MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Published by Google Inc. 2004

Presented by Sri Julapally, Sanjay Mohandas, Sujay Jagadeesh, Suraj Geddam

Georgia Tech

Background and Motivation

Processing large datasets is inherently complex and inefficient due to the sheer size of data being processed.

- Researchers at Google aimed to create a new paradigm based on principles from functional programming to address this challenge of processing large amounts of data using distributed computing.
- Employs powerful clusters of computers to efficiently process and return data based on the users specification of the map and reduce functions

Traditional Solutions to Large Data Processing

Many traditional systems aim to use existing paradigms to solve this issue, which results in some pitfalls due to inefficiencies

Batch Processing

- Dependent on software/hardware
- Not done in real time, not well suited for analysis

RDBMS

- Data is inflexible (must conform to schema)
- Same issues when it comes to large data processing, not optimized for distributed/parallel computing

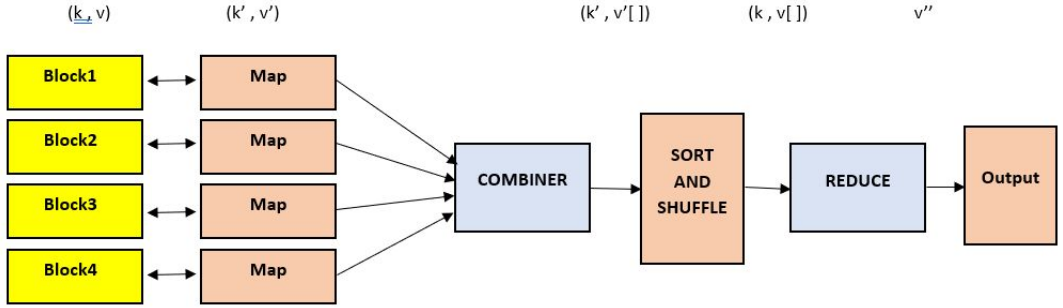
Main Idea of Paper

This paper introduces the MapReduce programming model.

- Map: maps key value pairs to intermediate key value pairs
- Reduce: reduces intermediate key value pairs to resultant data output which can take any form

Utilizes computing clusters to process and return data

While relatively simple, these two functions used in tandem result in very powerful tools for data processing and analysis.



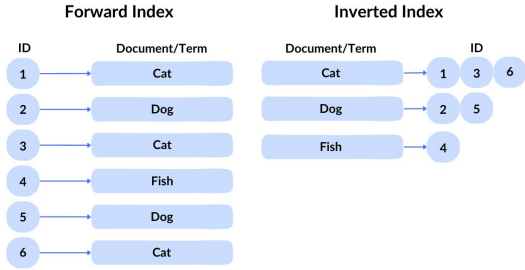
Acknowledgement:GeeksforGeeks. (2023, May 31). *Map reduce in Hadoop*. GeeksforGeeks. https://www.geeksforgeeks.org/map-reduce-in-hadoop/

Example: Inverted Index

An inverted index is similar to the index in the back of books. This is traditionally an involved process, requiring normalizing and tokenizing the words, creating intermediate data structures to store these with the corresponding IDs, and then restructuring the data to result in the inverted index.

With MapReduce its as simple as:

- Map: map <lineNum, line of text> to <text, lineNum>
- Reduce: <text, lineNum> -> <text, sortedLineNums>

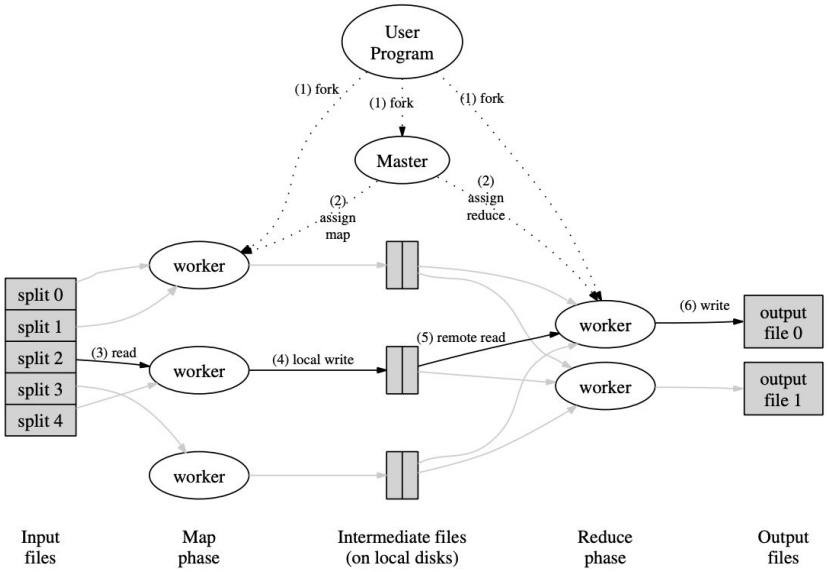


Acknowledgement: Otten, N. V. (2023, November 14). *How to implement inverted indexing [top 10 tools & future trends]*. Spot Intelligence. https://spotintelligence.com/2023/10/30/inverted-indexing/

Technical Details

Implementation: Execution Overview

- MapReduce library splits input files into M pieces (16-64 MB each).
- Program instances started on a cluster: one master and many workers.
- Master assigns map and reduce tasks to idle workers.
- Map tasks process input data and generate intermediate key/value pairs.
- Intermediate pairs are buffered and periodically written to local disks.
- Reduce tasks read intermediate data, sort it, and apply the Reduce function.
- Final output stored in R output files, one per reduce task.



Acknowledgement: Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Georgia Tech

Implementation: Master Data Structures & Fault Tolerance

- Master tracks the state and location of map and reduce tasks.
- Fault tolerance: Worker failures detected via periodic pings.
- Completed map tasks re-executed if a worker fails.
- Reduce tasks read data from completed map tasks; re-execution managed by the master.
- Atomic commits ensure consistency of output data.

Implementation: Locality & Task Granularity

- Locality: MapReduce schedules tasks close to data locations to minimize network transfer.
- Task granularity: M and R chosen to be much larger than the number of workers for load balancing.
- Large M and R values improve fault recovery and dynamic task assignment.

Implementation: Backup Tasks

- Backup tasks launched for remaining in-progress tasks to handle stragglers.
- Significantly reduces completion time with minimal additional resource usage.
- Example: Sort program completion time increased by 44% without backup tasks.

Refinements: Partitioning Function & Ordering Guarantees

- Partitioning function controls how intermediate data is distributed among reduce tasks.
- Custom partitioning functions can be provided for specific needs (e.g., URL partitioning by hostname).
- Ordering guarantees: Intermediate key/value pairs processed in increasing key order within each partition.

Refinements: Combiner Function

- Combiner function used for local aggregation to reduce data transfer.
- Operates on each machine that performs a map task.
- Often the same code is used for both the combiner and the reduce functions.

Input/Output Types: Side Effects & Skipping Bad Records

- Support for various formats, including text and binary. Custom formats can be implemented.
- Skipping bad records: Mechanism to skip records causing deterministic crashes, ensuring progress.
- Side effects: User code can produce additional outputs, but atomicity and idempotence must be ensured.

Input/Output Types: Status Information & Counters

- Status information: Master's internal HTTP server provides progress updates, worker status, and task details.
- Counters: Used for tracking occurrences of events (e.g., total words processed). Automatically maintained by the library.
- Example: Counters used to verify that output pairs match input pairs or to track document processing statistics.

```
Counter* uppercase;
uppercase = GetCounter("uppercase");

map(String name, String contents):
  for each word w in contents:
    if (IsCapitalized(w)):
      uppercase->Increment();
    EmitIntermediate(w, "1");
```

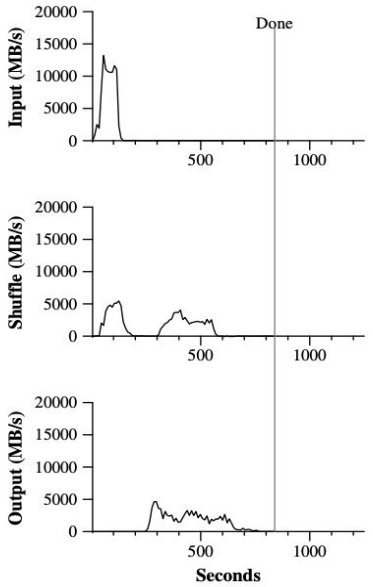
Evaluation

Key Observations

- **Scalability**: MapReduce efficiently processes large datasets across thousands of machines, showcasing its adaptability to varying computational demands.
- **Data Processing Rate**: The system achieves impressive data processing rates, reaching up to 30 GB/s in the Grep experiment and maintaining high rates in the Sort operation.
- **Fault Tolerance**: MapReduce's robustness is evident in its ability to handle machine failures, ensuring the reliability of the computation.
- **Efficiency of Backup Tasks**: The introduction of backup tasks significantly reduces the impact of stragglers, leading to faster and more uniform job completion times.
- **Impact on Real-world Applications**: The use of MapReduce in large-scale indexing at Google highlights its effectiveness in practical scenarios beyond theoretical use cases.

Detailed Analysis of Sort Experiment and Figure 3(a)

- **Experiment Setup**: Sorting 1 terabyte of data across approximately 1800 machines, with data split into 64MB chunks.
- **Data Processing Phases:** The operation includes reading input, shuffling data, and writing sorted output, as illustrated in Figure 3(a).
- **Performance Metrics**: Peak input reading rate of 13 GB/s, shuffling rate of 10 GB/s, and output writing rate of 2-4 GB/s.
- **Efficiency of Backup Tasks**: Backup tasks significantly reduce the tail of the job completion time, improving overall efficiency.
- **Key Observations**: The sort experiment demonstrates MapReduce's capability to handle large-scale data sorting with high efficiency and fault tolerance.



(a) Normal execution

Acknowledgement: Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Georgia Tech

Cluster Configuration and Grep Performance

- **Cluster Specs Summary**: Google's cluster for MapReduce includes dual-processor x86 systems with 4GB of RAM and gigabit Ethernet, providing robust support for data-intensive tasks.
- **Grep Experiment Overview**: A 10TB dataset was used to evaluate MapReduce's search capabilities, demonstrating swift processing of extensive text data to locate a simple pattern.
- **Performance Metrics Highlight**: MapReduce reached a peak rate of 30 GB/s during the Grep task, underscoring its ability to efficiently tackle large-scale data operations.
- **Graph of Data Transfer Rate (Figure 2)**: A sharp spike in data transfer rates is shown in Figure 2, exemplifying MapReduce's rapid scaling to full operational capacity and sustained performance.

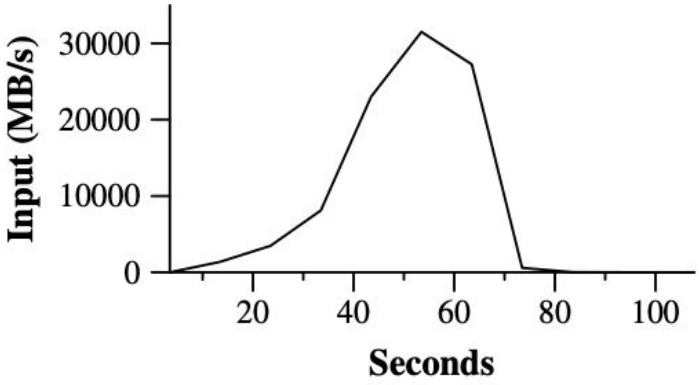


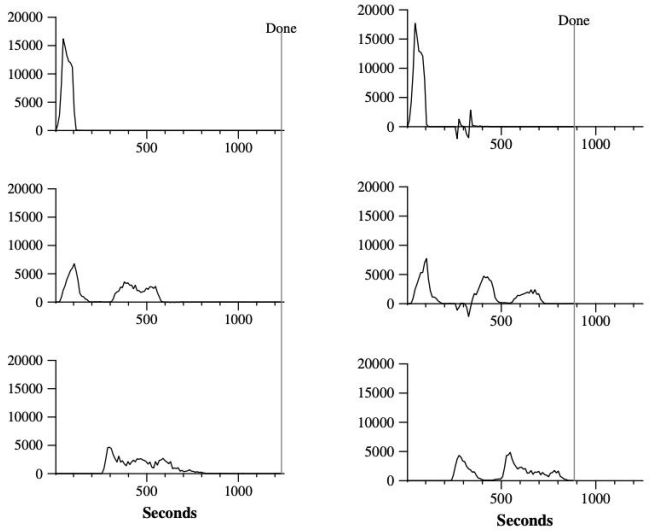
Figure 2: Data transfer rate over time

Acknowledgement: Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Georgia Tech

Resilience in MapReduce: Backup Tasks and Machine Failures

- **Backup Tasks Efficiency**: Backup tasks in MapReduce preempt straggler delays, effectively accelerating completion times for robust system throughput.
- **Machine Failure Management**: Built to cope with the reality of machine failures, MapReduce's automated task rerouting ensures continuous operation, highlighting its inherent fault tolerance.
- **Graphical Depiction (Refer to Figure 3)**:
    1. In Figure 3(b), observe the prolonged job times without backup tasks.
    2. Figure 3(c) illustrates MapReduce's resilience, with minimal impact on job times despite machine failures.



(b) No backup tasks

(c) 200 tasks killed

Acknowledgement: Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Georgia Tech

MapReduce in Practice: Applications and Experience

- **Large-Scale Data Processing:** MapReduce has been integral in processing extensive datasets for various Google products, demonstrating its versatility and reliability in production environments.
- **Indexing System Rewrite:** A noteworthy application of MapReduce is the complete rewrite of Google's indexing system. This task highlights the simplification of complex distributed computations and the subsequent benefits in operation and maintenance.
- **User-Friendly Framework:** The adoption of MapReduce by a wide range of developers, even those with no prior experience in parallel and distributed systems, underscores the framework's accessibility and ease of use.
- **Ecosystem Growth:** The growing ecosystem around MapReduce, including the development of open-source implementations like Apache Hadoop, reflects its widespread acceptance and the community's recognition of its value in big data processing.

Related Works

- HDFS Architecture Guide (updated 2022)
- Hadoop, The Definitive Guide (2009)
- The Google File System (2003)
- Spark: Cluster Computing with Working Sets (2010)

Conclusion

- Main Contributions:
  - Simplified data processing on large clusters
  - Automatic parallelization and distribution
  - Fault-tolerance mechanism
  - Scalability and efficiency in processing large data sets
- Limitations:
  - Not optimized for real-time data processing
  - Can be resource-intensive for small data sets
  - Limited to specific types of data processing tasks
- Future Work:
  - Enhancing real-time processing capabilities
  - Optimizing resource usage for various data sizes
  - Expanding the applicability to a broader range of tasks

Study Questions

Question 1: Application of MapReduce Counter

- In the context of MapReduce counter facility, which allows for tracking the occurrences of various events, how would a map function use a counter to keep track of a specific condition, such as the number of capitalized words within a dataset?

Question 2: Locality Optimization

- What is locality optimization in the context of MapReduce, and why is it important? How does MapReduce achieve this optimization, and what benefits does it bring to the processing of large data sets?