Concepts and Models of Parallel and Data-centric Programming

MapReduce Design Patterns – Data Organization Patterns

Lecture, Summer 2020

Simon Schwitanski
Dr. Christian Terboven

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Outline

0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. Distributed Shared Memory
7. Parallel Algorithms
8. Parallel I/O
9. **MapReduce**
10. Apache Spark

a. MapReduce Programming Model
b. Parallelizing MapReduce
c. Hadoop Ecosystem
d. Hadoop Distributed File System
e. Yet Another Resource Negotiator
f. Comparison to Other Approaches
g. MapReduce Design Patterns
    a. Summarization Patterns
    b. Filtering Patterns
    c. Data Organization Patterns

Data Organization Patterns (1)

- Reorganization of data often crucial for performance
- Partitioning, sharding and sorting can significantly affect runtime
- Structured to Hierarchical pattern
  - Create new records from data which has originally another structure
  - Migration from RDBMS to MapReduce: Transforming row-based data into hierarchical format like JSON or XML → More flexible, avoids joins
  - Example: Grouping a StackOverflow question together with answers and all comments

Data Organization Patterns (2)

- Partitioning pattern
  - Move records into categories (“partitioning”)
  - Uses partitioner of MapReduce framework and identity mapper and reducer
  - Write custom partitioner to change partitioning
  - Example: Partition users by last access date

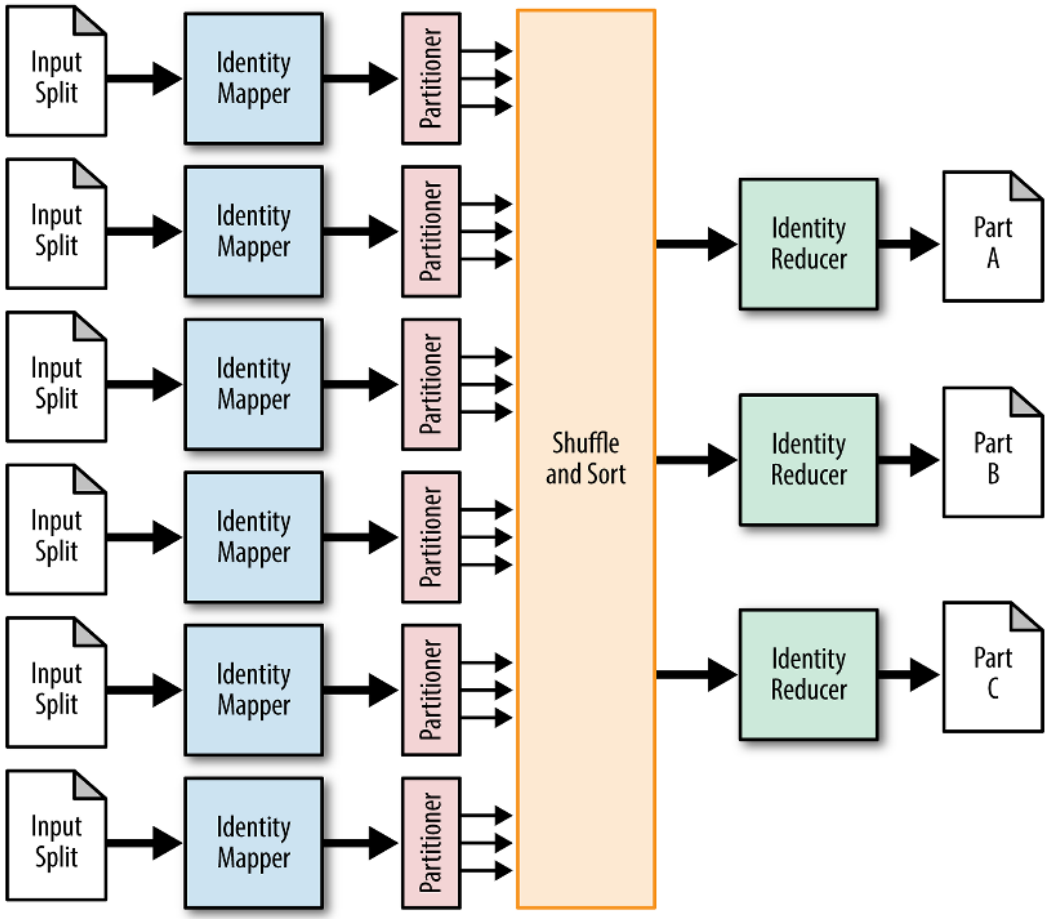

Illustration: Miner, Donald and Shook, Adam. “MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems”, p.84, O'Reilly Media, 2012

Data Organization Patterns (3)

- Binning pattern
  - Similar to partitioner pattern, but binning takes place directly in the mapper
  - No reduce phase
  - Each mapper produces one output file for each bin (huge number of files depending on number of bins and map tasks)

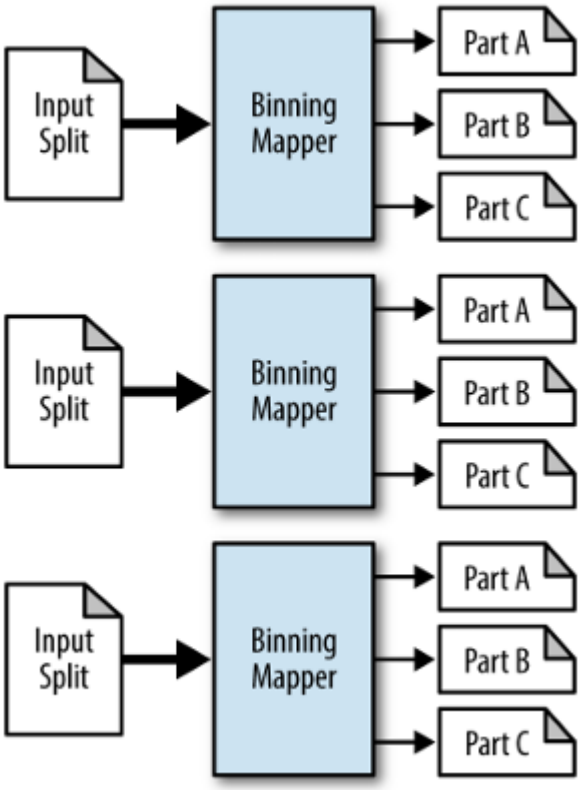

Illustration: Miner, Donald and Shook, Adam. "MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems", p.89, O'Reilly Media, 2012

Data Organization Patterns (4)

- Total Order Sorting pattern
  - Sorting data in parallel on a given sort key
  - Sorting in MapReduce not easy, because distributing total order sorting between tasks is difficult
  - Each Reducer sorts data by key (data sorted per partition), but no global sorting across all data
  - Main idea: Write custom partitioner that partitions data by sort key: Lowest range to first reducer, next range to second reducer, …
  - Challenge: Find suitable ranges for values
- Shuffling pattern
  - Randomizing set of records (opposite of total order sorting pattern)
  - Idea: Mapper outputs record with a random key, reducer sorts random keys and thus further randomizes data
  - Application: Anonymization

Concluding Remarks

Limitations of Hadoop MapReduce

- No support of caching data in memory, only disk-based
  - Each MapReduce job: Read from local disk, processing, write back to the file system
  - MapReduce job working on previous output has to read from local disk again
  - Serious performance penalty for iterative algorithms
  - Problem: Most data mining and machine learning algorithms are iterative
- Focus on batch processing, interactive analysis inefficient
- Inflexibility: Algorithms have to be broken down to *Map* and *Reduce* functions

Alternatives

- Specialized frameworks
  - HaLoop [1]: Iterative MapReduce interface (loop-aware scheduler and caching)
  - Pregel [2]: Model for iterative graph applications (keeps intermediate data in memory)
- Generalized framework: Apache Spark [3]
  - Implements a caching feature → Allows to reuse the outputs of previous operations
  - Beneficial for iterative algorithms
  - Interactive analysis supported → Continuously querying data possible
  - Processing of real-time data → Data from real time event streams (e.g., Twitter)

[1] Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst. “HaLoop: Efficient Iterative Data Processing on Large Clusters” PVLDB 3(1): 285-296 (2010)

[2] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski. “Pregel: a system for large-scale graph processing” SIGMOD Conference 2010: 135-146

[3] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. “Spark: Cluster Computing with Working Sets” HotCloud 2010

What you have learnt

What you have learnt

- MapReduce design patterns: Blueprints for solving common problems in reusable and general way
  - Summarization
  - Filtering
  - Data Organization
- Limitations of MapReduce: Only support for batch processing
  - Iterative jobs often limited by disk I/O, no caching supported
  - No interactive analyses possible
- Generalization of MapReduce approach: Apache Spark