CS 443
Map Reduce

Winter 2012 **Adapted from Suciu & Balazinska**

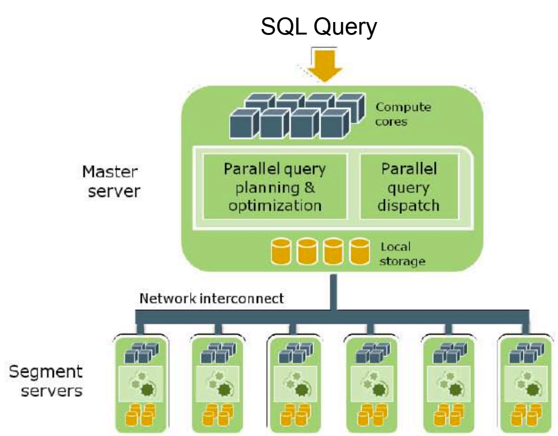
Parallel DBMS

2

- Intra-operator parallelism
  - An operator runs on multiple processors
  - For both OLTP and Decision Support
  - Main parallelism used in Parallel DBMS since 1980's
- Last week we discussed how to use data partitioning to parallelize main database operations like join and group by

Review: Parallel DBMS

Figure 5 - Master server performs global planning and dispatch



From: Greenplum Database Whitepaper

Parallel DBMS

4

- Parallel query plan: tree of parallel operators
  - Data streams from one operator to the next
  - Typically all cluster nodes process all operators
- Can run multiple queries at the same time
  - Queries will share the nodes in the cluster
- Notice that user does not need to know how his/her SQL query was processed

Cluster Computing

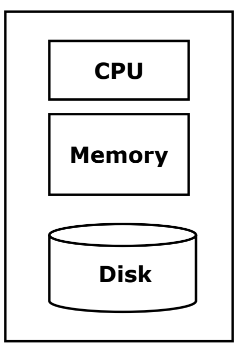
5

- Large number of commodity servers, connected by high speed, commodity network
- Rack: holds a small number of servers
- Data center: holds many racks
- Massive parallelism
  - 100s, or 1000s, or 10000s servers

Commodity Clusters

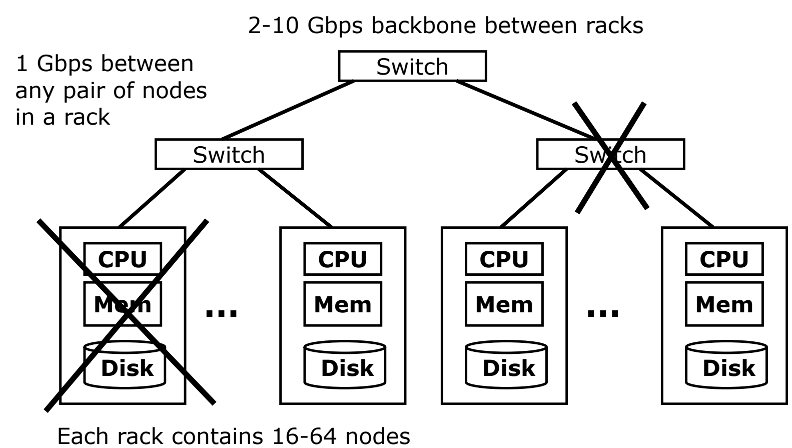
- Web data sets can be very large
  - Tens to hundreds of petabytes
- Cannot analyze on a single server
- Standard architecture
  - Cluster of commodity Linux nodes
  - Gigabit ethernet interconnect
- How to organize computations on this architecture?
  - Shared-nothing Parallel DBMS, right?
  - New performance issue: fault-tolerance
    - Mask issues such as hardware failure

Single-node architecture



Node architecture same as in shared nothing parallel DBMS

Cluster Architecture



Distributed File System

9

- For very large files: TBs, PBs
  - Each file is partitioned into *chunks*, typically 64MB
- Each chunk is replicated several times (≥3), on different racks, for fault tolerance
- Implementations:
  - Google's DFS: GFS, proprietary
  - Hadoop's DFS: HDFS, open source
- Typical usage pattern
  - Data is rarely updated in place
  - Reads and appends are common

Map-Reduce

10

- Google paper published 2004
  - Free variant: Hadoop
- Map-reduce = high-level programming model and implementation for large-scale parallel data processing

Data Model

11

- Based on file processing
- A file = a bag of (key, value) pairs
- A map-reduce program
  - Input: a bag of **(inputkey, value)** pairs
  - Output: a bag of **(outputkey, value)** pairs

Map

12

- User provides the MAP-function:
  - Input: (input key, value)
  - Output: bag of (intermediate key, value)
- System applies the map function in parallel to all (input key, value) pairs in the input file
  - Each mapper takes care one chunk of the file

Reduce

- User provides a REDUCE function:
  - Input: **(intermediate key, bag of values)**
  - Output: bag of output **(values)**
- System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

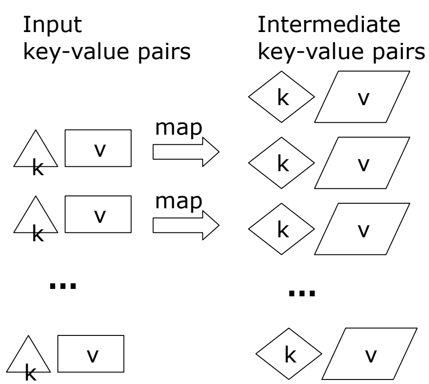
Example: Word Count

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Each Document Doc(did, word)
  - The key = document id (did)
  - The value = list of words (word)

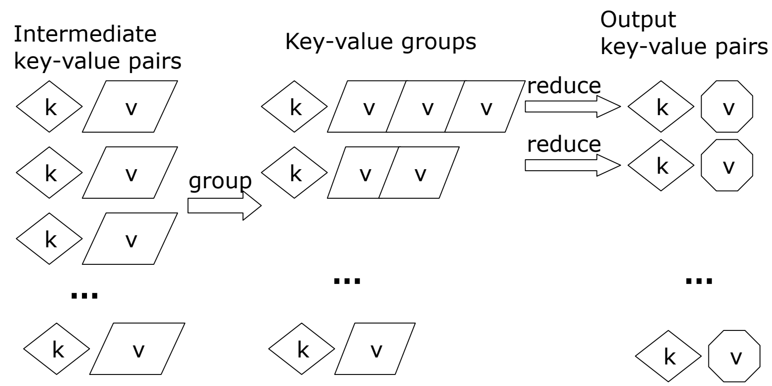
```
map(String key, String value):
  // key: document name
  // value: document contents
for each word w in value:
    EmitIntermediate(w, "1");
```

```
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));
```


MapReduce: The Map Step



MapReduce: The Reduce Step



Example

- Doc 1: the weather is good
- Doc 2: today is good
- Doc 3: good weather is good.

Map output

- Doc 1:
  - (the 1), (weather 1), (is 1), (good 1).
- Doc 2:
  - (today 1), (is 1), (good 1).
- Doc 3:
  - (good 1), (weather 1), (is 1), (good 1).

Reduce Input

- Key 1:
  - (the 1)
- Key 2:
  - (is 1), (is 1), (is 1)
- Key 3:
  - (weather 1), (weather 1)
- Key 4:
  - (today 1)
- Key 5:
  - (good 1), (good 1), (good 1), (good 1)

Reduce Output

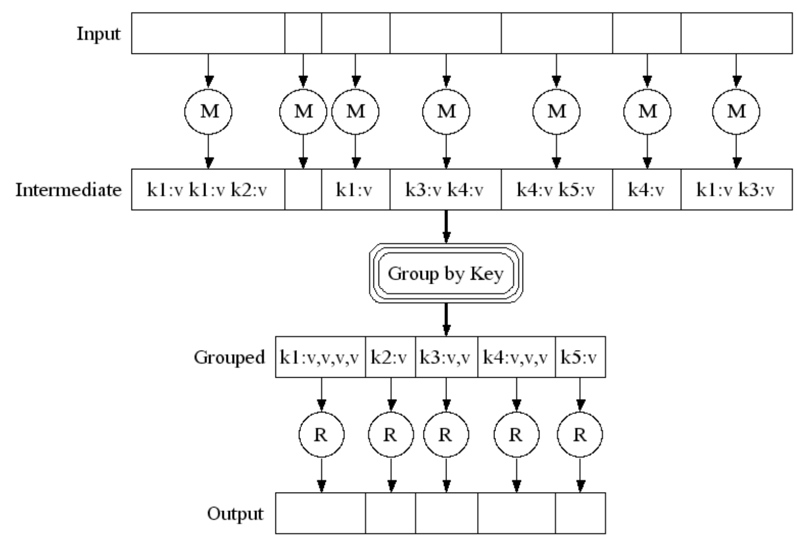
- Key 1:
  - (the 1)
- Key 2:
  - (is 3)
- Key 3:
  - (weather 2)
- Key 4:
  - (today 1)
- Key 5:
  - (good 4)

This example in SQL

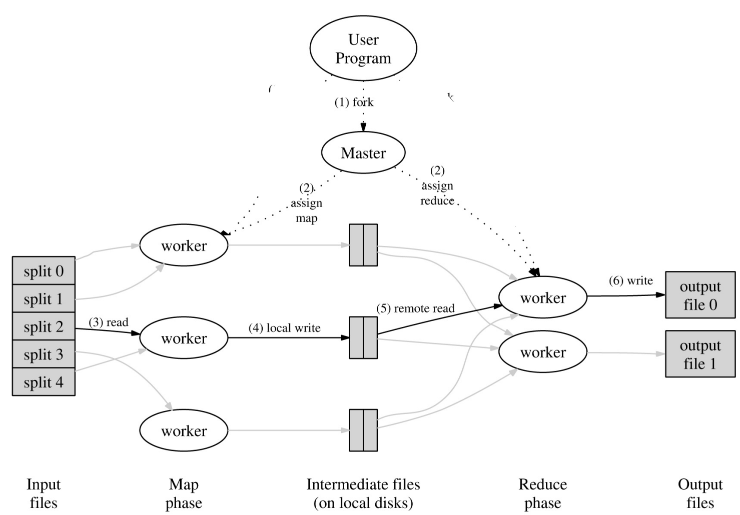
MAP = GROUP BY
REDUCE = Aggregate

Doc(did, word)

SELECT word, sum(1)
FROM Doc
GROUP BY word



Google MR Architecture



Worker

- A worker is a process that executes one task at a time
- Typically there is one worker per processor, hence 4, or 8 per node

File System

25

- All data transfer between workers occurs through distributed file system
  - – Support for split files
  - – Workers perform local writes
  - – Each **map** worker performs local or remote read of one or more input splits
  - – Each **reduce** worker performs remote read of multiple intermediate splits
  - – Output is left in as many splits as reduce workers

Data Partitioning

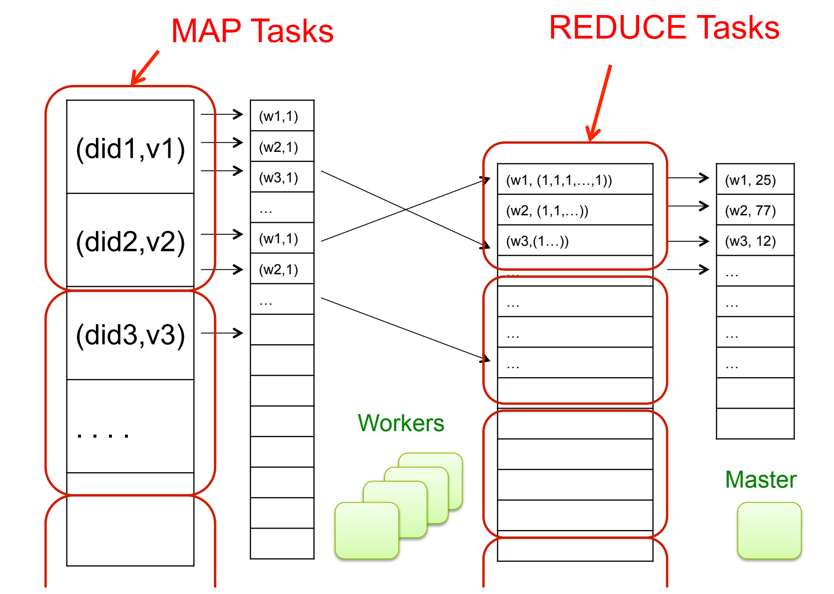
26

- Data partitioned (split) by hash on key
- Each worker responsible for certain hash bucket(s)
- How many workers/splits?
  - Best to have multiple splits per worker
    - Improves load balance
    - If worker fails, splits could be re-distributed across multiple other workers
  - Best to assign splits to "nearby" workers
  - Rules apply to both map and reduce workers

Job vs. Task

27

- A Map-Reduce Job
  - One single "query", e.g., count the words in all docs
  - More complex queries may consists of multiple jobs
- A Map Task or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker



Implementation

29

- There is one master node
- Master partitions input file into M splits, by key
- Master assigns workers (=servers) to the M map tasks, keeps track of their progress
- Workers write their output to local disk, partition into R regions (or intermediate splits)
- Master assigns workers to the R reduce tasks
- Reduce workers read regions from the map workers' local disks

Fault Tolerance

30

- Worker failure
  - Master pings workers periodically
  - If down then reassigns the task to another worker
  - Map/reduce tasks committed through master
- Master failure
  - Not covered in original implementation
  - Could be detected by user program or monitor
  - Could recover persistent state from disk

Performance

31

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks. E.g.:
  - Bad disk forces frequent correctable errors (30MB/s →→ 1MB/s)
  - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown
  - Solution: *pre-emptive backup execution of the* last few remaining in-progress tasks

Other Issues

32

- Handling bad records
  - Best is to debug and fix data/code
  - If master detects at least 2 task failures for a particular input record, skips record during 3rd attempt
  - Is this an issue in RDBMS?
- Debugging
  - Tricky in a distributed environment
  - Done through log messages and counters

Map-Reduce Summary

33

- Hides scheduling, fault recovery, and parallelization details
- Scales well, way beyond thousands of machines and terabytes of data
- Flexibility to handle heterogeneous unstructured data
- General enough for expressing many practical problems

Map-Reduce Summary

34

- One-input two-phase data flow rigid, hard to adapt
  - Does not allow for stateful multiple-step processing of records
  - Difficult to write more complex queries
    - Need multiple map-reduce jobs
- Procedural programming model requires (often repetitive) code for even the simplest operations (e.g., projection, filtering)
- Opaque nature of the map and reduce functions impedes optimization
- Solution: declarative query language!

Declarative languages on MR

35

- Hive (Facebook)
  - SQL + UDFs
- PIG Latin (Yahoo!)
  - New language, like Relational Algebra
- HadoopDB
  - MR + DB
- SQL /Tenzing (Google)
  - SQL on MR
  - Proprietary

Hive

36

- Is built on top of Hadoop (MapReduce + HDFS).
- Supports a declarative language (HiveQL) that is compatible with most SQL.
- Supports a combination of the declarative language and user defined functions.
- Compiles a query into a set of MapReduce jobs and executes these jobs in the MapReduce framework.

Pig Latin

- Is also built on top of Hadoop (MapReduce + HDFS).
- Is a procedure language that fits between the declarative style of SQL and the low-level MapReduce.
- Good
  - More straightforward for people who are not comfortable with SQL
  - Less coding compared to MapReduce.
- Bad
  - More coding compared to SQL.

HadoopDB

- Is a hybrid system of MapReduce and RDBMS.
- Replaces the storage layer by RDBMS.
  - Good
    - Many computations can be pushed to the database layer.
    - It can utilize nice database properties (e.g. indices)
  - Bad
    - Loading data is an overhead.
    - Some operations have to be done through MapReduce.

Parallel DBMS vs MR

- ParallelDBMS - **faster**
  - Indexing
  - Physical tuning
  - Can stream data from one op. to the next without blocking
- MapReduce - **fault-tolerant**
  - Can easily add nodes to the cluster (no need to even restart)
  - Uses less memory since processes one key-group at a time
  - Intra-query fault-tolerance thanks to results on disk
  - Handles adverse conditions: e.g., stragglers
  - Arguably **more scalable**... but also needs more nodes!