

CSE 444: Database Internals

Lectures 20-21
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

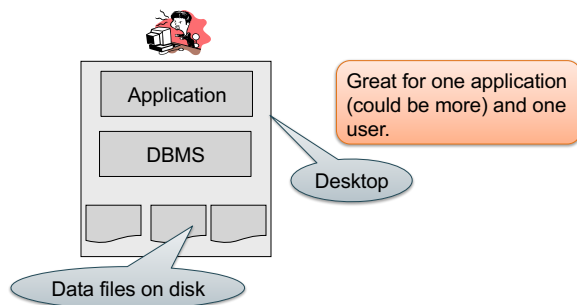
Where We Are Headed Next

- Scaling the execution of a query
 - Parallel DBMS
 - MapReduce
 - Spark and Myria
- Scaling transactions
 - Distributed transactions
 - Replication
- Scaling with NoSQL and NewSQL

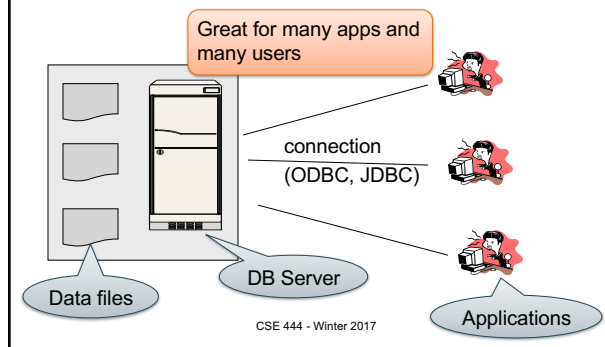
Reading Assignments

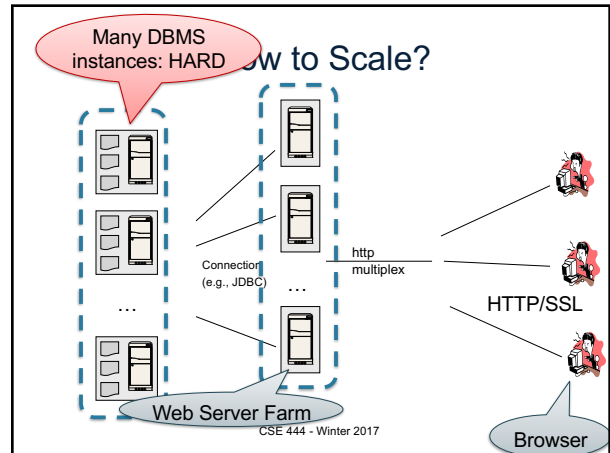
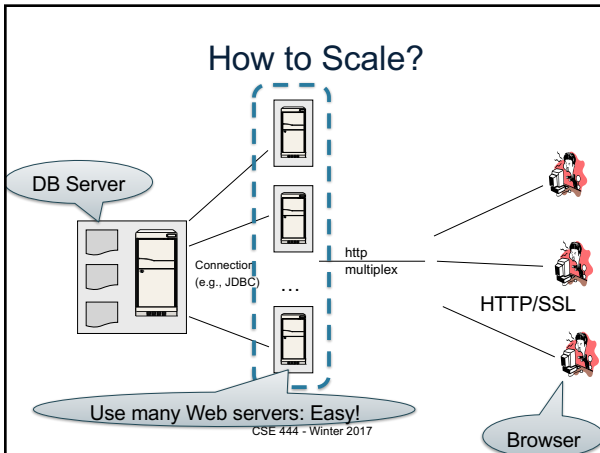
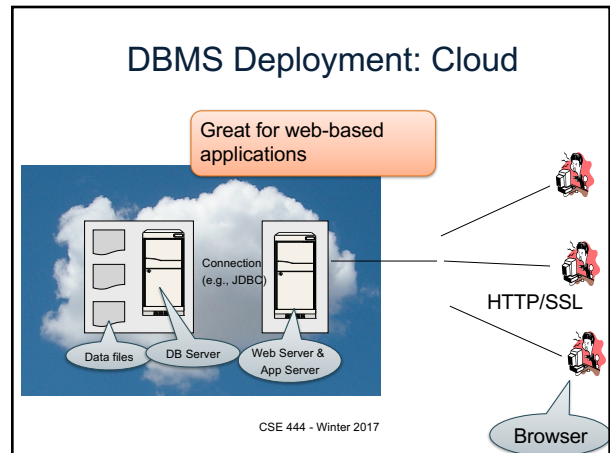
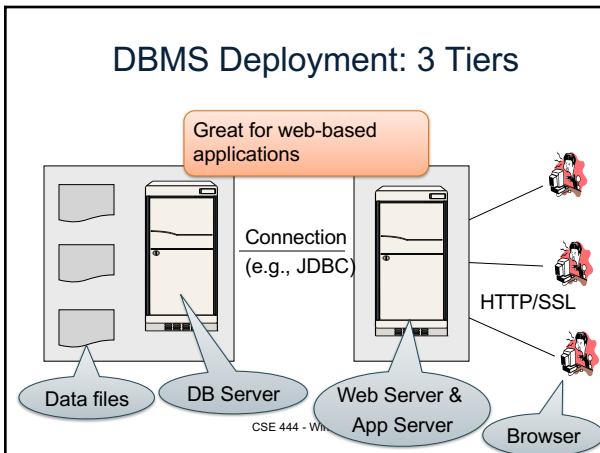
- Main textbook Chapter 20.1
- Database management systems. Ramakrishnan&Gehrke. Third Ed. Chapter 22.11

DBMS Deployment: Local

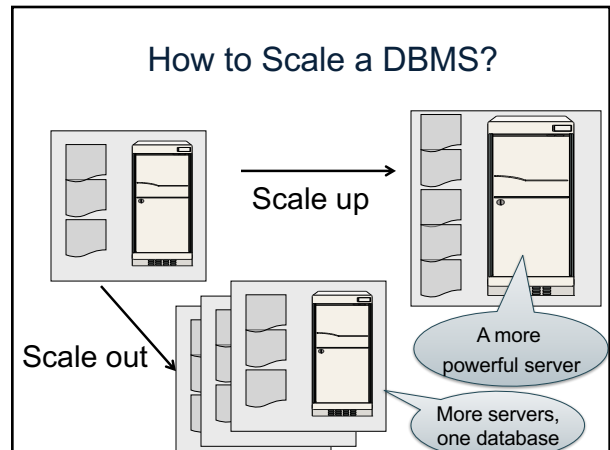


DBMS Deployment: Client/Server





- ### How to Scale?
- We can easily replicate the web servers and the application servers
 - We cannot so easily replicate the database servers, because the database is unique
 - We need to design ways to scale up the DBMS
- CSE 444 - Winter 2017 11



What to scale?

- OLTP: Transactions per second
 - OLTP = Online Transaction Processing
- OLAP: Query response time
 - OLAP = Online Analytical Processing

CSE 444 - Winter 2017

13

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

CSE 444 - Winter 2017

14

Scaling Single Query Response Time

- Goal is to scale OLAP workloads
- That means the analysis of massive datasets

CSE 444 - Winter 2017

15

This Week: Focus on Scaling a Single Query

CSE 444 - Winter 2017

16

Big Data

- Buzzword?
- Definition from industry:
 - High Volume <http://www.gartner.com/newsroom/id/1731916>
 - High Variety
 - High Velocity

CSE 444 - Winter 2017

17

Big Data

- Volume is not an issue
- Databases *do* parallelize easily; techniques available from the 80's
 - Data partitioning
 - Parallel query processing
 - SQL is *embarrassingly parallel*
 - We will learn how to do this

CSE 444 - Winter 2017

18

Big Data

New workloads are an issue

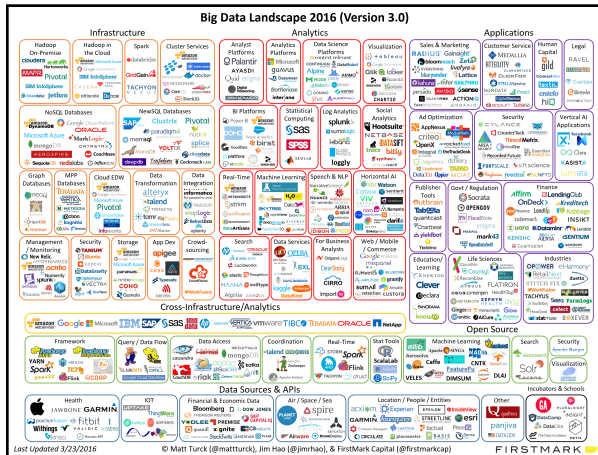
- Big volumes, small analytics
 - OLAP queries: join + group-by + aggregate
 - Can be handled by today's RDBMSs (e.g., Teradata)
- Big volumes, big analytics
 - More complex Machine Learning, e.g. click prediction, topic modeling, SVM, k-means
 - Requires innovation – Active research area

Data Analytics Companies

Ten years ago, explosion 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
- **DATAlegro** 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 (in two lectures). 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 data management, data mining/statistics, or machine learning!



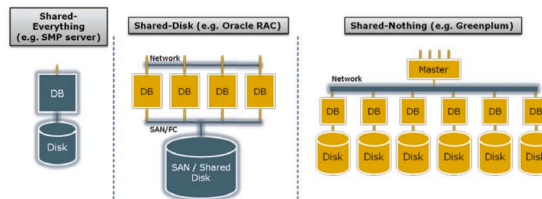
Two Fundamental Approaches to Parallel Data Processing

- **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 (in two lectures)
 - Only for **decision support queries**

Today we see convergence of the two approaches

Architectures for Parallel DMBSs

Figure 1 - Types of database architecture



From: Greenplum Database Whitepaper

SAN = "Storage Area Network"

Our Focus: Shared-Nothing DBMS

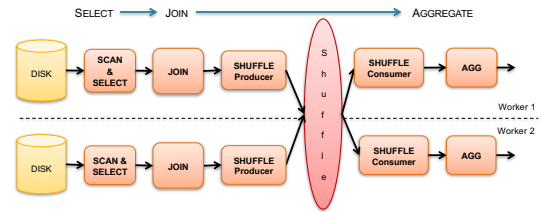
Parallel Query Evaluation

- Multiple DBMS instances (= processes) also called "nodes" execute on machines in a cluster
 - One instance plays role of the coordinator
 - Other instances play role of workers
- Applications interact with coordinator
- Workers execute queries
 - Typically **all workers execute the same plan**
 - Intra-operator parallelism & intra-query parallelism
 - Some operations may execute at subsets of workers
 - Workers can execute **multiple queries at the same time**
 - Inter-query parallelism

CSE 444 - Winter 2017

25

Parallel Query Execution



CSE 444 - Winter 2017

26

Parallel Query Evaluation

- New operator: **Shuffle**
- Origin: **Exchange** operator from Volcano system
 - Serves to re-shuffle data between processes
 - Handles data routing, buffering, and flow control
 - Two parts: **ShuffleProducer** and **ShuffleConsumer**
 - Producer:
 - Pulls data from child 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

CSE 444 - Winter 2017

27

Parallel DBMSs

- **Performance metrics**
 - **Speedup**: More nodes, same data → higher speed
 - **Scaleup**: More nodes, more data → same speed
 - Speed = query execution time
- **Key challenges**
 - Start-up costs
 - Interference
 - Skew

CSE 444 - Winter 2017

28

Parallel Query Processing

How do we **compute** these operations on a shared-nothing parallel db?

- **Selection**: $\sigma_{A=123}(R)$
- **Group-by**: $\gamma_{A, \text{sum}(B)}(R)$
- **Join**: $R \bowtie S$

Before we answer that: how do we **store** R (and S) on a shared-nothing parallel db?

CSE 444 - Winter 2017

29

Horizontal Data Partitioning

Data:

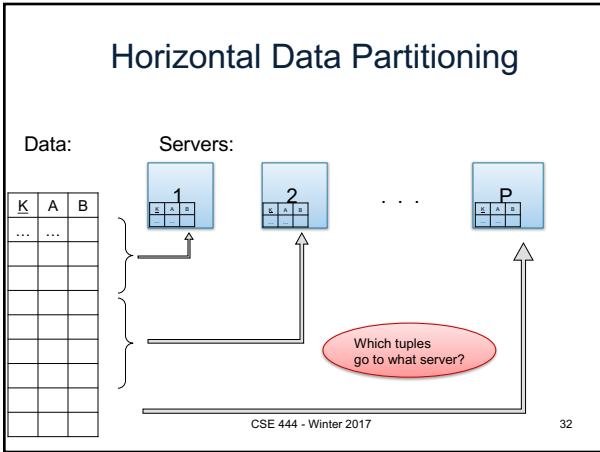
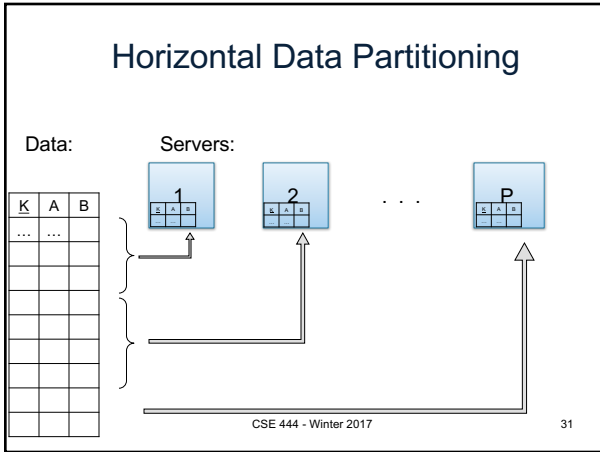
K	A	B
...	...	

Servers:



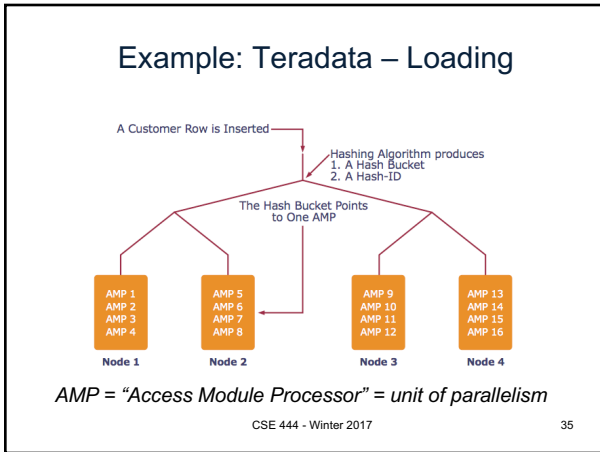
CSE 444 - Winter 2017

30



- ### Horizontal Data Partitioning
- Relation R split into P chunks R_0, \dots, R_{P-1} , stored at the P nodes
 - **Block partitioned**
 - Each group of k tuples goes to a different node
 - **Hash based partitioning on attribute A:**
 - Tuple t to chunk $h(t.A) \bmod P$
 - **Range based partitioning on attribute A:**
 - Tuple t to chunk i if $v_{i-1} < t.A < v_i$
 - For hash and range partitioning: Beware of skew
- CSE 444 - Winter 2017 33

- ### 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
- CSE 444 - Winter 2017 34



- ### Parallel Selection
- Compute $\sigma_{A=v_1}(R)$, or $\sigma_{v_1 < A < v_2}(R)$
- 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
- CSE 444 - Winter 2017 36

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v_1 < A < v_2}(R)$

- On a conventional database: cost = $B(R)$
- **Q:** What is the cost on a parallel database with P processors? **A:** $B(R) / P$, but
 - Block partitioned -- all servers do the work
 - Hash partitioned -- some servers do the work
 - Range partitioned -- some servers do the work

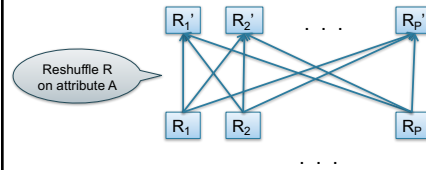
CSE 444 - Winter 2017

37

Basic Parallel GroupBy

Data: $R(K,A,B,C)$ -- hash-partitioned on K

Query: $\gamma_{A, \text{sum}(B)}(R)$



CSE 444 - Winter 2017

38

Basic Parallel GroupBy

- Step 1: each server i partitions its chunk R_i using a hash function $h(t.A) \bmod P$: $R_{i,0}, R_{i,1}, \dots, R_{i,P-1}$
- Step 2: server j computes $\gamma_{A, \text{sum}(B)}$ on $R_{0,j}, R_{1,j}, \dots, R_{P-1,j}$

CSE 444 - Winter 2017

39

Speedup and Scaleup

- Consider:
 - Query: $\gamma_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- **If we double the number of nodes P , what is the new running time?**
- **If we double both P and the size of R , what is the new running time?**

CSE 444 - Winter 2017

40

Speedup and Scaleup

- Consider:
 - Query: $\gamma_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- **If we double the number of nodes P , what is the new running time?**
 - Half (each server holds $\frac{1}{2}$ as many chunks)
- **If we double both P and the size of R , what is the new running time?**
 - Same (each server holds the same # of chunks)

CSE 444 - Winter 2017

41

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

CSE 444 - Winter 2017

42

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Distributive	Algebraic	Holistic
$\text{sum}(a_1+a_2+\dots+a_n)=$ $\text{sum}(\text{sum}(a_1+a_2+a_3)+$ $\text{sum}(a_4+a_5+a_6)+$ $\text{sum}(a_7+a_8+a_9))$	$\text{avg}(B) =$ $\text{sum}(B)/\text{count}(B)$	$\text{median}(B)$

YES

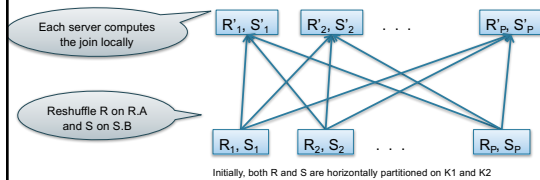
- Compute partial aggregates before shuffling

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

- **Data:** $R(K_1, A, C), S(K_2, B, D)$
- **Query:** $R(K_1, A, C) \bowtie_{A=B} S(K_2, B, D)$

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

- **Data:** $R(K_1, A, C), S(K_2, B, D)$
- **Query:** $R(K_1, A, C) \bowtie_{A=B} S(K_2, B, D)$



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

- Step 1
 - Every server holding any chunk of R partitions its chunk using a hash function $h(t.A) \bmod P$
 - Every server holding any chunk of S partitions its chunk using a hash function $h(t.B) \bmod P$
- Step 2:
 - Each server computes the join of its local fragment of R with its local fragment of S

Optimization for Small Relations

When joining R and S

- If $|R| \gg |S|$
 - Leave R where it is
 - Replicate entire S relation across nodes
- Also called a **small join** or a **broadcast join**

Other Interesting Parallel Join Implementation

Skew:

- Some partitions get more **input** tuples than others
 - Reasons:**
 - Range-partition instead of hash
 - Some values are very popular:
 - Heavy hitters values; e.g. 'Justin Bieber'
 - Selection before join with different selectivities
- Some partitions generate more **output** tuples than others

Some Skew Handling Techniques

If using range partition:

- Ensure each range gets same number of tuples
- E.g.: {1, 1, 1, 2, 3, 4, 5, 6} → [1,2] and [3,6]
- Eq-depth v.s. eq-width histograms

Some Skew Handling Techniques

Create more partitions than nodes

- And be smart about scheduling the partitions
- Note: MapReduce uses this technique

Some Skew Handling Techniques

Use subset-replicate (a.k.a. "skewedJoin")

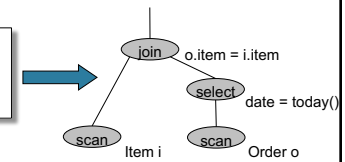
- Given $R \bowtie_{A=B} S$
- Given a heavy hitter value $R.A = 'v'$ (i.e. 'v' occurs very many times in R)
- Partition R tuples with value 'v' across all nodes e.g. block-partition, or hash on other attributes
- Replicate S tuples with value 'v' to all nodes
- R = the build relation
- S = the probe relation

Order(oid, item, date), Line(item, ...)

Example: Teradata – Query Execution

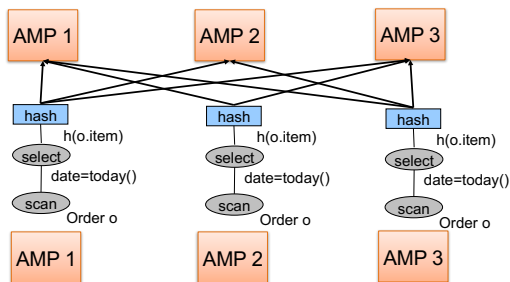
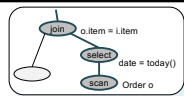
Find all orders from today, along with the items ordered

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



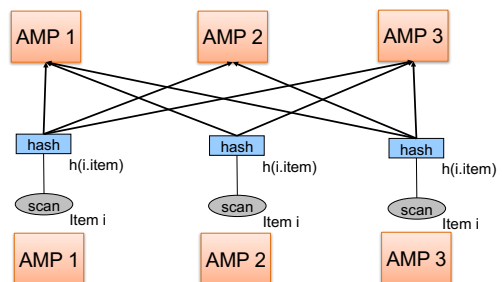
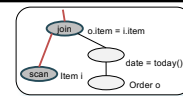
Order(oid, item, date), Line(item, ...)

Query Execution



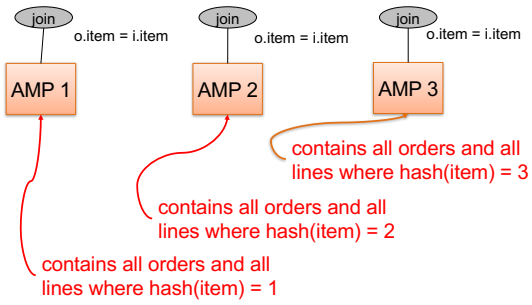
Order(oid, item, date), Line(item, ...)

Query Execution



Order(ojid, item, date), Line(item, ...)

Query Execution



CSE 444 - Winter 2017

55