Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters

Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, D. Stott Parker

presented by Nate Roberts
Outline

1. Introduction: principles of databases rather than the artifacts.
2. MapReduce
3. Map-Reduce-Merge: extending MapReduce
4. Using Map-Reduce-Merge to implement relational algebra operators
Principles of DB, Not the Artifacts

• New data-processing systems should consider alternatives to using big, traditional databases.

• MapReduce does a good job, in a limited context, with extraordinary simplicity.

• Map-Reduce-Merge will try to extend the applicability without giving up too much simplicity.
Introduction to MapReduce

1. Why MapReduce?
2. What is MapReduce?
3. How do you use it?
4. What’s it good for?
5. What are its limitations?
Why MapReduce?

• For (single core) CPUs, Moore’s Law is beginning to slow down

• The future is multi-core (large clusters of commodity hardware are the new supercomputers)

• But parallel programming is hard to think about!
What is MapReduce?

• MapReduce handles dispatching tasks across a large cluster

• You just have to define the tasks, in two stages:

  1. Map: (k1, v1) → [(k2, v2)]
  2. Reduce: (k2, [v2]) → [v3]
How do you use it?

• Example: count the number of occurrences of each word in a large collection of documents. For each document d with contents v:

  • map: given (d, v), for each word in v, emit (w, 1).

  • reduce: given (w, [v]), sum the counts in [v]. Emit the sum.
Word Counting

Behind the Scenes

• A single master server dispatches tasks and keeps a scoreboard.

1. Mappers are dispatched for each document. They do local writes with their results.

2. Once mapping finishes, a shuffle phase assigns reducers to each word. Reducers do remote reads from mappers.
MapReduce in Parallel

Three Optimizations

• Fault and slow node tolerance: once tasks are all dispatched and some nodes are finished, assign some unfinished tasks redundantly. (First to return wins.)

• Combiner: have mappers do some local reduction.

• Locality: assign mappers in such a way that most have their input available for local reading.
What’s it good for?

• Data processing tasks on homogeneous data sets:
  • Distributed Grep
  • Building an index mapping words to documents in which those words occur.
  • Distributed sort
What isn’t it good for?

• Not good at “heterogeneous” data sets.
Heterogeneous Data

<table>
<thead>
<tr>
<th>emp-id</th>
<th>dept-id</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>innov. award ($100)</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>hard worker ($50)</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>high perform. ($150)</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>innov. award ($100)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dept-id</th>
<th>bonus adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1.1</td>
</tr>
<tr>
<td>A</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Map-Reduce-Merge: Extending MapReduce

1. Change to reduce phase
2. Merge phase
3. Additional user-definable operations
   a. partition selector
   b. processor
   c. merger
   d. configurable iterators
Reduce & Merge Phases

1. Map: \((k_1, v_1) \rightarrow [(k_2, v_2)]\)

2. Reduce: \((k_2, [v_2]) \rightarrow [v_3]\)
   becomes:

   1. Map: \((k_1, v_1) \rightarrow [(k_2, v_2)]\)

   2. Reduce: \((k_2, [v_2]) \rightarrow (k_2, [v_3])\)

   3. Merge: \(((k_2, [v_3]), (k_3, [v_4])) \rightarrow (k_4, [v_5]))\)

Note that there are TWO sets of mappers & reducers, whose outputs are combined in the merge phase.
Programmer-Definable Operations

1. Partition selector - which data should go to which merger?

2. Processor - process data on an individual source.

3. Merger - analogous to the map and reduce definitions, define logic to do the merge operation.

4. Configurable iterators - how to step through each of the lists as you merge.

Why not implement processor as part of reduce?
## Employee Bonus Example, Revisited

The image illustrates the process of calculating employee bonuses across different departments. Here's a breakdown of the steps involved:

### LHS Mapper Computes Emp Bonuses
- **Table**: `emp-id`, `dept-id`, `bonus`
- **Data**:
  - `emp-id` | `dept-id` | `bonus`
  - 1 | B | $100
  - 1 | D | $80
  - 2 | A | $0
  - 3 | A | $150
  - 3 | A | $100

### RHS Mapper Retrieves Bonus Adjustment
- **Table**: `dept-id`, `bonus adjustment`
- **Data**:
  - B | 1.1
  - A | 0.9

### RHS Reducer Modified Bonus Adjustment and Sorts on Dept-id
- **Table**: `dept-id`, `bonus adjustment`
- **Data**:
  - B | 1.15
  - A | 0.95

### LHS Reducer Sorts on (dept-id, emp-id) Pair and Sums Up Emp Bonuses
- **Table**: `emp-id`, `dept-id`, `bonus sum`
- **Data**:
  - 2 | A | $80
  - 3 | A | $250
  - 3 | A | $150

### Match Keys on Dept-id
- **Table**: `dept-id`, `bonus adjustment`
- **Data**:
  - B | 1.15
  - A | 0.95

### Sort-Merge Merger Joins LHS and RHS Reduced Outputs, Then Computes Final Emp Bonuses
- **Table**: `emp-id`, `bonus`
- **Data**:
  - 2 | $80
  - 3 | $257.5
  - 1 | $172.5

This process ensures that bonuses are accurately calculated and distributed based on departmental performance and individual contributions.
Implementing Relational Algebra Operations

1. Projection
2. Aggregation
3. Selection
4. Set Operations: Union, Intersection, Difference
5. Cartesian Product
6. Rename
7. Join
Projection

- All we have to do is emit a subset of the data passed in.
- Just a mapper can do this.
Aggregation

• By choosing appropriate keys, can implement “group by” and aggregate SQL operators in MapReduce.

• (Do have to be careful here, though: choose badly, and you might not have enough tasks for MapReduce to do you any good.)
Selection

• If selection condition involves only the attributes of one data source, can implement in mappers.

• If it’s on aggregates or a group of values contained in one data source, can implement in reducers.

• If it involves attributes or aggregates from both data sources, implement in mergers.
Set Union

- Let each of the two MapReduces emit a sorted list of unique elements.
- Merges just iterate simultaneously over the lists:
  - store the lesser value and increment its iterator, if there is a lesser value.
  - if the two are equal, store one of the two, and increment both iterators.
Set Intersection

- Let each of the two MapReduces emit a sorted list of unique elements
- Merges just iterate simultaneously over the lists:
  - if there is a lesser value, increment its iterator
  - if the two are equal, store one of the two, and increment both iterators
Set Difference

To Compute A - B:

- Let each of the two MapReduces emit a sorted list of unique elements
- Merges just iterate simultaneously over the lists:
  - if A’s value is less than B’s, store A’s
  - if B’s value is less than A’s, increment it
  - if the two are equal, increment both
Cartesian Product

• Set the reducers up to output the two sets you want the Cartesian product of.

• Each merger will get one partition F from the first set of reducers, and the full set of partitions S from the second.

• Each merger emits F x S.
Rename

- Trivial
Sort-Merge Join

- Map: partition records into key ranges according to the values of the attributes on which you’re sorting, aiming for even distribution of values to mappers.
- Reduce: sort the data.
- Merge: join the sorted data for each key range.
Hash Join

- Map: use the same hash function for both sets of mappers.
- Reduce: produce a hash table from the values mapped.
- Merge: operates on corresponding hash buckets. Use one bucket as a build set, and the other as a probe.
Nested Loop Join

• Just like a hash join, except in the merge step, do a nested loop, scanning the right-hand relation for matches to the left.
Conclusion

- MapReduce & GFS represent a paradigm shift in data processing: use a simplified interface instead of overly-general DBMS
- Map-Reduce-Merge adds the ability to execute arbitrary relational algebra queries
- Next steps: develop SQL-like interface and a query optimizer
References


- A. Kimball, Problem Solving on Large-Scale Clusters. Lecture given on July 3, 2007. Available at http://www.youtube.com/watch?v=-vD6PUdf3Js