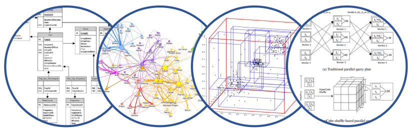


Database System Internals

Intro to Parallel DBMSs

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

1

Announcements

- Quiz 3+4 canceled ☺

2

What We Have Already Learned

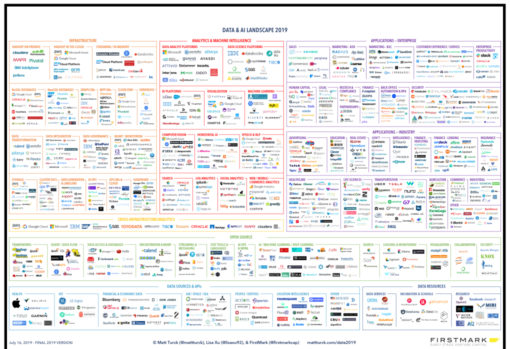
- Phase 1: Query Execution
  - Data Storage and Indexing
  - Buffer management
  - Query evaluation and operator algorithms
  - Query optimization
- Phase 2: Transaction Processing
  - Concurrency control: pessimistic and optimistic
  - Transaction recovery: undo, redo, and undo/redo
- Phase 3: Parallel Processing & Distributed Transactions

3

Where We Are Headed Next

- Scaling the execution of a query
  - Parallel DBMS
  - MapReduce
  - Spark
- Scaling transactions
  - Distributed transactions
  - Replication

4



5

How to Scale the DBMS?

- 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**

6

Building Our Parallel DBMS

Data model? Relational (SimpleDB!)

9

Building Our Parallel DBMS

Data model? Relational (SimpleDB!)

Scaleup goal?

10

Scaling Transactions Per Second

- OLTP: Transactions per second
  "Online Transaction Processing"
- Amazon
- Facebook
- Twitter
- … your favorite Internet application…
- Goal is to increase transaction throughput
- We will get back to this next week

11

Scaling Single Query Response Time

- OLAP: Query response time
  "Online Analytical Processing"
- Entire parallel system answers one query
- Goal is to improve query runtime
- Use case is analysis of massive datasets

12

Big Data

Volume alone is not an issue

- Relational 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!

13

Big Data

New **workloads** are an issue

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

14

Building Our Parallel DBMS

Data model? Relational

Scaleup goal? OLAP

15

Building Our Parallel DBMS

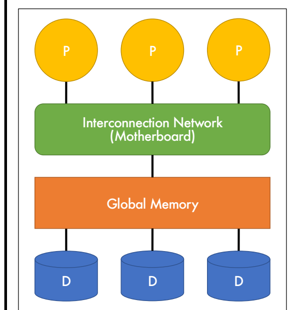
Data model? Relational

Scaleup goal? OLAP

Architecture?

16

Shared-Memory Architecture

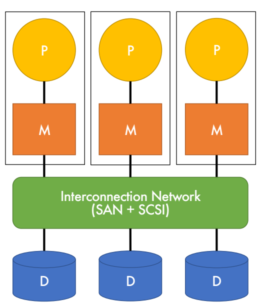


- Shared main memory and disks
- Your laptop or desktop uses this architecture
- Expensive to scale
- Easiest to implement on

Microsoft SQL Server
PostgreSQL
SQLite
MySQL

17

Shared-Disk Architecture

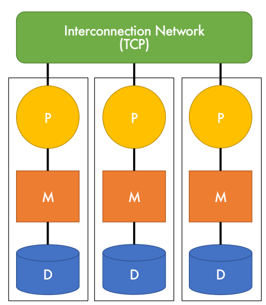


- Only shared disks
- No contention for memory and high availability
- Typically 1-10 machines

ORACLE
DATABASE

18

Shared-Nothing Architecture



- Uses cheap, commodity hardware
- No contention for memory and high availability
- Theoretically can **scale infinitely**
- Hardest to implement on

teradata.
APACHE Spark
MySQL Cluster

19

Building Our Parallel DBMS

Data model? Relational

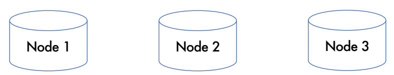
Scaleup goal? OLAP

Architecture? Shared-Nothing

20

Shared-Nothing Execution Basics

- Multiple DBMS instances (= processes) also called "nodes" execute on machines in a cluster
  - One node plays role of the coordinator
  - Other nodes play role of workers
- Workers execute queries
  - Typically **all workers execute the same plan**
  - Workers can execute multiple queries at the same time

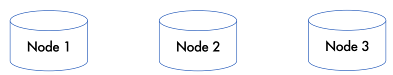


21

Shared-Nothing Database

We will assume a system that consists of multiple commodity machines on a common network

New problem: **Where does the data go?**



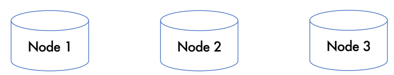
22

Shared-Nothing Database

We will assume a system that consists of multiple commodity machines on a common network

New problem: **Where does the data go?**

The answer will influence our execution techniques



23

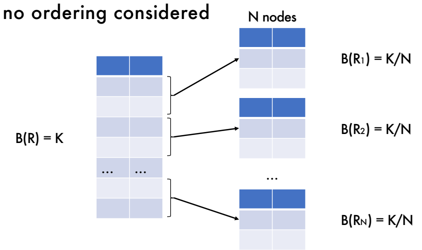
Option 1: Unpartitioned Table

- Entire table on just one node in the system
- Will bottleneck any query we need to run in parallel
- We choose partitioning scheme to divide rows among machines

24

Option 2: Block Partitioning

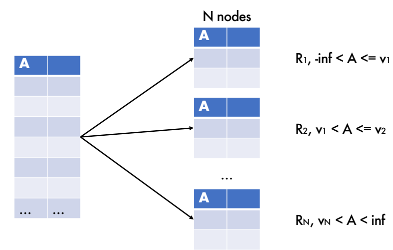
Tuples are horizontally (row) partitioned by raw size with no ordering considered



25

Option 3: Range Partitioning

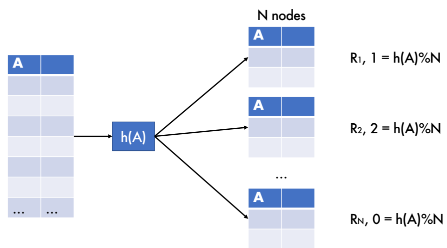
Node contains tuples in chosen attribute ranges



26

Option 4: Hash Partitioning

Node contains tuples with chosen attribute hashes



27

Skew: The Justin Bieber Effect

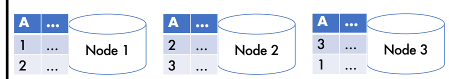
- Hashing data to nodes is very good when the attribute chosen better approximates a uniform distribution
- Keep in mind: Certain nodes will become bottlenecks if a poorly chosen attribute is hashed

28

Parallel Selection

Assume:
R is block partitioned

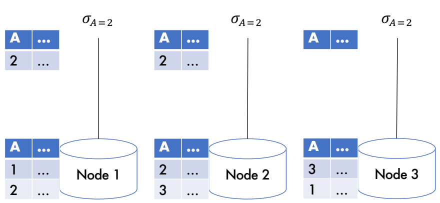
```
SELECT *
  FROM R
 WHERE A = 2
```



29

Parallel Selection

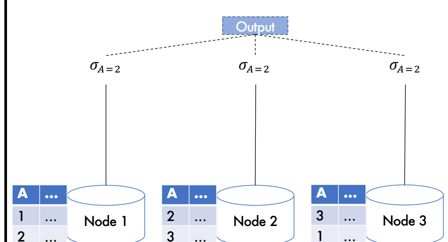
```
SELECT *
  FROM R
 WHERE A = 2
```



30

Implicit Union

Parallel query plans implicitly union at the end

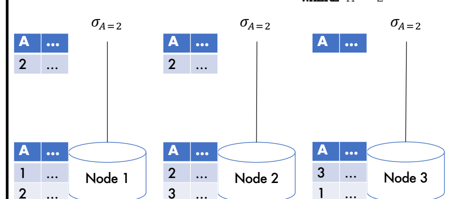


31

Parallel Selection

Data-parallel!

```
SELECT *
  FROM R
 WHERE A = 2
```



32

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 each node for a database with N nodes ?

A:

33

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 each node for a database with N nodes ?

A: B(R) / N block reads on each node

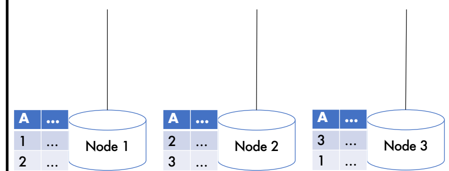
34

Parallel Selection

What if this query is not data-parallel?

Assume:
R is block partitioned

```
SELECT *
  FROM R
  ......
```

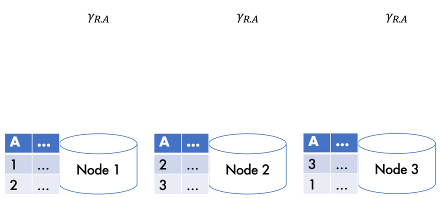


35

Partitioned Aggregation

Assume:
R is block partitioned

```
SELECT *
  FROM R
 GROUP BY R.A
```

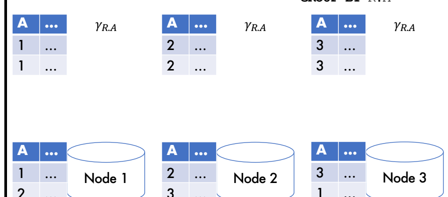


36

Partitioned Aggregation

Assume:
R is block partitioned

```
SELECT *
  FROM R
 GROUP BY R.A
```



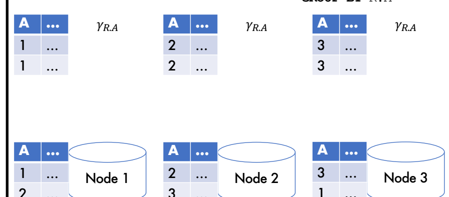
37

Partitioned Aggregation

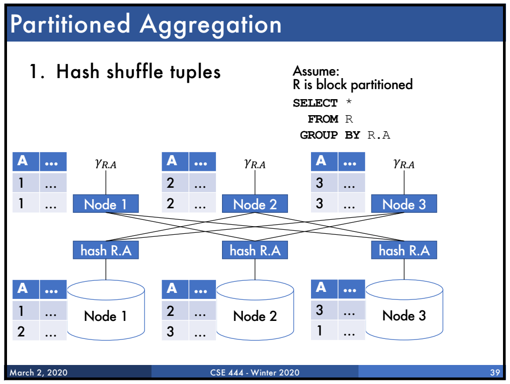
1. Hash shuffle tuples

Assume:
R is block partitioned

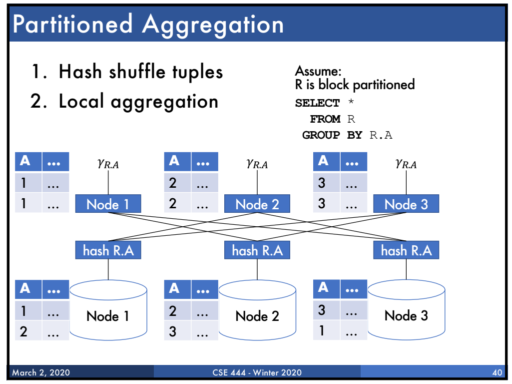
```
SELECT *
  FROM R
 GROUP BY R.A
```



38



39



40

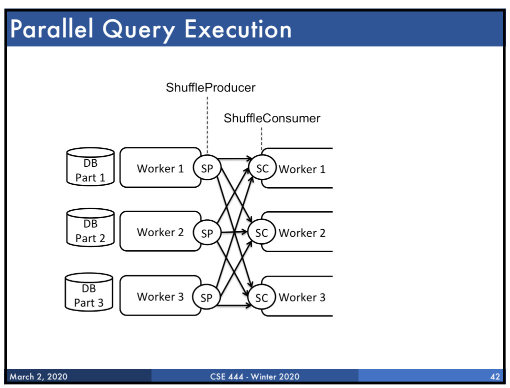
Parallel Query Evaluation

New operator: Shuffle

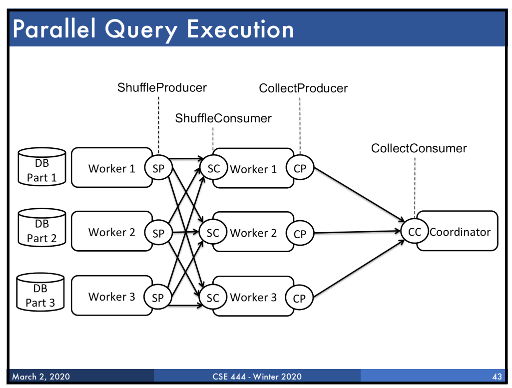
- Serves to re-shuffle data between processes
  - Handles data routing, buffering, and flow control
- Two parts: ShuffleProducer and ShuffleConsumer
- Producer:
  - Pulls data from child operator and sends to n consumers
  - Producer acts as driver for operators below it in query plan
- Consumer:
  - Buffers input data from n producers and makes it available to operator through getNext() interface

March 2, 2020 CSE 444 - Winter 2020 41

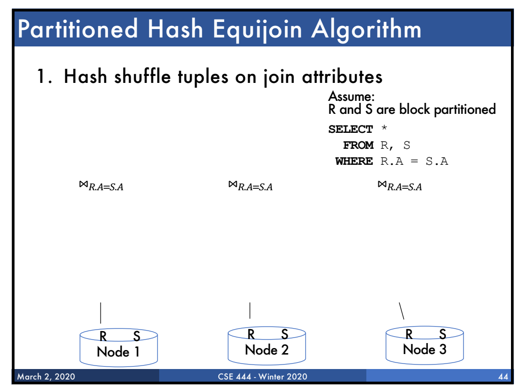
41



42



43



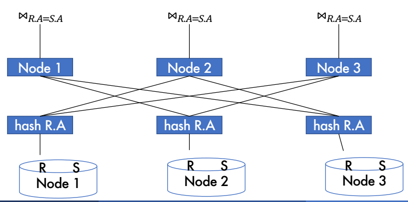
44

Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes

Assume:
R and S are block partitioned

```
SELECT *
  FROM R, S
 WHERE R.A = S.A
```



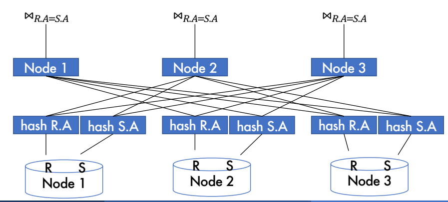
45

Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes

Assume:
R and S are block partitioned

```
SELECT *
  FROM R, S
 WHERE R.A = S.A
```



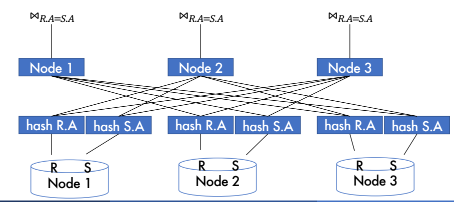
46

Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes
2. Local join

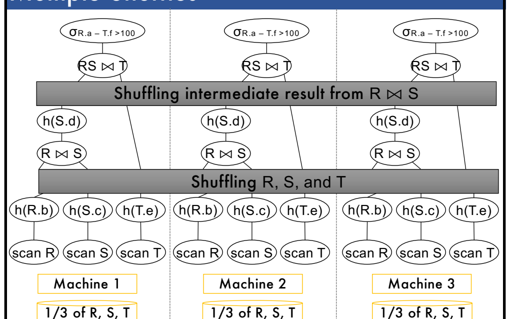
Assume:
R and S are block partitioned

```
SELECT *
  FROM R, S
 WHERE R.A = S.A
```



47

Multiple Shuffles



48

Summary

- With one new operator, we've made SimpleDB an OLAP-ready parallel DBMS!
- Next lecture:
  - Skew handling
  - Algorithm refinements

49

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - 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?

50

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - 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 ½ as many chunks)
- If we double both P and the size of R, what is the new running time?

51

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - 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 ½ 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)

52

Basic Parallel GroupBy

Can we do better?

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

53

Basic Parallel GroupBy

Can we do better?

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

Distributive	Algebraic	Holistic
$sum(a_1+a_2+...+a_9)= sum(sum(a_1+a_2+a_3)+ sum(a_4+a_5+a_6)+ sum(a_7+a_8+a_9))$	avg(B) = sum(B)/count(B)	median(B)

54

Basic Parallel GroupBy

Can we do better?

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

Distributive	Algebraic	Holistic
$sum(a_1+a_2+...+a_9)= sum(sum(a_1+a_2+a_3)+ sum(a_4+a_5+a_6)+ sum(a_7+a_8+a_9))$	avg(B) = sum(B)/count(B)	median(B)

YES

- Compute partial aggregates before shuffling

55

Basic Parallel GroupBy

Can we do better?

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

Distributive	Algebraic	Holistic
$sum(a_1+a_2+...+a_9)= sum(sum(a_1+a_2+a_3)+ sum(a_4+a_5+a_6)+ sum(a_7+a_8+a_9))$	avg(B) = sum(B)/count(B)	median(B)

YES

- Compute partial aggregates before shuffling

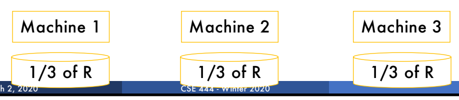
MapReduce implements this as “Combiners”

56

Exercise (www.draw.io is fast!)

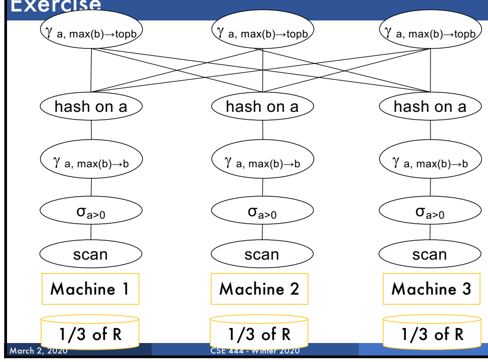
Example Query with Group By

SELECT a, max(b) as topb
FROM R WHERE a > 0
GROUP BY a



57

Exercise



58

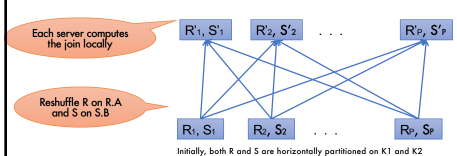
Parallel Join: R $\bowtie_{A=B}$ S

- Data: R(<u>K1</u>,A, C), S(<u>K2</u>, B, D)
- Query: R(<u>K1</u>,A,C) ⋈ S(<u>K2</u>,B,D)

59

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

- Data: R(<u>K1</u>,A, C), S(<u>K2</u>, B, D)
- Query: R(<u>K1</u>,A,C) ⋈ S(<u>K2</u>,B,D)



62

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

63

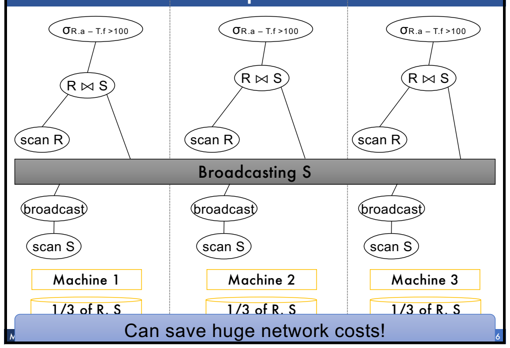
Optimization for Small Relations

When joining R and S

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

65

Broadcast Join Example



66

Justin Biebers Re-visited

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

67

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

68

Some Skew Handling Techniques

Create more partitions than nodes

- And be smart about scheduling the partitions
  - E.g. One node ONLY does Justin Biebers
- Note: MapReduce uses this technique

69

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

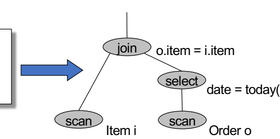
70

Example: Teradata – Query Execution

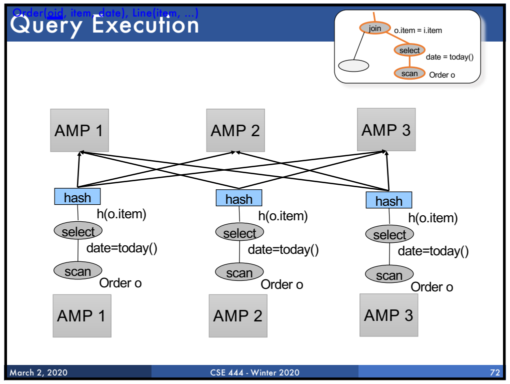
Order(oid, item, date), Line(item, ...)

Find all orders from today, along with the items ordered

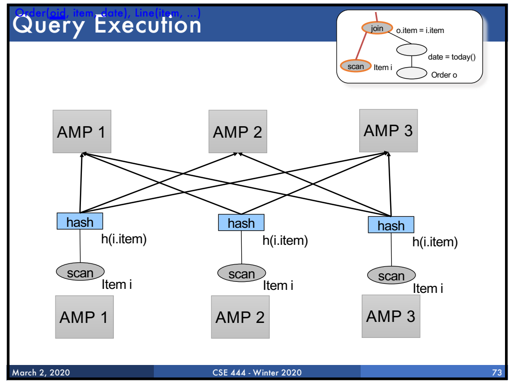
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



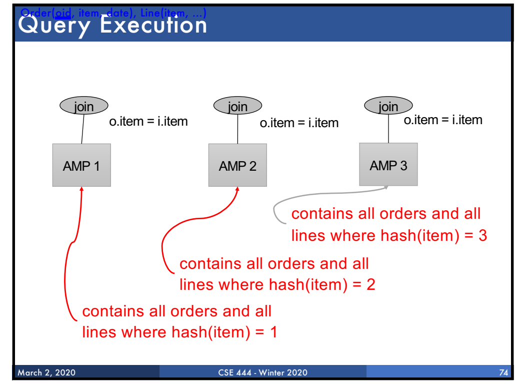
71



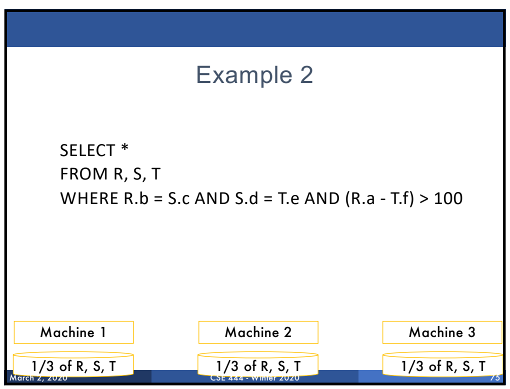
72



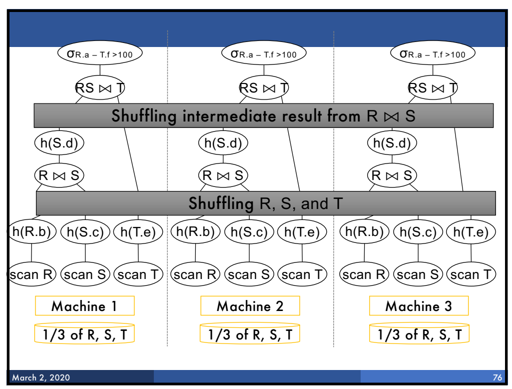
73



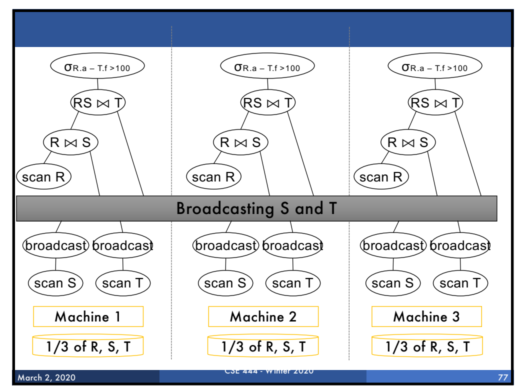
74



75



76



77