Parallel Data Processing

Prof. Yanlei Diao

Slides Courtesy of R. Ramakrishnan and J. Gehrke
In 50 years, computers will be intelligent, with a storage capacity of about $10^9$.

--- Alan Turing, 1950

Moore’s Law predicts the CPU and memory capacities to double about every 18 months.
If exponential growth continues...
Limitations of Moore’s Law

- Concerns regarding if Moore’s law can continue...
- Disk IO bandwidth did not increase much (7200 - 15000 rpm)
- In some domains data grows faster than Moore’s law
- New data processing needs grow fast (“democratization of data”)
Another technology breakthrough is...

Parallel Data Processing
Data Analytics for A Social Network

User profiles:
Background, pics, postings, friends…

Web Server  Web Server  Web Server

Click Streams:
many billion rows/day
many TB/day

Data Loading:
High Volume + Transformation

Data Processing Backend

Quick lookups and updates:
Update your own profile, read friends’ profiles, write msgs,…

Analysis Queries:
Ad targeting, fraud detection, resource provisioning…
Some (Old) Numbers about Facebook

1.5 billion daily active users

22% Internet time of Americans

4 petabyte of new data each day

>250 billion photos; 350 million photos and 5.8 billion likes a day.

Initial software: PHP + MySQL cluster + Memcached

One of the largest MySQL cluster (OLTP)
- serve images/profiles

Hive, FB’s data warehouse (OLAP), has 300 petabytes of data

>30,000 servers
~180,000 servers or more
Google AlphaGo
Three Forms of Parallelism

**Data Parallelism**

**Pipeline Parallelism**

**Model Parallelism**
Topics

1. Parallel databases (80’s - 90’s)

2. MapReduce (2004 - present)

3. Relational processing on MapReduce
Parallel Databases 101

• Rise of parallel databases: late 80’s

• Architecture: **shared-nothing** systems
  - A number of nodes connected by fast Ethernet switches
  - Inter-machine messages are the only way for communication
  - But used special-purpose hardware (costly, slow to evolve)
  - Small scale (hence did not focus on fault tolerance)

• Typical systems
  - Gamma: U. of Wisconsin Madison
  - TeraData: Wal-Mart’s 7.5TB sales data in hundreds of machines
  - Tandem
  - IBM / DB2
  - Informix…
Some Parallel (||) Terminology

• **Speed-Up**
  - Holds the problem size, grows the system
  - Reports *serial time/||-time*
  - Ideally, linear

• **Scale-Up**
  - Grows both the system and the problem, reports running time
  - Ideally, constant
Some Parallel (||) Terminology

- **Speed-Up**
  - Holds the problem size, grows the system
  - Reports serial time/parallel-time
  - Ideally, linear

- **Utilization**
  - Speed-up / degree of ||-ism
  - Ideally, constant
Data Partitioning Schemes

Partitioning a table:

**Range by R.a**
- sequential scan
- associative search
- sorting
- may have data skew

**Hash by R.a**
- sequential scan
- equality search
- equijoins if matching the hash attribute
- range search, operations that do not match the hash attr.
- can have data skew

**Round Robin**
- sequential scan
- useless for other query operations
1) Parallel Scans & Index Lookups

- **Scan** in parallel, and merge.
- **Selection** queries may not require all sites for range or hash partitioning.
  - Want to restrict selection to a few nodes, or restrict “small” queries to a few nodes.
  - Indexes can be built at each partition.
  - What happens during data inserts and lookups?
2) Parallel Sorting

- **Sort R by R.b**, where R is currently partitioned by R.a

- **Idea:**
  - Scan in parallel, and range-partition by R.b as you go.
  - As tuples come into each node, begin “local” sorting.
  - Resulting data is sorted, and range-partitioned.
  - Problem: *skew!*
  - Solution: “sample” the data at start to determine partition boundaries.
Parallel Sort by R.b

Denote a range partitioning function as \( F \):

1. Run sampling to build an equi-depth histogram and set \( F \) for \( R.b \)
2. Partition all data by \( F \) on \( R.b \)
3. Local sort
3) Partitioned Join

- For equi-joins, *partition* the input relations by the join attribute on all nodes, and compute the join locally.
- Can use either *range partitioning* or *hash partitioning*, on the join attribute
  - $R$ and $S$ each are partitioned into $n$ partitions, denoted as $R_0, R_1, ..., R_{n-1}$ and $S_0, S_1, ..., S_{n-1}$.
  - Partitions $R_i$ and $S_i$ are sent to node $i$.
  - Each node locally computes the join using any method.
Dataflow Network for Parallel Join

- Good use of split/merge makes it easier to build parallel versions of sequential join code.
4) Fragment-and-Replicate Join

- Partitioning not possible for non-equi join conditions
  - E.g., R.A > S.B.

- Use the fragment and replicate technique
  - 1) Special case: only one relation is partitioned
    - R is partitioned; any partitioning technique can be used.
    - The other relation, S, is replicated across all the nodes.
    - Node i then locally computes the join of Ri with all of S using any join technique.
    - Works well when S is small.
  - 2) General case: both R and S are partitioned
    - Need to replicate all R partitions or all S partitions
    - Depicted on the next slide
Illustrating Fragment-and-Replicate Join

(a) Asymmetric fragment and replicate

(b) Fragment and replicate
5) Parallel Aggregates

• For each aggregate function, need a decomposition:
  – **Distributive**: \( \text{count}(S) = \Sigma \text{count}(s(i)) \), ditto for \( \text{sum}() \)
  – **Algebraic**: \( \text{avg}(S) = (\Sigma \text{sum}(s(i))) / \Sigma \text{count}(s(i)) \)
  – **Holistic**: e.g., median, quantiles

• For group-by aggregation:
  – How would you implement parallel group by? See slides later..
  – How do you add aggregation to each group?