CS 343
Map Reduce
Diane Horton and Renee Miller

Adapted from Suciu & Balazinska

Parallel DBMS

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

Parallel DBMS

- Parallel query plan: tree of parallel operators
  - Data streams from one operator to the next
  - Typically ***all cluster nodes process all operators***
    - ***but only on a subset (partition) of data***
- 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

- 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

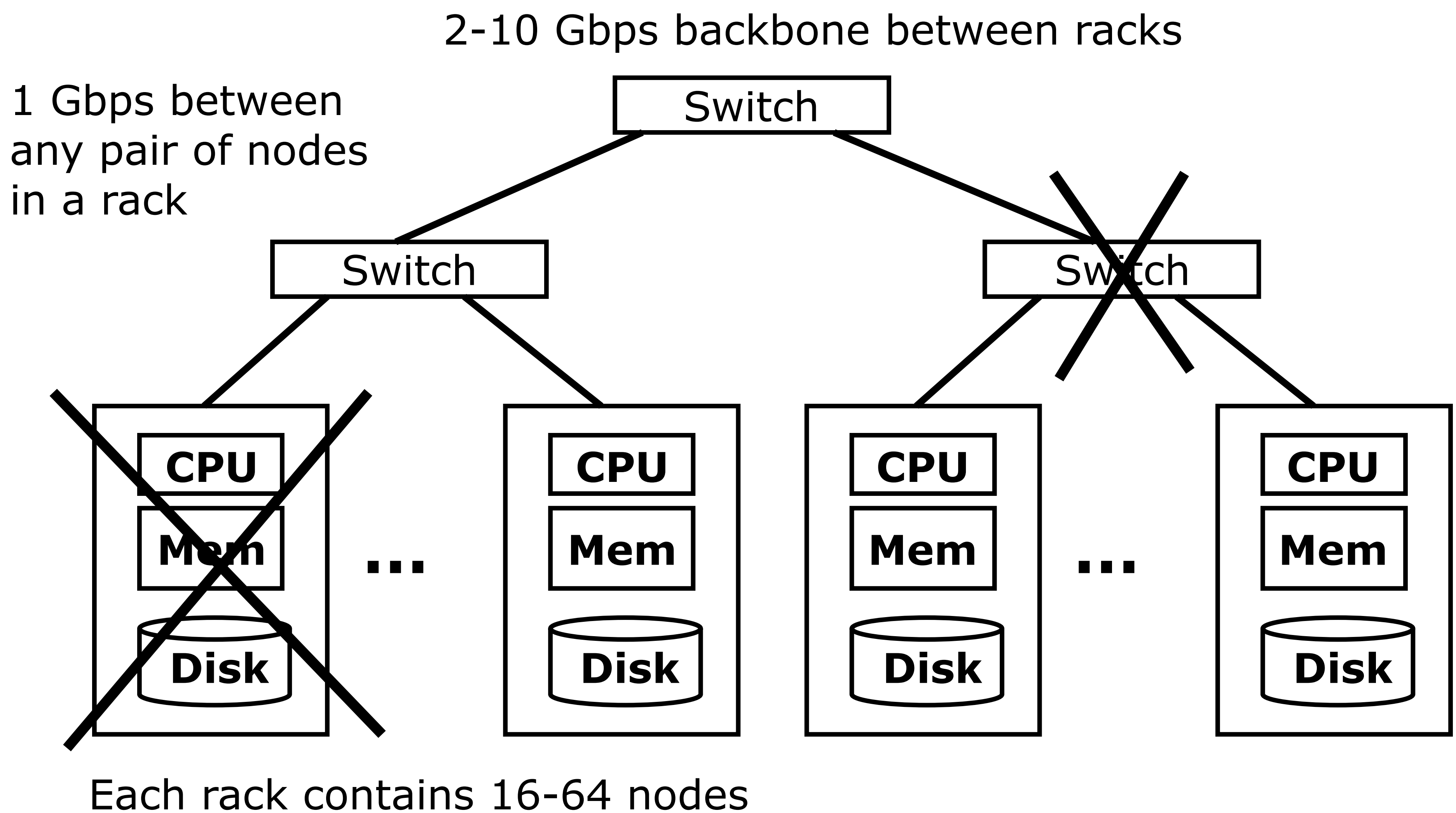
UofT:DB Group

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

UofT:DB Group

Cluster Architecture



Node architecture same as in shared nothing parallel DBMS

Distributed File System

- For very large files: TBs, PBs
  - Each file is partitioned into *chunks*, typically 64MB
- Each chunk is replicated several times ($\geq$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

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

Data Model

- 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

- 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

```
SELECT word, count(*)
FROM Doc
GROUP BY word
```

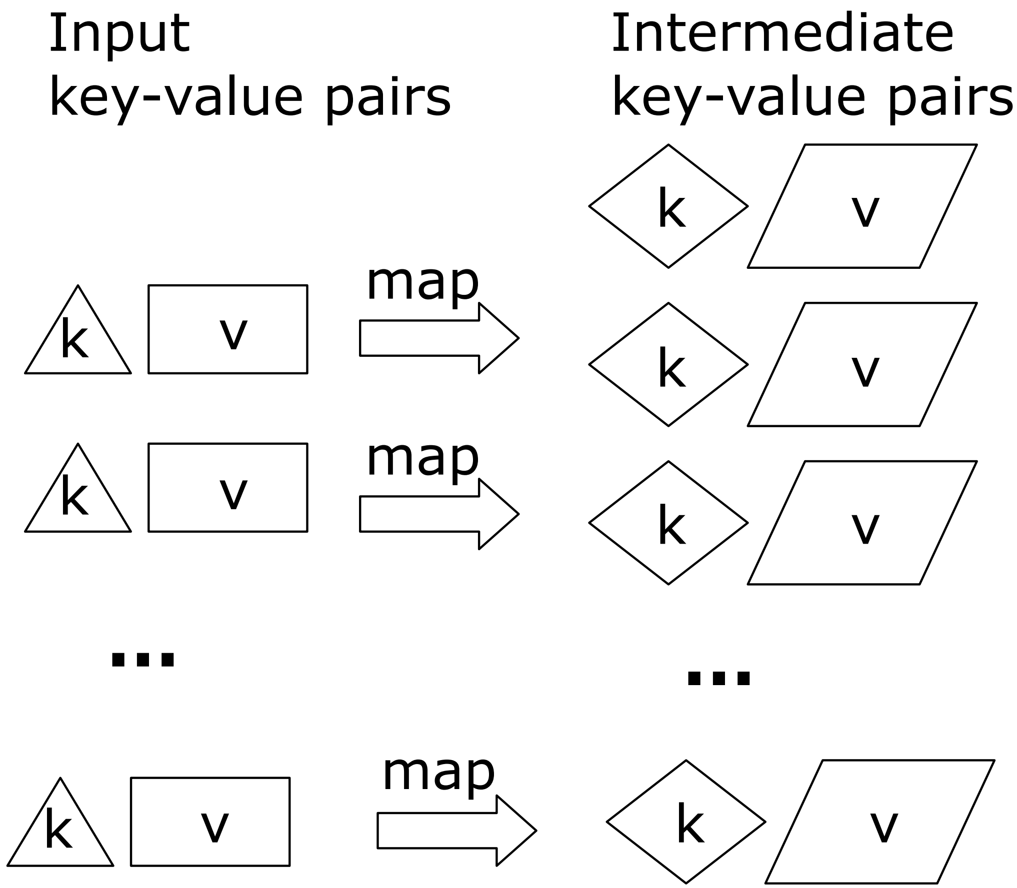

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Each Document Doc(did, word)
  - The key = document id (did)
  - The value = list of words (word)

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

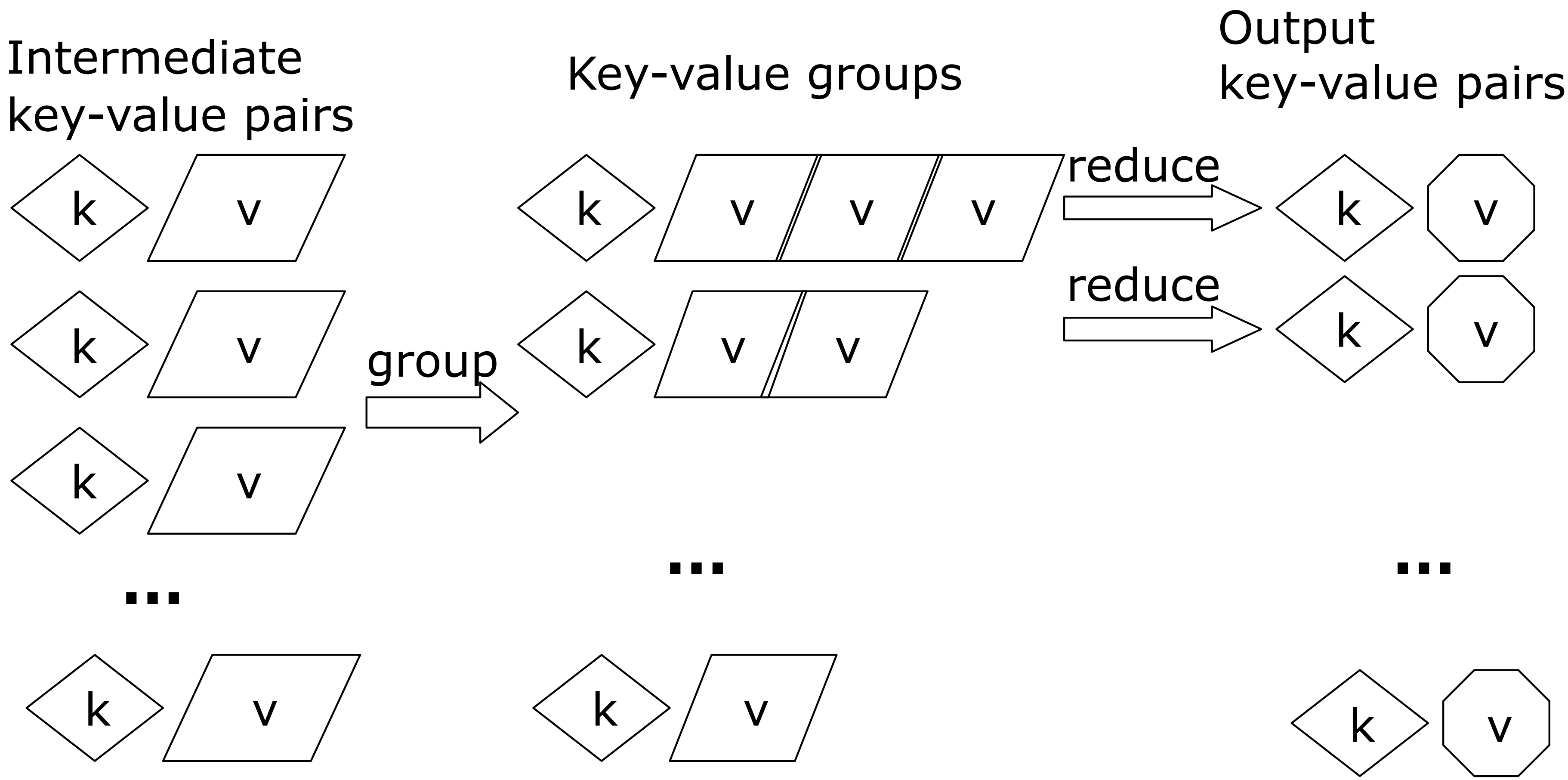
```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
int result = 0;
for each v in values:
  result += ParseInt(v);
Emit(result);
```

UofT:DB Group

MapReduce: The Map Step



MapReduce: The Reduce Step



File System

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

Data Partitioning

- 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

Implementation

- 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

- 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

- *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

UofT:DB Group

Map-Reduce Summary

- 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

UofT:DB Group

Map-Reduce Summary

- One-input two-phase data flow rigid, hard to adapt
  - No 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!
  - Have been (are being) added

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!

UofT:DB Group