LECTURE 22

Parallel and Distributed Computing

April 15, 2024

Data 101/Info 258, Spring 2024 @ UC Berkeley

Aditya Parameswaran https://data101.org/sp24

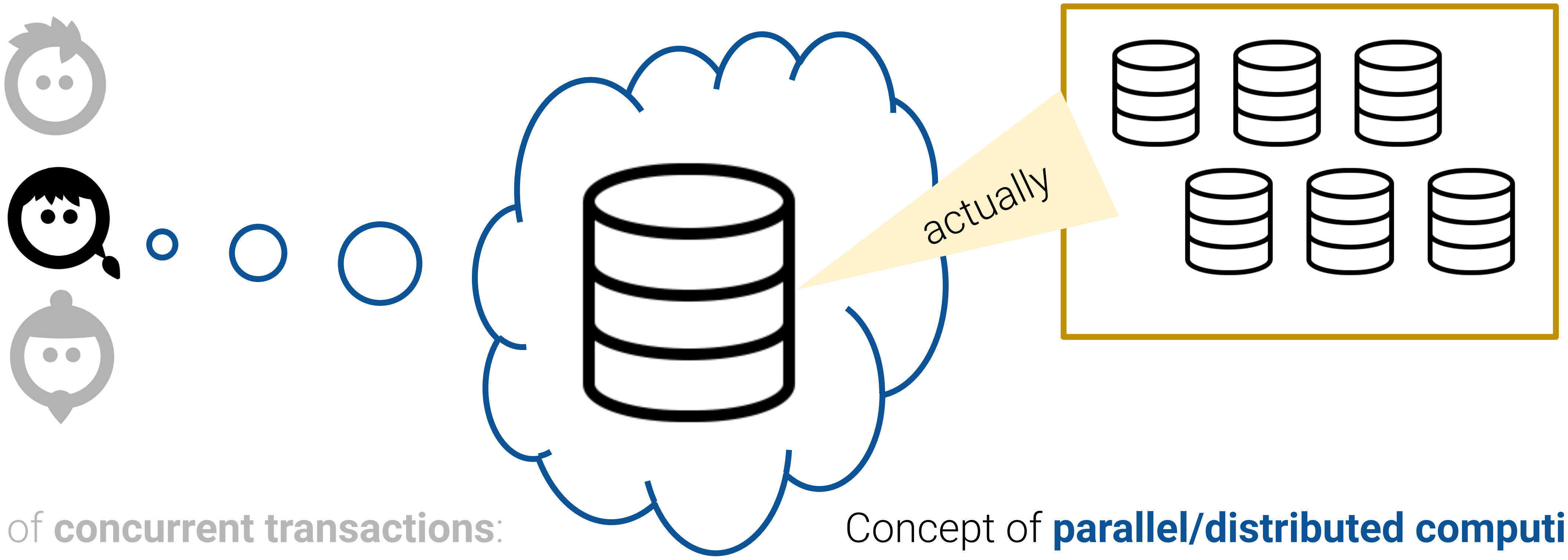
From Concurrency to Distributed Computing

Concept of **concurrent transactions**:

- **Multiple users** using the DBMS in parallel
- Multiple users appear to be working concurrently (interleaved transaction schedules)

(We have just done this)

From Concurrency to **Distributed Computing**



Concept of **concurrent transactions**:

- Multiple users using the DBMS in parallel
- Multiple users appear to be working concurrently (interleaved transaction schedules)

(We have just done this)

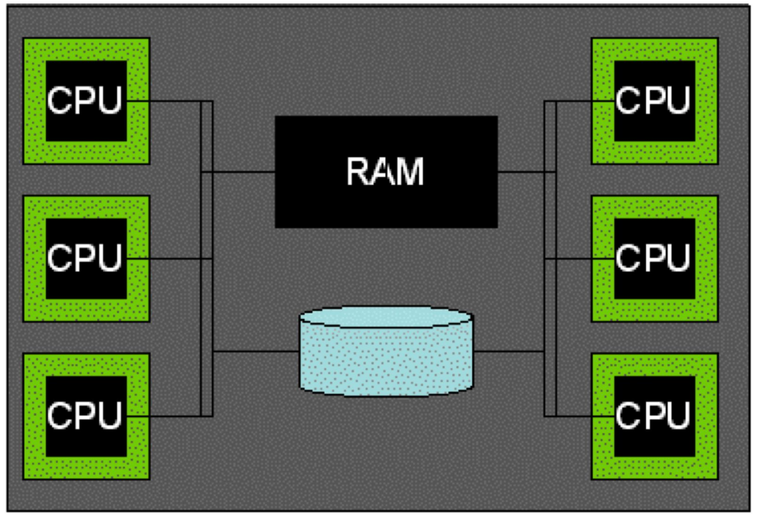
Concept of **parallel/distributed computing**:

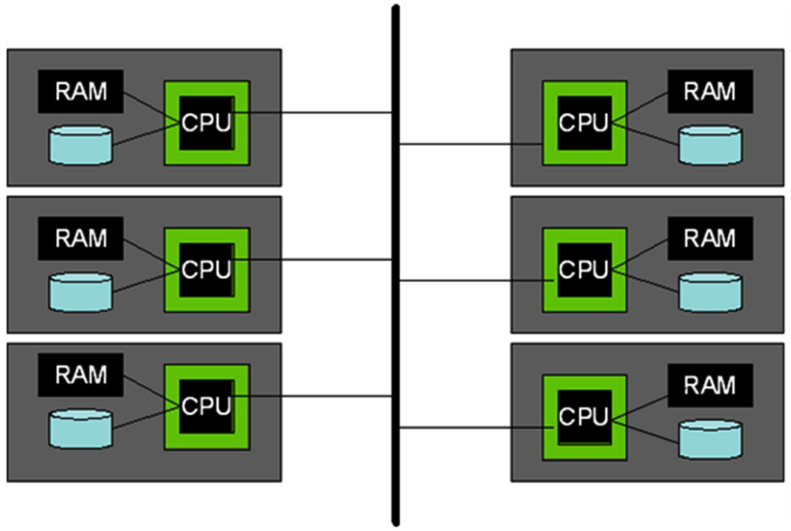
- **Multiple computing resources** in parallel achieving the goals of a **single user**
- DBMS appears to operate as a single machine, despite the parallel architecture.

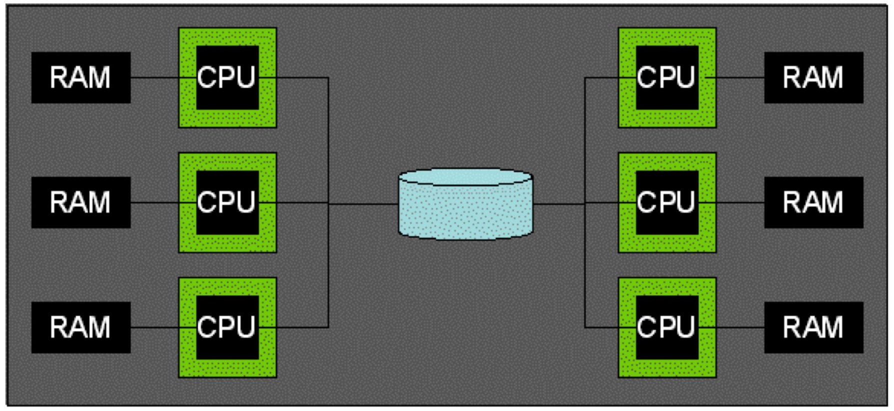
(we are now going to do this)

Focus on relational setting, but also applies to semi-structured data!

What are some Parallel DBMSs/Parallel Computation Architectures?







Shared Memory

Multiple cores accessing shared memory and disk

(This is what you get when you reserve a "beefy machine")

Shared Nothing

Multiple commodity nodes each with their own memory and disk

- A manager node will coordinate with worker nodes
- Each worker responsible for a piece of the data

Shared Disk

Worker nodes have own memory and may have local disk as a cache

Each may be "responsible" for a piece of data from shared disk.

- Re-emerging as a promising alternative in the cloud era
- Example: object store / S3

Parallel Data Processing Considerations

Data Partitioning

- Where and how do we break up the datasets into pieces across many machines to be operated on in parallel?

Parallel Data Processing

- Given that the datasets have been broken up, what parallel computation alternatives exist?
- Latency hierarchy: reading from memory << from disk << across the network

Hardware edge cases

- How do we deal with failures and stragglers?
- If we have 1000 servers and a server dies once every 100 days, then on average 10 will die per day.

We'll only get to discuss the first two items (partitioning, processing). But know that physical restrictions are the most important in daily operation!

Partitioning Strategies

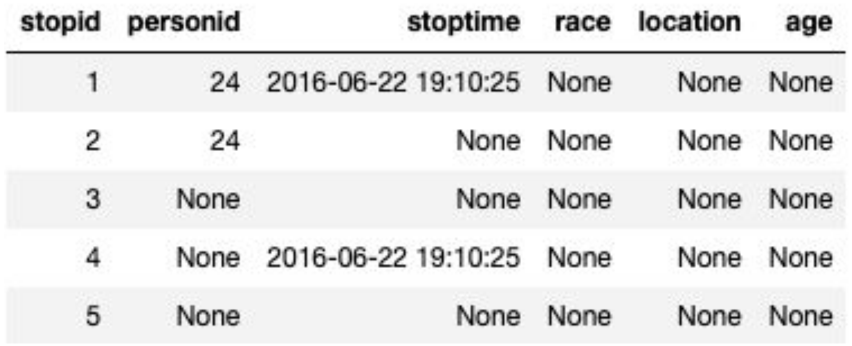
Lecture 22, Data 101/Info 258 Spring 2024

Partitioning Approaches

How we partition our data depends on the data model itself.

Vertical partitioning:

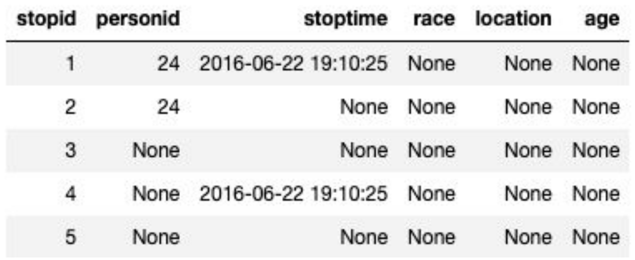
- By columns (features)
- Natural in column stores,
  but not particularly generalizable



stopid	personid	stoptime	race	location	age
1	24	2016-06-22 19:10:25	None	None	None
2	24	None	None	None	None
3	None	None	None	None	None
4	None	2016-06-22 19:10:25	None	None	None
5	None	None	None	None	None

Horizontal partitioning:

- By records (documents, rows, etc.)
- Works for both relations/column stores
  and document stores!

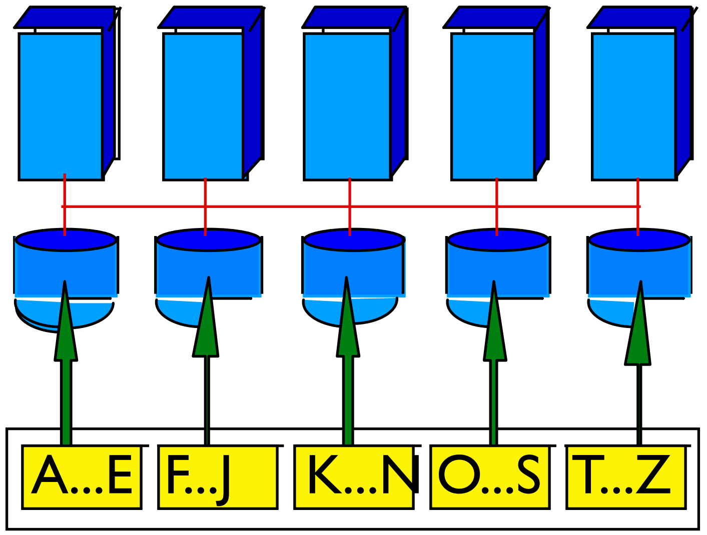


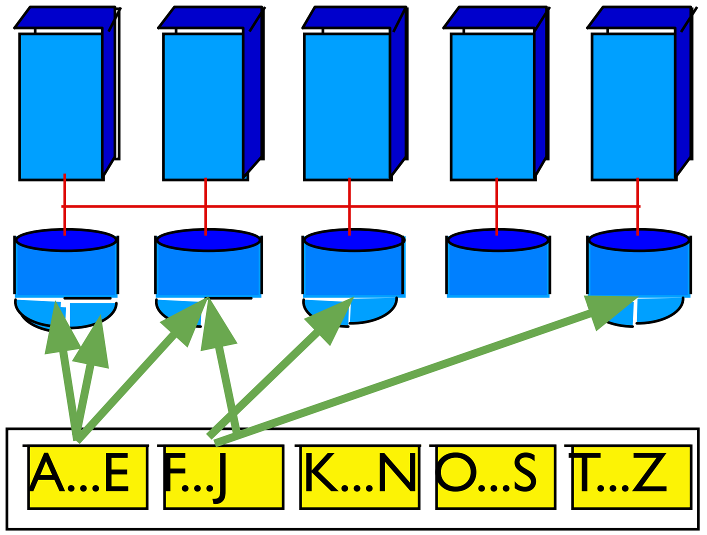
stopid	personid	stoptime	race	location	age
1	24	2016-06-22 19:10:25	None	None	None
2	24	None	None	None	None
3	None	None	None	None	None
4	None	2016-06-22 19:10:25	None	None	None
5	None	None	None	None	None

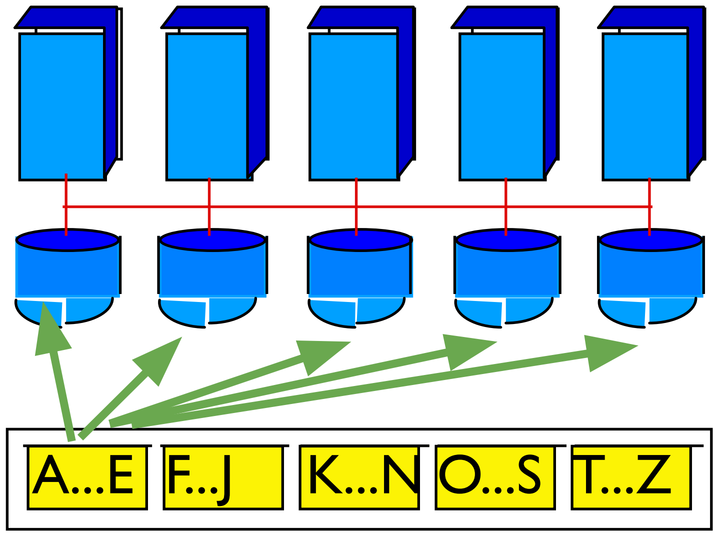
```
{
  "ID": "22222",
  "name": {
    "firstname": "Albert",
    "lastname": "Einstein"
  },
  "deptname": "Physics",
  "children": [
    {
      "firstname": "Hans",
      "lastname": "Einstein"
    },
    {
      "firstname": "Eduard",
      "lastname": "Einstein"
    }
  ]
}
```

We focus mostly on **horizontal partitioning**, as that is the more generalizable strategy.

Three (Horizontal) Partitioning Approaches

1.


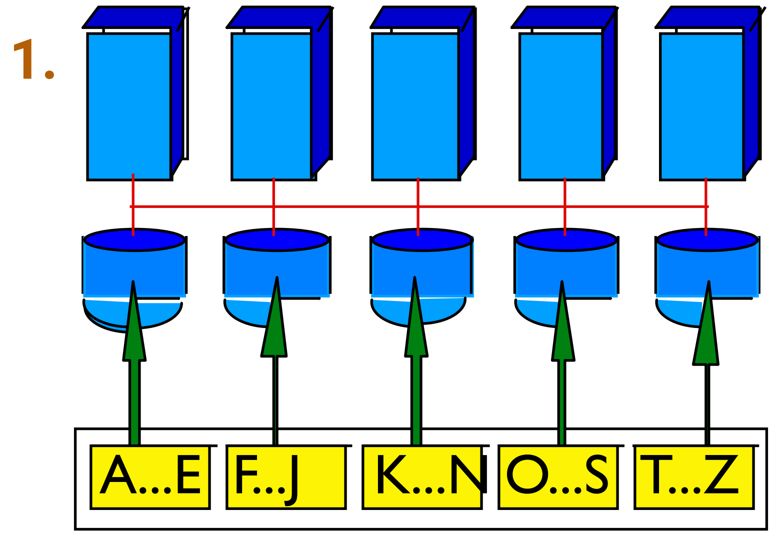
2.


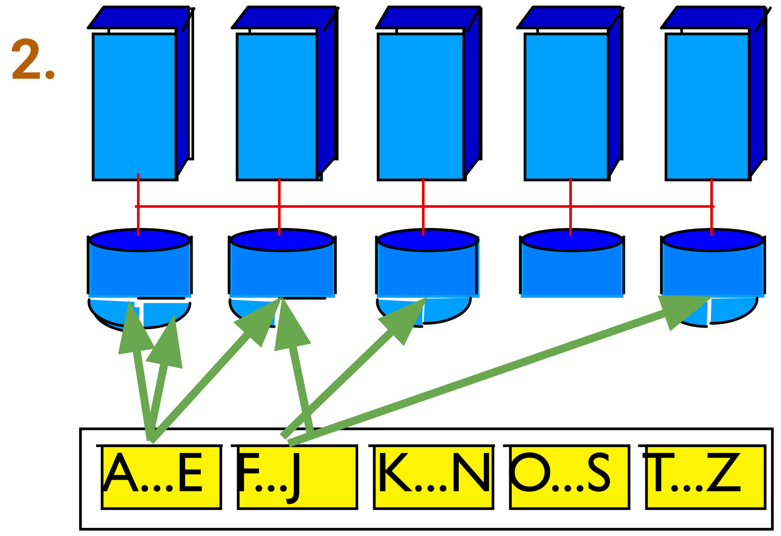
3.


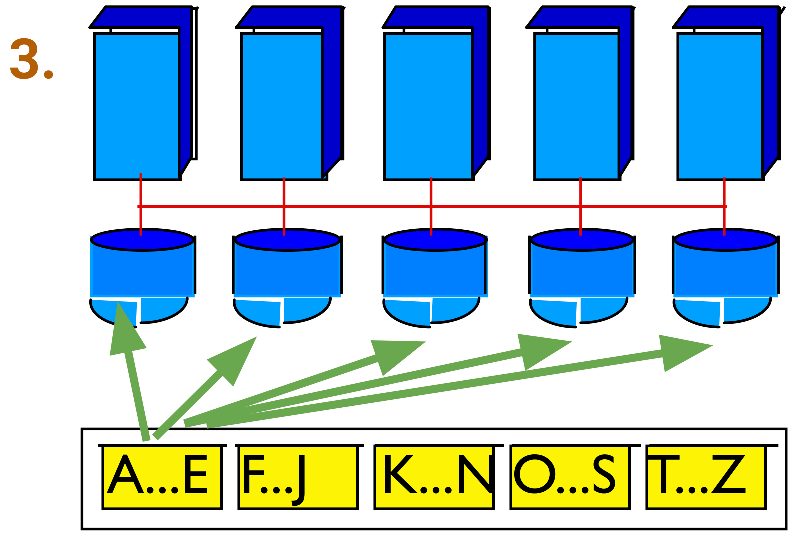
Which partitioning approach matches to each diagram?

A. Range-based partitioning
B. Hash-based partitioning
C. Round-robin partitioning

Three (Horizontal) Partitioning Approaches







A. Range-based partitioning

Pick processor `j` for tuple `i` if its value for some field is within range `i`.

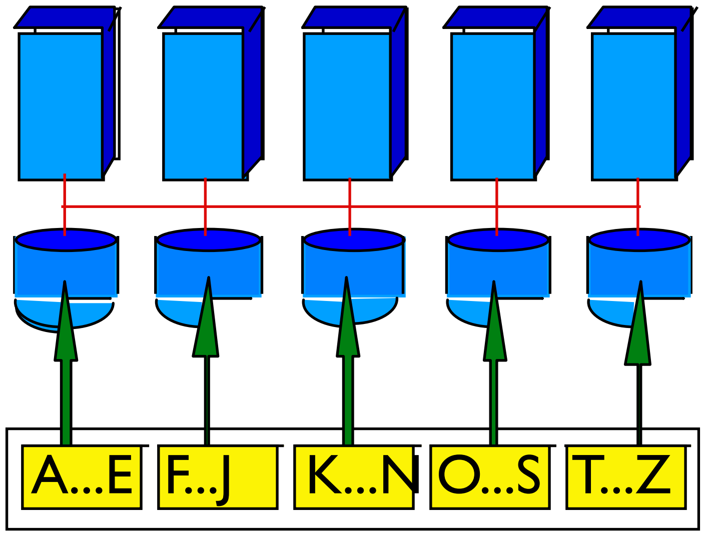
B. Hash-based partitioning

Pick a field. Pick processor for a tuple by hashing with this field.

C. Round-robin partitioning

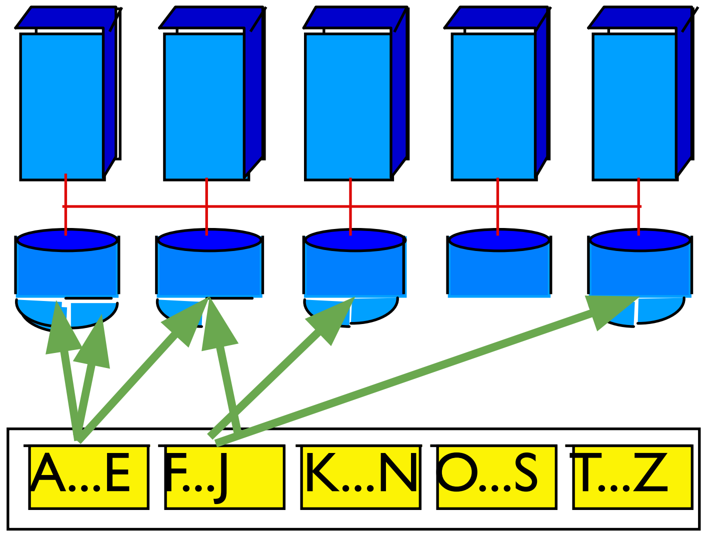
Hash tuple `i` to processor `i % n`

Three (Horizontal) Partitioning Approaches

1.


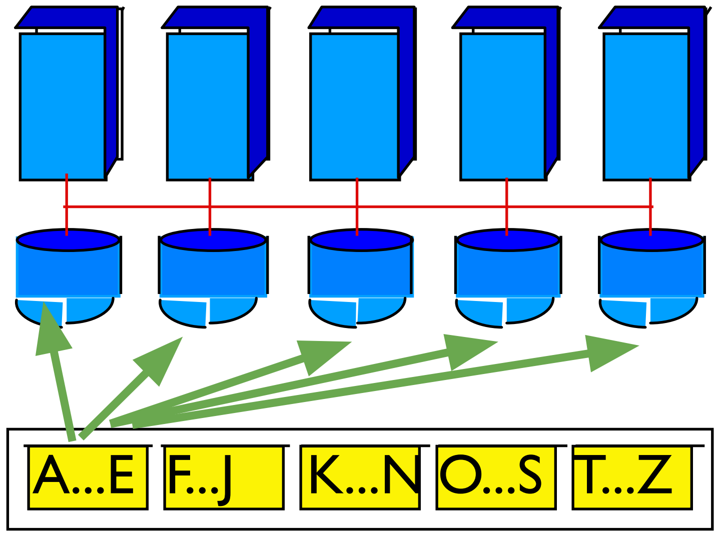
Range-based partitioning

Pick processor `j` for tuple `i` if its value for some field is within range `i`.

2.


Hash-based partitioning

Pick a field. Pick processor for a tuple by hashing with this field.

3.


Round-robin partitioning

Hash tuple `i` to processor `i % n`

Also: What do we partition on? Think of partitioning as coarse-grained indexing.

- So we can partition on records that hold specific values in "important" columns.
  - e.g., those used in WHERE clauses, PK/FK, etc.

Each approach has tradeoffs!
We'll discuss one: **skew**.

Skew

Skew is a common problem when dealing with parallel processing across many machines.

Reason 1: **Uneven data distribution**

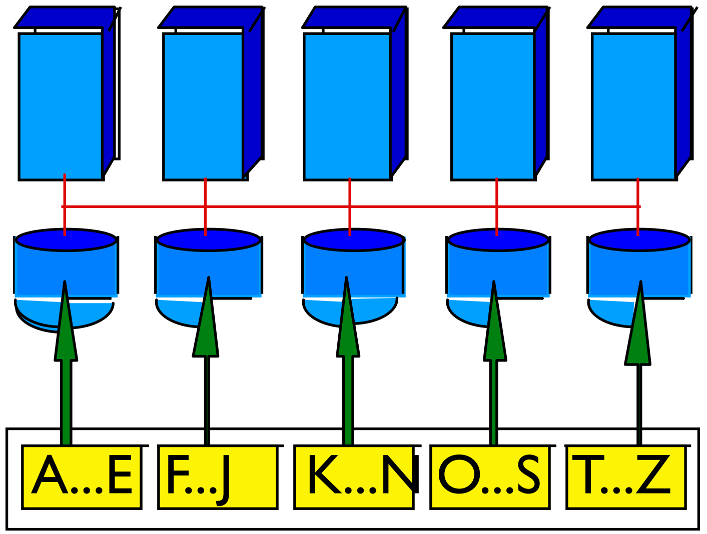
- R has many tuples with value of A = a.
- Example: UC Berkeley students partitioned by state of residency, where many State = CA

Reason 2: **Uneven access patterns**

- Tuples in R corresponding to A = a are more frequently accessed than others.
- Example: more recent data is more frequently accessed.

Three (Horizontal) Partitioning Approaches and Skew

1.

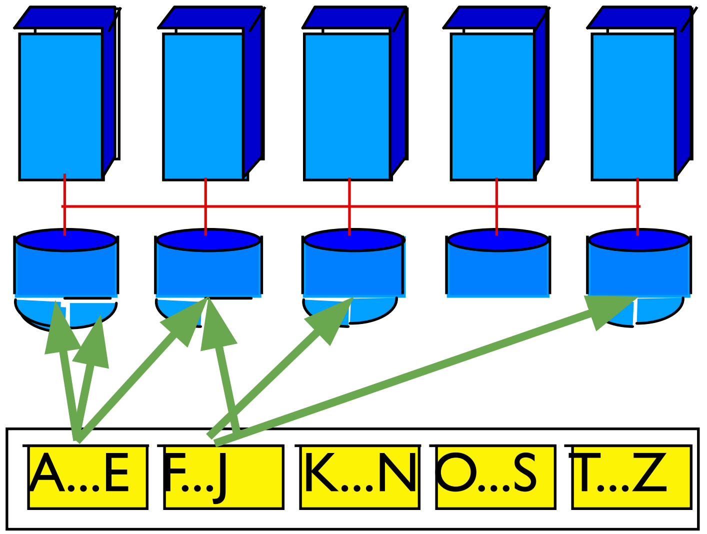


Range-based partitioning

Pick processor `j` for tuple `i` if its value for some field is within range `i`.

Fairly susceptible to skew if there are some partitioning attribute ranges that are very popular (from either standpoint: data or access)

2.

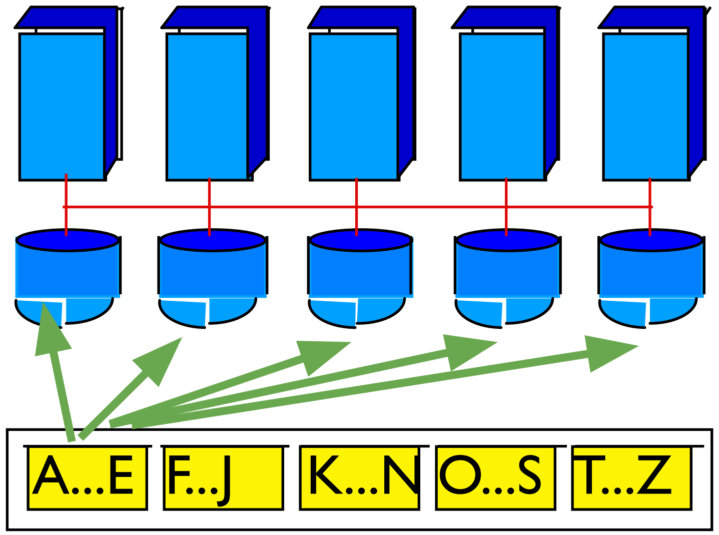


Hash-based partitioning

Pick a field. Pick processor for a tuple by hashing with this field.

Somewhat susceptible to skew if there are some partitioning attribute values that are very popular (from either standpoint: data or access)

3.



Round-robin partitioning

Hash tuple `i` to processor `i % n`

Not susceptible to skew since work is equally divided, but **there is more work** involved since all partitions need to be consulted

Partitioned operations

Data Partitioning

- Where and how do we break up the datasets into pieces across many machines to be operated on in parallel?

Parallel Data Processing

- Given that the datasets have been broken up, what parallel computation alternatives exist?
- Latency hierarchy: reading from memory << from disk << across the network

Hardware edge cases

- How do we deal with failures and stragglers?
- If we have 1000 servers and a server dies once every 100 days, then on average 10 will die per day.

Two types of parallel operations we discuss:

- **Tuple-wise operations**: selection, updates, operation
- **Merge-style operations**: joins, sorts, group/agg

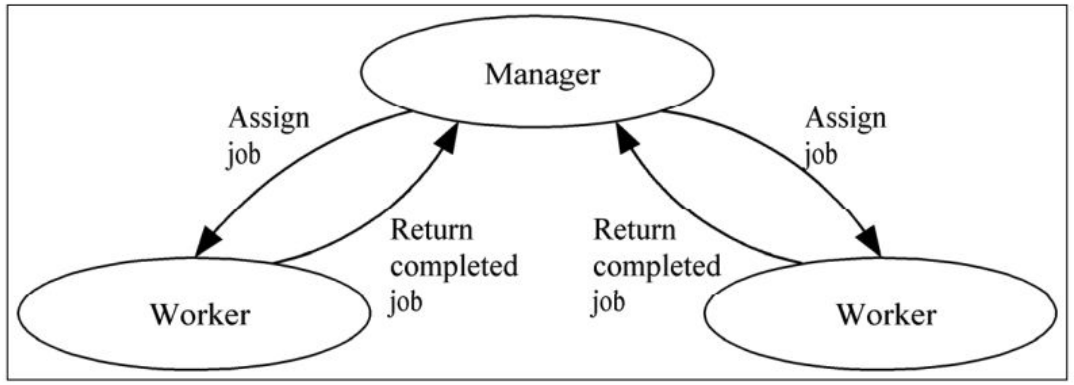
Partitioned Operators and Pipelining

Lecture 22, Data 101/Info 258 Spring 2024

Partitioned Operators, Manager-Worker, and Operation Pipelining

Many parallel operations use **computing nodes** in two ways:

- **Manager**: Delegates work
- **Worker**: Does work and sends back to manager
- Sometimes manager is also a worker



Quinn, *Parallel Programming*, 2008.

Partitioned operators used to speed up a given DB op, where workers all perform the same task in parallel designated by a manager.

Partitioned operators can also be **pipelined**.

- Operators can proceed in parallel reading, applying selection/projection, sending data, etc. (more next time...)
- As opposed to **blocking**, which is waiting for the previous operation to complete in full

Operator Speedup with Partitioning and Pipelining

Scans

- Scan each partition in parallel!
- K nodes implies a K-way speedup
- Parallel speedup! Can also be pipelined!

Selection/Projection

- Pipelined speedup:
  - Apply selection/projection during scanning.
  - If result to be produced at the manager node,
    then less data transferred from the K worker nodes to the manager.
- How does partitioned selection work? (See next example)

Grouped Aggregation, Joins, Sorting

- Partition operation: Yes, we can!
- However, no pipelining speedups. These operations are **blocking**.
  - (esp. some flavors of hash and sort-merge)
  - Essentially need to wait "until the inputs arrive"
  - This is just like the single node setting

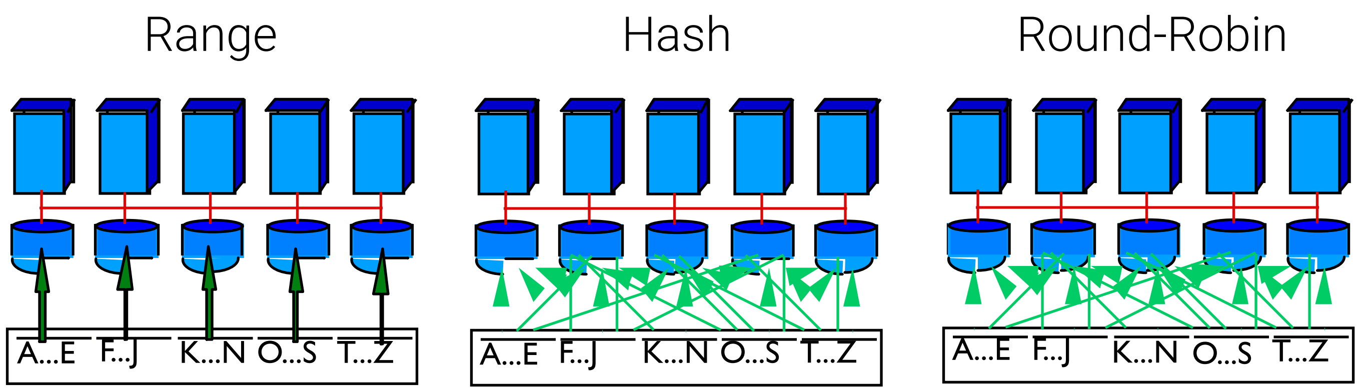
How do Partitioned Operators Work? Selection

Point Selection (i.e., Selection by Equality)   `SELECT * … WHERE Name = 'Shana';`

- Range, Hash:
  - If data partitioned by Name (or function of Name), only access 1 relevant node.
  - Otherwise, route to all nodes.
- Round-robin:
  - Route to all nodes.

Range Selection (i.e., Selection by Equality)   `SELECT * … WHERE Name LIKE 'S%';`

- Range, Hash:
  - If data partitioned by Name (or function of Name), only access relevant node(s).
  - Otherwise, route to all nodes.
- Round-robin:
  - Route to all nodes.


Range
Hash
Round-Robin
A...E F...J K...N O...S T...Z
A...E F...J K...N O...S T...Z
A...E F...J K...N O...S T...Z

How do Partitioned Operators Work? Updates and Insertions

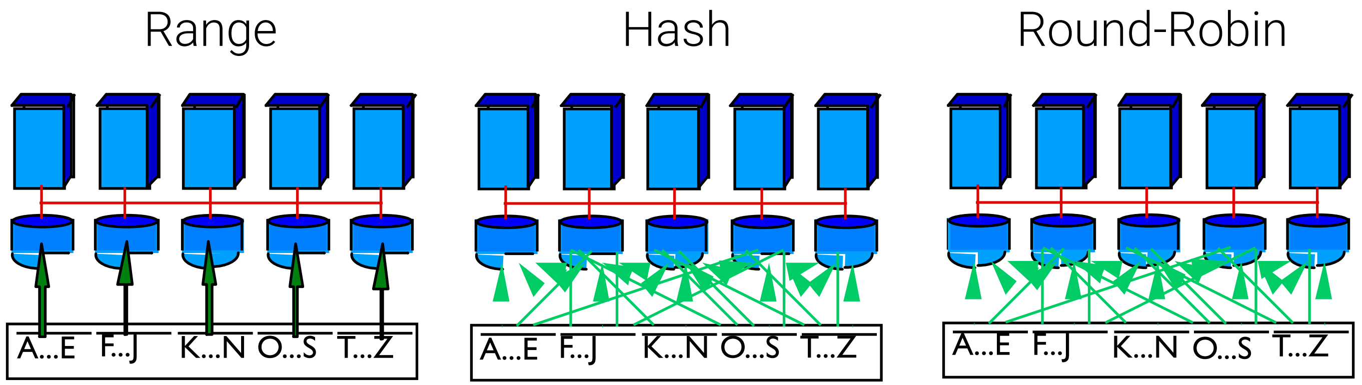
Updates

```
UPDATE … SET … WHERE Name = 'Shana';
```

- Range, Hash:
  - If update based on partitioning field, only access 1 relevant node.
  - Otherwise, route to all nodes.
- Round-robin:
  - Route to all nodes.

Insertion

- Range, Hash:
  - Only access 1 relevant node based on if insertion is with partitioning field.
- Round-robin:
  - Route to the node specified by the round-robin protocol.



From now on, assume the worst case scenario.

Given the conditions on which we partition, it's sometimes easier to assume that **all partitions need to be consulted** on any selection/insertion/update.

…unless **indexes** exist!

If we have indexes, then we look at indexes first, then access data partitions.

- **Global indexes** over the database: Which **partitions** should we look at?
- **Local index per partition**: Which **pages** in the partition should we look at?

Partitioned Joins and Aggregations

Lecture 22, Data 101/Info 258 Spring 2024

How do Partitioned Operators Work? Joins

Natural join R and S on A

Neither R nor S is partitioned on A

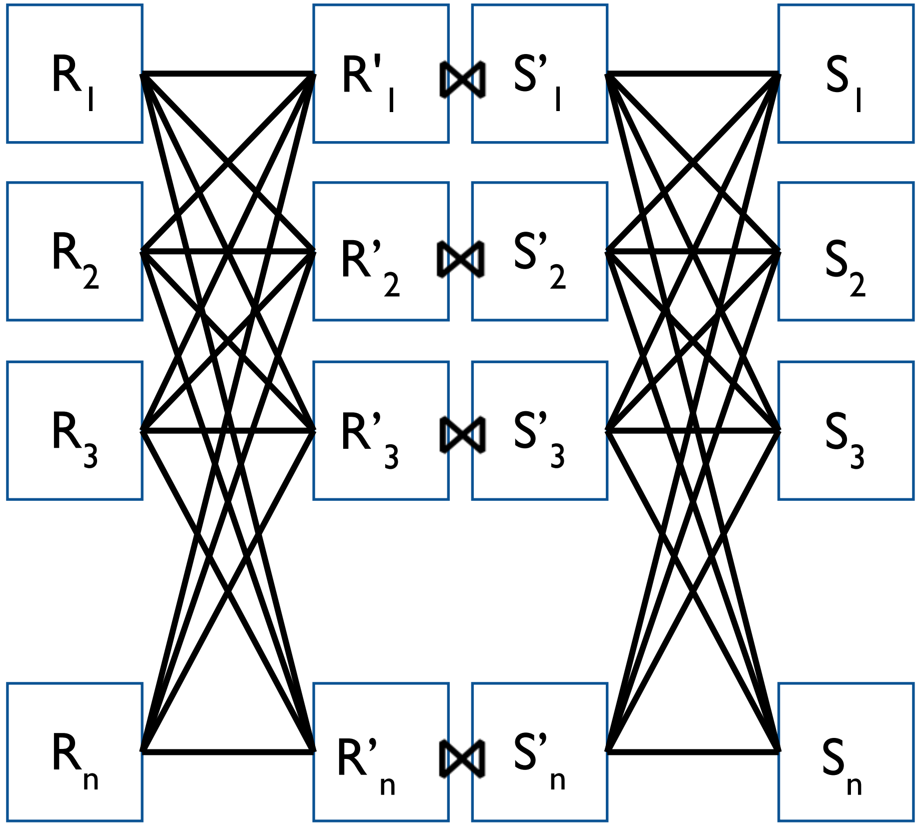
Q: How would we do this join?

Repartition R on A
Repartition S on A

- Hash or range (same for both)

Then each node individually does a hash join, a merge join, a nested loops, etc.

- Or **streaming variants** that can start producing results without waiting for entire input(s) to arrive



Parallel Joins with Complex Join Conditions

When the join conditions become a little more complex

- multiple predicates, <>, …
- e.g., Join R(A, B) and S(C, D) where R.A + S.C > 10

Can't simply apply repartitioning as is

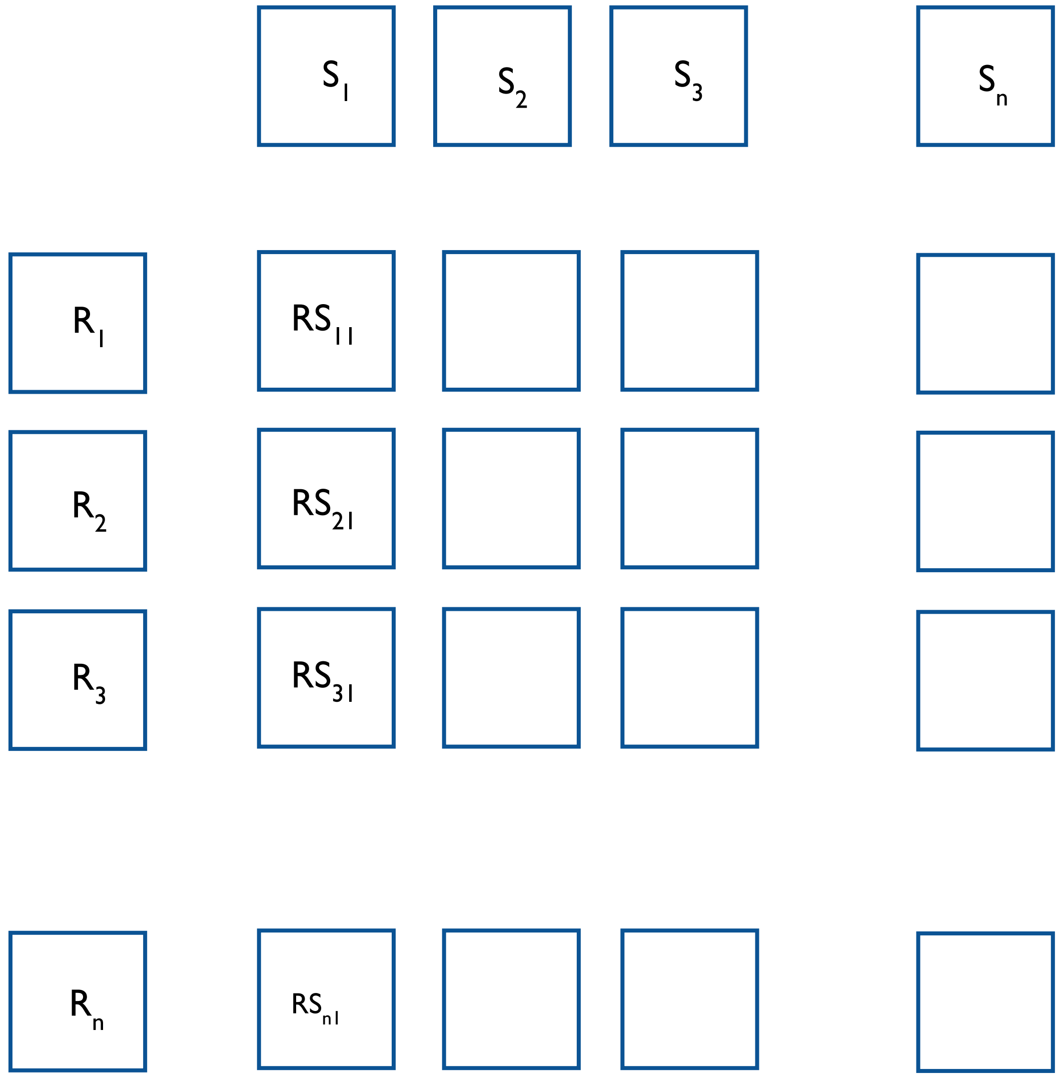
Q: What would we do for this?

Parallel Join: Fragment-Replicate Join

Every fragment in R is joined with every fragment in S

Extreme case: only one fragment of R (when it is small)

- Called a broadcast or asymmetric fragment-replicate join



How do Partitioned Operators Work? Aggregation

Example: SELECT A, SUM(B) FROM R GROUP BY A

Easy to do if GROUP BY is on partitioning attribute (A)

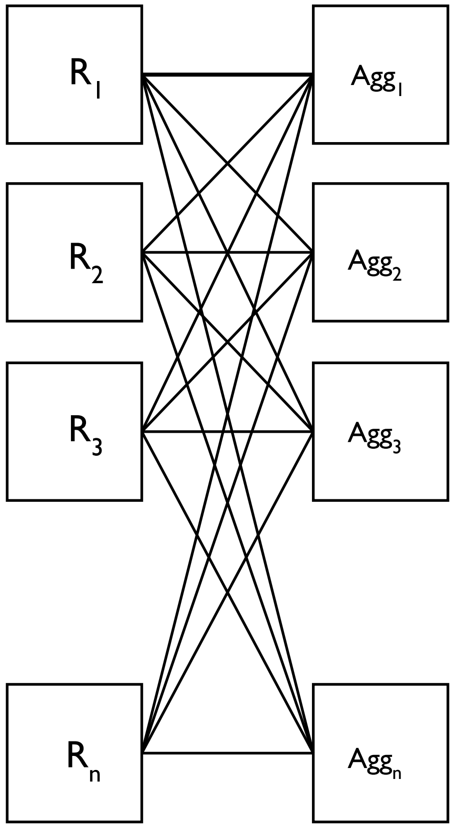
Else:

- Repartition relation based on the groups of A
  - Hash or range
  - Each group A = aj accumulates at a given node
- Perform aggregation for each group at corresponding node

Q: Any room for optimization?

A: Can avoid communicating all of the tuples

- Only communicate partial results of aj, SUM(B) from each partition Ri



MapReduce

Lecture 22, Data 101/Info 258 Spring 2024

Parallel Data Processing Considerations

Data Partitioning ✅

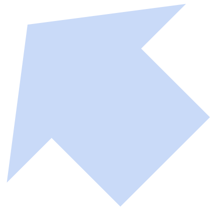
- Where and how do we break up the datasets into pieces across many machines to be operated on in parallel?

Parallel Data Processing

- Given that the datasets have been broken up, what parallel computation alternatives exist?
- Latency hierarchy: reading from memory << from disk << across the network

~~Hardware edge cases~~

- ~~How do we deal with failures and stragglers?~~
- ~~If we have 1000 servers and a server dies once every 100 days, then on average 10 will die per day.~~



next up

We'll only get to discuss the first two items (partitioning, **processing**). But know that physical restrictions are the most important in daily operation!

MapReduce

[Google Research link]

MapReduce is a parallel programming model that was introduced in 2004 at Google.

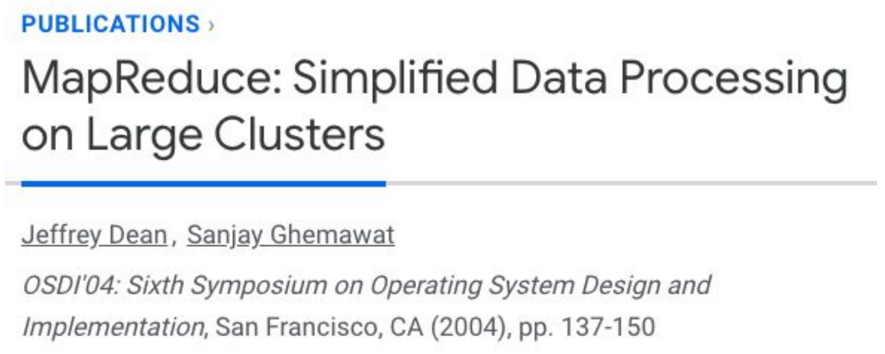
The programmer specifies two methods:

`Map(k, v) → <k', v'>*`

- Takes a key-value pair and outputs a set of key-value pairs
- There is one `Map` function call for each `(k,v)` pair

`Reduce(k', <v'>*) → <k', v''>*`

- All values `v'` with same key `k'` are reduced together and processed in `v'` order
- There is one `Reduce` function call for each unique key `k'`

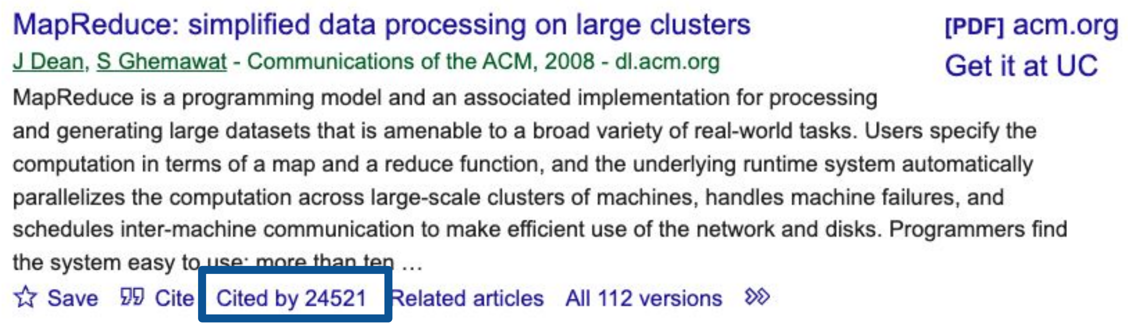


PUBLICATIONS ›

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean, Sanjay Ghemawat

OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150



MapReduce: simplified data processing on large clusters — [PDF] acm.org — Get it at UC

J Dean, S Ghemawat - Communications of the ACM, 2008 - dl.acm.org

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten …

On the surface, MapReduce will seem entirely different to every paradigm you've seen before. But it's not!

But it turns out that MapReduce, like partitioning ops, leverages **parallel operations and manager-worker nodes**.

MapReduce

[Google Research link]

MapReduce is a parallel programming model that was introduced in 2004 at Google.

The programmer specifies two methods:

`Map(k, v) → <k', v'>*`

- Takes a key-value pair and outputs a set of key-value pairs
- There is one `Map` call for every `(k,v)` pair

Map: item-by-item processing
Read a lot of data and extract something of value

(**shuffle**: implicit in-between step to group by keys such that reduce works properly)

`Reduce(k', <v'>*) → <k', v''>*`

- All values `v'` with same key `k'` are reduced together and processed in `v'` order
- There is one `Reduce` function call per unique key `k'`

Reduce: collect items corresponding to the same key, and process them together

Fundamental idea: Many parallel data processing algorithms (on both relational and semi-/un-structured data) can be captured using these two primitives: **map** and **reduce**.

MapReduce Example 1: Word Counts

Suppose that a crawl of the entire world wide web is stored across N machines.

We want to count the # of times each word appears on the world wide web.

MapReduce Example 1: Word Counts

[Google Research link]

MapReduce is a parallel programming model that was introduced in 2004 at Google.

The programmer specifies two methods:

`Map(k, v) → <k', v'>*`

- Takes a key-value pair and outputs a set of key-value pairs
- There is one `Map` call for every `(k,v)` pair

Map: item-by-item processing
Read a lot of data and extract something of value

- For each document, emit kv pairs <word: 1> for each word as it appears

(**shuffle**: implicit in-between step to group by keys such that reduce works properly)

- Group/sort pairs based on word

<word 1, 1>, <word 1, 1>, … <word 1, 1> <word2, 1> , …

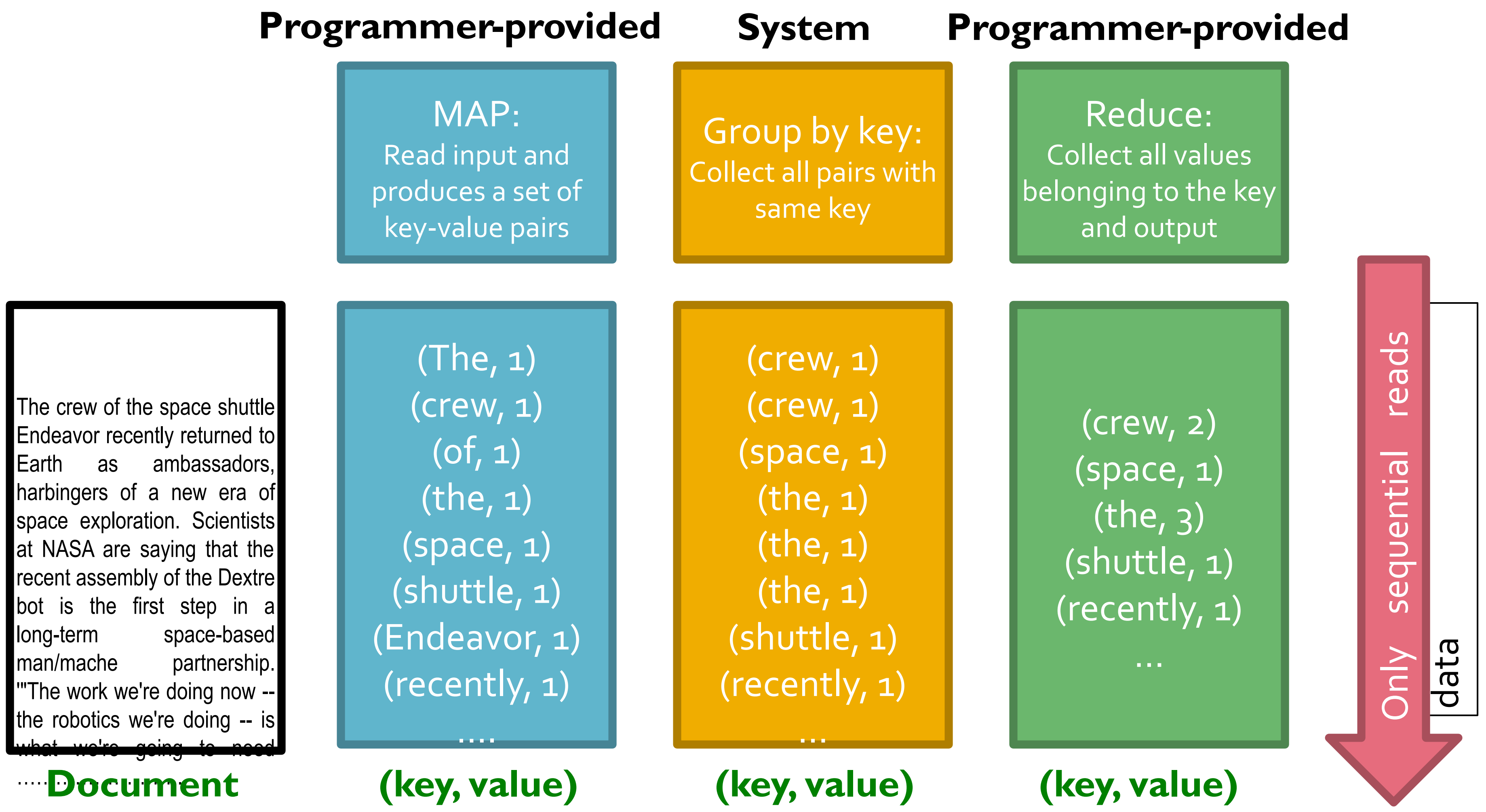
`Reduce(k', <v'>*) → <k', v''>*`

- All values `v'` with same key `k'` are reduced together and processed in `v'` order
- There is one `Reduce` function call per unique key `k'`

Reduce: collect items corresponding to the same key, and process them together

- For each word, sum up the total number of 1s in the value list.

MapReduce Example 1: Word Counts



The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need
……………………..

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

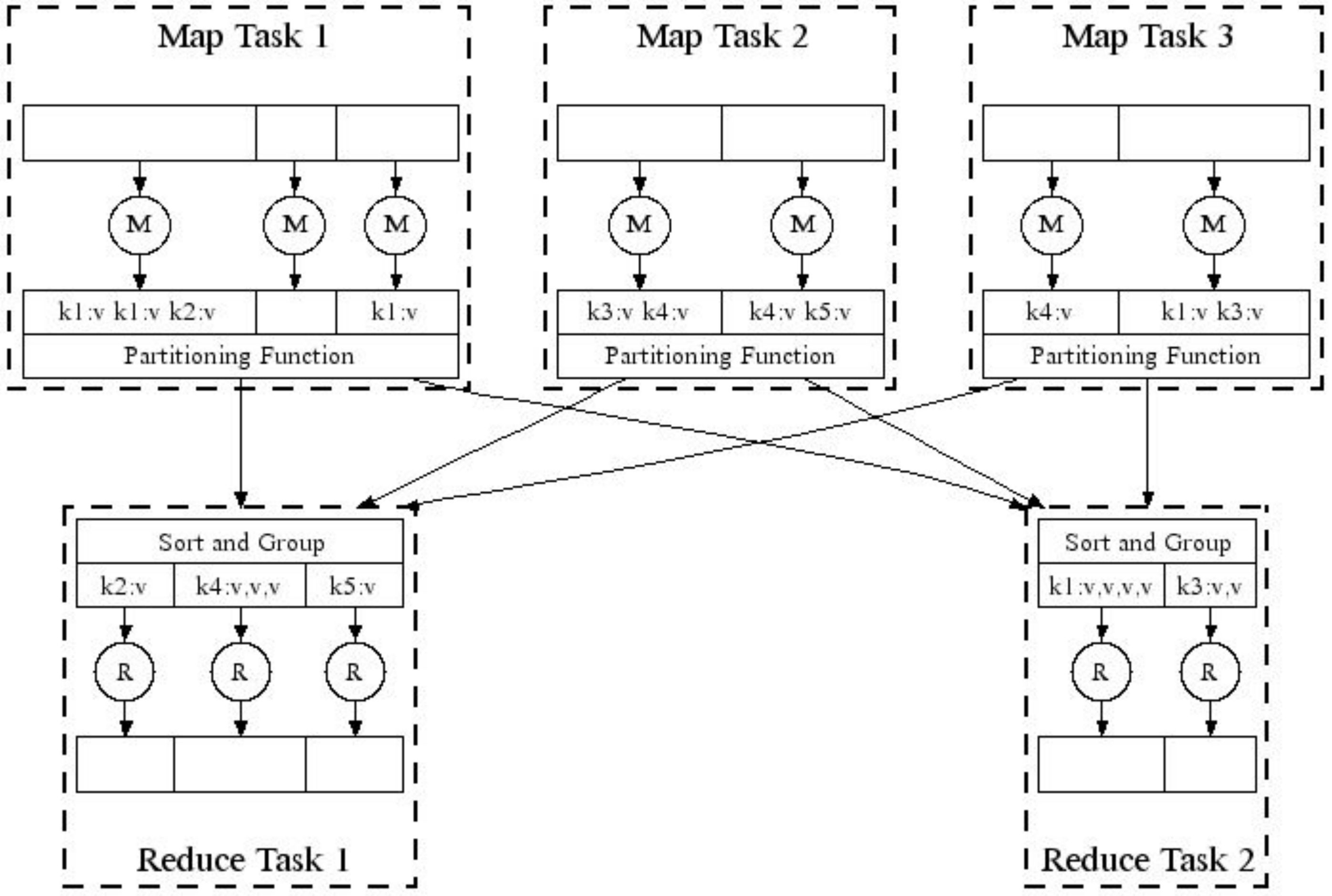
(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

Each document is processed on a separate worker

reorg by key and redistribute to workers

each key-value list is processed on a separate worker

[at home] MapReduce Example 2: Partitioned Parallel Join



[at home] MapReduce Example 2: Partitioned Parallel Join

Input relations:

- R(A, B), S(B, C)
- partitioned across many nodes

Map:

Input: R (or S)

Output <B: (R, A)> (or <B: (S, C)>)

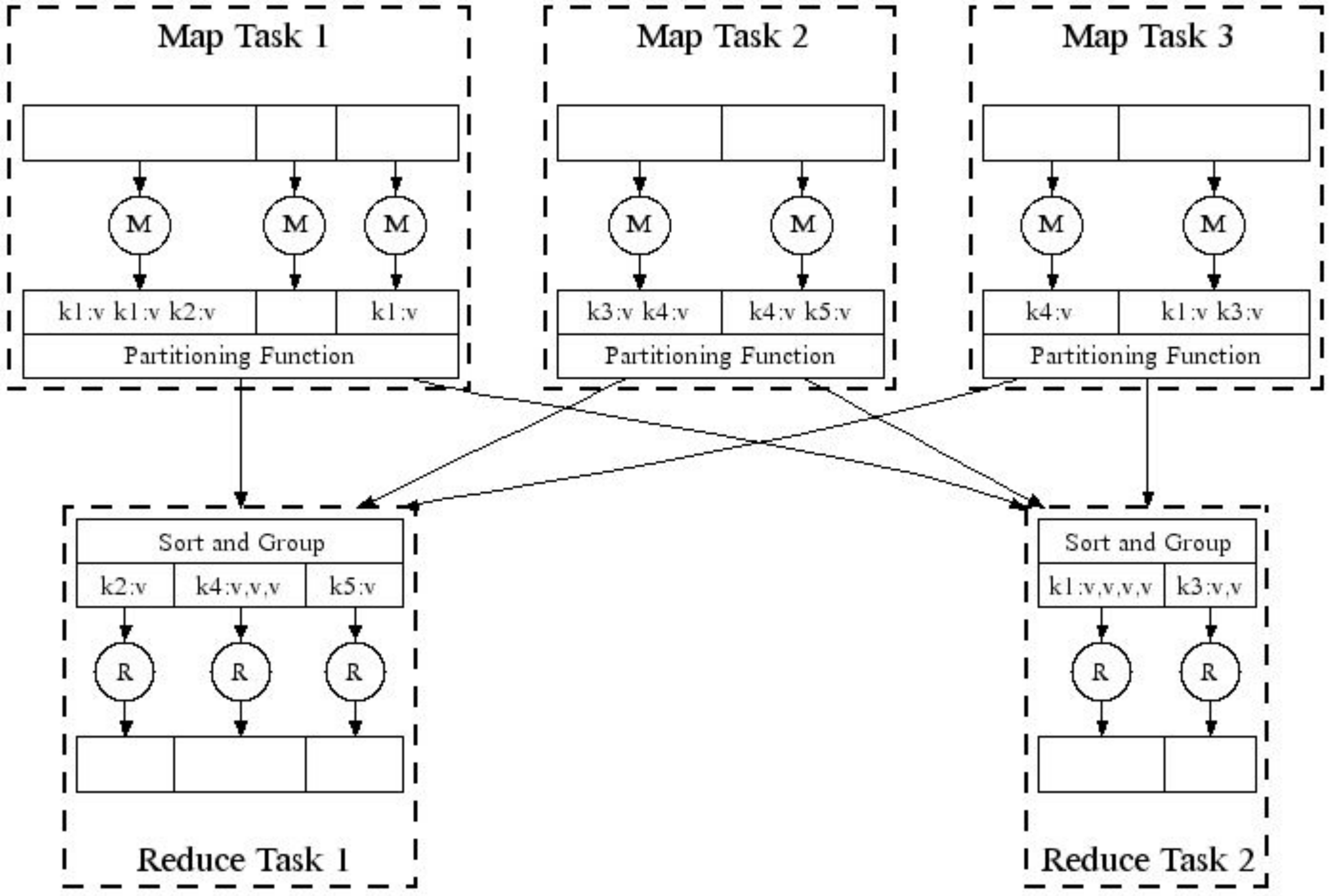
(**Shuffle:** all B values on same worker)

Reduce:

Input <B: (R, A1)…(R, An), (S, C1)…(S, Cm)>

Output: cross product of Ai and Cj

(A1, B, C1), (A1, B, C2), …, (An, B, Cm)



MapReduce's Legacy and Spark

Lecture 22, Data 101/Info 258 Spring 2024

MapReduce beyond Map and Reduce

Other than the Map and Reduce functions defined by the programmer,
the system handles everything else:

- Assigning tasks to different worker machines
- Performing the “shuffle” step
- Handling failures and inter-machine communication (via “materialization,” or creation of intermediate data copies between each step)
- In some case, performs parallel aggregation before larger shuffling across the network
  - This “combiner” during the map task can reduce the amount of data shuffled/reorganized across the full network.

Is MapReduce Still Used?

There are **several downsides** of MapReduce:

- User must handwrite the Map and Reduce steps;

we've seen a glimpse of how hard it is to think in this paradigm!

Is MapReduce Still Used?

There are **several downsides** of MapReduce:

- User must handwrite the Map and Reduce steps;
- No indexing, query optimization;
- No pipelining; intermediate data copies must be materialized;
- Cannot "declare" query processing steps; and
- Overall, must rely on system's shuffle step. Not very flexible!

we've seen a glimpse of how hard it is to think in this paradigm!

While convenient to do **arbitrary large scale** parallel data processing on **arbitrary data**…
…MapReduce isn't actually all that efficient compared to **parallel relational databases**!

- Lots of accumulated wisdom; parallel relational databases research since 1990s!

Is MapReduce Still Used?

There are **several downsides** of MapReduce:

- User must handwrite the Map and Reduce steps;
- No indexing, query optimization;
- No pipelining; intermediate data copies must be materialized;
- Cannot "declare" query processing steps; and
- Overall, must rely on system's shuffle step. Not very flexible!

While convenient to do **arbitrary large scale** parallel data processing on **arbitrary data**… …MapReduce isn't actually all that efficient compared to **parallel relational databases**!

- Lots of accumulated wisdom; parallel relational databases research since 1990s!

At this point, MapReduce is largely overshadowed by **parallel databases** that addresses several of these limitations.

However, note that various MapReduce implementations still exist today:

- **Hadoop MapReduce** (2006), open-sourced alternative to Google's. Still exists today for open-ended large scale data processing.
- MongoDB, CouchDB, and **doc stores. Key-value stores** are a natural fit for MapReduce algos

Spark: Somewhere In Between

Spark (2014) introduced the notion of **resilient distributed datasets** (RDDs).

- RDDs: Fancy name for data cached in memory with a “query” attached to it

Originally, introduced as a way to bridge the gap between MapReduce and Parallel DBs

- Supported a small set of data manipulation operators
  - More than relational databases, less than dataframes. Lots of Data Science/ML

Ultimately ended up much faster than MapReduce…

- …while providing many similar benefits: distributed computation, arbitrary data types, etc.
- Features: query optimization (b/c of restricted operator set), pipelining, etc.

Now: More like a parallel database with additional machine-learning-oriented bells and whistles.

- Supports SQL as its primary language

Matei Zaharia
Associate Professor
UC Berkeley

Beyond: Streaming Data Applications

- Parallel DBs: Partitioning Operations
- MapReduce
- Spark

These all only perform **batch jobs**: A single query on a bunch of data with one result.

Modern applications need much more, e.g., **streaming jobs**, or live results as data is updated.

[Extra] Parallel DB: Sort-Merge, Sort

Lecture 22, Data 101/Info 258 Spring 2024

Parallel Two-Pass Sort-Merge

- Recall regular sort:
  - Pass 1:
    - Sort B(R)/M subsets of M blocks of R each
    - Each is output to disk as a *run*
  - Pass 2:
    - “Merge” these runs by bringing in one block for each of them

- Pass 1: Locally sort data at each node
- Pass 2: Shuffle runs across the network
- Pass 3: Merge these partitioned runs across nodes
  - Node 1 gets all runs corresponding to [1-100] and then then merges them
  - Node 2 gets all runs corresponding to [101-200] and then then merges them
  - …
- Pass 4: The sorted runs at Node 1, Node 2, … are kept as is in a partitioned manner, or are concatenated to give an overall sort

Parallel Sort Approach 2: Partitioned Sort

- Pass 1: Each node does partitioning, sending tuples across the network:
  - Node 1 gets all tuples corresponding to [1-100]
  - Node 2 gets all tuples corresponding to [101-200]
  - …
- Pass 2: Each receiving node then sorts all the tuples locally
- Pass 3: The sorted runs at Node 1, Node 2, … are kept as is in a partitioned manner, or are concatenated to give an overall sort

- Both approaches are roughly equivalent…

[Extra] Volcano Framework for Parallelism

Lecture 22, Data 101/Info 258 Spring 2024

Volcano Framework: A Helpful Way to think about Parallelism

The Volcano Framework (90s) by Graefe

- Considering so many parallel variants for every operator can be quite a challenge!
- Instead, define a new operator called **exchange** that easily incorporates parallelism into query processor.
- Other operators are unaware of parallelism and perform local ops on local data.

Exchange functions:

- Hash/range-based partitioning of data
- Replicating data to all nodes (broadcasting)
- Sending all data to a single node

Destination operators can read data from multiple “streams” via

- Random merge (order doesn’t matter)
- Ordered merge

Volcano Framework Examples

Partitioned parallel join

1. Exchange operator with hash or range partitioning
2. Local join

Asymmetric fragment and replicate join

1. Exchange operator using broadcast replication
2. Local join

Partitioned parallel aggregation

1. Exchange operator with hash or range partitioning
2. Local aggregation

MapReduce in terms of other parallel operations

MapReduce is akin to what we've been discussing so far:

- Map is analogous to **tuple-wise operations** such as projection/selection
- Shuffle is analogous to **exchange** (via repartitioning/broadcast)
- Reduce is analogous to **merge-style operations**: joins, sorts, grouping/aggregation
  - Where there is a need to ensure that all tuples satisfying certain conditions end up at a certain node