

## MapReduce

Sunday, April 19, 2020 5:22 PM

### Map:

- Process individual records to generate intermediate key/value pairs

Ex.

Input: "welcome Everyone Hello everyone"	Key	Value	
welcome	1	1	Map Task 1
Everyone	1	1	
Hello	1	1	Map Task 2
Everyone	1	1	

- very large data sets can be "sharded", i.e. split into multiple map tasks.

### Reduce:

- reduce processes and merges all intermediate values associated per key:

Key	Value	Key	Value
welcome	1	Everyone	2
Everyone	1	Hello	1
Hello	1		
Everyone	1	Welcome	1

- each Key is assigned to one reduce task

- parallelly processes and merges all intermediate values by partitioning keys

Key	Value	Reduce Task 1	Key	Value
welcome	1		Everyone	2
Everyone	1		Hello	1
Hello	1			
Everyone	1		Welcome	1

- Hash partitioning is a popular method for task assignment

↳ Key is assigned to reduce  $\# = \text{hash}(\text{key}) \% \text{number of reduce servers}$

↳ hashing leads to fairly good load balancing in the system.

## Examples

### Distributed Cgrep:

- input: large set of files
- output: lines that match pattern

MAP → emits a line if it matches the supplied pattern

REDUCE → copies the intermediate data to the output

### Reverse Web-Link Graph

- input: Web graph: tuples  $(a, b)$  where page  $a \rightarrow$  page  $b$
- output: for each page, list of pages that link to it

target = Key

MAP → processes web log and for each input & source, target  $\rightarrow$ , it outputs  $\langle \text{target}, \text{source} \rangle$

REDUCE → emits  $\langle \text{target}, \text{list}(\text{source}) \rangle$

### Count of URL access frequency

- input: log of accessed URL's e.g. from a proxy server
- for each URL, % of total accesses for that URL

MAP → processes web log and outputs  $\langle \text{URL}, 1 \rangle$

REDUCE → emits  $\langle \text{URL}, \text{URL\_count} \rangle$

MAP → processes  $\langle \text{URL}, \text{URL\_count} \rangle$  and outputs  $\langle \text{URL}, \langle \text{URL}, \text{URL\_count} \rangle \rangle \rightarrow$

REDUCE → sums up URL\_count to get total count and then emits multiple  $\langle \text{URL}, \text{URL\_count}/\text{overall\_count} \rangle$

## MapReduce Scheduling:

### Programming MapReduce:

Externally: For User:

- 1) write a Map program (short), write a Reduce program (short)
- 2) Submit job, wait for result
- 3) Need to know nothing about distributed/parallel computing

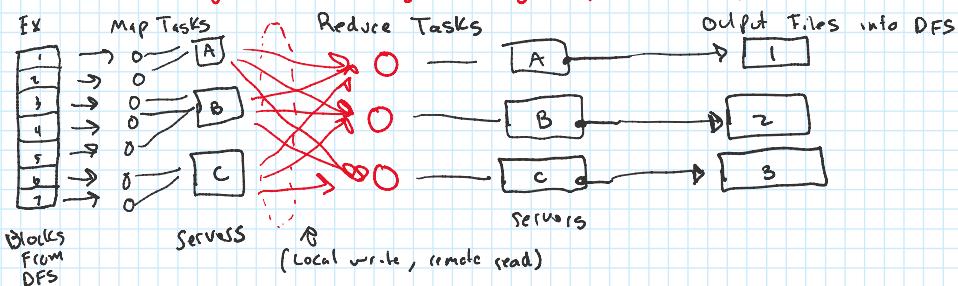
Internally : For the partitioner and scheduler

- 1) Parallelize Map
- 2) Transfer data from Map to Reduce
- 3) Parallelize Reduce
- 4) Implement storage for Map input, Map output, Reduce Input, and Reduce Output
- 5) It also ensures the barrier between the Map phase and the Reduce phase.  
↳ no reduce can begin until all Maps complete (in general)

Inside MapReduce :

- 1) Parallelize Map: easy! each map task is independent of the others.
  - all map output records with same key assigned to same Reduce.
- 2) Transfer data from Map to Reduce:
  - all map output records with same key are assigned to the same reduce task
  - we can use a partitioning function like  $\text{hash}(\text{key}) \% \text{number of reducers}$
- 3) Parallelize Reduce: easy! each reduce task is independent of one another!
- 4) Implement storage for Map input, Map output, Reduce Input, and Reduce Output
  - Map input: from distributed file system
  - Map output: to local disk (at Map Node) - uses local file system
  - Reduce input: from (multiple) remote disks, uses local file systems
  - Reduce output: to distributed file system

• local file systems = Linux FS, etc  
• distributed file systems = GFS (Google File System), HDFS (Hadoop Distributed File System)



\* Resource Manager (assigns maps to reduce servers)

\* Each map task may produce key value pairs for any of the reduce tasks

Yarn Scheduler :

- Used in Hadoop 2.x+
- Yet Another Resource Negotiator
- Treats each server as a collection of containers
  - container = some CPU + some memory
- Has 3 components
  - Global Resource Manager (RM)
  - Scheduling
  - Per-Server Node Manager (NM)
- Per-application (job) Application Master (AM)
  - container negotiation with RM and NMs
  - Detecting task failures of that job

Yarn : How a job gets a container

- the application master of some node tells the resource manager that it needs a container. The task is queued.
- some node who's task is finished tells the resource manager a container is available
- the resource manager informs the requesting node that a container is available at some node X
- the requesting node requests node X to complete its task.

Fault Tolerance :

- most common failure is a server failure
- NM sends "heartbeats" to the RM
  - RM lets all affected AM's know and AM's take action
- NM keeps a track of each task running on its server
  - if task fails while in-progress, mark the task as idle and restart it
- AM heartbeats to RM
  - on failure, RM restarts AM, which then syncs up with its running tasks.

RM Failure

- use old checkpoints and bring up secondary RM
- ... and to meanwhile another renominate 1. .... and another renominate

- RM Failure
  - use old checkpoints and bring up secondary RM
- Heartbeats also used to piggyback container requests to avoid extra messages

### Slow Servers

- slow nodes (stragglers) slow down the entire system.
  - ↳ causes: bad disk, bad network bandwidth, CPU or memory
- this is handled by keeping track of "progress" of each task (% completed)
  - perform back up (replicated) tasks of the straggler task.
  - ↳ task is considered done when the first replica completes

} Speculative (if %o is low, Execution replicate execution)

### Locality

- since the clouds have a hierarchical topology (e.g. racks) ~ ~
- GFS / HDFS store 3 replicas of each chunk (e.g. 64 MB size)
  - ↳ maybe 2 on same rack A, 1 on same rack B
- MapReduce attempts to schedule a map task in the following order:
  - schedule map task on a machine that contains a replica of corresponding input data
  - else
  - schedule map task on the same rack as a machine containing a replica of the input data
  - else
  - anywhere
  - each try may fail because all containers are full

### Summary

- MapReduce uses parallelization + aggregation to schedule applications across clusters
- Needs to deal with failure
- Plenty of ongoing research work in scheduling and fault tolerance for MapReduce and Hadoop.