

**Parallel Data Processing<sup>†</sup>**

CompSci 316

Introduction to Database Systems

<sup>†</sup>Most contents are drawn and adapted from slides by  
Madga Balazinska at U. Washington

---

---

---

---

---

---

---

---

2

**Announcements (Tue. Dec. 4)**

- ❖ Homework #4 due today!
- ❖ Project demo slots finalized—see email
- ❖ Final exam 2-5pm Dec. 12
  - Open book, open notes; focus on the second half
  - Sample solution to 2011 final emailed
- ❖ Course evaluation is online this year for us
  - Please complete by this Thursday
  - See my email for instructions
  - Notify me if there is any glitch—this is a pilot run!

---

---

---

---

---

---

---

---

3

**Parallel processing**

- ❖ Improve performance by executing multiple operations in parallel
- ❖ Cheaper to scale than relying on a single increasingly more powerful processor
- ❖ Performance metrics
  - Speedup, in terms of completion time
  - Scaleup, in terms of time per unit problem size
  - Cost: completion time  $\times$  # processors  $\times$  (cost per processor per unit time)

---

---

---

---

---

---

---

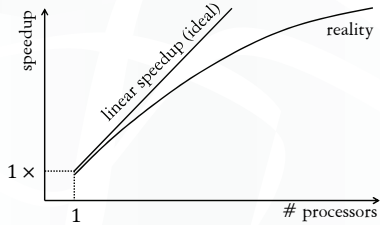
---

## Speedup

4

❖ Increase # processors → how much faster can we solve the same problem?

- Overall problem size is fixed



---

---

---

---

---

---

---

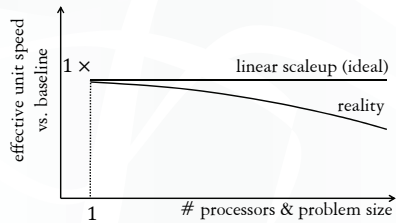
---

## Scaleup

5

❖ Increase # processors and problem size proportionally → can we solve bigger problems in the same time?

- Per-processor problem size is fixed



---

---

---

---

---

---

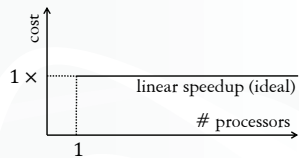
---

---

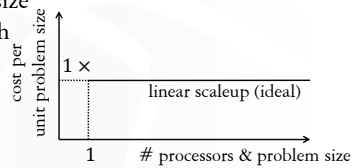
## Cost

6

❖ Fix problem size



❖ Increase problem size proportionally with # processors



---

---

---

---

---

---

---

---

## Challenges to linear speedup/scaleup <sup>7</sup>

- ❖ Startup
  - Overhead of starting useful work on many processors
- ❖ Communication
  - Cost of exchanging data/information among processors
- ❖ Interference
  - Contention for resources among processors
- ❖ Skew
  - Slowest processor becomes the bottleneck

---

---

---

---

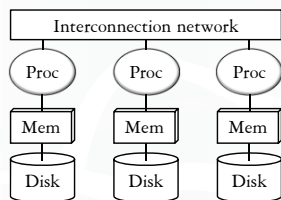
---

---

---

---

## Shared-nothing architecture <sup>8</sup>



- ❖ Most scalable (vs. shared-memory and shared-disk)
  - Minimizes interference by minimizing resource sharing
  - Can use commodity hardware
- ❖ Also most difficult to program

---

---

---

---

---

---

---

---

## Parallel query evaluation opportunities <sup>9</sup>

- ❖ Inter-query parallelism
  - Each query can run on a different processor
- ❖ Inter-operator parallelism
  - A query runs on multiple processors
  - Each operator can run on a different processor
- ❖ Intra-operator parallelism
  - An operator can run on multiple processors, each working on a different “split” of data/operation

---

---

---

---

---

---

---

---

## A brief tour of two systems

10

- ❖ Parallel DBMS (e.g., Teradata)
  - Provides the same abstractions (e.g., relational data model, SQL, transactions) as a regular DBMS
  - Parallelization handled behind the scene
- ❖ MapReduce (e.g., Hadoop)
  - Supports easy scaling out (e.g., adding lots of commodity servers) and failure handling
  - Does not require loading data into tables
  - Exposes parallelism to programmers
    - Other tools built on top of MapReduce can provide higher-level abstractions

---

---

---

---

---

---

---

---

## Horizontal data partitioning

11

- ❖ Split a table  $R$  into  $p$  chunks, each stored at one of the  $p$  processors
- ❖ Splitting strategies:
  - Round robin assigns the  $i$ -th row assigned to chunk  $(i \bmod p)$
  - Hash-based partitioning on attribute  $A$  assigns row  $r$  to chunk  $(h(t.A) \bmod p)$
  - Range-based partitioning on attribute  $A$  partitioning the range of  $R.A$  values into  $p$  ranges, and assigns row  $r$  to the chunk whose corresponding range contains  $r.A$

---

---

---

---

---

---

---

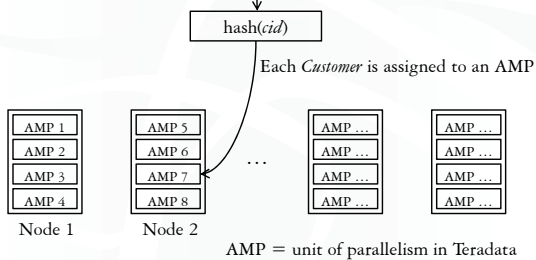
---

## Teradata: an example parallel DBMS

12

- ❖ Hash-based partitioning of *Customer* on *cid*

A *Customer* row is inserted



---

---

---

---

---

---

---

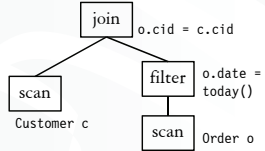
---

## Example query in Teradata

13

❖ Find all orders today, along with the customer info

```
SELECT *
FROM Order o, Customer c
WHERE o.cid = c.cid
AND o.date = today();
```




---

---

---

---

---

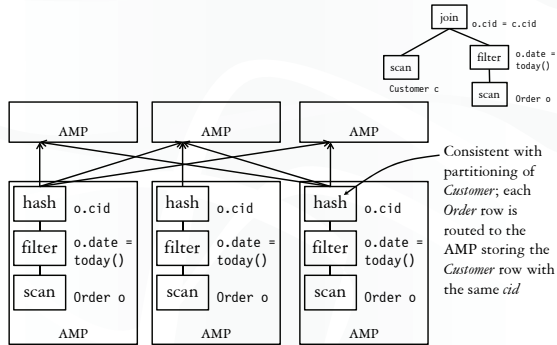
---

---

---

## Teradata example: scan-filter-hash

14




---

---

---

---

---

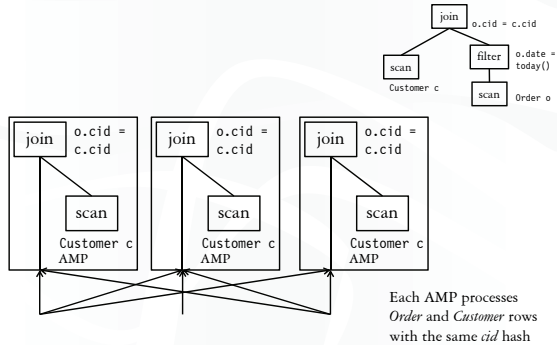
---

---

---

## Teradata example: hash join

15




---

---

---

---

---

---

---

---

## MapReduce: motivation

16

- ❖ Many problems can be processed in this pattern:
  - Given a lot of unsorted data
  - Map: extract something of interest from each record
  - Shuffle: group the intermediate results in some way
  - Reduce: further process (e.g., aggregate, summarize, analyze, transform) each group and write final results(Customize map and reduce for problem at hand)
- ☞ Make this pattern easy to program and efficient to run
  - Original Google paper in OSDI 2004
  - Yahoo!'s Hadoop is the most popular open-source implementation

---

---

---

---

---

---

---

---

## M/R programming model

17

- ❖ Input/output: each a collection of key/value pairs
- ❖ Programmer specifies two functions
  - $\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$ 
    - Processes each input key/value pair, and produces a list of intermediate key/value pairs
  - $\text{reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$ 
    - Processes all intermediate values associated with the same key, and produces a list of result values (usually just one for the key)

---

---

---

---

---

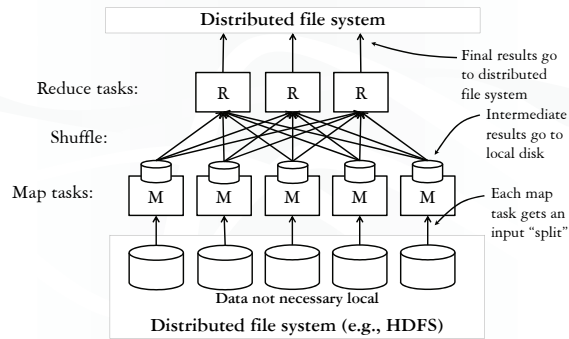
---

---

---

## M/R execution

18



---

---

---

---

---

---

---

---

## M/R example: word count

19

- ❖ Expected input: a huge file (or collection of many files) with millions of lines of English text
- ❖ Expected output: list of (word, count) pairs
- ❖ Implementation
  - $\text{map}(\_, \text{line}) \rightarrow \text{list}(\text{word}, \text{count})$ 
    - Given a line, split it into words, and output  $(w, 1)$  for each word  $w$  in the line
  - $\text{reduce}(\text{word}, \text{list}(\text{count})) \rightarrow (\text{word}, \text{count})$ 
    - Given a word  $w$  and list  $L$  of counts associated with it, compute  $s = \sum_{\text{count} \in L} \text{count}$  and output  $(w, s)$
  - Optimization: before shuffling, map can pre-aggregate word counts locally so there is less data to be shuffled
    - This optimization can be implemented in Hadoop as a “combiner”

---

---

---

---

---

---

---

---

## Some implementation details

20

- ❖ There is one “master” node
- ❖ Input file gets divided into  $m$  “splits,” each a contiguous piece of the file
- ❖ Master assigns  $m$  map tasks (one per split) to “workers” and tracks their progress
- ❖ Map output is partitioned into  $r$  “regions”
- ❖ Master assigns  $r$  reduce tasks (one per region) to workers and tracks their progress
- ❖ Reduce workers read regions from the map workers’ local disks

---

---

---

---

---

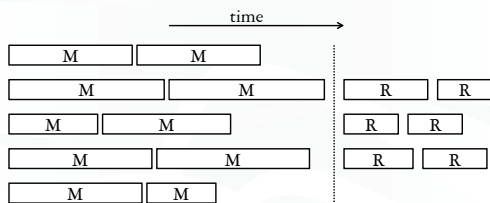
---

---

---

## M/R execution timeline

21



- ❖ When there are more tasks than workers, tasks execute in “waves”
  - Boundaries between waves are usually blurred
- ❖ Reduce tasks can’t start until all map tasks are done

---

---

---

---

---

---

---

---

## More implementation details

22

- ❖ Numbers of map and reduce tasks
  
- ❖ Worker failure
  - Master pings workers periodically
  - If one is down, reassign its split/region to another worker
- ❖ “Straggler”: a machine that is exceptionally slow
  - Pre-emptively run the last few remaining tasks redundantly as backup

---

---

---

---

---

---

---

---

## M/R example: Hadoop TeraSort

23

- ❖ Expected input: a collection of (key, payload) pairs
- ❖ Expected output: sorted (key, payload) pairs
- ❖ Implementation

---

---

---

---

---

---

---

---

## Parallel DBMS vs. MapReduce

24

- ❖ Parallel DBMS
  - Schema + intelligent indexing/partitioning
  - Can stream data from one operator to the next
  - SQL + automatic optimization
- ❖ MapReduce
  - No schema, no indexing
  - Higher scalability and elasticity
    - Just throw new machines in!
  - Better handling of failures and stragglers
  - Black-box map/reduce functions → hand optimization
  - ☞ Work underway to add schema, indexing, declarative languages, and automatic optimization

---

---

---

---

---

---

---

---