CSE 414: Section 7
Parallel Databases

November 8th, 2018

Agenda for Today

This section:

- Quick touch up on parallel databases
-

Distributed Query Processing

In this class, only **shared-nothing architecture** and **intra-operator parallelism**

Horizontal Data Partitioning:

- Block Partition
- Hash partitioned on attribute A
- Range partitioned on attribute A

Distributed Query Processing

In this class, only **shared-nothing architecture** and **intra-operator parallelism**

Horizontal Data Partitioning:

WTH?

- Block Partition
- Hash partitioned on attribute A
- Range partitioned on attribute A

Distributed Query Processing

In this class, only **shared-nothing architecture** and **intra-operator parallelism**

Horizontal Data Partitioning:

- Block Partition
- Hash partitioned on attribute A
- Range partitioned on attribute A

WTH?

"Our processor/storage nodes are separate from each other and deal with only one operation at a time. We toss around whole tuples."

Moving Data

We have a “network layer” to move tuples temporarily between nodes.

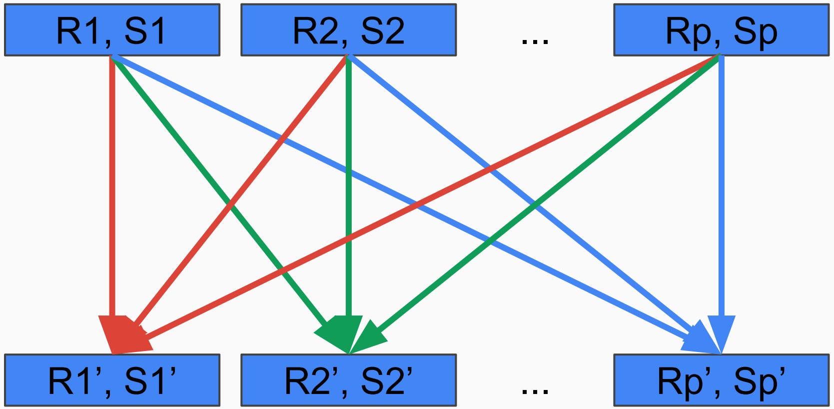
Transferring data is expensive so we need to be efficient (especially on joins and grouping).

Moving Data:
Partitioned Hash-Join Mechanism

We have p machines

We wish to join on some attribute (say R.x and S.y)

Call our hash function h(z)



Contains tuples s.t. h(R.x) = h(S.y) = red

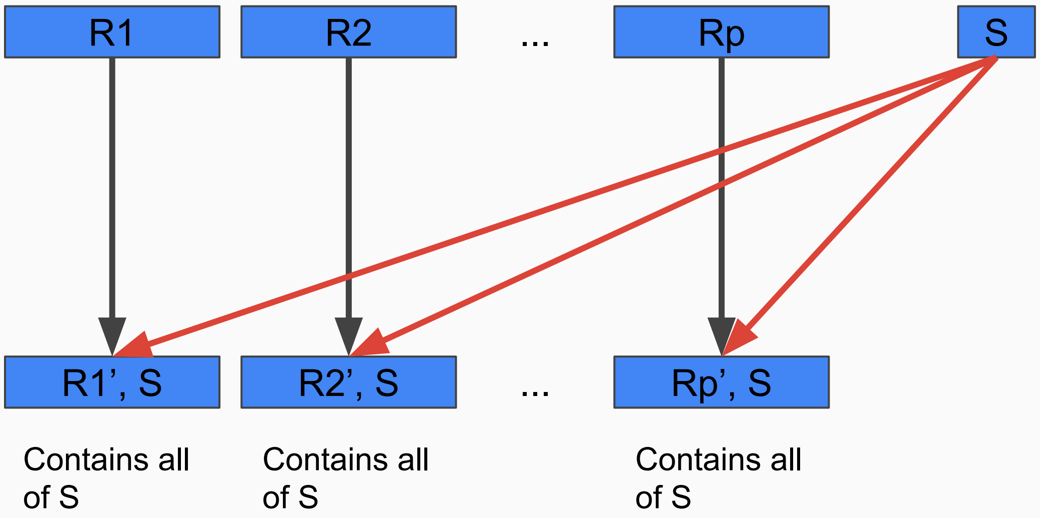
Contains tuples s.t. h(R.x) = h(S.y) = green

Contains tuples s.t. h(R.x) = h(S.y) = blue

Moving Data: Broadcast Join (Map-Side Join) Mechanism

We want to think about how to prevent sending all data through the network.

Take advantage of small datasets (meaning the whole dataset can fit into main memory)

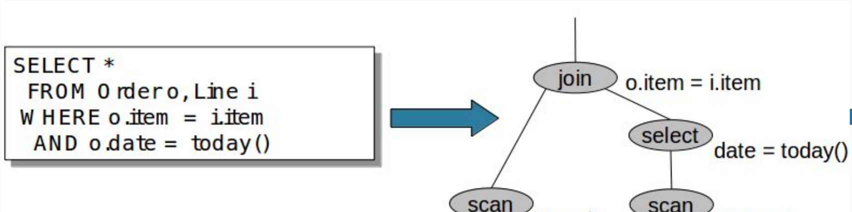


Now What?

“Cool. I know how to split data up and move it around efficiently. What does that have to do with my queries?”

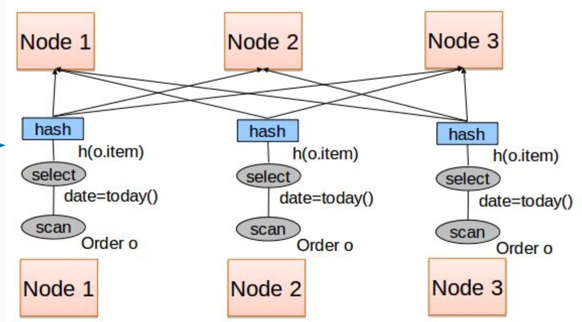
Now What?

“Cool. I know how to split data up and move it around efficiently. What does that have to do with my queries?”



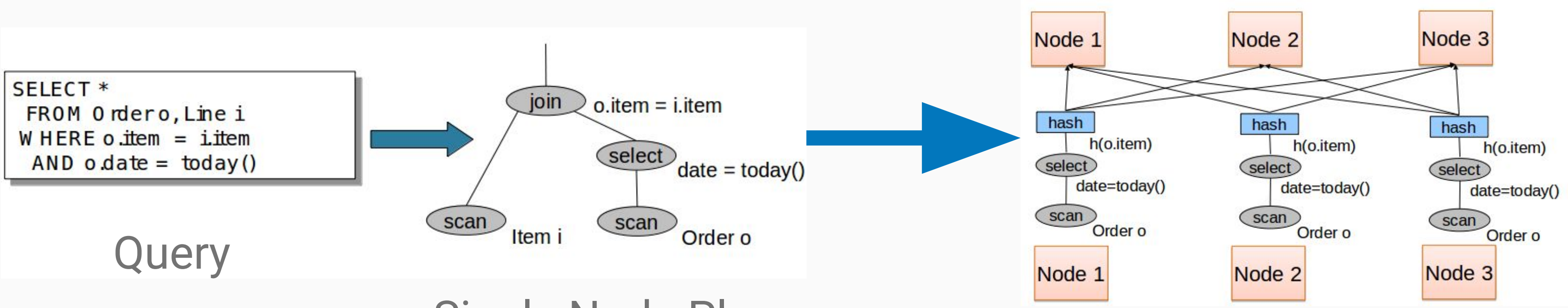
Query

Single Node Plan



Multi-Node Plan

Parallel Query Plans

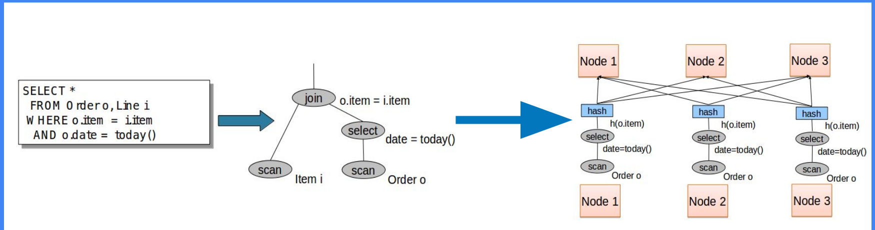


Single Node Plan

Multi-Node Plan

Know how to derive parallel plans though this pipeline.

Parallel Query Plans



Know how to derive parallel plans from your single node plans.

- Which RA operations can you do without talking to other nodes?
- Which RA operations require moving tuples?
- Can we take advantage of how our data is already stored? (partitioning)

$\bowtie$ σ π

Parallel DB Practice!

We have a distributed database that hold the relations:
Drug(spec VARCHAR(255), compatibility INT)
Person(name VARCHAR(100) PK, compatibility INT)

We want to compute: *This is a pretty hard question*

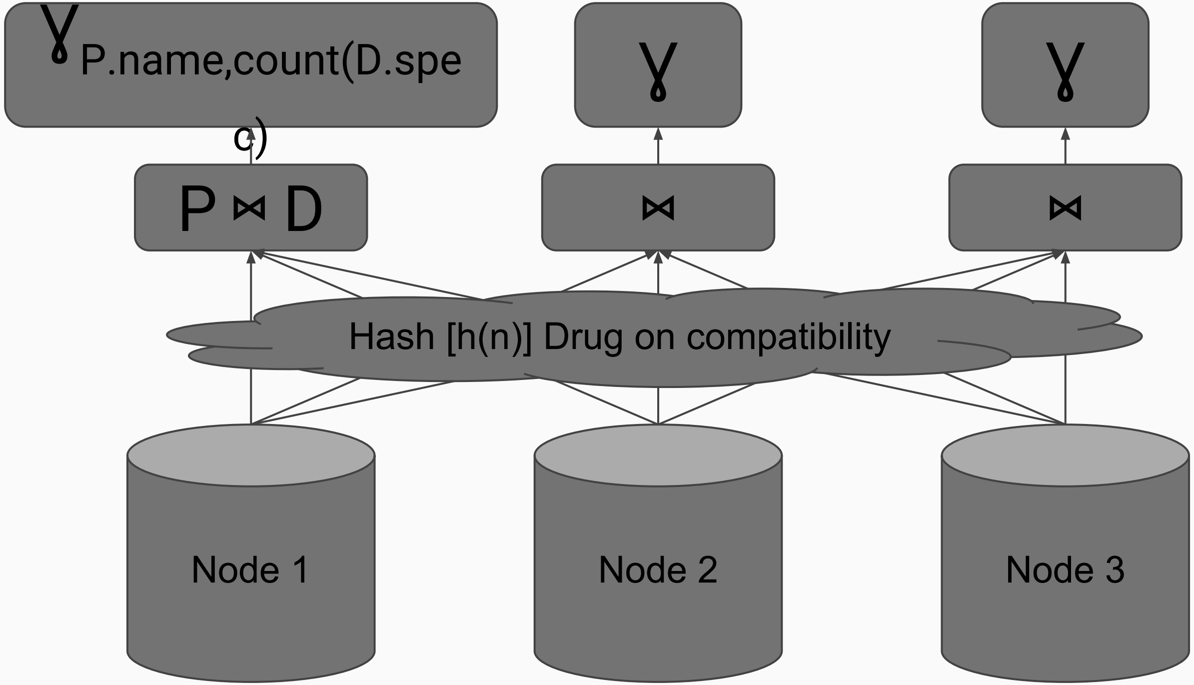
```
SELECT P.name, count(D.spec)
  FROM Person AS P, Drug AS D
 WHERE P.compatibility = D.compatibility
 GROUP BY P.name;
```

Drug is block-partitioned
Person is hash-partitioned on compatibility [h(n)]
You have three nodes. **Draw a parallel query plan.**

$\gamma_{P.name,\ count(D.spec)}(P \bowtie D)$

Take advantage of:

1. Hash partitioning of [h(n)]
2. The PK uniqueness of name



Apache

Cluster-computing framework

Apache Hadoop Mapreduce vs. Apache Spark

https://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html

“Hadoop MapReduce”

Distributed File System (DFS) -> Hadoop Distributed File System (HDFS)

MapReduce Job:

- Map Task (EmitIntermediate)
- Reduce Task (Emit)

Fault Tolerance (replicated chunks, write intermediate files to disk)

Word Counting in MapReduce

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
    emitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    emit(AsString(result));
```

“Spark” (HW6)

Resilient Distributed Datasets (RDD)

High level commands:

- Transformations (map, join, sort...) -> **Lazy**
- Actions (count, reduce, save...) -> **Eager**

Fault Tolerance (main memory and lineage)

Spark Objects for HW6

Row

RowFactory.create(Objects...)

Dataset<Row>

JavaRDD<Row>

JavaPairRDD<K, V>

Tuple2<> you can leave the generics empty

Spark Methods for HW6

`spark.sql("SELECT ... FROM ...")` spark must be a SparkSession

`d.filter(t -> f(t) == true/false)`

`d.distinct()`

`d.map()` d must be a JavaRDD

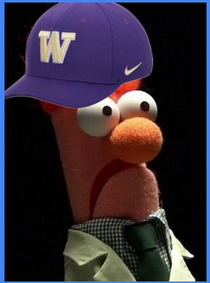
`d.mapToPair(t -> new Tuple2<>(K, V))`

`d.reduceByKey((v1, v2) -> f(v1, v2))` d must be a JavaPairRDD


MIDTERM IS TOMORROW!!!

(you'll be fine)





About ~~Midterms~~ Celebrations of Knowledge

Understand content in the lecture slides

Look at previous tests to try problems
(we use a pretty standard format for questions)

Tests are usually pretty long so don't feel obligated to complete everything

This quarter's materials are in a different order than other quarters