CSE 132C
Database System Implementation

Arun Kumar

Topic 6: Parallel RDBMSs and Dataflow Systems

Chapter 22 till 22.5 of Cow Book; extra references listed
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
Parallel DBMSs: Motivation
Parallel DBMSs: Motivation

❖ **Scalability**: Database is too large for a single node’s disk
Parallel DBMSs: Motivation

❖ **Scalability**: Database is too large for a single node’s disk
❖ **Performance**: Exploit multiple cores/disks/nodes
Parallel DBMSs: Motivation

❖ **Scalability**: Database is too large for a single node’s disk
❖ **Performance**: Exploit multiple cores/disks/nodes
❖ … while maintaining almost all other benefits of (R)DBMSs!
Three Paradigms of Parallelism
Three Paradigms of Parallelism

Shared-Disk Parallelism

Interconnect
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
  - Interconnect
  - Multiple CPUs connected to multiple disks

- **Shared-Memory Parallelism**
  - Interconnect
  - Multiple CPUs connected to a shared memory space

These paradigms differ in how data is accessed and processed by the system.
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
- **Shared-Memory Parallelism**
- **Shared-Nothing Parallelism**
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
  - Contention
  - Interconnect

- **Shared-Memory Parallelism**
  - Interconnect

- **Shared-Nothing Parallelism**
  - Interconnect
Three Paradigms of Parallelism

Shared-Disk Parallelism

Shared-Memory Parallelism

Shared-Nothing Parallelism

Interconnect

Contention
Three Paradigms of Parallelism

- Shared-Disk Parallelism
- Shared-Memory Parallelism
- Shared-Nothing Parallelism

Contention

Interconnect

Data/Partitioned Parallelism
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
  - Interconnect
  - Contention

- **Shared-Memory Parallelism**
  - Interconnect
  - Contention

- **Data/Partitioned Parallelism**
  - Interconnect

**Shared-Nothing Parallelism**
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
- **Shared-Memory Parallelism**
- **Shared-Nothing Parallelism**

**Data/Partitioned Parallelism**

- **Contention**
- **Interconnect**

**Symmetric Multi-Processing (SMP)**
Three Paradigms of Parallelism

- Shared-Disk Parallelism
- Shared-Memory Parallelism
- Shared-Nothing Parallelism

Interconnect

Contention

Data/Partitioned Parallelism

Symmetric Multi-Processing (SMP)

Massively Parallel Processing (MPP)
Shared-Nothing Parallelism
Shared-Nothing Parallelism

- Followed by almost all parallel RDBMSs (and “Big Data” sys.)
Shared-Nothing Parallelism

- Followed by almost all parallel RDBMSs (and “Big Data” sys.)
- 1 manager node orchestrates multiple worker nodes
Shared-Nothing Parallelism

- Followed by almost all parallel RDBMSs (and “Big Data” sys.)
- 1 manager node orchestrates multiple worker nodes
- Need partitioned parallel implementation algorithms for relational op implementations and query proc.; modify QO
Shared-Nothing Parallelism

❖ Followed by almost all parallel RDBMSs (and “Big Data” sys.)
❖ 1 manager node orchestrates multiple worker nodes
❖ Need partitioned parallel implementation algorithms for relational op implementations and query proc.; modify QO

**Q:** If we give 10 workers (CPUs/nodes) for processing a query in parallel, will its runtime go down by a factor of 10?
Shared-Nothing Parallelism

- Followed by almost all parallel RDBMSs (and “Big Data” sys.)
- 1 manager node orchestrates multiple worker nodes
- Need partitioned parallel implementation algorithms for relational op implementations and query proc.; modify QO

**Q:** If we give 10 workers (CPUs/nodes) for processing a query in parallel, will its runtime go down by a factor of 10?

It depends!
(Access patterns of the query’s operators, communication of intermediate data, relative startup overhead, etc.)
Shared-Nothing Parallelism
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Speedup plot / Strong scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Speedup plot / Strong scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Speedup plot / Strong scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Number of workers

Speedup plot / Strong scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Speedup plot / Strong scaling
Shared-Nothing Parallelism

**Runtime speedup (fixed data size)**

- **Linear Speedup**
- **Sublinear Speedup**

**Number of workers**

1  4  8  12

**Factor (# workers, data size)**

1  4  8  12

**Speedup plot / Strong scaling**

**Scaleup plot / Weak scaling**
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

- Linear Speedup
- Sublinear Speedup

Number of workers:
1  4  8  12

Factor (# workers, data size):
1  4  8  12

Speedup plot / Strong scaling

Scaleup plot / Weak scaling
Shared-Nothing Parallelism

**Runtime speedup (fixed data size)**

- **Linear Speedup**
- **Sublinear Speedup**

**Number of workers**

<table>
<thead>
<tr>
<th>Workers</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

**Factor (# workers, data size)**

- **Linear Scaleup**

**Speedup plot / Strong scaling**

**Scaleup plot / Weak scaling**
Shared-Nothing Parallelism

**Speedup** plot / Strong scaling

**Scaleup** plot / Weak scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

- Linear Speedup
- Sublinear Speedup

Number of workers

- Speedup plot / Strong scaling

Runtime speedup

- Linear Scaleup

Factor (# workers, data size)

- Scaleup plot / Weak scaling
Shared-Nothing Parallelism

Runtime speedup (fixed data size)

Number of workers

1  4  8  12

1  4  8  12

Linear Speedup

Sublinear Speedup

Runtime speedup

Factor (# workers, data size)

1  4  8  12

1  0.5

Linear Scaleup

Sublinear Scaleup

Speedup plot / Strong scaling

Scaleup plot / Weak scaling
Q: Is *superlinear* speedup/scaleup possible?
Shared-Nothing Parallelism: Outline

❖ Data Partitioning
❖ Parallel Operator Implementations
❖ Parallel Query Optimization
❖ Parallel vs “Distributed” DBMSs
Data Partitioning
Data Partitioning

❖ A part of **ETL** (Extract-Transform-Load) for database
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given k machines):
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given k machines):
  - **Round-robin**: assign tuple \( i \) to machine \( i \mod k \)
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given k machines):
  - **Round-robin**: assign tuple $i$ to machine $i \mod k$
  - **Hashing-based**: needs partitioning attribute(s)
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given $k$ machines):
  - **Round-robin**: assign tuple $i$ to machine $i \mod k$
  - **Hashing-based**: needs partitioning attribute(s)
  - **Range-based**: needs ordinal partitioning attribute(s)
Data Partitioning

- A part of ETL (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given k machines):
  - **Round-robin**: assign tuple i to machine i MOD k
  - **Hashing-based**: needs partitioning attribute(s)
  - **Range-based**: needs ordinal partitioning attribute(s)
- **Tradeoffs**: Round-robin often inefficient for parallel query processing (why?); range-based good for range queries but faces new kind of “skew”; hashing-based is most common
Data Partitioning

- A part of **ETL** (Extract-Transform-Load) for database
- Typically, record-wise/horizontal partitioning (aka “sharding”)
- Three common schemes (given k machines):
  - **Round-robin**: assign tuple i to machine i MOD k
  - **Hashing-based**: needs partitioning attribute(s)
  - **Range-based**: needs ordinal partitioning attribute(s)
- **Tradeoffs**: Round-robin often inefficient for parallel query processing (why?); range-based good for range queries but faces new kind of “skew”; hashing-based is most common
- **Replication** often used for more availability, performance
Parallel Scans and Select
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- **Filescan:**
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!
- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
Parallel Scans and Select

- Intra-operator parallelism is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- Filescan:
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)

- **Indexed:**
Parallel Scans and Select

❖ **Intra-operator parallelism** is our primary focus
  ❖ Inter-operator and inter-query parallelism also possible!

❖ **Filescan:**
  ❖ Trivial! Worker simply scans its partition and streams it
  ❖ Apply selection predicate (if any)

❖ **Indexed:**
  ❖ Depends on data partitioning scheme and predicate!
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)

- **Indexed:**
  - Depends on data partitioning scheme and predicate!
  - Same tradeoffs: Hash index vs B+ Tree index
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)

- **Indexed:**
  - Depends on data partitioning scheme and predicate!
  - Same tradeoffs: Hash index vs B+ Tree index
  - Each worker can have its own (sub-)index
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!

- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)

- **Indexed:**
  - Depends on data partitioning scheme and predicate!
  - Same tradeoffs: Hash index vs B+ Tree index
  - Each worker can have its own (sub-)index
  - Manager *routes* query based on “matching workers”
Parallel Sorting
Parallel Sorting

- Naive algorithm:
Parallel Sorting

- **Naive algorithm:**
  1. Each worker sorts local partition (EMS)
Parallel Sorting

❖ Naive algorithm:

1. Each worker sorts local partition (EMS)
2. Manager merges all locally sorted runs
Parallel Sorting

❖ Naive algorithm:

1. Each worker sorts local partition (EMS)
2. Manager merges all locally sorted runs

❖ Issue: Parallelism is limited during merging phase!
Parallel Sorting

❖ Naive algorithm:

1. Each worker sorts local partition (EMS)
2. Manager merges all locally sorted runs

❖ Issue: Parallelism is limited during merging phase!

❖ Faster algorithm:
Parallel Sorting

❖ **Naive algorithm:**

1. Each worker sorts local partition (EMS)
2. Manager merges all locally sorted runs

❖ **Issue:** Parallelism is limited during merging phase!

❖ **Faster algorithm:**

1. Scan in parallel and *range partition* data (most likely a repartitioning) based on SortKey
Parallel Sorting

❖ **Naive algorithm:**
  1. Each worker sorts local partition (EMS)
  2. Manager merges all locally sorted runs
❖ **Issue:** Parallelism is limited during merging phase!

❖ **Faster algorithm:**
  1. Scan in parallel and *range partition* data (most likely a repartitioning) based on SortKey
  2. Each worker sorts local allotted range (EMS); result is globally sorted and conveniently range-partitioned
Parallel Sorting

❖ **Naive algorithm:**
  1. Each worker sorts local partition (EMS)
  2. Manager merges all locally sorted runs

❖ **Issue:** Parallelism is limited during merging phase!

❖ **Faster algorithm:**
  1. Scan in parallel and *range partition* data (most likely a repartitioning) based on SortKey
  2. Each worker sorts local allotted range (EMS); result is globally sorted and conveniently range-partitioned

❖ **Potential Issue:** Skew in range partitions; handled by roughly estimating distribution using sampling
Parallel Sorting

Original Partitions

Manager

Worker 1

Worker 2

... 

Worker n
Parallel Sorting

Original Partitions

Manager

Worker 1

Worker 2

Worker n

Assign SortKey

Range splits

\( V_1 \) to \( V_2 \)

\( V_2 \) to \( V_3 \)

... 

\( V_{n-1} \) to \( V_n \)
Parallel Sorting

Original Partitions

Manager

Worker 1

Worker 2

...  

Worker n

Assign SortKey

Range splits

$V_1$ to $V_2$

$V_2$ to $V_3$

$V_{n-1}$ to $V_n$

Re-partitioning
Parallel Sorting

Original Partitions

Manager

Worker 1

Worker 2

Worker n

Assign SortKey
Range splits

V₁ to V₂

V₂ to V₃

Vₙ₋₁ to Vₙ

Range-partitioned

Manager

Worker 1

Worker 2

Worker n

V₁ to V₂

V₂ to V₃

Vₙ₋₁ to Vₙ

Re-partitioning
Parallel Sorting

Original Partitions

Manager

Worker 1

\( V_1 \) to \( V_2 \)

Worker 2

\( V_2 \) to \( V_3 \)

... 

Worker \( n \)

\( V_{n-1} \) to \( V_n \)

Assign SortKey

Range splits

Range-partitioned

Manager

Worker 1

\( V_1 \) to \( V_2 \)

Worker 2

\( V_2 \) to \( V_3 \)

... 

Worker \( n \)

\( V_{n-1} \) to \( V_n \)

Local EMS

Assign SortKey

Range splits

Re-partitioning
Parallel Sorting

Original Partitions

Manager

Worker 1

$V_1$ to $V_2$

Worker 2

$V_2$ to $V_3$

...$

Worker n$

$V_{n-1}$ to $V_n$

Assign SortKey

Range splits

Range-partitioned

Manager

Worker 1

$V_1$ to $V_2$

Worker 2

$V_2$ to $V_3$

...$

Worker n$

$V_{n-1}$ to $V_n$

Re-partitioning

Globally Sorted

Manager

Worker 1

$V_1$ to $V_2$

Worker 2

$V_2$ to $V_3$

...$

Worker n$

$V_{n-1}$ to $V_n$
Parallel Aggregates and Group By
Parallel Aggregates and Group By

- Without Group By List:
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
Parallel Aggregates and Group By

- **Without Group By List:**
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  - MEDIAN requires parallel sorting (why?)
Parallel Aggregates and Group By

❖ Without Group By List:
  ❖ Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  ❖ MEDIAN requires parallel sorting (why?)

❖ With Group By List:
Parallel Aggregates and Group By

❖ Without Group By List:
  ❖ Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  ❖ MEDIAN requires parallel sorting (why?)

❖ With Group By List:
  1. If AggFunc allows, pre-compute partial aggregates
Parallel Aggregates and Group By

- **Without Group By List:**
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  - MEDIAN requires parallel sorting (why?)

- **With Group By List:**
  1. If AggFunc allows, pre-compute partial aggregates
  2. Manager assigns each worker a set of groups (**hash partition**)
Parallel Aggregates and Group By

- **Without Group By List:**
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  - MEDIAN requires parallel sorting (why?)

- **With Group By List:**
  1. If AggFunc allows, pre-compute partial aggregates
  2. Manager assigns each worker a set of groups (hash partition)
  3. Each worker communicates its partial aggregate for a group to that group’s assigned worker (aka “shuffle”)
Parallel Aggregates and Group By

- **Without Group By List:**
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  - MEDIAN requires parallel sorting (why?)

- **With Group By List:**
  1. If AggFunc allows, pre-compute partial aggregates
  2. Manager assigns each worker a set of groups (hash partition)
  3. Each worker communicates its partial aggregate for a group to that group’s assigned worker (aka “shuffle”)
  4. Each worker finishes aggregating for all its assigned groups
Parallel Group By Aggregate

Original Partitions

Manager

Worker 1

Worker 2

...

Worker n
Parallel Group By Aggregate

Original Partitions

Manager

Local GrpBY

Worker 1

Worker 2

...

Worker n
Parallel Group By Aggregate

Original Partitions

Worker 1

Manager

Local GrpBY

Worker 2

...

Worker n

Partial Aggs

Worker 1

Manager

Worker 2

...

Worker n
Parallel Group By Aggregate

Original Partitions

Worker 1

Worker 2

Worker n

Manager

Local GrpBY

Partial Aggs

Worker 1

Worker 2

Worker n

Manager

Assign GroupingList

Hash splits

G₁

G₂

Gₙ
Parallel Group By Aggregate

Original Partitions

Worker 1

Worker 2

Worker n

Partial Aggs

Manager

Worker 1

Worker 2

Worker n

Assign GroupingList

Hash splits

G1

G2

Gn

Re-partitioning

Local GrpBY
Parallel Group By Aggregate

Original Partitions

Worker 1

Worker 2

Worker n

Local GrpBY

Worker 1

Worker 2

Worker n

Manager

Partial Aggs

Manager

Worker 1

Worker 2

Worker n

Assign GroupingList
Hash splits

Re-partitioned Partial Aggs

Manager

Worker 1

Worker 2

Worker n

G1

G2

Gn

Assign GroupingList

Hash splits

Re-partitioning

G1

G2

Gn
Parallel Group By Aggregate

Original Partitions

Worker 1
Worker 2
...
Worker n
Manager

Local GrpBY

Partial Aggs

Worker 1
Worker 2
...
Worker n
Manager

Assign GroupingList

Hash splits

Re-partitioned Partial Aggs

Worker 1
Worker 2
...
Worker n
Manager

Local GrpBY Again

Partial Aggs

Worker 1
Worker 2
...
Worker n
Manager

Assign GroupingList

Hash splits

Re-partitioning

Worker 1
Worker 2
...
Worker n
Manager

Local GrpBY Again

G_1
G_2
G_n
Parallel Group By Aggregate

Original Partitions

Worker 1
Worker 2
Worker n

Manager
Local GrpBY
Local
GrpBY
Again

Partial Aggs

Worker 1
Worker 2
Worker n

Manager
Local GroupingList
Hash splits

Assign
G1
G2
Gn

Re-partitioned Partial Aggs

Worker 1
Worker 2
Worker n

Manager
Local GrpBY
Again

Final Aggs

Worker 1
Worker 2
Worker n

Manager

Local
GrpBY
Again

Partial Aggs

Worker 1
Worker 2
Worker n

Manager
Local GroupingList
Hash splits

Assign
G1
G2
Gn

Re-partitioning

Final Aggs

Worker 1
Worker 2
Worker n

Manager

Local
GrpBY
Again
Parallel Project
Parallel Project

- Non-deduplicating Project:
  - Trivial! Pipelined with Scans/Select
Parallel Project

- Non-deduplicating Project:
  - Trivial! Pipelined with Scans/Select

- Deduplicating Project:
Parallel Project

❖ **Non-deduplicating Project:**
  ❖ Trivial! Pipelined with Scans/Select

❖ **Deduplicating Project:**
  1. Each worker deduplicates its partition on ProjectionList
Parallel Project

❖ **Non-deduplicating Project:**
  ❖ Trivial! Pipelined with Scans/Select

❖ **Deduplicating Project:**

1. Each worker deduplicates its partition on ProjectionList
2. If estimated output size is small (catalog?), workers communicate their results to Manager to finish dedup.
Parallel Project

❖ Non-deduplicating Project:
   ❖ Trivial! Pipelined with Scans/Select

❖ Deduplicating Project:
   1. Each worker deduplicates its partition on ProjectionList
   2. If estimated output size is small (catalog?), workers communicate their results to Manager to finish dedup.
   3. If estimated output size is too large for Manager’s disk, similar algorithm as Parallel Aggregate with Group By, except, there is no AggFunc computation
Parallel Nested Loops Join
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Manager assigns range/hash splits on JoinAttribute to workers
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Manager assigns range/hash splits on JoinAttribute to workers
  2. Repartitioning of A and B separately using same splits on JoinAttribute (unless pre-partitioned on it!)
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Manager assigns range/hash splits on JoinAttribute to workers
  2. Repartitioning of A and B separately using same splits on JoinAttribute (unless pre-partitioned on it!)
  3. Worker i applies BNLJ locally on its partitions Ai and Bi
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Manager assigns range/hash splits on JoinAttribute to workers
  2. Repartitioning of A and B separately using same splits on JoinAttribute (unless pre-partitioned on it!)
  3. Worker i applies BNLJ locally on its partitions Ai and Bi
  4. Overall join output is just collection of all n worker outputs
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Manager assigns range/hash splits on JoinAttribute to workers
  2. Repartitioning of A and B separately using same splits on JoinAttribute (unless pre-partitioned on it!)
  3. Worker i applies BNLJ locally on its partitions Ai and Bi
  4. Overall join output is just collection of all n worker outputs
- If join is not equi-join, there might be a lot of communication between workers; worst-case: all-to-all for cross-product!
Parallel “Split” and “Merge” for Joins
Parallel “Split” and “Merge” for Joins

- Repartitioning quite common for parallel (equi-)joins
Parallel “Split” and “Merge” for Joins

- Repartitioning quite common for parallel (equi-)joins
- Functionality abstracted as two new physical operators:
Parallel “Split” and “Merge” for Joins

- Repartitioning quite common for parallel (equi-)joins
- Functionality abstracted as two new physical operators:
  - **Split**: each worker sends a subset of its partition to another worker based on Manager’s command (hash/range)
Parallel “Split” and “Merge” for Joins

- Repartitioning quite common for parallel (equi-)joins
- Functionality abstracted as two new physical operators:
  - **Split**: each worker sends a subset of its partition to another worker based on Manager’s command (hash/range)
  - **Merge**: each worker unions subsets sent to it by others and constructs its assigned (re)partitioned subset
Parallel “Split” and “Merge” for Joins

- Repartitioning quite common for parallel (equi-)joins
- Functionality abstracted as two new physical operators:
  - **Split**: each worker sends a subset of its partition to another worker based on Manager’s command (hash/range)
  - **Merge**: each worker unions subsets sent to it by others and constructs its assigned (re)partitioned subset
- Useful for parallel BNLJ, Sort-Merge Join, and Hash Join
Parallel Sort-Merge and Hash Join

- For SMJ, split is on ranges of (ordinal) JoinAttribute; for HJ, split is on hash function over JoinAttribute
- Worker i does local join of Ai and Bi using SMJ or HJ
Improved Parallel Hash Join
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance
- **Idea:** Previous version hash partitions JoinAttribute to n (same as # workers); instead, decouple the two and do a 2-stage process: partition phase and join phase
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance
- **Idea:** Previous version hash partitions JoinAttribute to n (same as # workers); instead, decouple the two and do a 2-stage process: partition phase and join phase

**Partition Phase:** Say |A| < |B|; divide A and B into k (can be > n) partitions using h1() s.t. each F x |Ai| < Cluster RAM
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance

- **Idea:** Previous version hash partitions JoinAttribute to n (same as # workers); instead, decouple the two and do a 2-stage process: partition phase and join phase

- **Partition Phase:** Say \(|A| < |B|\); divide A and B into k (can be > n) partitions using \(h1()\) s.t. each \(F \times |A_i| < \) Cluster RAM

- **Join Phase:** Repartition an \(A_i\) into n partitions using \(h2()\); build hash table on new \(A_{ij}\) at worker j as tuples arrive; repartition \(B_i\) using \(h2()\); local HJ of \(A_{ij}\) and \(B_{ij}\) on worker j in parallel for \(j = 1\) to n; repeat all these steps for each \(i = 1\) to k
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance
- **Idea:** Previous version hash partitions JoinAttribute to n (same as # workers); instead, decouple the two and do a 2-stage process: partition phase and join phase

  - **Partition Phase:** Say |A| < |B|; divide A and B into k (can be > n) partitions using h1() s.t. each F x |Ai| < Cluster RAM

  - **Join Phase:** Repartition an Ai into n partitions using h2(); build hash table on new Aij at worker j as tuples arrive; repartition Bi using h2(); local HJ of Aij and Bij on worker j in parallel for j = 1 to n; repeat all these steps for each i = 1 to k

- Uses all n workers for join of each subset pair $A_i \bowtie B_i$
Parallel Query Optimization

- Far more complex than single-node QO!
- I/O cost, CPU cost, and communication cost for each phy. op.
- Space of PQPs explodes: each node can have its own different local sub-plan (e.g., filescan v indexed)
- Pipeline parallelism and partitioned parallelism can be interleaved in complex ways!
- Join order enumeration affected: bushy trees can be good!
- … (we will skip more details)
Parallel vs “Distributed” RDBMSs
Parallel vs “Distributed” RDBMSs

❖ A parallel RDBMS layers distribution atop the file system
❖ Can handle dozens of nodes (Gamma, Teradata, etc.)
Parallel vs “Distributed” RDBMSs

❖ A parallel RDBMS layers distribution atop the file system
  ❖ Can handle dozens of nodes (Gamma, Teradata, etc.)
❖ Raghu’s “distributed”: collection of “independent” DBMSs
  ❖ Quirk of terminology; “federated” more accurate term
  ❖ Each base RDBMS can be at a different location
  ❖ Each RDBMS might host a subset of the database files
  ❖ Might need to ship entire files for distributed QP
  ❖ … (we will skip more details)
❖ These days: “Polystores,” federated DBMSs on steroids!
Outline

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
Cloud Computing
Cloud Computing

- Compute, storage, memory, networking are virtualized and exist on remote servers; rented by application users
Cloud Computing

Compute, storage, memory, networking are virtualized and exist on remote servers; rented by application users

- **Manageability**: Managing hardware is not user's problem!
- **Pay-as-you-go**: Fine-grained pricing economics based on actual usage (granularity: seconds to years!)
- **Elasticity**: Can dynamically add or reduce capacity based on actual workload’s demand
Cloud Computing

- Compute, storage, memory, networking are virtualized and exist on remote servers; rented by application users
  - **Manageability**: Managing hardware is not user's problem!
  - **Pay-as-you-go**: Fine-grained pricing economics based on actual usage (granularity: seconds to years!)
  - **Elasticity**: Can dynamically add or reduce capacity based on actual workload’s demand
- Infrastructure-as-a-Service (IaaS); Platform-as-a-Service (PaaS); Software-as-a-Service (SaaS)
Cloud Computing

Resources Managed at each Layer:

- **Application Layer**
  - End Users
  - Software as a Service (SaaS)
  - Software Frameworks (Java/Python/.NET)
  - Storage (Database/File)

- **Platform Layer**
  - Software Developers
  - Platform as a Service (PaaS)
  - Virtual Machines

- **Infrastructure Layer**
  - Network/System Administrators
  - Infrastructure as a Service (IaaS)
  - CPU, Memory, Disk
  - Hardware Layer
Q: How to redesign a parallel RDBMS to best exploit the cloud’s capabilities?
Evolution of Cloud Infrastructure
Evolution of Cloud Infrastructure

- **Data Center**: Physical space from which a cloud is operated.
Evolution of Cloud Infrastructure

❖ **Data Center**: Physical space from which a cloud is operated

❖ **3 generations of data centers/clouds:**
  ❖ **Cloud 1.0 (Past)**: Networked servers; user rents/time-sliced access to servers needed for data/software
Evolution of Cloud Infrastructure

- **Data Center**: Physical space from which a cloud is operated

- **3 generations of data centers/clouds:**
  - **Cloud 1.0 (Past)**: Networked servers; user rents/time-sliced access to servers needed for data/software
  - **Cloud 2.0 (Current)**: “Virtualization” of networked servers; user rents amount of resource capacity; cloud provider has a lot more flexibility on provisioning (multi-tenancy, load balancing, more elasticity, etc.)
Evolution of Cloud Infrastructure

- **Data Center**: Physical space from which a cloud is operated

- **3 generations of data centers/clouds**:
  - **Cloud 1.0 (Past)**: Networked servers; user rents/time-sliced access to servers needed for data/software
  - **Cloud 2.0 (Current)**: “Virtualization” of networked servers; user rents amount of resource capacity; cloud provider has a lot more flexibility on provisioning (multi-tenancy, load balancing, more elasticity, etc.)
  - **Cloud 3.0 (Ongoing Research)**: “Serverless” and disaggregated resources all connected to fast networks
Revisiting Parallelism in the Cloud

Shared-Disk Parallelism

Shared-Nothing Parallelism

Interconnect
Revisiting Parallelism in the Cloud

Networks have become much faster: 100GbE to even TbE!

Shared-Disk Parallelism

Shared-Nothing Parallelism
Revisiting Parallelism in the Cloud

Networks have become much faster: 100GbE to even TbE!

Such bundling could under-utilize some resources
Revisiting Parallelism in the Cloud

Networks have become much faster: 100GbE to even TbE!

Such bundling could under-utilize some resources

Q: How to exploit cloud’s virtualization of compute, memory, and storage resources to improve speed and utilization?
The promise of full serverless / resource disaggregation:
All resources (compute, memory, storage) are network-attached and can be elastically added/removed
The promise of full serverless / resource disaggregation:
All resources (compute, memory, storage) are network-attached and can be elastically added/removed
Revisiting Parallelism in the Cloud

The promise of full serverless / resource disaggregation:
All resources (compute, memory, storage) are network-attached and can be elastically added/removed

Get more memory for some phy. ops.
The promise of full serverless / resource disaggregation:
All resources (compute, memory, storage) are network-attached and can be elastically added/removed

Get more memory for some phy. ops.

Get more CPUs to better parallelize aggregates
Revisiting Parallelism in the Cloud

The promise of full serverless / resource disaggregation:
All resources (compute, memory, storage) are network-attached and can be elastically added/removed

Q: How to fulfill the promise with minimal added latency?
Cloud-Native Parallel RDBMSs
Cloud-Native Parallel RDBMSs

❖ Not just running a regular parallel RDBMS on IaaS!
❖ Higher levels (data model, SQL, parser, etc.) preserved
❖ Need to revisit, redesign, and reimplement all internals: storage subsystem, memory management, query processing and optimization, transaction management, and more
Cloud-Native Parallel RDBMSs

- Not just running a regular parallel RDBMS on IaaS!
  - Higher levels (data model, SQL, parser, etc.) preserved
  - Need to revisit, redesign, and reimplement all internals: storage subsystem, memory management, query processing and optimization, transaction management, and more
- Cloud providers, traditional database companies, startups

- Amazon Aurora
- Amazon Redshift
- Amazon Athena
- Google Cloud Spanner
- Azure SQL Database
- Snowflake
Regular MPP (shared-nothing style)

Heterogeneous and elastic compute capacities

Wide variety of storage formats

Spectrum supports ad-hoc remote reads from S3 vs local storage

https://www.intermix.io/blog/amazon-redshift-architecture/#amazon_redshift_architecture_and_the_life_of_a_query
Each virtual warehouse is an independent MPP compute cluster

Compressed columnar format

Shared-disk style + elastic compute

https://docs.snowflake.net/manuals/user-guide/intro-key-concepts.html#snowflake-architecture
Key Example:

Serverless! Remote reads from S3

Schema-on-read ETL not needed Many data formats

Simple interactive queries Federated possible

https://www.xenonstack.com/blog/amazon-athena-quicksight/
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
Outline

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
Beyond RDBMSs: A Brief History
Relational model and RDBMSs are **too restrictive**:

1. “Flat” tables with few data/attribute types
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are **too restrictive**:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are **too restrictive**:  
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first
Relational model and RDBMSs are too restrictive:

1. “Flat” tables with few data/attribute types
2. Restricted language interface (SQL)
3. Need to know schema first
4. Optimized for static dataset
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are **too restrictive**:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first
  4. Optimized for static dataset

But the DB community has addressed these issues already!
Relational model and RDBMSs are too restrictive:

1. “Flat” tables with few data/attribute types

**Object-Relational DBMSs:** UDT, UDFs, text, multimedia, etc.

2. Restricted language interface (SQL)

3. Need to know schema first

4. Optimized for static dataset

But the DB community has addressed these issues already!
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are too restrictive:
  1. “Flat” tables with few data/attribute types

  **Object-Relational DBMSs:** UDT, UDFs, text, multimedia, etc.

  2. Restricted language interface (SQL)

  **PL/SQL:** recursive SQL; embedded SQL; QBE; visual interfaces

  3. Need to know schema first

  4. Optimized for static dataset

But the DB community has addressed these issues already!
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are **too restrictive**:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first
  4. Optimized for static dataset

  **Object-Relational DBMSs**: UDT, UDFs, text, multimedia, etc.

  **PL/SQL**; recursive SQL; embedded SQL; QBE; visual interfaces

  “Schema-later” **semi-structured**: XML+XQuery; JSON; YAML

But the DB community has addressed these issues already!
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are too restrictive:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first
  4. Optimized for static dataset

- Object-Relational DBMSs: UDT, UDFs, text, multimedia, etc.
- PL/SQL; recursive SQL; embedded SQL; QBE; visual interfaces
- “Schema-later” semi-structured: XML+XQuery; JSON; YAML
- Stream data model; “standing” queries; time windows

But the DB community has addressed these issues already!
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are **too restrictive**:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first
  4. Optimized for static dataset

  **Object-Relational DBMSs**: UDT, UDFs, text, multimedia, etc.

  **PL/SQL**: recursive SQL; embedded SQL; QBE; visual interfaces

  **“Schema-later” semi-structured**: XML+XQuery; JSON; YAML

  **Stream** data model; “standing” queries; time windows

**Ad**: Take DSC 104; CSE 132B; CSE 135 on these DB capabilities
So, why did people still need to look beyond RDBMSs?
Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

Google

Amazon
Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

- 4 new concerns of Web giants vs RDBMSs built for enterprises:
Beyond RDBMSs: A Brief History

❖ DB folks got blindsided by the rise of Web/Internet giants

Google  
amazon

❖ 4 new concerns of Web giants vs RDBMSs built for enterprises:

❖ **Developability**: Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

- 4 new concerns of Web giants vs RDBMSs built for enterprises:
  - **Developability**: Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
  - **Fault Tolerance**: Need to scale to 1000s of machines; need for graceful handling of worker failure
Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

- 4 new concerns of Web giants vs RDBMSs built for enterprises:
  - **Developability**: Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
  - **Fault Tolerance**: Need to scale to 1000s of machines; need for graceful handling of worker failure
  - **Elasticity**: Need to be able to easily upsize or downsize cluster size based on workload
Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

- 4 new concerns of Web giants vs RDBMSs built for enterprises:
  - **Developability**: Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
  - **Fault Tolerance**: Need to scale to 1000s of machines; need for graceful handling of worker failure
  - **Elasticity**: Need to be able to easily upsize or downsize cluster size based on workload
  - **Cost**: Commercial RDBMSs licenses too costly; hired own software engineers to build custom new systems
A new breed of parallel data systems called **Dataflow Systems** jolted the DB folks from being smug and complacent!
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
❖ The MapReduce/Hadoop Craze
❖ Spark and Other Dataflow Systems
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
❖ The MapReduce/Hadoop Craze
❖ Spark and Other Dataflow Systems
“Big Data”
“Big Data”

❖ Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”
“Big Data”

- Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”
- Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”
“Big Data”

- Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”
- Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”
- Typical characterization by 3 Vs:
“Big Data”

❖ Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”

❖ Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”

❖ Typical characterization by 3 Vs:

❖ **Volume**: larger-than-RAM; >= TBs, even Exabytes!
“Big Data”

❖ Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”

❖ Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”

❖ Typical characterization by 3 Vs:
  ❖ **Volume**: larger-than-RAM; >= TBs, even Exabytes!
  ❖ **Variety**: relations, webpages, docs, tweets, multimedia, etc.
“Big Data”

❖ Marketing term; think “Big” as in “Big Oil” or “Big Government”, not “big building” or “big car”

❖ Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”

❖ Typical characterization by 3 Vs:
  ❖ **Volume**: larger-than-RAM; >= TBs, even Exabytes!
  ❖ **Variety**: relations, webpages, docs, tweets, multimedia, etc.
  ❖ **Velocity**: high generation rate, e.g., sensors, surveillance, etc.
Why “Big Data” now? 1. Applications
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
Why “Big Data” now? 1. Applications

❖ New “data-driven mentality” in almost all applications:
❖ **Web**: search, e-commerce, e-mails, social media
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
- **Web**: search, e-commerce, e-mails, social media
- **Science**: satellite imagery, CERN’s LHC, document corpora
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
  - **Logistics**: sensors, GPS, “Internet of Things”
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
  - **Logistics**: sensors, GPS, “Internet of Things”
  - **Finance**: high-throughput trading, monitoring
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
  - **Logistics**: sensors, GPS, “Internet of Things”
  - **Finance**: high-throughput trading, monitoring
  - **Humanities**: digitized books/literature, social media
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
  - **Logistics**: sensors, GPS, “Internet of Things”
  - **Finance**: high-throughput trading, monitoring
  - **Humanities**: digitized books/literature, social media
  - **Governance**: e-voting, targeted campaigns, NSA 😊
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - Web: search, e-commerce, e-mails, social media
  - Science: satellite imagery, CERN’s LHC, document corpora
  - Medicine: pharmacogenomics, precision medicine
  - Logistics: sensors, GPS, “Internet of Things”
  - Finance: high-throughput trading, monitoring
  - Humanities: digitized books/literature, social media
  - Governance: e-voting, targeted campaigns, NSA 😊
- …
Why “Big Data” now? 2. Storage

Worldwide Byte Shipments by Storage Media Type

Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018
Outline

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems
Outline

❖ Parallel RDBMSs
❖ Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems aka Dataflow Systems
❖ The MapReduce/Hadoop Craze
❖ Spark and Other Dataflow Systems
What is MapReduce?

What is MapReduce?

- A programming model for parallel programs on **sharded data** + **distributed system** architecture
- **Map** and **Reduce** are terms from functional PL; software/data/ML engineer implements logic of Map, Reduce
- System handles data distribution, parallelization, fault tolerance, etc. under the hood
What is MapReduce?

- A programming model for parallel programs on *sharded data* + *distributed system* architecture
- **Map** and **Reduce** are terms from functional PL; software/data/ML engineer implements logic of Map, Reduce
- System handles data distribution, parallelization, fault tolerance, etc. under the hood
- Created by Google to solve “simple” data workload: index, store, and search the Web!
- Google’s engineers started with MySQL! Abandoned it due to reasons listed earlier (developability, fault tolerance, elasticity, etc.)
What is MapReduce?
What is MapReduce?

❖ **Standard example**: count word occurrences in a doc corpus
❖ **Input**: A set of text documents (say, webpages)
❖ **Output**: A dictionary of unique words and their counts
What is MapReduce?

- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

Hmmm, sounds suspiciously familiar! :)
What is MapReduce?

- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

```java
function map (String docname, String doctext) :
    for each word w in doctext :
        emit (w, 1)
```
What is MapReduce?

❖ **Standard example**: count word occurrences in a doc corpus
❖ **Input**: A set of text documents (say, webpages)
❖ **Output**: A dictionary of unique words and their counts

```java
function map (String docname, String doctext) :
    for each word w in doctext :
        emit (w, 1)
```

*Part of MapReduce API*
What is MapReduce?

- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

```java
function map (String docname, String doctext) :
    for each word w in doctext :
        emit (w, 1)

function reduce (String word, Iterator partialCounts) :
    sum = 0
    for each pc in partialCounts :
        sum += pc
    emit (word, sum)
```

Part of MapReduce API
What is MapReduce?

- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

```java
function map (String docname, String doctext) :
    for each word w in doctext :
        emit (w, 1)

function reduce (String word, Iterator partialCounts) :
    sum = 0
    for each pc in partialCounts :
        sum += pc
    emit (word, sum)
```

Part of MapReduce API
How MapReduce Works
How MapReduce Works

Parallel flow of control and data during MapReduce execution:

The overall MapReduce word count process

- Input
  - Deer Bear River
  - Car Car River
  - Deer Car Bear

- Splitting
  - Deer Bear River
  - Car Car River
  - Deer Car Bear

- Mapping
  - Deer, 1
  - Bear, 1
  - River, 1

- Shuffling
  - Bear, 1
  - Bear, 1
  - Car, 1
  - Car, 1
  - Car, 1

- Reducing
  - Bear, 2
  - Car, 3
  - Deer, 2
  - River, 2

- Final result
  - Bear, 2
  - Car, 3
  - Deer, 2
  - River, 2
Parallel flow of control and data during MapReduce execution:

Under the hood, each **Mapper** and **Reducer** is a separate process; Reducers face barrier synchronization (BSP)
How MapReduce Works

Parallel flow of control and data during MapReduce execution:

The overall MapReduce word count process

Under the hood, each **Mapper** and **Reducer** is a separate process; Reducers face barrier synchronization (BSP)

Fault tolerance achieved using **data replication**
Benefits and Catch of MapReduce
Benefits and Catch of MapReduce

❖ **Goal:** High-level *functional* ops to simplify data-intensive programs
Benefits and Catch of MapReduce

❖ **Goal:** High-level *functional* ops to simplify data-intensive programs
❖ **Key Benefits:**
  ❖ Map() and Reduce() are highly general; any data types/structures; great for ETL, text/multimedia
  ❖ Native scalability, large cluster parallelism
  ❖ System handles fault tolerance automatically
  ❖ Decent FOSS stacks (Hadoop and later, Spark)
Benefits and Catch of MapReduce

❖ **Goal:** High-level *functional* ops to simplify data-intensive programs

❖ **Key Benefits:**
  ❖ Map() and Reduce() are highly general; any data types/structures; great for ETL, text/multimedia
  ❖ Native scalability, large cluster parallelism
  ❖ System handles fault tolerance automatically
  ❖ Decent FOSS stacks (Hadoop and later, Spark)

❖ **Catch:** Users must learn “art” of casting program as MapReduce
  ❖ Map operates record-wise; Reduce aggregates globally
  ❖ But MR libraries now available in many PLs: C/C++, Java, Python, R, Scala, etc.
Abstract Semantics of MapReduce
Abstract Semantics of MapReduce

- **Map()**: Process one “record” at a time *independently*
  - A record can physically *batch* multiple data examples/tuples
  - Dependencies across Mappers *not* allowed
  - *Emit* 1 or more key-value pairs as output(s)
  - Data types of input vs. output can be different
Abstract Semantics of MapReduce

- **Map()**: Process one “record” at a time *independently*
  - A record can physically *batch* multiple data examples/tuples
  - Dependencies across Mappers *not* allowed
  - *Emit* 1 or more key-value pairs as output(s)
  - Data types of input vs. output can be different

- **Reduce()**: Gather all Map outputs across workers sharing same key into an Iterator (list)
  - Apply *aggregation* function on Iterator to get final output(s)
Abstract Semantics of MapReduce

- **Map()**: Process one “record” at a time *independently*
  - A record can physically *batch* multiple data examples/tuples
  - Dependencies across Mappers *not* allowed
  - *Emit* 1 or more key-value pairs as output(s)
  - Data types of input vs. output can be different

- **Reduce()**: Gather all Map outputs across workers sharing same key into an Iterator (list)
  - Apply *aggregation* function on Iterator to get final output(s)

- **Input Split**:
  - Physical-level shard to batch many records to one file “block” (HDFS default: 128MB?)
  - User/application can create *custom* Input Splits
Emulate MapReduce in SQL?
Emulate MapReduce in SQL?

Q: How would you do the word counting in RDBMS / in SQL?
**Q:** How would you do the word counting in RDBMS / in SQL?

- **First step:** Transform text docs into relations and load:
  - Part of the ETL stage
  - Suppose we pre-divide each doc into words w/ schema:
    - **DocWords** (DocName, Word)
Q: How would you do the word counting in RDBMS / in SQL?

❖ **First step:** Transform text docs into relations and load:

   Part of the ETL stage
   Suppose we pre-divide each doc into words w/ schema:
   **DocWords** (DocName, Word)

❖ **Second step:** a single, simple SQL query!

```sql
SELECT Word, COUNT(*) FROM DocWords GROUP BY Word
[ORDER BY Word]
```
Emulate MapReduce in SQL?

Q: How would you do the word counting in RDBMS / in SQL?

❖ **First step:** Transform text docs into relations and load:
  Part of the ETL stage
  Suppose we pre-divide each doc into words w/ schema:
  **DocWords** (DocName, Word)

❖ **Second step:** a single, simple SQL query!

```
SELECT Word, COUNT (*)
FROM DocWords
GROUP BY Word
[ORDER BY Word]
```

Parallelism, scaling, etc. done by RDBMS under the hood
More MR Examples: Select Operation

- **Input Split:**
  - Shard table tuple-wise

- **Map():**
  - On tuple, apply selection condition; if satisfies, emit KV pair with dummy key, entire tuple as value

- **Reduce():**
  - Not needed! No cross-shard aggregation here

- These kinds of MR jobs are called “Map-only” jobs
More MR Examples: Simple Agg.

❖ Suppose it is *algebraic* aggregate (SUM, AVG, MAX, etc.)

❖ **Input Split:**
  ❖ Shard table tuple-wise

❖ **Map():**
  ❖ On agg. attribute, compute incr. stats; emit pair with single global dummy key and incr. stats as value

❖ **Reduce():**
  ❖ Since only one global dummy key, Iterator has *all* suff. stats to unify into global agg.
More MR Examples: GROUP BY Agg.

- Assume it is *algebraic* aggregate (SUM, AVG, MAX, etc.)
- **Input Split:**
  - Shard table tuple-wise
- **Map():**
  - On agg. attribute, compute incr. stats; emit pair with *grouping attribute* as key and stats as value
- **Reduce():**
  - Iterator has all suff. stats *for a single group*; unify those to get result for that group
  - Different reducers will output different groups’ results
More MR Examples: Matrix Norm

- Assume it is *algebraic* aggregate ($L_{p,q}$ norm)
- Very similar to simple SQL aggregates

**Input Split:**
- Shard table tuple-wise

**Map():**
- On agg. attribute, compute incr. stats; emit pair with single global dummy key and stats as value

**Reduce():**
- Since only one global dummy key, Iterator has *all* suff. stats to unify into global agg.
What is Hadoop then?
What is Hadoop then?

- FOSS system implementation with MapReduce as prog. model and HDFS as filesystem
- MR user API; input splits, data distribution, shuffling, fault tolerances handled by Hadoop under the hood
What is Hadoop then?

- FOSS system implementation with MapReduce as prog. model and HDFS as filesystem
- MR user API; input splits, data distribution, shuffling, fault tolerances handled by Hadoop under the hood
- Exploded in popularity in 2010s: 100s of papers, 10s of products
- A “revolution” in scalable+parallel data processing that took the DB world by surprise
- But nowadays Hadoop largely supplanted by Spark
What is Hadoop then?

❖ FOSS system implementation with MapReduce as prog. model and HDFS as filesystem
❖ MR user API; input splits, data distribution, shuffling, fault tolerances handled by Hadoop under the hood
❖ Exploded in popularity in 2010s: 100s of papers, 10s of products
❖ A “revolution” in scalable+parallel data processing that took the DB world by surprise
❖ But nowadays Hadoop largely supplanted by Spark

**NB:** Do not confuse MR for Hadoop or vice versa!
A Spectacular “War of the Worlds”
A Spectacular “War of the Worlds”

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | Trackbacks (1)

MapReduce has been lauded as the key technology for the new paradigm of cloud computing. This recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of low-end machines to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshmen how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is...
A Spectacular “War of the Worlds”

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | Trackbacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our comments on MapReduce [1]. The recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of lower-end servers to address a computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on
MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | Trackbacks (1)

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here with our response to that request. In particular, the recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of commodity servers to solve computing problems. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on
A Spectacular “War of the Worlds”

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our response to this reader. The rapid-fire rate of change in the recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of a small number of servers to solving the computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

No declarativity!
Filesca-based!
A Spectacular “War of the Worlds”

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | Trackbacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our response. In recent years, Facebook and other companies have been focusing considerable effort on data intensive computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all – it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

No declarativity!
Files wiping engines!
DeWitt’s work on parallel DBMSs!
MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our response to it. The recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of low-performance processing units to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. **A giant step backward in the programming paradigm** for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMSs
5. Incompatible with all of the tools DBMS users have come to depend on

No declarativity!

Filescan-based!

DeWitt’s work on parallel DBMSs!

Cheap rip-off of RDBMSs!
“Young Turks” vs. “Old Guard”? 
Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on
“Young Turks” vs. “Old Guard”?

Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on
"Young Turks" vs. "Old Guard"?

Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
“Young Turks” vs. “Old Guard”?

Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level **text ETL**

Meant for “offline” fault-tolerant workloads on cheap nodes
Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
Meant for “offline” fault-tolerant workloads on cheap nodes
Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL

Meant for “offline” fault-tolerant workloads on cheap nodes

Google awarded a patent for MapReduce (ahem)!
Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
Meant for “offline” fault-tolerant workloads on cheap nodes
Google awarded a patent for MapReduce (ahem)!
“Young Turks” vs. “Old Guard”?

Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
Meant for “offline” fault-tolerant workloads on cheap nodes
Google awarded a patent for MapReduce (ahem)!
MapReduce/Hadoop not meant to be an RDBMS replacement
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:
Enter Hybrid Systems!

❖ Clever DB researches: “Let’s get the best of both worlds!”
❖ Numerous projects on hybrid systems in industry/academia:

*Programming model-level*: Bring declarativity from RDBMS world to MapReduce/Hadoop world
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  **Programming model-level**: Bring declarativity from RDBMS world to MapReduce/Hadoop world

  **Systems-level**: Intermix system implementation ideas
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  - *Programming model-level*: Bring declarativity from RDBMS world to MapReduce/Hadoop world
  - *SQL dialect* over Hadoop

  - *Systems-level*: Intermix system implementation ideas
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  **Programming model-level**: Bring declarativity from RDBMS world to MapReduce/Hadoop world

  SQL dialect over Hadoop

  Dataflow language over Hadoop

  **Systems-level**: Intermix system implementation ideas
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  **Programming model-level**: Bring declarativity from RDBMS world to MapReduce/Hadoop world

  - SQL dialect over Hadoop
  - Dataflow language over Hadoop

  **Systems-level**: Intermix system implementation ideas

  HadoopDB from Yale U.
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  **Programming model-level**: Bring declarativity from RDBMS world to MapReduce/Hadoop world

  - SQL dialect over Hadoop
  - Dataflow language over Hadoop

  **Systems-level**: Intermix system implementation ideas

  - HadoopDB from Yale U.
  - Microsoft Polybase
Enter Hybrid Systems!

- Clever DB researches: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

**Programming model-level:** Bring declarativity from RDBMS world to MapReduce/Hadoop world

- SQL dialect over Hadoop
- Dataflow language over Hadoop

**Systems-level:** Intermix system implementation ideas

- HadoopDB from Yale U.
- Microsoft Polybase
“Big Data” / Dataflow Systems

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems
“Big Data” / Dataflow Systems

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems
Apache Spark
Apache Spark

- **Dataflow programming** model (subsumes most of RA; MR)
  - Inspired by Python Pandas style of chaining functions
  - Unified storage of relations, text, etc.; custom programs
  - System impl. (re)designed from scratch
Apache Spark

- **Dataflow programming** model (subsumes most of RA; MR)
  - Inspired by Python Pandas style of chaining functions
  - Unified storage of relations, text, etc.; custom programs
  - System impl. (re)designed from scratch
- Tons of sponsors, gazillion bucks, unbelievable hype!
- **Key idea vs Hadoop**: exploit distributed memory to cache data
- **Key novelty vs Hadoop**: lineage-based fault tolerance
- Open-sourced to Apache; commercialized as Databricks
Distributed Architecture of Spark

https://spark.apache.org/docs/latest/cluster-overview.html
## Spark’s Dataflow Programming Model

**Transformations** are relational ops, MR, etc. as functions

**Actions** are what force computation; aka *lazy evaluation*

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map(f : T \rightarrow U)</code></td>
<td>RDD[T] \rightarrow RDD[U]</td>
</tr>
<tr>
<td><code>filter(f : T \rightarrow Bool)</code></td>
<td>RDD[T] \rightarrow RDD[T]</td>
</tr>
<tr>
<td><code>flatMap(f : T \rightarrow Seq[U])</code></td>
<td>RDD[T] \rightarrow RDD[U]</td>
</tr>
<tr>
<td><code>sample(fraction : Float)</code></td>
<td>RDD[T] \rightarrow RDD[T] (Deterministic sampling)</td>
</tr>
<tr>
<td><code>sortByKey()</code></td>
<td>RDD[(K, V)] \rightarrow RDD[(K, Seq[V])]</td>
</tr>
<tr>
<td><code>reduceByKey(f : (V, V) \rightarrow V)</code></td>
<td>RDD[(K, V)] \rightarrow RDD[(K, V)]</td>
</tr>
<tr>
<td><code>union()</code></td>
<td>(RDD[T],RDD[T]) \rightarrow RDD[T]</td>
</tr>
<tr>
<td><code>join()</code></td>
<td>(RDD[(K, V)],RDD[(K, W)]) \rightarrow RDD[(K, (V, W))]</td>
</tr>
<tr>
<td><code>cogroup()</code></td>
<td>(RDD[(K, V)],RDD[(K, W)]) \rightarrow RDD[(K, (Seq[V], Seq[W]))]</td>
</tr>
<tr>
<td><code>crossProduct()</code></td>
<td>(RDD[T],RDD[U]) \rightarrow RDD[(T, U)]</td>
</tr>
<tr>
<td><code>mapValues(f : V \rightarrow W)</code></td>
<td>RDD[(K, V)] \rightarrow RDD[(K, W)] (Preserves partitioning)</td>
</tr>
<tr>
<td><code>sort(c : Comparator[K])</code></td>
<td>RDD[(K, V)] \rightarrow RDD[(K, V)]</td>
</tr>
<tr>
<td><code>partitionBy(p : Partitioner[K])</code></td>
<td>RDD[(K, V)] \rightarrow RDD[(K, V)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count()</code></td>
<td>RDD[T] \rightarrow Long</td>
</tr>
<tr>
<td><code>collect()</code></td>
<td>RDD[T] \rightarrow Seq[T]</td>
</tr>
<tr>
<td><code>reduce(f : (T,T) \rightarrow T)</code></td>
<td>RDD[T] \rightarrow T</td>
</tr>
<tr>
<td><code>lookup(k : K)</code></td>
<td>RDD[(K, V)] \rightarrow Seq[V] (On hash/range partitioned RDDs)</td>
</tr>
<tr>
<td><code>save(path : String)</code></td>
<td>Outputs RDD to a storage system, e.g., HDFS</td>
</tr>
</tbody>
</table>
Word Count Example in Spark

Spark RDD API available in Python, Scala, Java, and R

```python
from pyspark import SparkContext

text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```scala
import org.apache.spark.{SparkContext, RDD}

val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

```java
import org.apache.spark.RDD;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import java.util.Arrays;
import java.util.Iterator;

JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

Ad: Take Yoav’s CSE 255 to learn more Spark programming
Word Count Example in Spark

Spark RDD API available in Python, Scala, Java, and R

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
   .map(lambda word: (word, 1))
   .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
   .map(word => (word, 1))
   .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
   .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
   .mapToPair(word -> new Tuple2<>(word, 1))
   .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");

Spark DataFrame API of SparkSQL offers an SQL interface
Can also interleave SQL with DF-style function chaining!

**Ad:** Take Yoav’s CSE 255 to learn more Spark programming
Spark DF API and SparkSQL

- Databricks now recommends SparkSQL/DataFrame API; avoid RDD API unless really needed!
- **Key Reason:** Automatic query optimization becomes more feasible
  - AKA (painfully) re-learn 40 years of database systems research! :)

Query Optimization in Spark

- Common automatic query optimizations (from RDBMS world) are now performed in Spark’s Catalyst optimizer:
  - **Projection pushdown:**
    - Drop unneeded columns early on
  - **Selection pushdown:**
    - Apply predicates close to base tables
  - **Join order optimization:**
    - Not all joins are equally costly
  - Fusing of aggregates
  - ...

Query Optimization in Spark

```python
def add_demographics(events):
    u = sqlCtx.table("users")
    events \n        .join(u, events.user_id == u.user_id) \n        .withColumn("city", zipToCity(df.zip)) # Run udf to add city column
    events = add_demographics(sqlCtx.load("/data/events", "parquet"))
    training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```

Databricks is building yet another parallel RDBMS! :)

Reinventing the Wheel?
Spark-based Ecosystem of Tools

The Berkeley Data Analytics Stack (BDAS)
New Paradigm of Data “Lakehouse”

❖ **Data “Lake”**: *Loose coupling* of data file format and data/query processing stack (vs RDBMS’s tight coupling); many frontends

(a) *First-generation platforms.*

(b) *Current two-tier architectures.*

(c) *Lakehouse platforms.*

If interested, check out this vision paper on the future of data lakes and data lakehouses:

… which too is a form of DBMS! :)

Time for my annual rant on data science systems. 😈

<rant>
A data lakehouse is a DBMS.
A DataFrame system is a DBMS.
An ML platform is a DBMS.
A model store is a DBMS.
A feature store is a DBMS.

Those who fail to learn (+ves & -ves) from history are doomed to repeat it.
</rant>
... which too is a form of DBMS! :)

While I agree with you, note that there's nothing wrong with "repeat and relearn". Many use cases' first order problems are not "data independence" (which is about change), but about being able to get the task done first.

A data
A Data
An ML
A model
A feature

Those are doom,
</rant>

Yes, agreed on your point. I put it more colorfully. 👇 A wheel is a wheel but we don't use an Egyptian chariot wheels for a car nor a car's for a space shuttle. 😊
References and More Material

❖ MapReduce/Hadoop:

❖ Spark:
  ❖ Online Guide: https://spark.apache.org/docs/2.1.0/sql-programming-guide.html
“Big Data” / Dataflow Systems

- Parallel RDBMSs
- Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems aka Dataflow Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems