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

- I. 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

- I. 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: $(kl, vl) \rightarrow [(k2, v2)]$
 - 2. Reduce: $(k2, [v2]) \rightarrow [v3]$

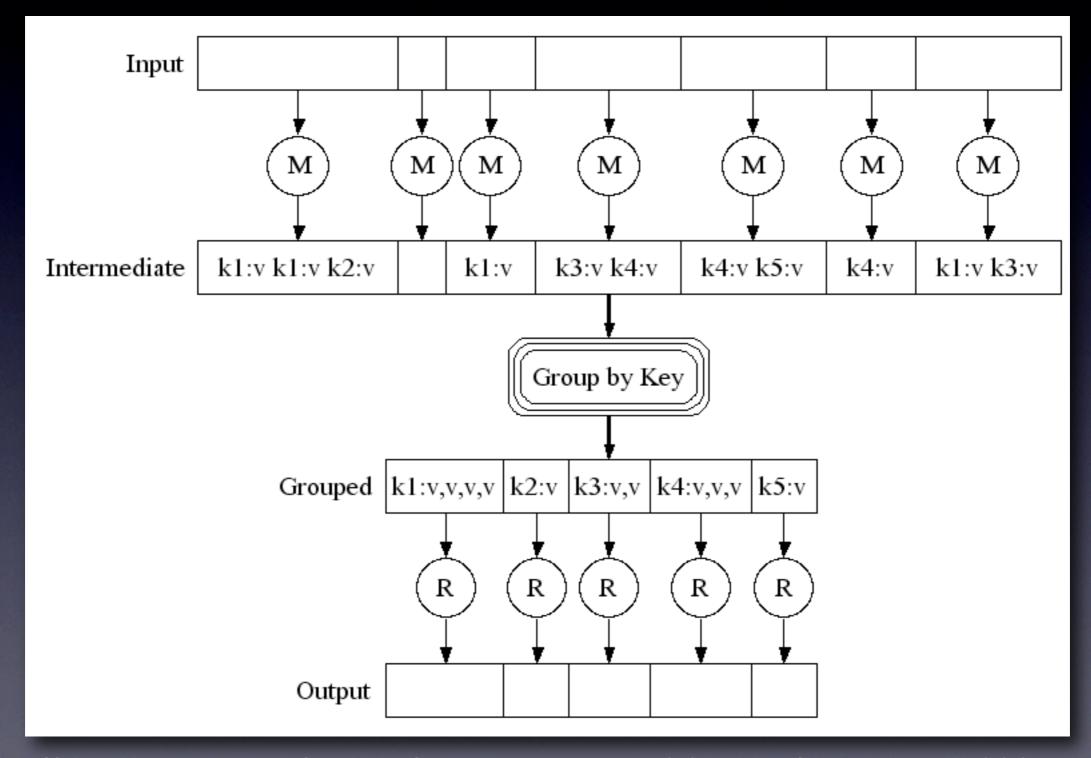
How do you use it?

- 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, I).
 - reduce: given (w, [v]), sum the counts in [v]. Emit the sum.

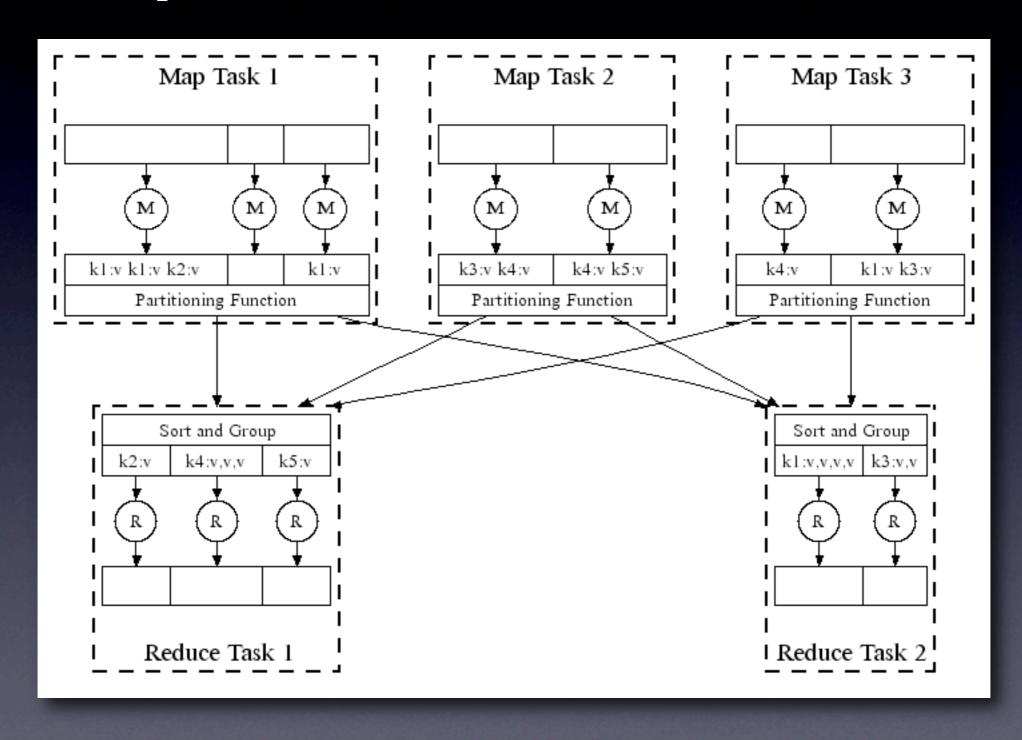
Word Counting Behind the Scenes

- A single master server dispatches tasks and keeps a scoreboard.
 - I. 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 Schematic



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

emp-id	dept-id	bonus
	В	innov. award (\$100)
	В	hard worker (\$50)
2	A	high perform. (\$150)
3	A	innov. award (\$100)

dept-id	bonus adjustment
В	1.1
A	0.9

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: $(k1, v1) \rightarrow [(k2, v2)]$
- 2. Reduce: (k2, [v2]) → [v3] becomes:
- I. Map: $(kI, vI) \rightarrow [(k2, v2)]$
- 2. Reduce: $(k2, [v2]) \rightarrow (k2, [v3])$
- 3. Merge: $((k2, [v3]), (k3, [v4])) \rightarrow (k4, [v5])$

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

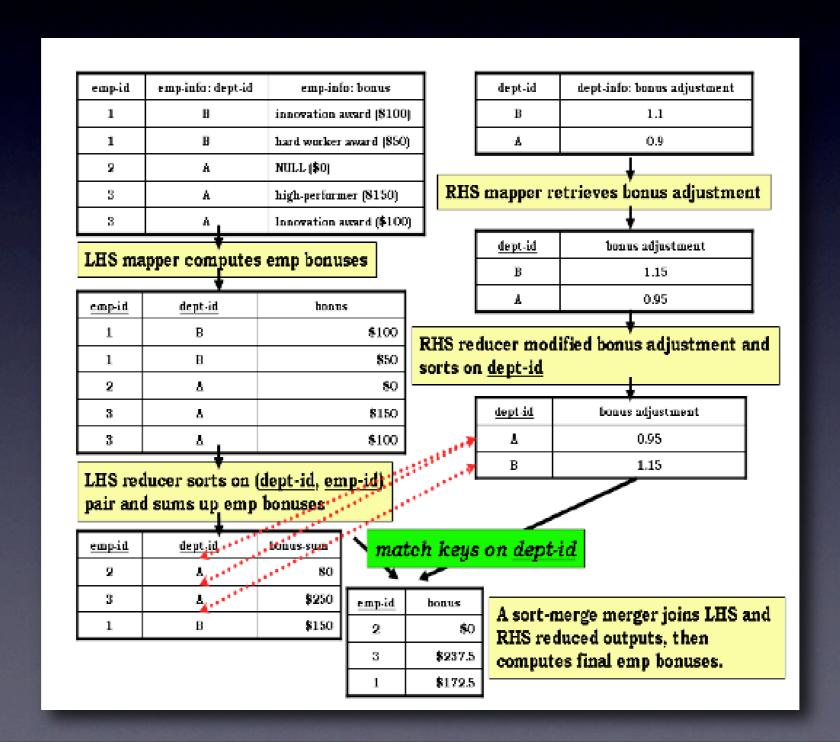
Programmer-Definable Operations

- I. Partition selector which data should go to which merger?
- 2. Processor process data on an individua source.

Why not implement processor as part of reduce?

- 3. Merger analogous to the map and reduded definitions, define logic to do the merge operation.
- 4. Configurable iterators how to step through each of the lists as you merge.

Employee Bonus Example, Revisited



Implementing Relational Algebra Operations

- I. 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 th values mapped.
- Merge: operates on corresponding has buckets. Use one bucket as a build set, and the other as a probe.

Note sure whether reduce really has any

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

- J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, pages 137-150, 2004. Slides available: http://labs.google.com/papers/mapreduce-osdi04-slides/index.html
- A. Kimball, Problem Solving on Large-Scale Clusters.
 Lecture given on July 3, 2007. Available at http://www.youtube.com/watch?v=-vD6PUdf3Js