CSE 444: Database Internals

Lectures 23-24
Parallel DBMSs

References

- Book Chapter 20.1
- **Database management systems.**
  Ramakrishnan and Gehrke.
  Third Ed. **Chapter 22.11**
  (more info than our main book)

Parallel v.s. Distributed Databases

- Distributed database system:
  - Data is stored across several sites, each site managed by a DBMS capable of running independently
- Parallel database system:
  - Improve performance through parallel implementation

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

Demonstration

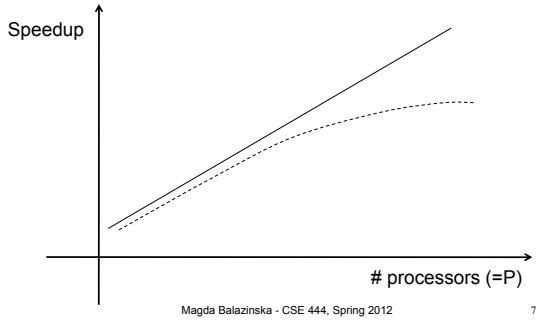
- Lab 6 demonstration

Performance Metrics for Parallel DBMSs

Speedup

- More processors ➔ higher speed
- Individual queries should run faster
- Should do more transactions per second (TPS)
- Fixed problem size *overall*, vary # of processors ("strong scaling")

Linear v.s. Non-linear Speedup

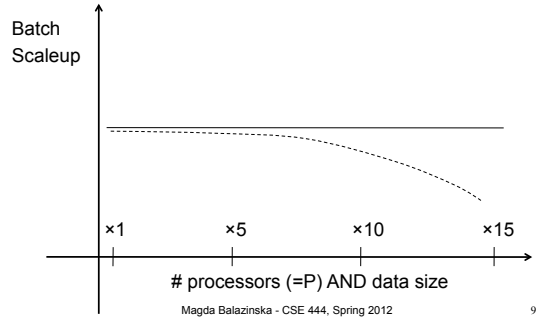


Performance Metrics for Parallel DBMSs

Scaleup

- More processors ➔ can process more data
- Fixed problem size *per processor,* vary # of processors ("weak scaling")
- 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 Scaleup



Warning

- Be careful. Commonly used terms today:
  - "scale up" = use an increasingly more powerful server
  - "scale out" = use a larger number of servers

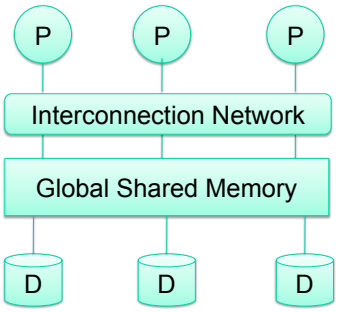
Challenges to Linear Speedup and Scaleup

- 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

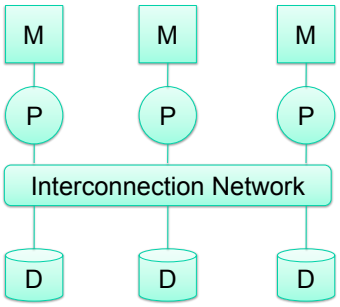
- Shared memory
- Shared disk
- Shared nothing

Shared Memory



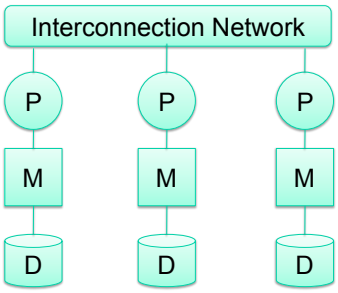
13

Shared Disk



14

Shared Nothing



15

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
  - "Processor" != core
- P = number of nodes

We will focus on shared nothing

16

Question

- What can we parallelize in a parallel DBMS ?

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

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 $v_{i-1} < t.A < v_i$

Horizontal Data Partitioning

- All three choices are just special cases:
  - For each tuple, compute *bin = f(t)*
  - Different properties of the function *f* determine hash vs. range vs. round robin vs. anything

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 if cost is response time
- 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 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: $R_{i0}, R_{i1}, \ldots, R_{i,P-1}$
- Step 2: server i sends partition R_{ij} to serve j
- Step 3: server j computes $\gamma_{A, sum(B)}$ on $R_{0j}, R_{1j}, \ldots, R_{P-1,j}$

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

- Can we do better?
- Sum?
- Count?
- Avg?
- Max?
- Median?

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

- $Sum(B) = Sum(B_0) + Sum(B_1) + \ldots + Sum(B_n)$
- $Count(B) = Count(B_0) + Count(B_1) + \ldots + Count(B_n)$
- $Max(B) = Max(Max(B_0), Max(B_1), \ldots, Max(B_n))$

distributive

- Avg(B) = Sum(B) / Count(B)

algebraic

- Median(B) =

holistic

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: $R_{i0}, R_{i1}, \ldots, R_{i,P-1}$
  - For all servers in [k+1,P], server j partitions chunk S_j using a hash function h(t.A) mod P: $S_{j0}, S_{j1}, \ldots, R_{j,P-1}$
- 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
  - Assuming both R and S are spread across all servers
- 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
  - 4(B(R)+B(S))/P otherwise

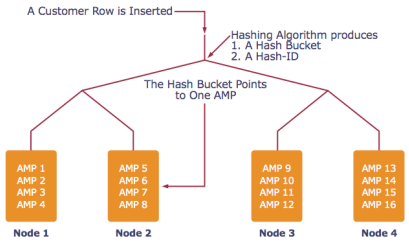
Parallel Dataflow Implementation

- Use relational operators unchanged
- Add a special *shuffle* operator
  - Handle data routing, buffering, and flow control
  - Inserted between consecutive operators in the query plan
  - Two components: ShuffleProducer and ShuffleConsumer
  - 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

Modern Shared Nothing Parallel DBMSs

- *Greenplum* founded in 2003 acquired by EMC in 2010
- *Vertica* founded in 2005 and acquired by HP in 2011
- *DATAllegro* founded in 2003 acquired by Microsoft in 2008
- *Netezza* founded in 2000 and acquired by IBM in 2010
- *Aster Data* Systems founded in 2005 acquired by Teradata in 2011
  - MapReduce-based data processing system (next week)

Example System: Teradata

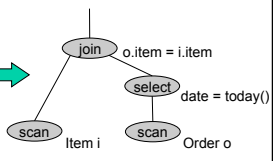


AMP = unit of parallelism

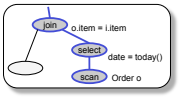
Example System: Teradata

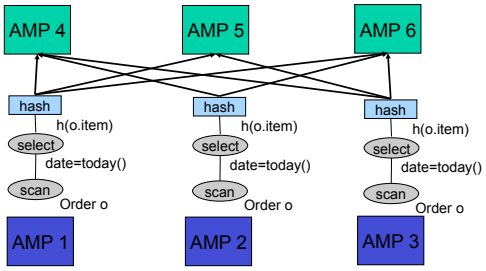
Find all orders from today, along with the items ordered

```
SELECT *
  FROM Orders o, Lines i
 WHERE o.item = i.item
   AND o.date = today()
```

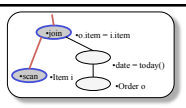


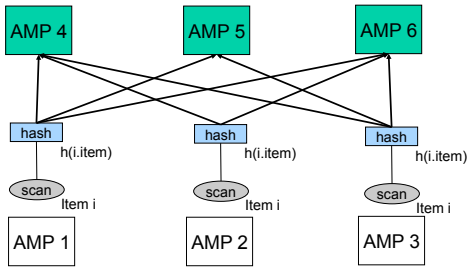
Example System: Teradata





Example System: Teradata





Example System: Teradata