CSE 444: Database Internals

Lectures 19-20
Parallel DBMSs

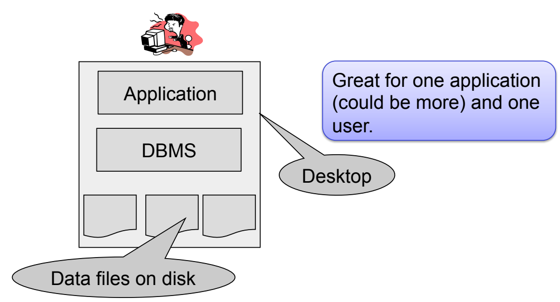
What We Have Already Learned

- Overall architecture of a DBMS
- Internals of query execution:
  - Data storage and indexing
  - Buffer management
  - Query evaluation including operator algorithms
  - Query optimization
- Internals of transaction processing:
  - Concurrency control: pessimistic and optimistic
  - Transaction recovery: undo, redo, and undo/redo

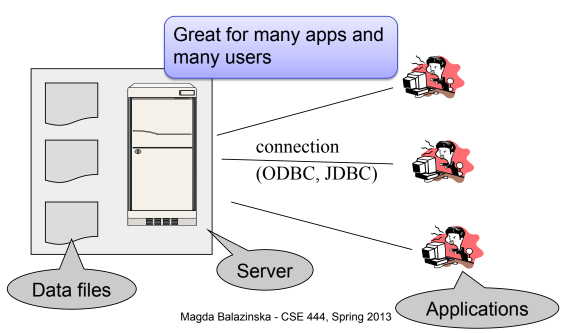
Where We Are Headed Next

- Scaling the execution of a query (this week)
  - Parallel DBMS
  - MapReduce
  - Distributed query processing and optimization
- Scaling transactions (next week)
  - Distributed transactions
  - Replication
- Scaling with NoSQL and NewSQL (in two weeks)

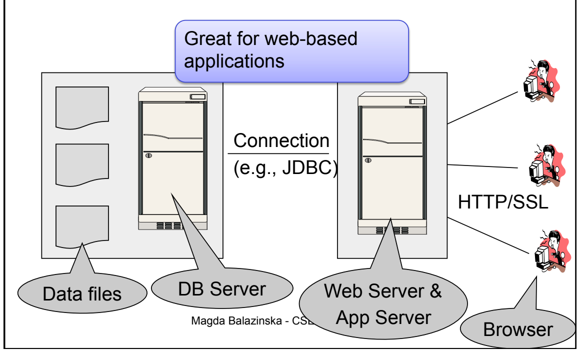
DBMS Deployment: Local



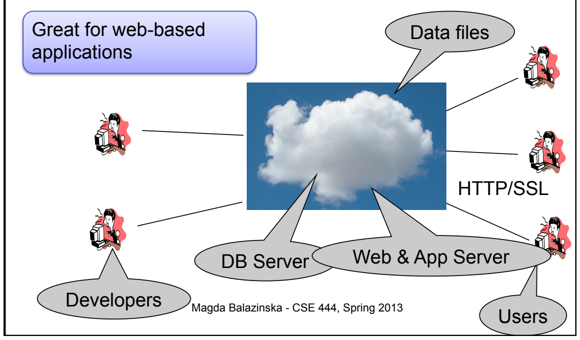
DBMS Deployment: Client/Server



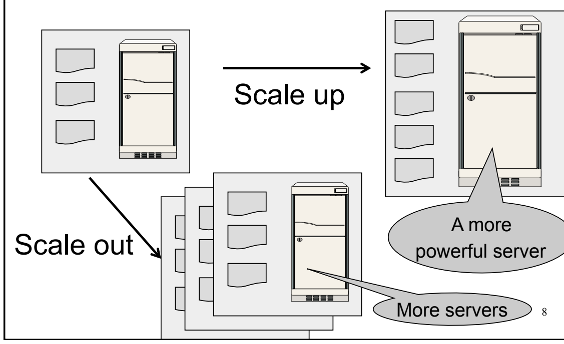
DBMS Deployment: 3 Tiers



DBMS Deployment: Cloud



How to Scale a DBMS?



Why Do I Care About Scaling Transactions Per Second?

- Amazon
- Facebook
- Twitter
- … your favorite Internet application…

- Goal is to scale OLTP workloads

- We will get back to this next week

Why Do I Care About Scaling A Single Query?

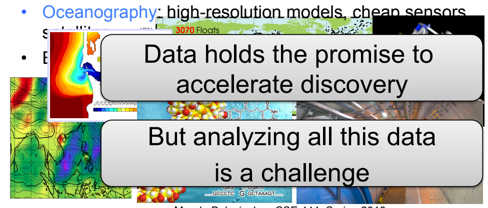
- Goal is to scale OLAP workloads

- That means the analysis of massive datasets

This Week: Focus on Scaling a Single Query

Science is Facing a Data Deluge!

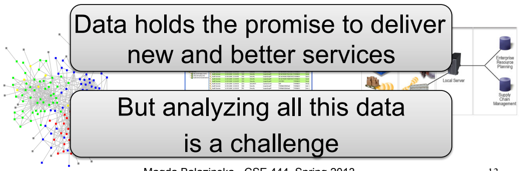
- Astronomy: High-resolution, high-frequency sky surveys (SDSS, LSST)
- Medicine: ubiquitous digital records, MRI, ultrasound
- Biology: lab automation, high-throughput sequencing
- Oceanography: high-resolution models, cheap sensors,



Industry is Facing a Data Deluge!

- Clickstreams, search logs, network logs, social networking data, RFID data, etc.
- Examples: Facebook, Twitter, Google, Microsoft, Amazon, Walmart, etc.

Data holds the promise to deliver new and better services



But analyzing all this data is a challenge

Big Data

- Companies, organizations, scientists have data that is **too big, too fast, and too complex** to be managed without changing tools and processes
- Relational algebra and SQL are easy to parallelize and parallel DBMSs have already been studied in the 80's!

Data Analytics Companies

As a result, we are seeing an explosion of and a huge success of db analytics companies

- Greenplum founded in 2003 acquired by EMC in 2010; A parallel shared-nothing DBMS (this lecture)
- Vertica founded in 2005 and acquired by HP in 2011; A parallel, column-store shared-nothing DBMS (see 444 for discussion of column-stores)
- DATAllegro founded in 2003 acquired by Microsoft in 2008; A parallel, shared-nothing DBMS
- Aster Data Systems founded in 2005 acquired by Teradata in 2011; A parallel, shared-nothing, MapReduce-based data processing system (next lecture). SQL on top of MapReduce
- Netezza founded in 2000 and acquired by IBM in 2010. A parallel, shared-nothing DBMS.

Great time to be in the data management, data mining/statistics, or machine learning!

Two Approaches to Parallel Data Processing

- Parallel databases, developed starting with the 80s (this lecture)
  - For both OLTP (transaction processing)
  - And for OLAP (Decision Support Queries)
- MapReduce, first developed by Google, published in 2004 (next lecture)
  - Only for Decision Support Queries

Today we see convergence of the two approaches (Greenplum,Tenzing SQL)

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 (early next week):
  - Data is stored across several sites, each site managed by a DBMS capable of running independently
- Parallel database system (today):
  - Improve performance through parallel implementation

Parallel DBMSs

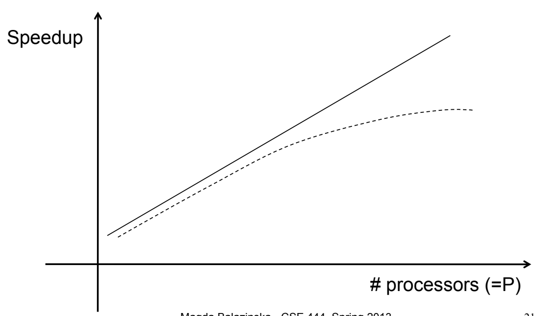
- Goal
  – Improve performance by executing multiple operations in parallel
- Key benefit
  – Cheaper to scale than relying on a single increasingly more powerful processor
- Key challenge
  – Ensure overhead and contention do not kill performance

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

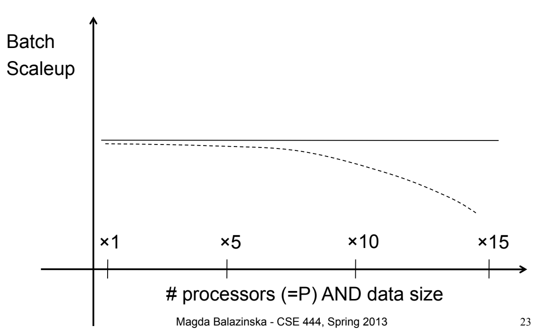


Performance Metrics for Parallel DBMSs

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

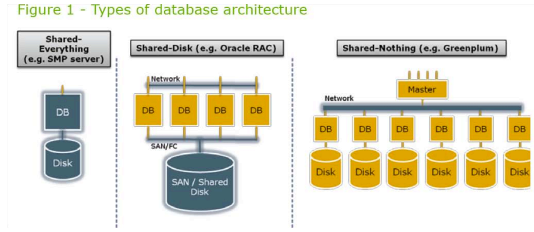
- Be careful. Commonly used terms today:
  – "scale up" = use an increasingly more powerful server
  – "scale out" = use a larger number of servers

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

Figure 1 - Types of database architecture



From: Greenplum Database Whitepaper

SAN = “Storage Area Network”

Shared Memory

- Nodes share both RAM and disk
- Dozens to hundreds of processors

Example: SQL Server runs on a single machine and can leverage many threads to get a query to run faster (see query plans)

- Easy to use and program
- But very expensive to scale

Shared Disk

- All nodes access the same disks
- Found in the largest "single-box" (non-cluster) multiprocessors

Oracle dominates this class of systems

Characteristics:
- Also hard to scale past a certain point: existing deployments typically have fewer than 10 machines

Shared Nothing

- Cluster of machines on high-speed network
- Called "clusters" or "blade servers”
- Each machine has its own memory and disk: lowest contention.

NOTE: Because all machines today have many cores and many disks, then shared-nothing systems typically run many "nodes” on a single physical machine.

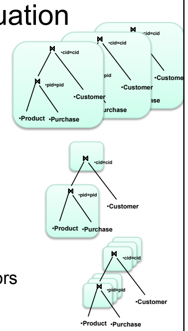
Characteristics:
- Today, this is the most scalable architecture.
- Most difficult to administer and tune.

We discuss only Shared Nothing in class

In Class

- You have a parallel machine. Now what?
- How do you speed up your DBMS?

Approaches to Parallel Query Evaluation



- Inter-query parallelism
  - Each query runs on one processor
  - Only for OLTP queries
- Inter-operator parallelism
  - A query runs on multiple processors
  - An operator runs on one processor
  - For both OLTP and Decision Support
- Intra-operator parallelism
  - An operator runs on multiple processors
  - For both OLTP and Decision Support

We study only intra-operator parallelism: most scalable

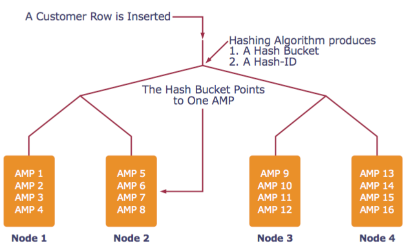
Horizontal Data Partitioning

- Relation R split into P chunks $R_0, \ldots, R_{P-1}$, stored at the P nodes
- Block partitioned
  - Each group of k tuples go to a different node
- 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 $v_{i-1} < t.A < v_i$

Uniform Data v.s. Skewed Data

- Let R(K,A,B,C); which of the following partition methods may result in skewed partitions?
- Block partition — Uniform
- Hash-partition
  - On the key K — Uniform (Assuming uniform hash function)
  - On the attribute A — May be skewed (E.g. when all records have the same value of the attribute A, then all records end up in the same partition)
- Range-partition
  - On the key K
  - On the attribute A — May be skewed (Difficult to partition the range of A uniformly.)

Example from Teradata



AMP = unit of parallelism

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 ?
  - Block partitioned
  - Hash partitioned
  - Range partitioned

Parallel Selection

- Q: What is the cost on a parallel database with P nodes ?
- A: B(R) / P in all cases if cost is response time
- However, different processors do the work:
  - Block: all servers do the work
  - Hash: one server for $\sigma_{A=v}(R)$, all for $\sigma_{v1<A<v2}(R)$
  - Range: some servers only

Data Partitioning Revisited

What are the pros and cons ?

- Block based partitioning
  - Good load balance but always needs to read all the data
- Hash based partitioning
  - Good load balance
  - Can avoid reading all the data for equality selections
- Range based partitioning
  - Can suffer from skew (i.e., load imbalances)
  - Can help reduce skew by creating uneven partitions

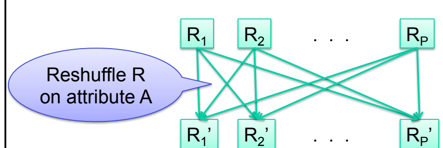
Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

- 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}, …, $R_{P-1,j}$

Parallel GroupBy

$\gamma_{A,sum(C)}(R)$

- If R is partitioned on A, then each node computes the group-by locally
- Otherwise, hash-partition R(<u>K</u>,A,B,C) on A, then compute group-by locally:



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) + … + Sum(B_n)
- Count(B) = Count(B_0) + Count(B_1) + … + Count(B_n)
- Max(B) = Max(Max(B_0), Max(B_1), …, Max(B_n))

distributive

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

algebraic

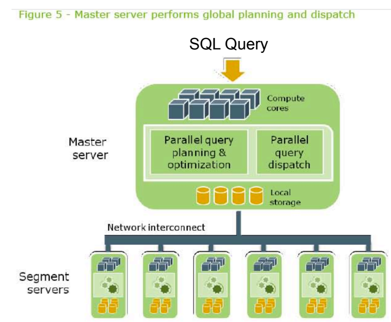
- Median(B) =

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}, …, $R_{i,P-1}$
  – 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}

Overall Architecture

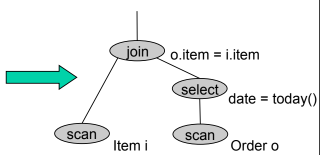


From: Greenplum Database Whitepaper

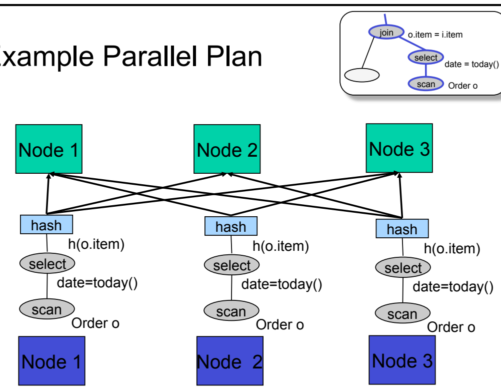
Example of Parallel Query Plan

Find all orders from today, along with the items ordered

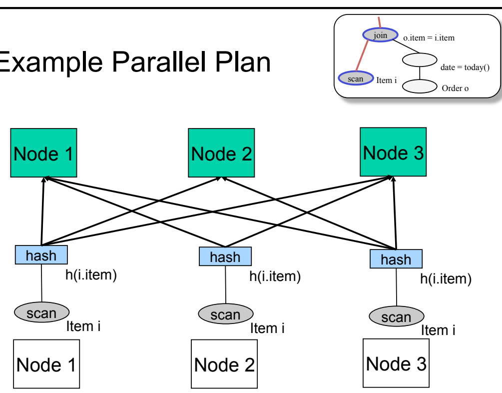
```
SELECT *
  FROM Orders o, Lines i
 WHERE o.item = i.item
   AND o.date = today()
```



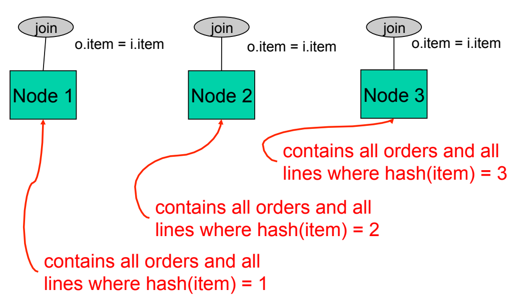
Example Parallel Plan



Example Parallel Plan



Example Parallel Plan



Optimization for Small Relations

- When joining R and S
- If |R| >> |S|
  - Leave R where it is
  - Replicate entire S relation across nodes
- Sometimes called a "small join"

Other Interesting Parallel Join Implementation

Problem of skew during join computation

- Some join partitions get more **input** tuples than others
  - Reason 1: Base data unevenly distributed across machines
    - Because used a range-partition function
    - Or used hashing but some values are very popular
  - Reason 2: Selection before join with different selectivities
  - Reason 3: Input data got unevenly rehashed (or otherwise repartitioned before the join)
- Some partitions **output** more tuples than others

Some Skew Handling Techniques

1. Use range- instead of hash-partitions
   - Ensure that each range gets same number of tuples
   - Example: {1, 1, 1, 2, 3, 4, 5, 6 } → [1,2] and [3,6]
2. Create more partitions than nodes
   - And be smart about scheduling the partitions
3. Use subset-replicate (i.e., "skewedJoin")
   - Given an extremely common value 'v'
   - Distribute R tuples with value v randomly across k nodes (R is the build relation)
   - Replicate S tuples with value v to same k machines (S is the probe relation)

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)