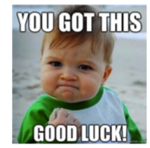
Introduction to Database Systems
CSE 414

Lecture 16: Query Evaluation

Announcements

- HW5 + WQ5 due tomorrow
- Midterm this Friday in class!
  - Review session this Wednesday evening
  - See course website
- HW6 will be released later this week
  - Due on Friday 5/11
  - No WQ6 (yet)!

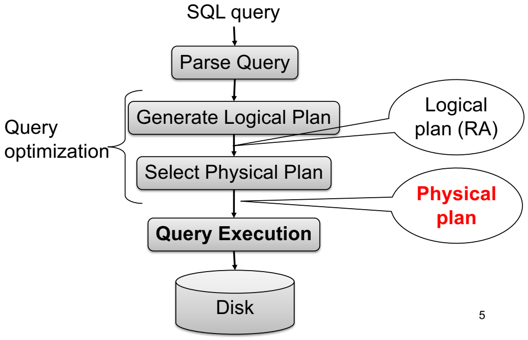


Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
- Unit 3: Non-relational data
- Unit 4: RDMBS internals and parallel query processing
- Unit 5: DBMS usability, conceptual design
- Unit 6: Transactions
- Unit 7: Advanced topics

From Logical RA Plans to Physical Plans

Query Evaluation Steps Review



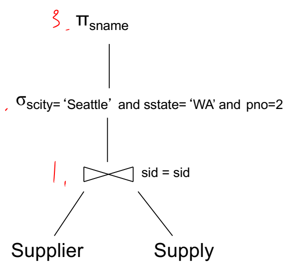
Logical vs Physical Plans

- Logical plans:
  - Created by the parser from the input SQL text
  - Expressed as a relational algebra tree
  - Each SQL query has many possible logical plans
- Physical plans:
  - Goal is to choose an efficient implementation for each operator in the RA tree
  - Each logical plan has many possible physical plans

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Review: Relational Algebra

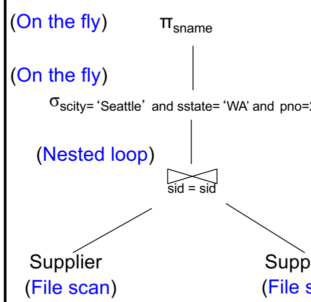
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



Relational algebra expression is also called the "logical query plan"


```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Physical Query Plan 1



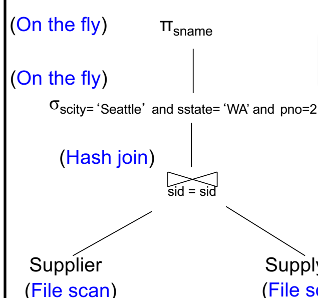
A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Physical Query Plan 2



Same logical query plan
Different physical plan

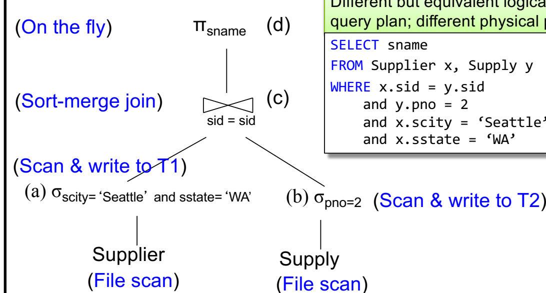
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Physical Query Plan 3

Different but equivalent logical query plan; different physical plan




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```


Query Optimization Problem

- For each SQL query… many logical plans
- For each logical plan… many physical plans
- Choosing the best one among them is the goal of *query optimization*
- More on this later in the quarter

Distributed query processing

Why compute in parallel?

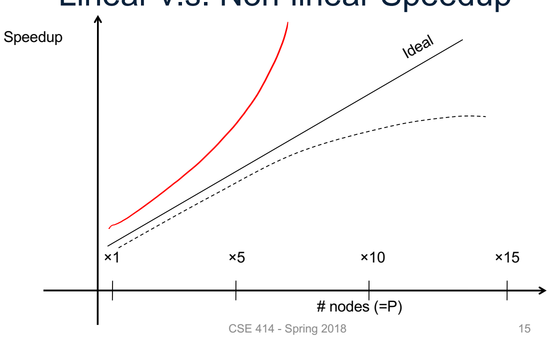
- Multi-cores:
  - Most processors have multiple cores
  - This trend will likely increase in the future
- Big data: too large to fit in main memory
  - Distributed query processing on 100x-1000x servers
  - Widely available now using cloud services
  - Recall HW3 and motivation for NoSQL!

Performance Metrics for Parallel DBMSs

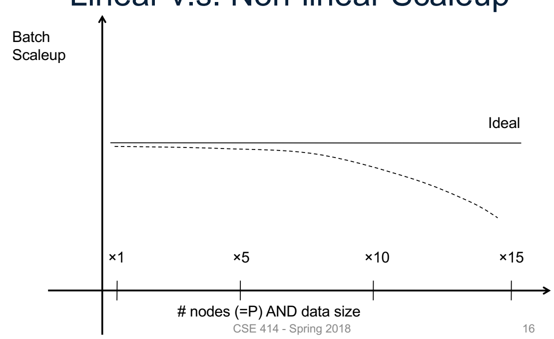
Nodes = processors, computers

- Speedup:
  - More nodes, same data ➔ higher speed
- Scaleup:
  - More nodes, more data ➔ same speed

Linear v.s. Non-linear Speedup



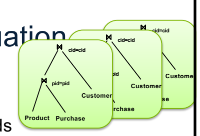
Linear v.s. Non-linear Scaleup



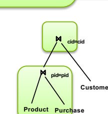
Why Sub-linear Speedup and Scaleup?

- Startup cost
  - Cost of starting an operation on many nodes
- Interference
  - Contention for resources between nodes
- Skew
  - Slowest node becomes the bottleneck

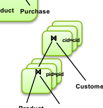
Approaches to Parallel Query Evaluation



- Inter-query parallelism
  - One query per node
  - Good for transactional (OLTP) workloads



- Inter-operator parallelism
  - Operator per node
  - Good for analytical (OLAP) workloads



- Intra-operator parallelism
  - Operator on multiple nodes
  - Good for both?

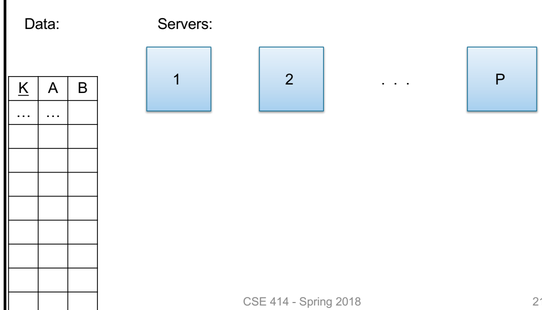
We study only intra-operator parallelism: most scalable

Parallel Data Processing in the 20th Century

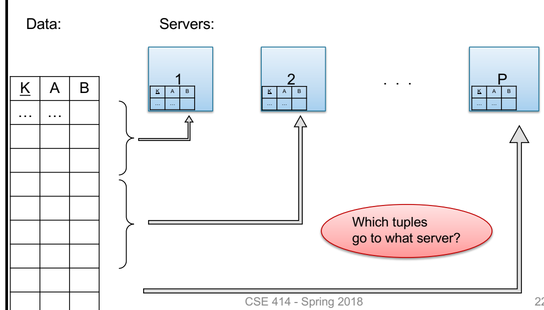
Let's parallelize RDBMS

- Data is horizontally partitioned on many servers
- Operators may require data reshuffling
- First let's discuss how to distribute data across multiple nodes / servers

Horizontal Data Partitioning



Horizontal Data Partitioning



Recall: Horizontal Data Partitioning

- Block Partition:
  - Partition tuples arbitrarily s.t. $size(R_1) \approx \ldots \approx size(R_P)$
- Hash partitioned on attribute A:
  - Tuple t goes to chunk i, where i = h(t.A) mod P + 1
  - Recall: calling hash fn's is free in this class
- Range partitioned on attribute A:
  - Partition the range of A into $-\infty = v_0 < v_1 < \ldots < v_P = \infty$
  - Tuple t goes to chunk i, if $v_{i-1} < t.A < v_i$

Uniform Data v.s. Skewed Data

- Let R(<u>K</u>,A,B,C); which of the following partition methods may result in skewed partitions?
- Block partition — Uniform
- Hash-partition
  - On the key K — Uniform
  - On the attribute A — May be skewed

Assuming good hash function

E.g. when all records have the same value of the attribute A, then all records end up in the same partition

Keep this in mind in the next few slides

Parallel Execution of RA Operators: Grouping

Data: R(K,A,B,C)
Query: $\gamma_{A,sum(C)}(R)$

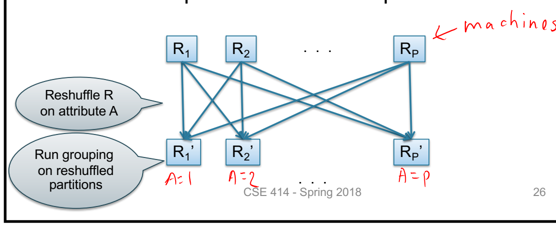
How to compute group by if:

- R is hash-partitioned on A ?
- R is block-partitioned ?
- R is hash-partitioned on K ?

Parallel Execution of RA Operators: Grouping

Data: R(K,A,B,C)
Query: $\gamma_{A,sum(C)}(R)$

- R is block-partitioned or hash-partitioned on K



Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - Runtime: only consider I/O costs
- 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)

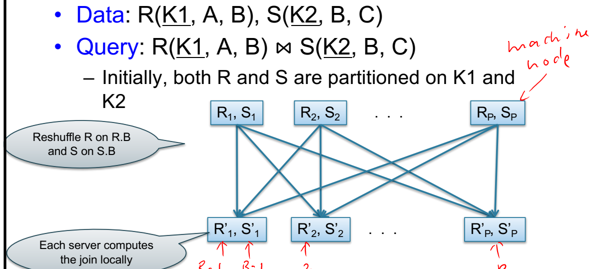
But only if the data is without skew!

Skewed Data

- R(K,A,B,C)
- Informally: we say that the data is skewed if one server holds much more data that the average
- E.g., we hash-partition on A, and some value of A occurs very many times ("Justin Bieber")
- Then the server holding that value will be skewed

Parallel Execution of RA Operators: Partitioned Hash-Join

- Data: R(K1, A, B), S(K2, B, C)
- Query: R(K1, A, B) ⋈ S(K2, B, C)
  - Initially, both R and S are partitioned on K1 and K2



Data: R(K1,A, B), S(K2, B, C)
Query: R(K1,A,B) ⋈ S(K2,B,C)

Parallel Join Illustration

Partition

M1: R1

K1	B
1	20
2	50

M1: S1

K2	B
101	50
102	50

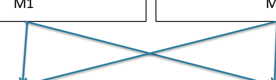
M2: R2

K1	B
3	20
4	20

M2: S2

K2	B
201	20
202	50

Shuffle on B



Local Join

M1: R1' ⋈ S1'

K1	B
1	20
3	20
4	20

K2	B
201	20

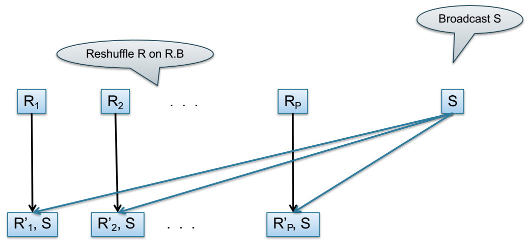
M2: R2' ⋈ S2'

K1	B
2	50

K2	B
101	50
102	50
202	50

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

Broadcast Join

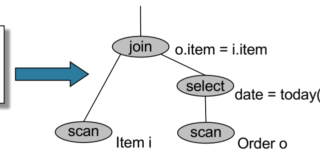


Why would you want to do this?

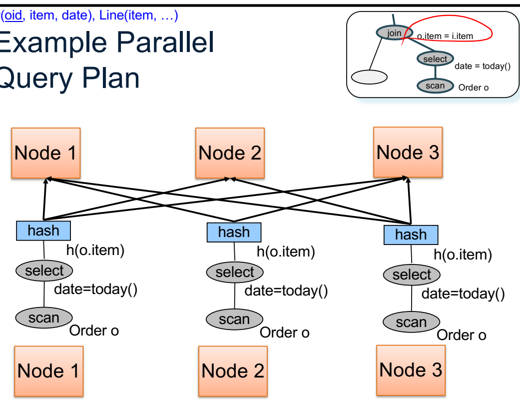
Putting it Together: Example Parallel Query Plan

Find all orders from today, along with the items ordered

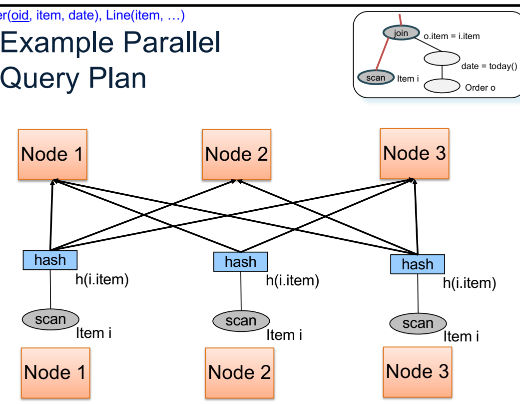
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



Example Parallel Query Plan



Example Parallel Query Plan



Example Parallel Query Plan