MapReduce: Simplified Data Processing on Large Clusters

Khue Le and Izn E Allah

Intro

Google tackles vast amounts of raw data, from web documents to user logs.
The goal: Extract valuable information like inverted indices, web document structures, and user query patterns.

Challenge:

While the computations are conceptually simple, the sheer volume of data demands distribution across numerous machines.

Parallelizing tasks, managing data distribution, and ensuring fault tolerance introduce complexities.

Solution: MapReduce:

A new abstraction inspired by functional programming concepts like 'map' and 'reduce'.

This model simplifies large-scale computations, ensuring automatic parallelization and fault tolerance through re-execution.

Programming model

Core Principle:

- MapReduce is a functional programming model designed to process and generate large datasets with a parallel, distributed algorithm on a cluster.

Two Key Functions:

1. **Map:** Takes input pairs and produces a set of intermediate key/value pairs.
   - Example: If the task is to count the number of occurrences of words in a document, the map function breaks down the document into words and creates a key/value pair for each word, typically (word, 1).
2. **Reduce:** Merges all intermediate values associated with the same key.
   -Example: For the word counting task, the reduce function would sum up the values for each word, producing an output of (word, total count).

Data Flow:

- Input data is split into chunks and processed by the map tasks in parallel.
- The MapReduce library groups the intermediate key/value pairs by keys.
- The reduce tasks then operate on the grouped data, producing the final output.

Sample code

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

The map function emits each word plus an associated count of occurrences (just ‘1’ in this simple example). The reduce function sums together all counts emitted for a particular word

Implementation

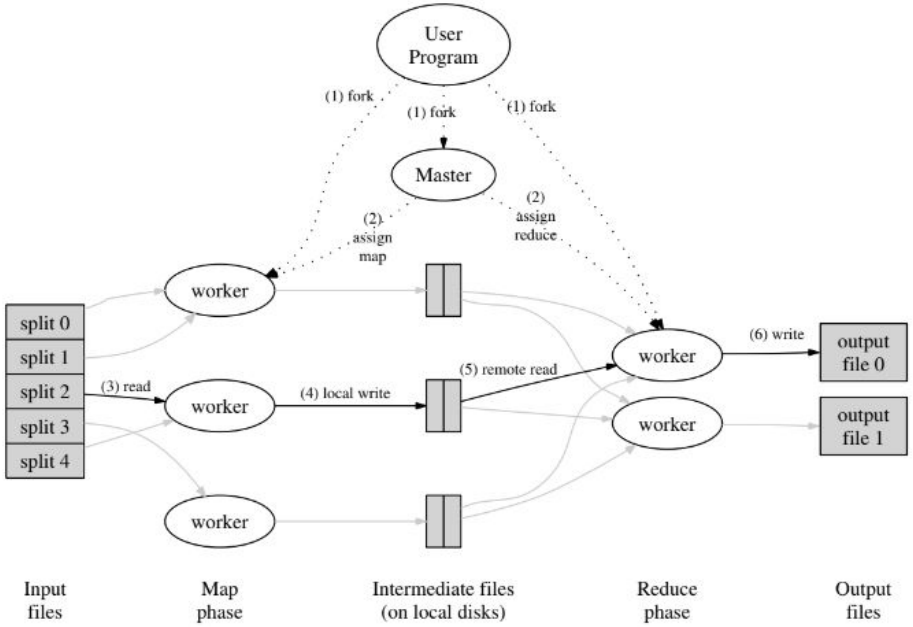


Figure 1: Execution overview

Fault tolerant

- Worker failure: Master ping workers periodically. If no response is received from a worker in a certain amount of time, the master marks the worker as fail. The task is then assigned to another worker and the whole progress restart again. Complete map task are re-executed and complete reduce task do not.
- MapReduce is resilient to large-scale worker failures.
- Master failure: Unlikely, but If the master task dies, a new copy can be started from the last checkpointed state. But most of the time the implementation aborts the MapReduce computation if master fails.
- Back-up tasks : When a MapReduce operation is close to completion, the master schedules backup executions of the remaining in-progress tasks.

Refinement

- **Partitioning Function:**
  - Users specify desired number of reduce tasks/output files.
  - Default partitioning uses hashing for balanced partitions.
  - Custom partitioning functions allow for specific data groupings, e.g., grouping URLs by host.

- **Ordering Guarantees:**
  - Within a partition, key/value pairs are processed in increasing key order.
  - Facilitates sorted output files for efficient key-based lookups.

- **Combiner Function:**
  - For tasks with repetitive intermediate keys, a combiner function can partially merge data before network transfer.
  - Acts on each machine performing a map task, often using the same code as the reduce function.

- **Input and Output Types:**
  - Supports various data formats, from text to key/value pairs.
  - Users can define custom readers for unique data sources, like databases.

Refinement

Side-effects:
- Allows auxiliary files as additional outputs.
- Ensures atomic and idempotent side-effects, typically using temporary files.

Skipping Bad Records:
- Mechanism to skip records causing deterministic crashes.
- Helps in making forward progress despite bugs in user code.

Local Execution:
- For debugging, an alternative implementation executes all work sequentially on the local machine.

Status Information:
- Master provides an HTTP server displaying computation progress, worker failures, and task outputs.

Counters:
- Facility to count event occurrences, e.g., number of words processed.
- Useful for sanity checks and monitoring live computation progress.

Performance

- Compare the performance of on two computations running on a large cluster of machines.
- Grep program search for rare 3-character pattern approximately 1 terabyte of data
- The sort program also sort approximately 1 terabyte of data, 10^10 100-byte records

- Grep program data transfer rate:

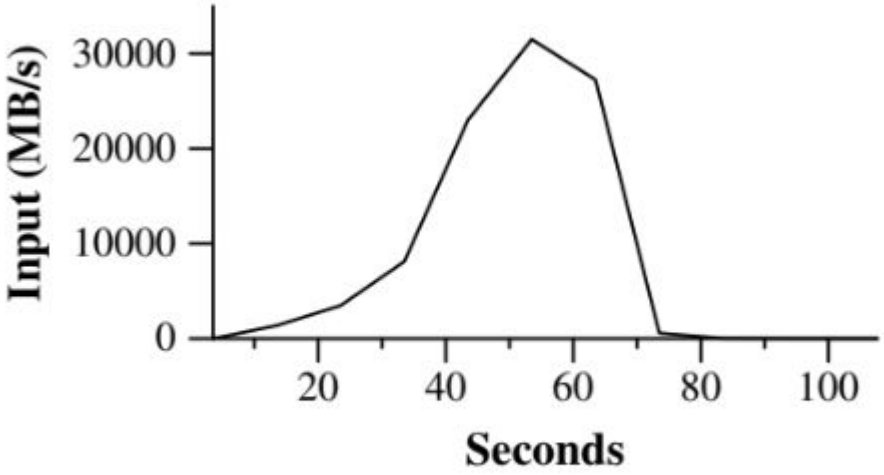
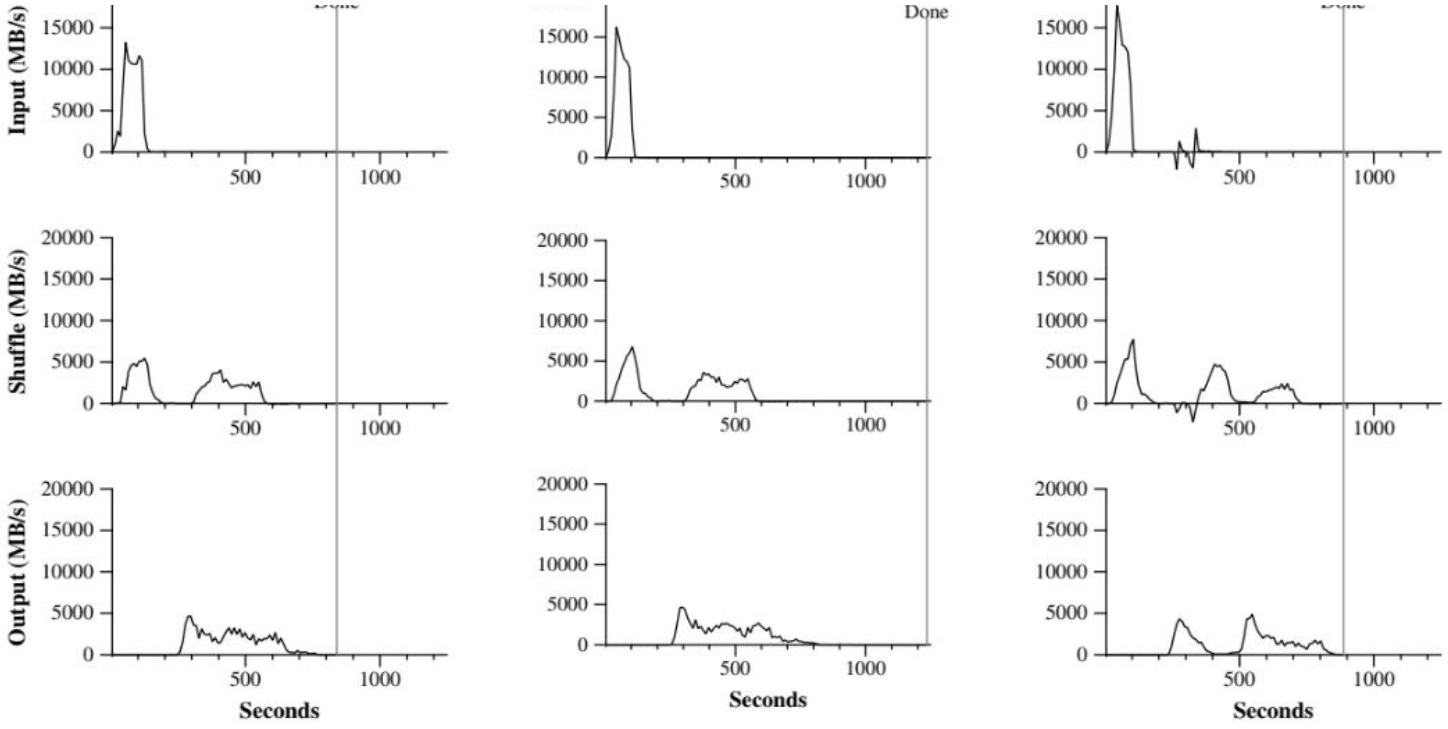


Figure 2: Data transfer rate over time



(a) Normal execution (b) No backup tasks (c) 200 tasks killed

Figure 3: Data transfer rates over time for different executions of the sort program

Experience and Application

- Large-scale machine learning problems
- Clustering problems for the Google News and Google products
- Extraction of data used to produce reports of popular queries (e.g. Google Zeitgeist)
- Extraction of properties of web pages for new experiments and products (e.g. extraction of geographical locations from a large corpus of web pages for localized search)
- Large-scale graph computations.

Conclusion

- Easy to use, even for programmers without experience with parallel and distributed systems
- A large variety of problems are easily expressible as MapReduce computations: web search service, sorting, data mining, machine learning,etc.
- Solving large computational problem with the resources from thousands of machines.

Thank you!