Lecture 22: Parallel Databases

Wednesday, May 26, 2010

Overview

- Parallel architectures and operators: Ch. 20.1
- Map-reduce: Ch. 20.2
- Semijoin reductions, full reducers: Ch. 20.4
  - We covered this a few lectures ago

Parallel v.s. Distributed Databases

- Parallel database system:
  - Improve performance through parallel implementation

- Distributed database system:
  - Data is stored across several sites, each site managed by a DBMS capable of running independently

Parallel DBMSs

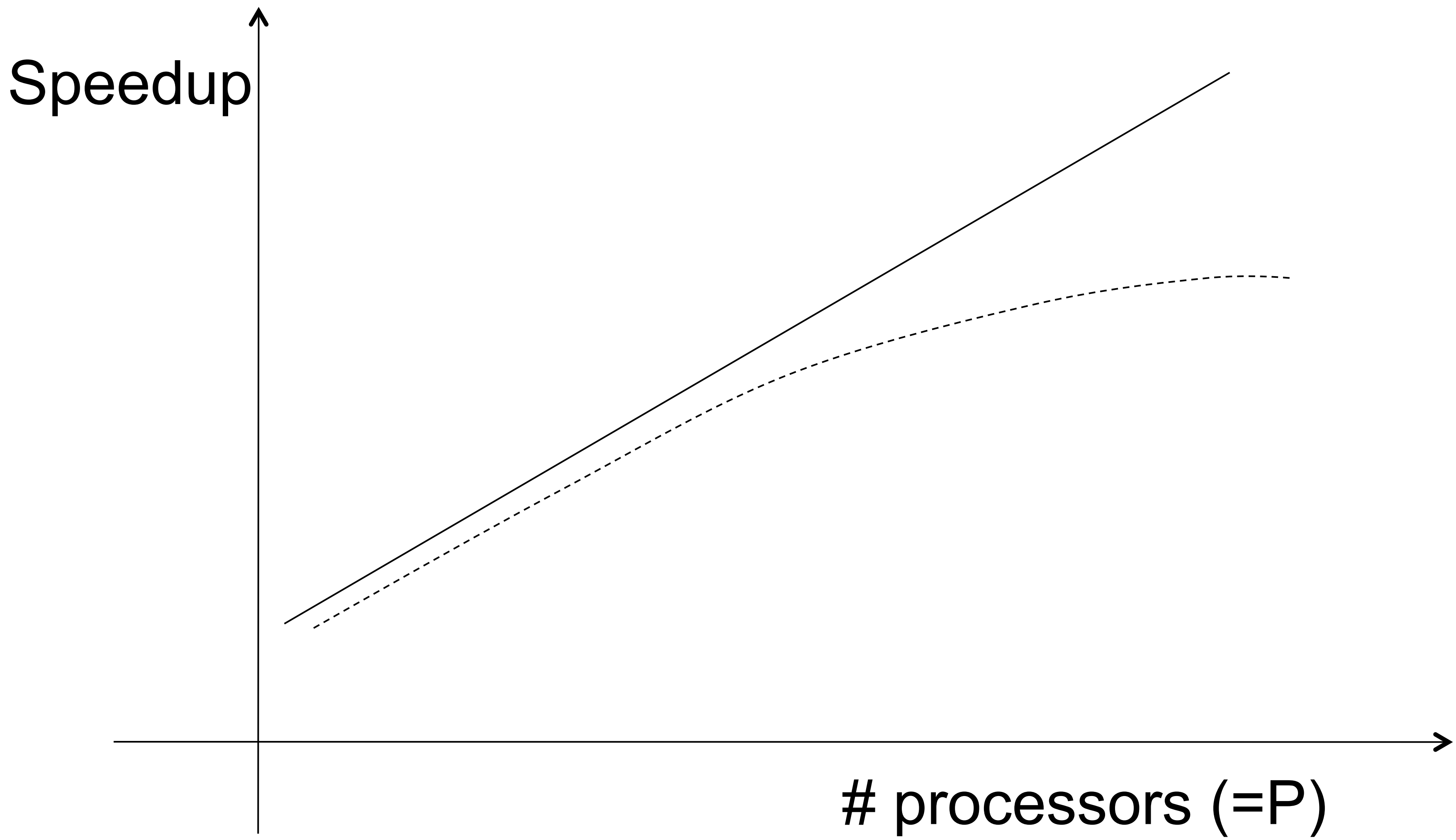
- Goal
  - Improve performance by executing multiple operations in parallel
- Key benefit
  - Cheaper to scale than relying on a single increasingly more powerful processor
- Key challenge
  - Ensure overhead and contention do not kill performance

Performance Metrics for Parallel DBMSs

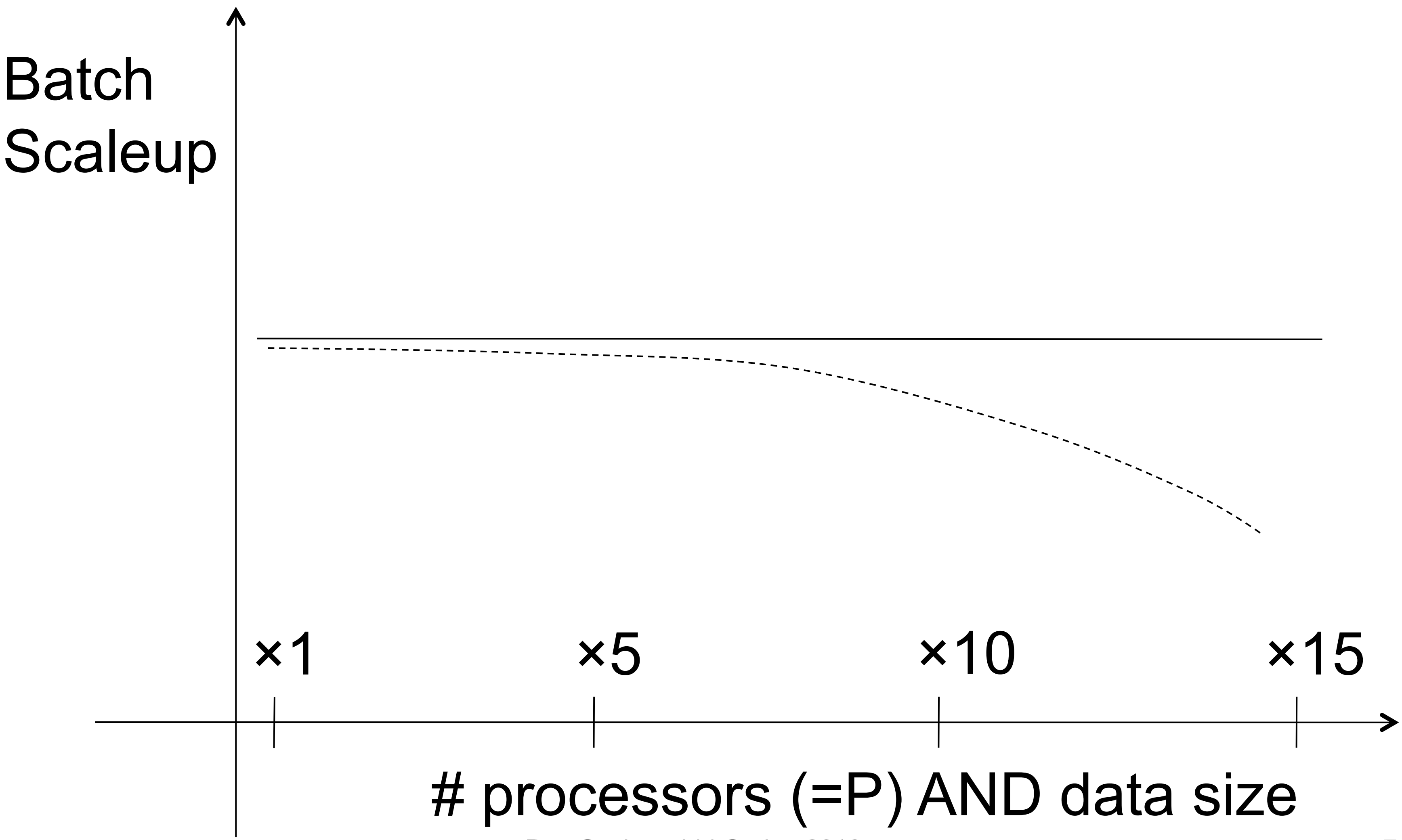
- Speedup
  - More processors → higher speed
  - Individual queries should run faster
  - Should do more transactions per second (TPS)

- Scaleup
  - More processors → can process more data
  - Batch scaleup
    - Same query on larger input data should take the same time
  - Transaction scaleup
    - N-times as many TPS on N-times larger database
    - But each transaction typically remains small

Linear v.s. Non-linear Speedup



Linear v.s. Non-linear Scaleup



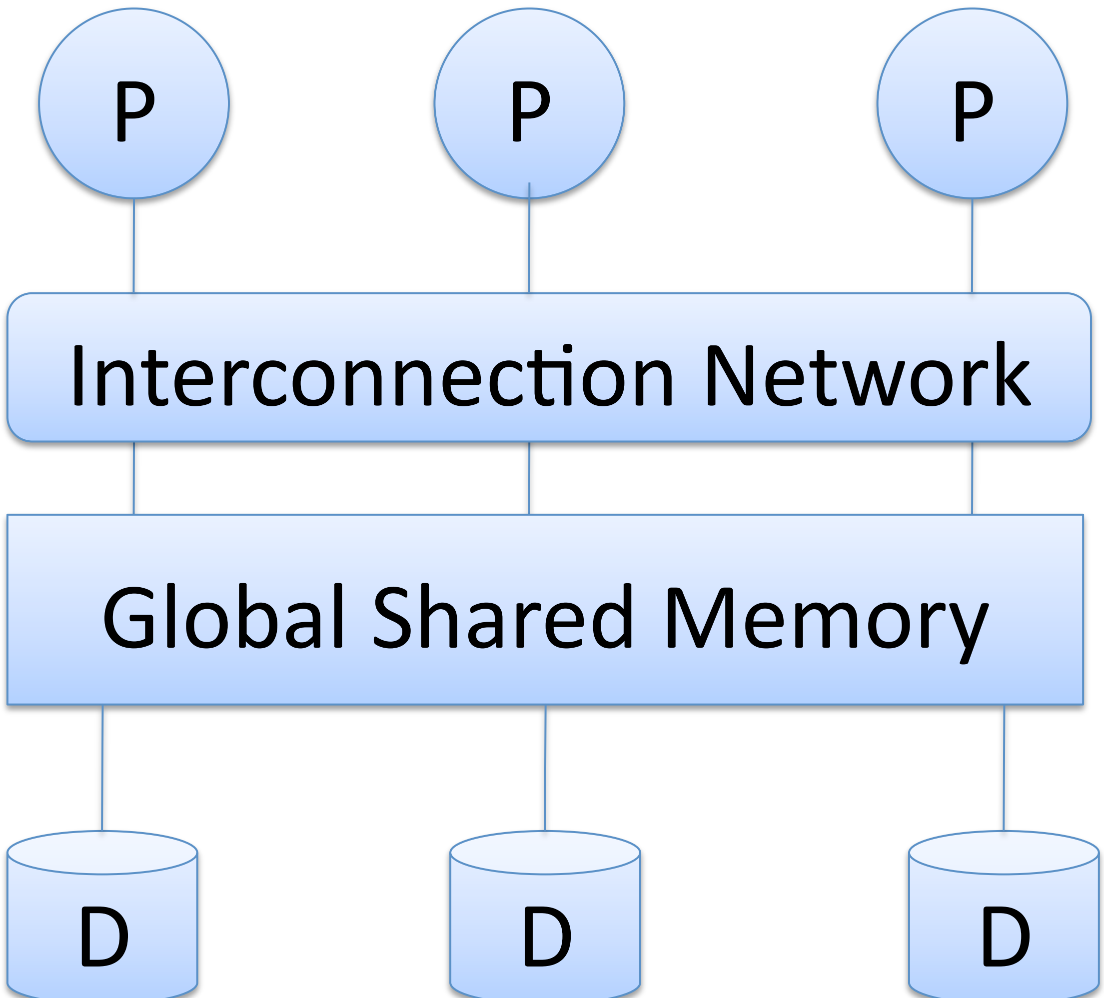
Challenges to Linear Speedup and Scaleup

- Startup cost
  - Cost of starting an operation on many processors
- Interference
  - Contention for resources between processors
- Skew
  - Slowest processor becomes the bottleneck

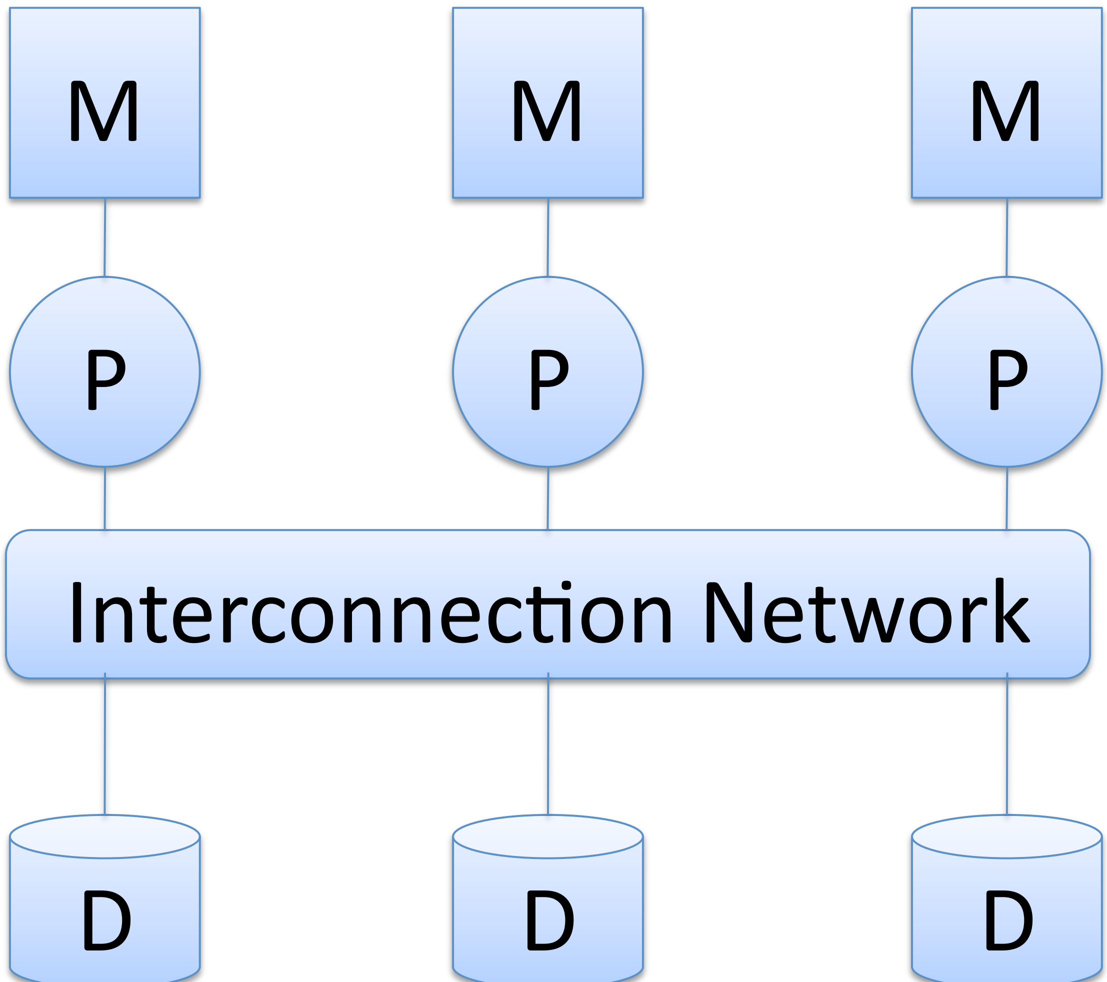
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

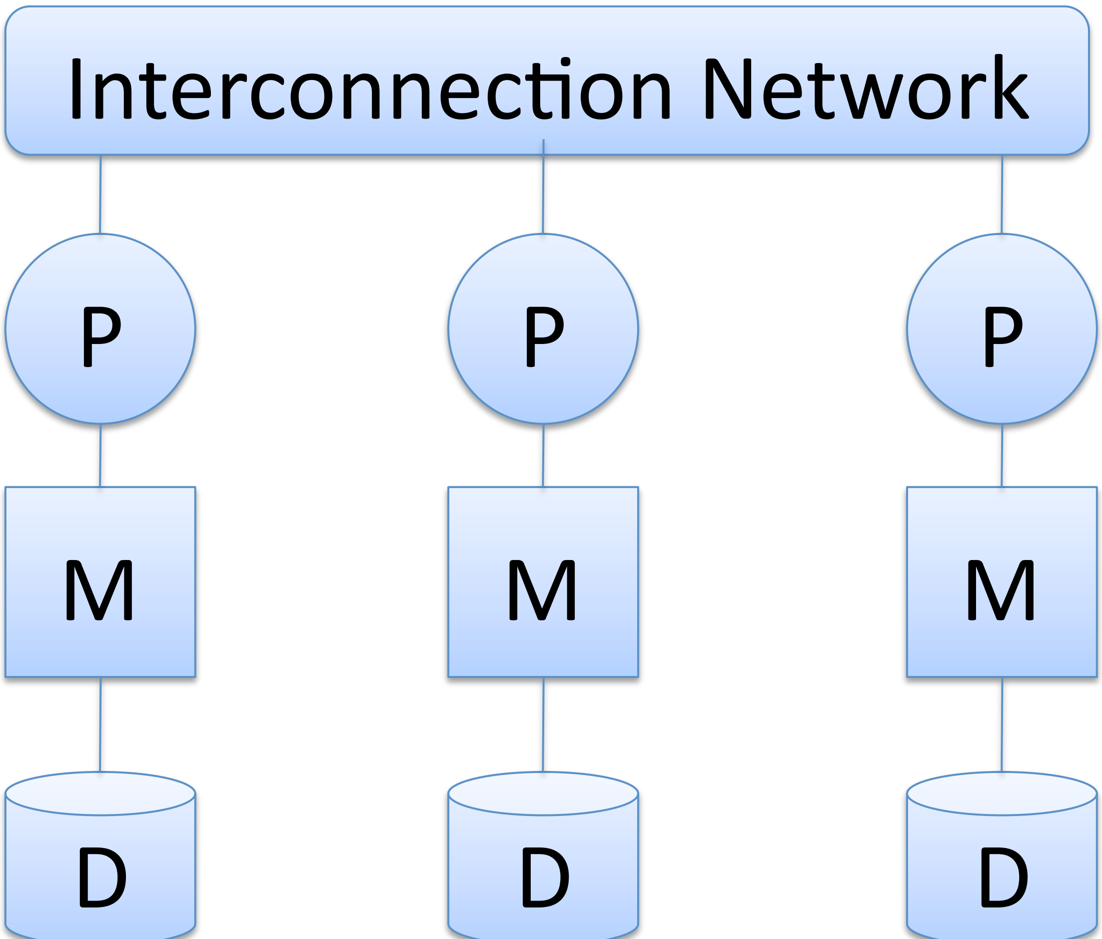
Shared Memory



Shared Disk



Shared Nothing



Shared Nothing

- Most scalable architecture
  - Minimizes interference by minimizing resource sharing
  - Can use commodity hardware
- Also most difficult to program and manage
- Processor = server = node
- P = number of nodes

We will focus on shared nothing

Question

- What exactly can we parallelize in a parallel DB ?

Taxonomy for Parallel Query Evaluation

- Inter-query parallelism
  - Each query runs on one processor
- Inter-operator parallelism
  - A query runs on multiple processors
  - An operator runs on one processor
- Intra-operator parallelism
  - An operator runs on multiple processors

We study only intra-operator parallelism: most scalable

Horizontal Data Partitioning

- Relation R split into P chunks $R_0, \ldots, R_{P-1}$, stored at the P nodes
- Round robin: tuple t_i to chunk (i mod P)
- Hash based partitioning on attribute A:
  - Tuple t to chunk h(t.A) mod P
- Range based partitioning on attribute A:
  - Tuple t to chunk i if $v_{i-1} < t.A < v_i$

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1<A<v2}(R)$

- On a conventional database: cost = B(R)
- Q: What is the cost on a parallel database with P processors ?
  - Round robin
  - Hash partitioned
  - Range partitioned

Parallel Selection

- Q: What is the cost on a parallel database with P processors ?

- A: B(R) / P in all cases

- However, different processors do the work:
  - Round robin: all servers do the work
  - Hash: one server for $\sigma_{A=v}(R)$, all for $\sigma_{v1<A<v2}(R)$
  - Range: one server only

Data Partitioning Revisited

What are the pros and cons ?

- Round robin
  - Good load balance but always needs to read all the data
- Hash based partitioning
  - Good load balance but works only for equality predicates and full scans
- Range based partitioning
  - Works well for range predicates but can suffer from data skew

Parallel Group By: $\gamma_{A, sum(B)}(R)$

- Step 1: server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$
- Step 2: server i sends partition R_{ij} to serve j
- Step 3: server j computes $\gamma_{A, sum(B)}$ on R_{0j}, R_{1j}, …, $R_{P-1,j}$

Cost of Parallel Group By

Recall conventional cost = 3B(R)

- Cost of Step 1: B(R)/P I/O operations
- Cost of Step 2: (P-1)/P B(R) blocks are sent
  - Network costs assumed to be much lower than I/O
- Cost of Step 3: 2 B(R)/P
  - Why ?
  - When can we reduce it to 0 ?

Total = 3B(R) / P + communication costs

Parallel Join: $R \bowtie_{A=B} S$

- Step 1
  - For all servers in [0,k], server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$
  - For all servers in [k+1,P], server j partitions chunk S_j using a hash function h(t.A) mod P: S_{j0}, S_{j1}, …, $R_{j,P-1}$
- Step 2:
  - Server i sends partition R_{iu} to server u
  - Server j sends partition S_{ju} to server u
- Steps 3: Server u computes the join of R_{iu} with S_{ju}

Cost of Parallel Join

- Step 1: (B(R) + B(S))/P
- Step 2: 0
  - (P-1)/P (B(R) + B(S)) blocks are sent, but we assume network costs to be << disk I/O costs
- Step 3:
  - 0 if smaller table fits in main memory: B(S)/p <=M
  - 2(B(R)+B(S))/P otherwise

Parallel Dataflow Implementation

- Use relational operators unchanged
- Add special split and merge operators
  - Handle data routing, buffering, and flow control
- Example: exchange operator
  - Inserted between consecutive operators in the query plan
  - Can act as either a producer or consumer
  - Producer pulls data from operator and sends to n consumers
    - Producer acts as driver for operators below it in query plan
  - Consumer buffers input data from n producers and makes it available to operator through getNext interface

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

- Files !
- A file = a bag of (key, value) pairs
- A map-reduce program:
  - Input: a bag of (input key, value) pairs
  - Output: a bag of (output key, value) pairs

Step 1: the MAP Phase

- User provides the MAP-function:
  - Input: one (input key, value)
  - Ouput: a bag of (intermediate key, value) pairs
- System applies the map function in parallel to all (input key, value) pairs in the input file

Step 2: the REDUCE Phase

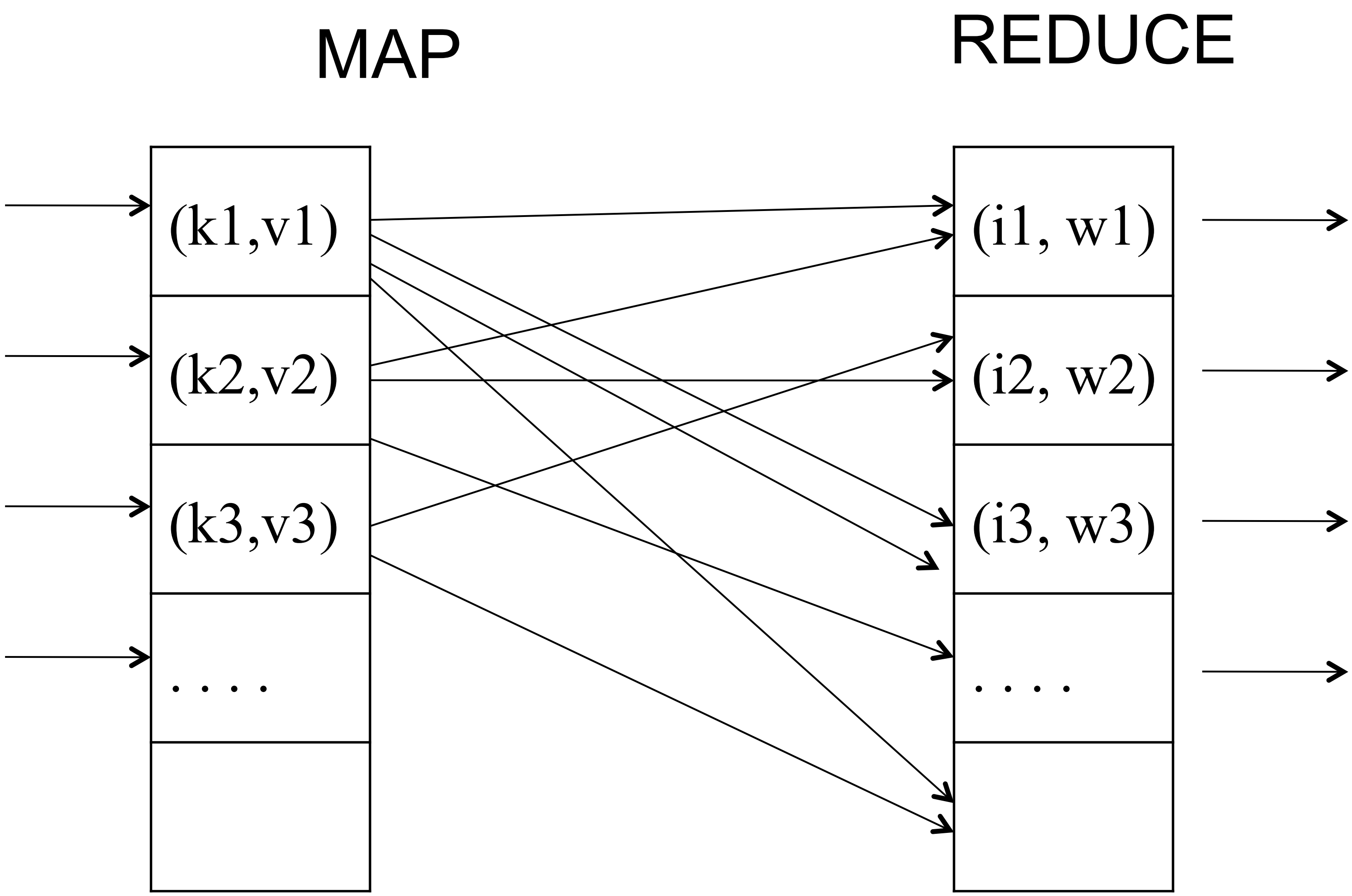
- User provides the REDUCE function:
  - Input: intermediate key, and bag of values
  - Output: bag of output values
- System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- Counting the number of occurrences of each word in a large collection of documents

```
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));
```


MAP
REDUCE
(k1,v1)
(k2,v2)
(k3,v3)
. . . .
(i1, w1)
(i2, w2)
(i3, w3)
. . . .

Map = GROUP BY, Reduce = Aggregate

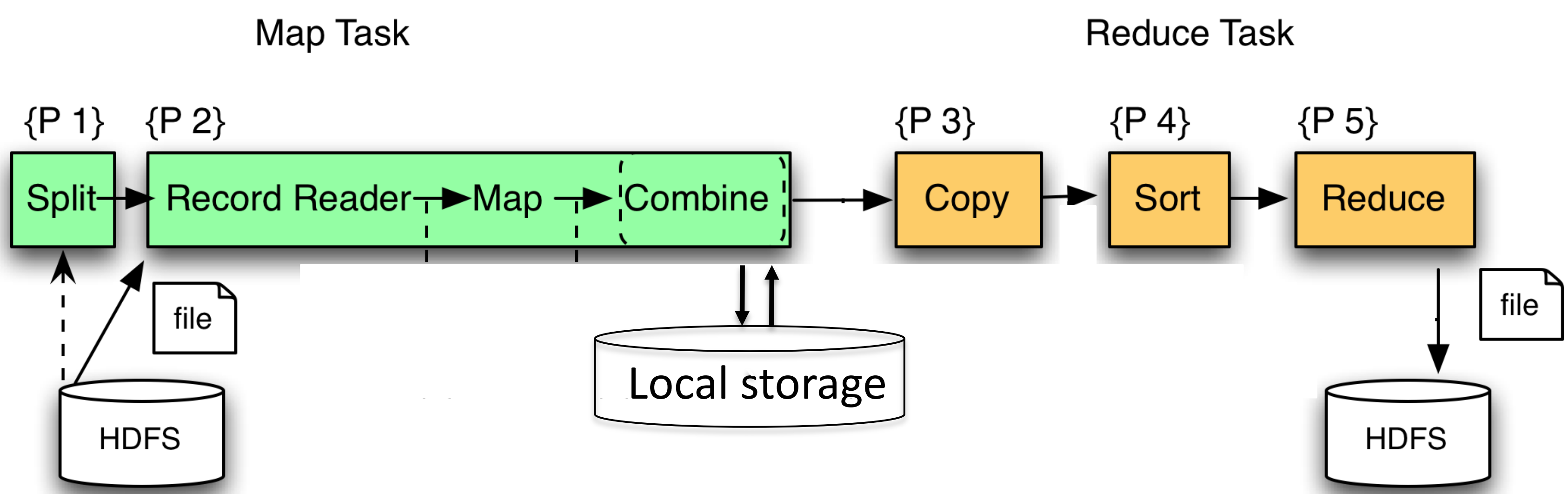
R(documentKey, word)

```
SELECT word, sum(1)
FROM R
GROUP BY word
```

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*
- Master assigns workers to the *R reduce tasks*
- Reduce workers read regions from the map workers' local disks

MR Phases



Map Task
Reduce Task
{P 1}
{P 2}
{P 3}
{P 4}
{P 5}
Split
Record Reader
Map
Combine
Copy
Sort
Reduce
file
HDFS
Local storage
file
HDFS

Interesting Implementation Details

- Worker failure:
  - Master pings workers periodically,
  - If down then reassigns its splits *to all other* workers → good load balance
- Choice of M and R:
  - Larger is better for load balancing
  - Limitation: master needs $O(M \times R)$ memory

Interesting Implementation Details

Backup tasks:

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks. Eg:
  - 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*

Map-Reduce Summary

- Hides scheduling and parallelization details
- However, very limited queries
  - Difficult to write more complex tasks
  - Need multiple map-reduce operations
- Solution:

PIG-Latin !

- Others:
  - Scope (MS): SQL ! But proprietary…
  - DryadLINQ (MS): LINQ ! But also proprietary…