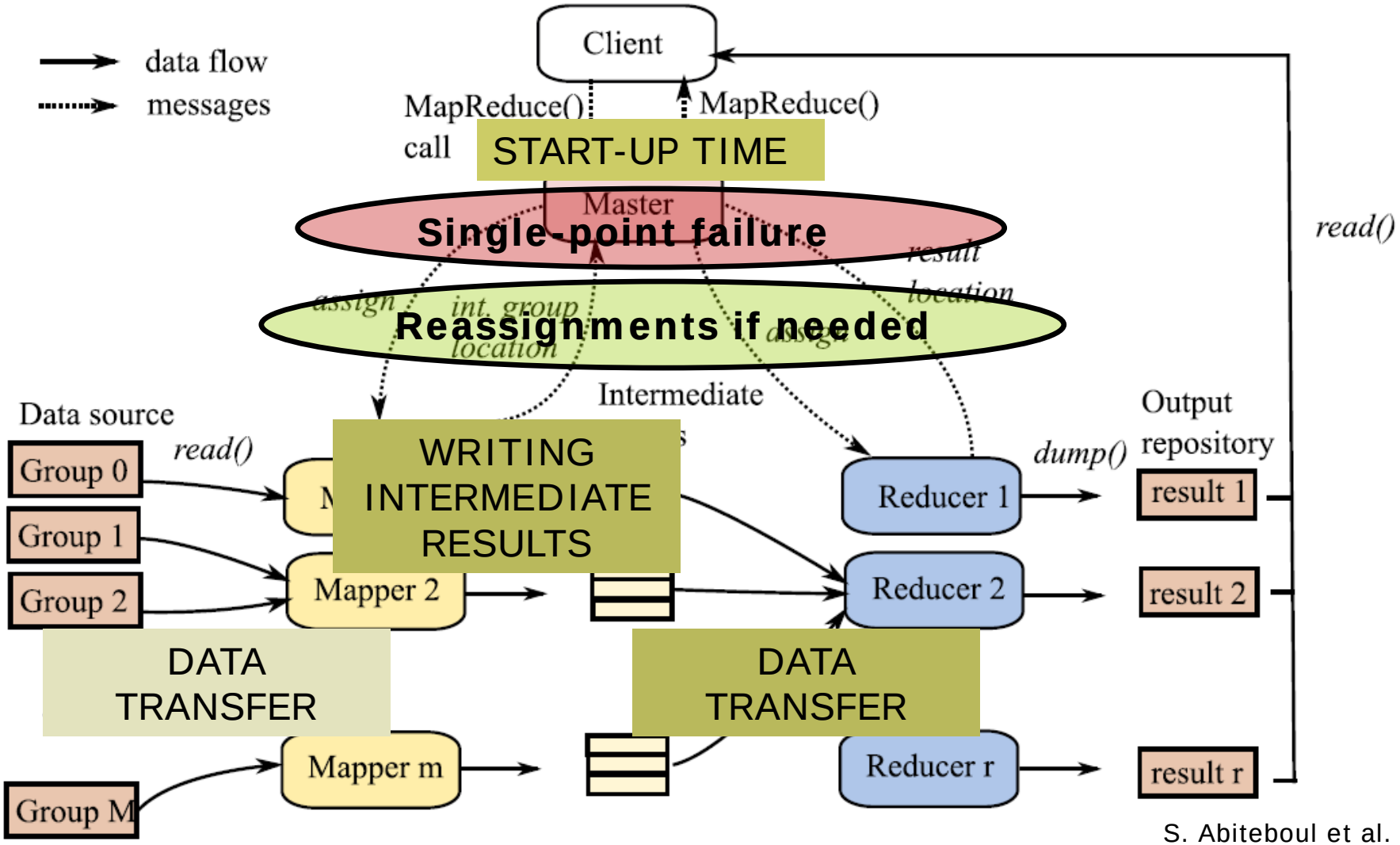
Design decisions

- Number of Mappers
  - Mappers should take at least a minute to execute
  - One per chunk in the input
    - Ideally, to exploit data locality, 10*N < M <100*N
- Number of Reducers
  - Many can produce an explosion of intermediate files
    - For immediate launch: 0.95*N*MaxTasks
    - For load balancing: 1.75*N*MaxTasks

Fault tolerance mechanisms

- Worker failure
  - Master pings workers periodically (heartbeat)
    - Assumes failure if no response
  - Completed/in-progress map and in-progress reduce tasks on failed worker are rescheduled on a different worker node
    - Use chunk replicas
- Master failure
  - Since there is only one, it is less likely it fails
  - Keep checkpoints of data structure

Tasks and Data Flows



S. Abiteboul et al.

Performance Problems

- Defines the execution plan on the fly
  - Schedules one block at a time
    - Which allows to adapt to workload and performance differences between nodes in the cluster
  - Heavy start-up process
- Writes intermediate results to disk
  - Reduce tasks pull intermediate data
    - Which improves fault tolerance
- The Map and Reduce code are black boxes
  - The system knows nothing about what is going on there
- Although query-shipping-oriented, the shuffle phase of the reducer implies massive data transfer
- Does not benefit from compression
  - Without any reason inherent to the model
- In general, it hardly beats a RDBMS until a cluster of >8 nodes is considered
  - Distributed scans compensate the inherent overheads

Improvements

- Direct access to disk
  - The API of the storage system generates overhead
- Implementing a query optimizer
  - The Map & Reducer should define pre- post-conditions (Stratosphere project)
- Index usage
  - No index is used currently
- Record parsing with mutable objects
  - E.g., users should avoid using Java String
- Implement different grouping algorithms
  - Merge-sort is not always the best option
- Avoid fine grain scheduling
  - Blocks may be too small

High Level Languages

- Is the *de facto* standard for robust execution of large data-oriented tasks
- Has been massively criticized for being too low-level
  - APIs for Ruby, Python, Java, C++, etc.
- There are not better solutions
  - Support in HBase, MongoDB, CouchDB, etc.
- Attempts to build declarative languages on top of it
  - Hive
  - Pig Latin
  - Cassandra Query Language (CQL)
    - Resembles SQL

MapReduce at First Sight

- The MapReduce programming paradigm is computationally complete
  - Any program can be adapted to it
    - Not necessarily efficient
- MapReduce's signature is closed
  - MapReduce iterations can be nested
    - Fault tolerance is not guaranteed in between
- Some tasks better adapt to it than others
  - Matrix-vector multiplication
  - Relational algebra

Relational operations: Projection

$$\pi_{a_{i_1},\ldots,a_{i_n}}(T) \Rightarrow \begin{cases} \texttt{map}(\text{key } k, \text{value } v) \mapsto [(\texttt{prj}_{a_{i_1},\ldots,a_{i_n}}(k \oplus v), 1)] \\ \texttt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto [(ik)] \end{cases}$$

Relational operations: Cross Product

$$
T \times S \quad \mapsto \quad
\begin{cases}
\texttt{map}(\text{key } k, \text{value } v) \mapsto \\
\begin{cases}
[(\mathrm{h}_T(k) \bmod D, k \oplus v)] & \text{if input}(k \oplus v) = T, \\
[(0, k \oplus v), .., (D-1, k \oplus v)] & \text{if input}(k \oplus v) = S. \\
\end{cases} \\
\texttt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto \\
\Big[\texttt{crossproduct}(T_{ik}, S) \mid \\
\quad T_{ik} = \{iv \mid iv \in ivs \wedge \text{input}(iv) = T\}, \\
\quad S = \{iv \mid iv \in ivs \wedge \text{input}(iv) = S\}\Big]
\end{cases}
$$

Activity

- *Objective: Practice MapReduce paradigm*
- *Tasks:*
  1. *(10') Individually solve two exercises*
     a) *Selection & Difference*
     b) *Join & Union*
     c) *Aggregation & Intersection*
  2. *(20') Explain the solution to the others*
  3. *Hand in the six solutions*

- *Roles for the team-mates during task 2:*
  a) *Explains his/her material*
  b) *Asks for clarification of blur concepts*
  c) *Mediates and **controls time***

Summary

- Processes in MapReduce
- Fault tolerance mechanisms in MapReduce
- Data transfer in MapReduce
- MapReduce expressivity

Bibliography

- J. Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04
- J. Dittrich et al. *Hadoop++: Making a yellow elefant run like a cheetah (without it even noticing).* Proceedings of VLDB 3(1), 2010
- A. Pavlo et al. *A Comparison of Approaches to Large-Scale Data Analysis*. SIGMOD, 2009
- A. Rajaraman et al. *Mining massive data sets*. Cambridge University Press, 2012

Resources

- http://hadoop.apache.org
- http://hive.apache.org
- http://pig.apache.org
- http://www.cloudera.com
- http://flink.incubator.apache.org
  - Former http://stratosphere.eu
- https://spark.apache.org