Map Reduce

: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

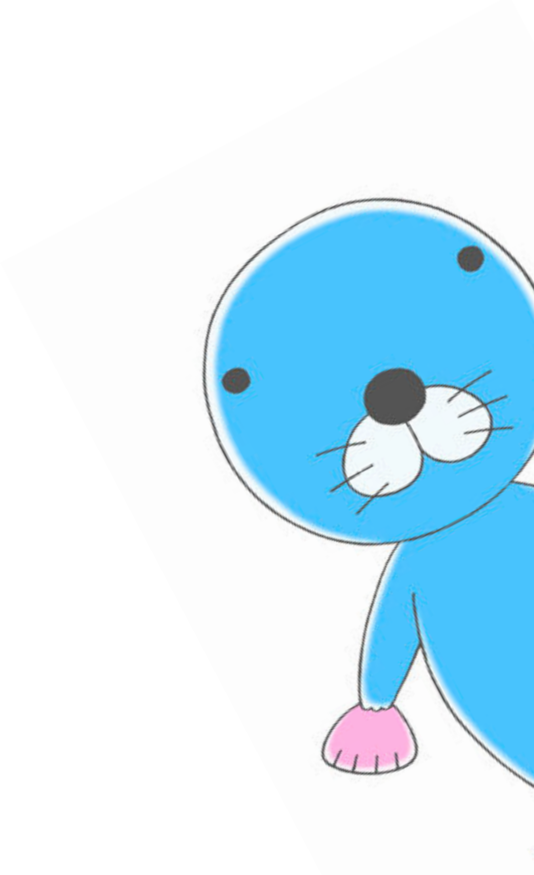
Google, Inc

presented by Joon-Hyun Jeong

Outline

1. Intro
2. Implementation of Map Reduce
3. Refinements
4. Results
5. Conclusion

Intro



Motivation

- **Processing large amounts of raw data in reasonable time**

  - How to parallelize the computation?
  - How to distribute the data?
  - How to handle machine failures?

- **Observation: most google computations applied map, reduce operation**

Where to go?

Functional model with user specified map and reduce operations

- easily parallelize large computations
- fault tolerance by re-execution mechanism

Programming model with example

```
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));
```

input to Map function: (k1, v1)

output from Map function: list (k2, v2)

Input to Reduce function: (k2, list (v2))

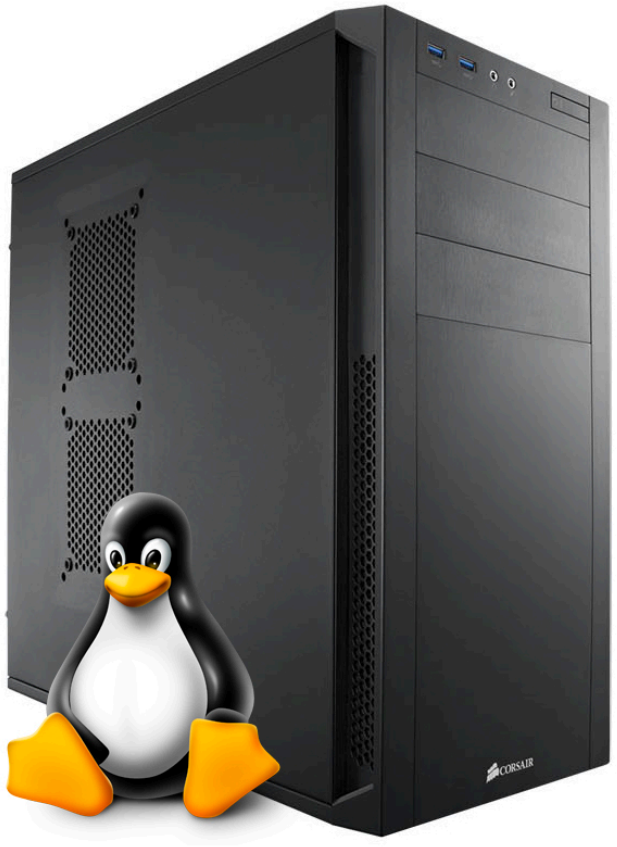
Output from Reduce function: list(v2)

Implementation

of Map Reduce

Implementation environment

A Cluster has hundreds or thousands of commodity machines.



CPU: dual-processor x86 processors

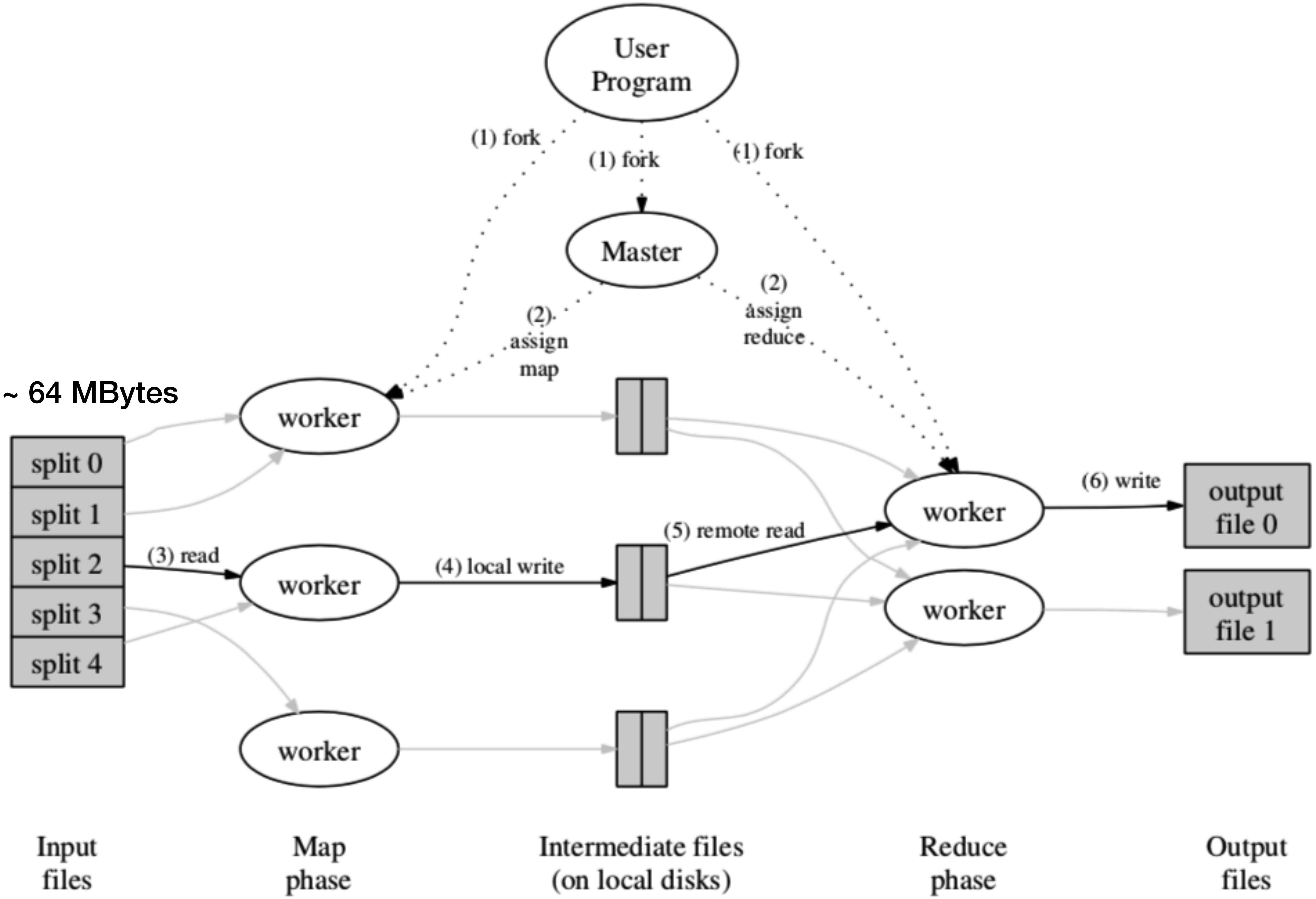
OS: Linux

RAM: 2-4 GB

Storage: inexpensive IDE disks

Execution overview

16 ~ 64 MBytes



Worker failure tolerance

- Every failed worker and map task is reset to Idle state by Master for rescheduling.

- Completed map tasks are re-executed on a failure while completed reduce tasks aren’t.

  - writing to local disks or global file system.

Master failure tolerance

- Writing periodic checkpoints of the master data structures.

- New copy from the last checkpoint state of Master - using this, master can be restarted if master dies.

Locality

- To minimize network bandwidth, Replication of the input data block is used.

  - master schedules a map task on a machine that contains a replica or is near a replica.

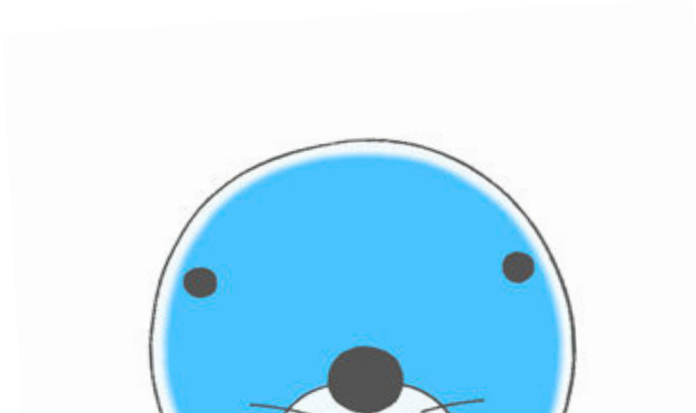
Backup tasks

- When MapReduce operation is close to completion, the master schedules backup executions of the remaining in-progress tasks.

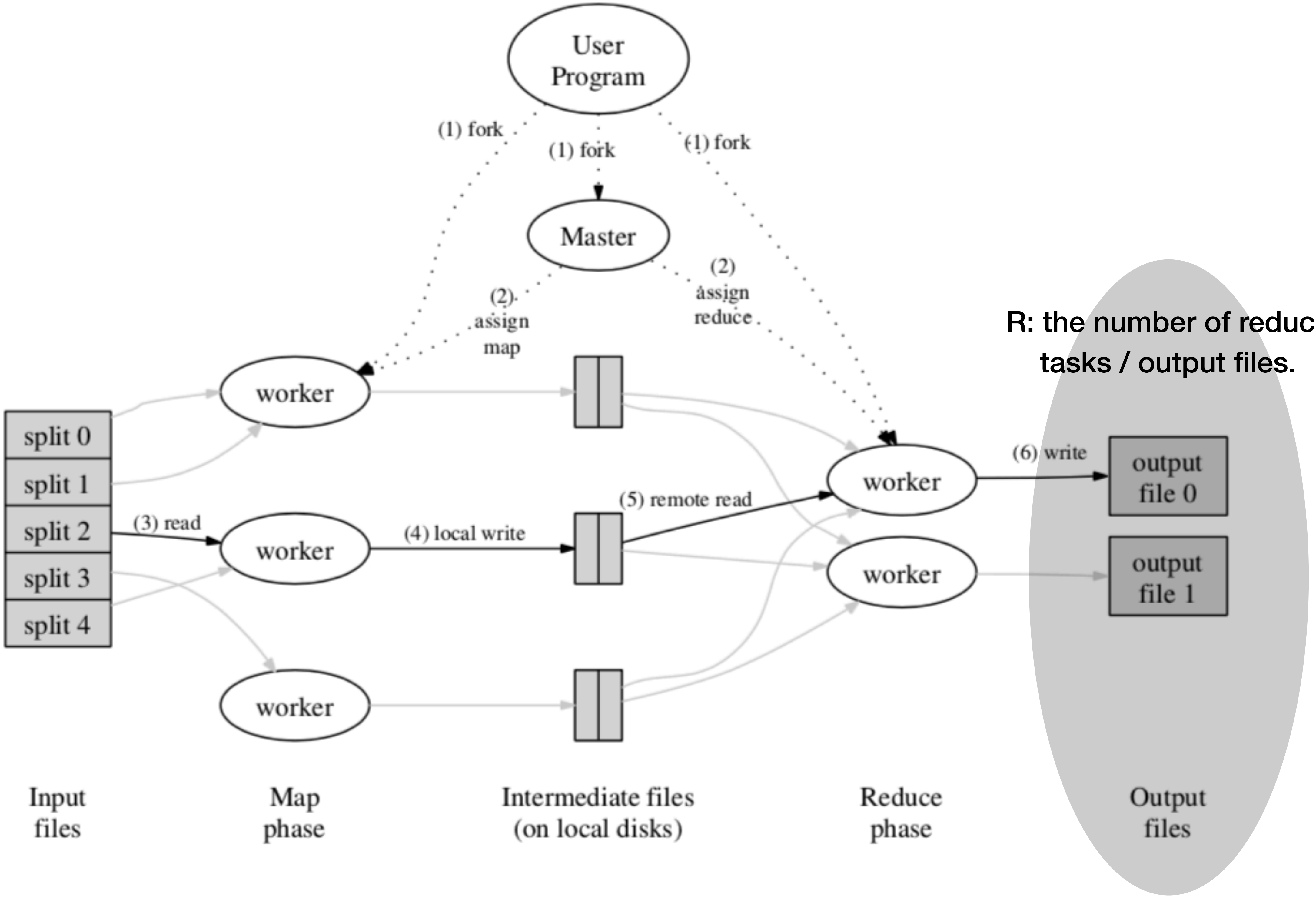
  - Because remaining in progress tasks can be stragglers, which can increase overall run time.

sorting program takes 44% shorter to complete using Backup tasks!

Refinements



Partitioning Function



R: the number of reduce tasks / output files.

Partitioning Function

- Default partitioning function : $Hash(key)\ mod\ \ R$

- special partitioning function: $Hash(Hostname(urlKey)\ mod\ \ R$

  - In specific cases (e.g. all entries for a single host to end up in the same output file)

Combiner Function

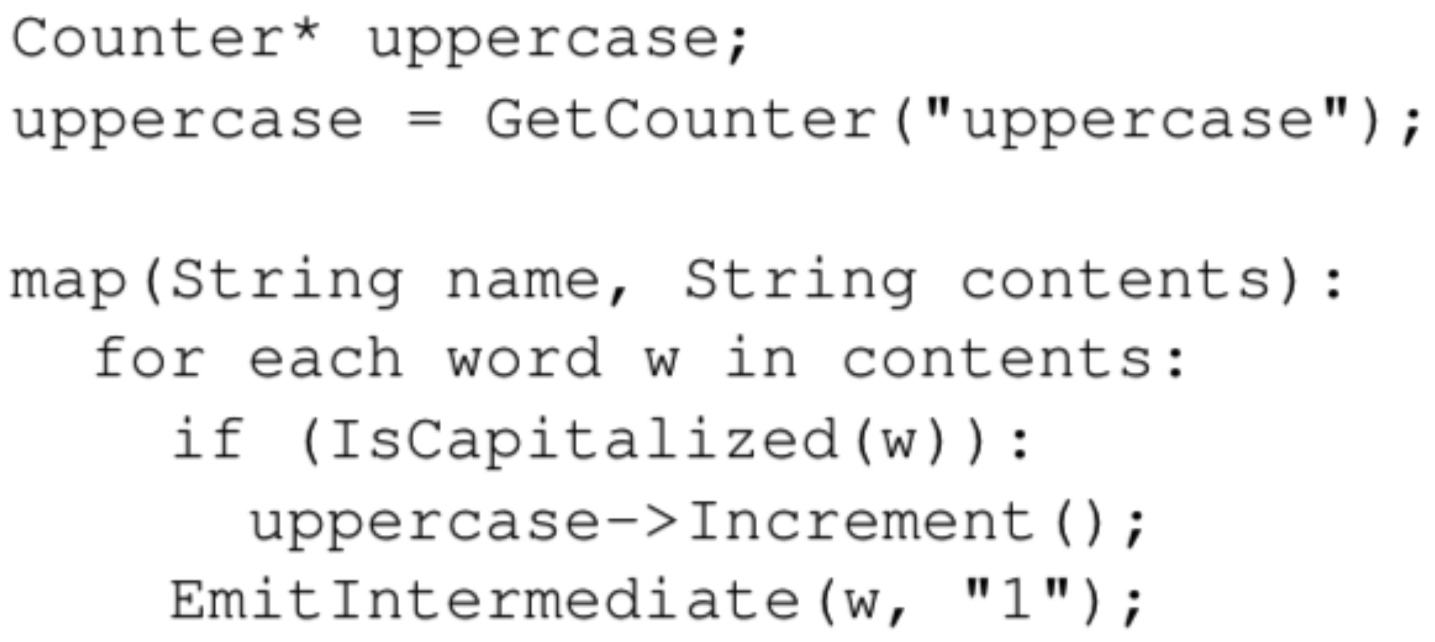
- Partial merging of thousands of records of intermediate key value pair

  - Written to an intermediate file before reduce task.

- Combiner function code and Reduce function code is exactly the same.

Significantly speeds up!

Counters



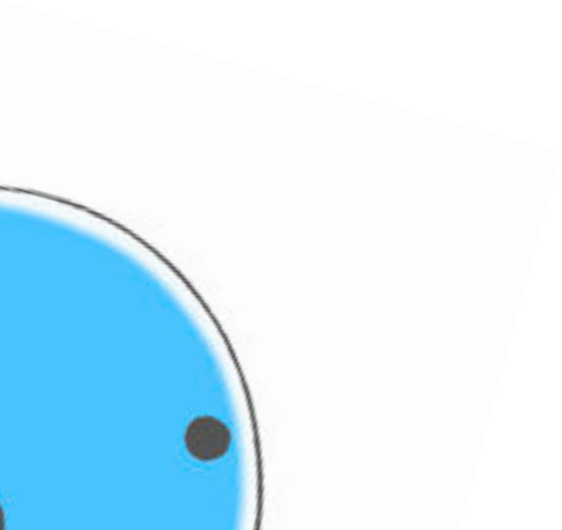
```
Counter* uppercase;
uppercase = GetCounter("uppercase");

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

- A facility to count occurrences of various events.
- Master aggregates the counter values from successful map and reduce tasks.

Useful for sanity checking the behavior of MapReduce operations.

Results



Cluster configuration

1800 commodity pcs

- 2GHz Intel Xeon cpu with hyper threading
- 4GB RAM
- 160GB IDE disks
- Gigabit Ethernet link

Performance

input data split: 64MB * 15000

output file: 1

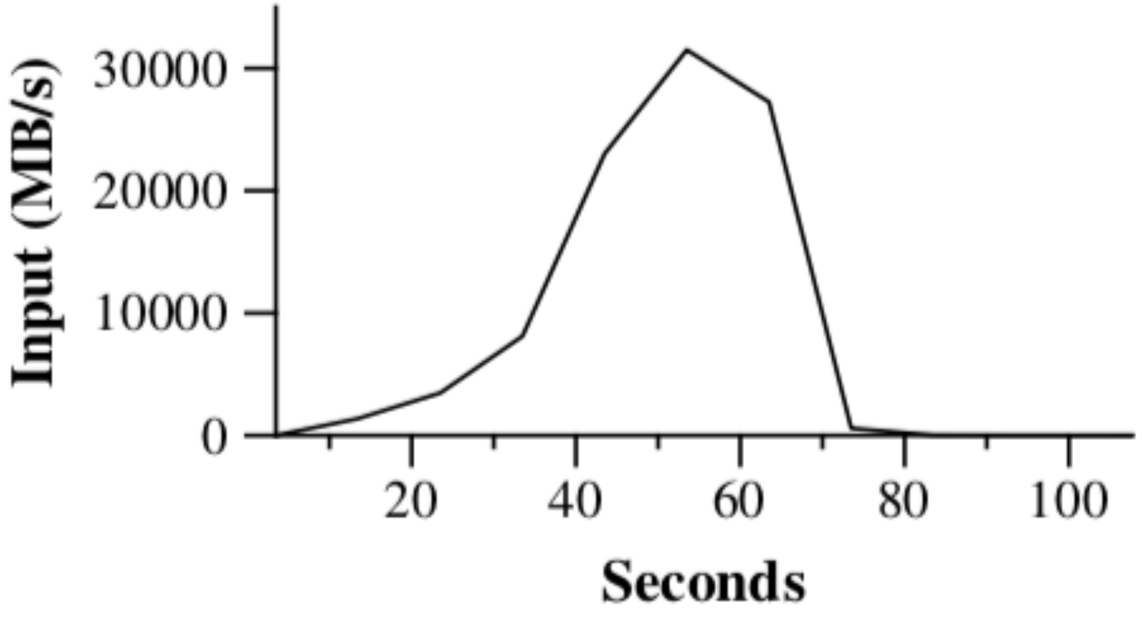
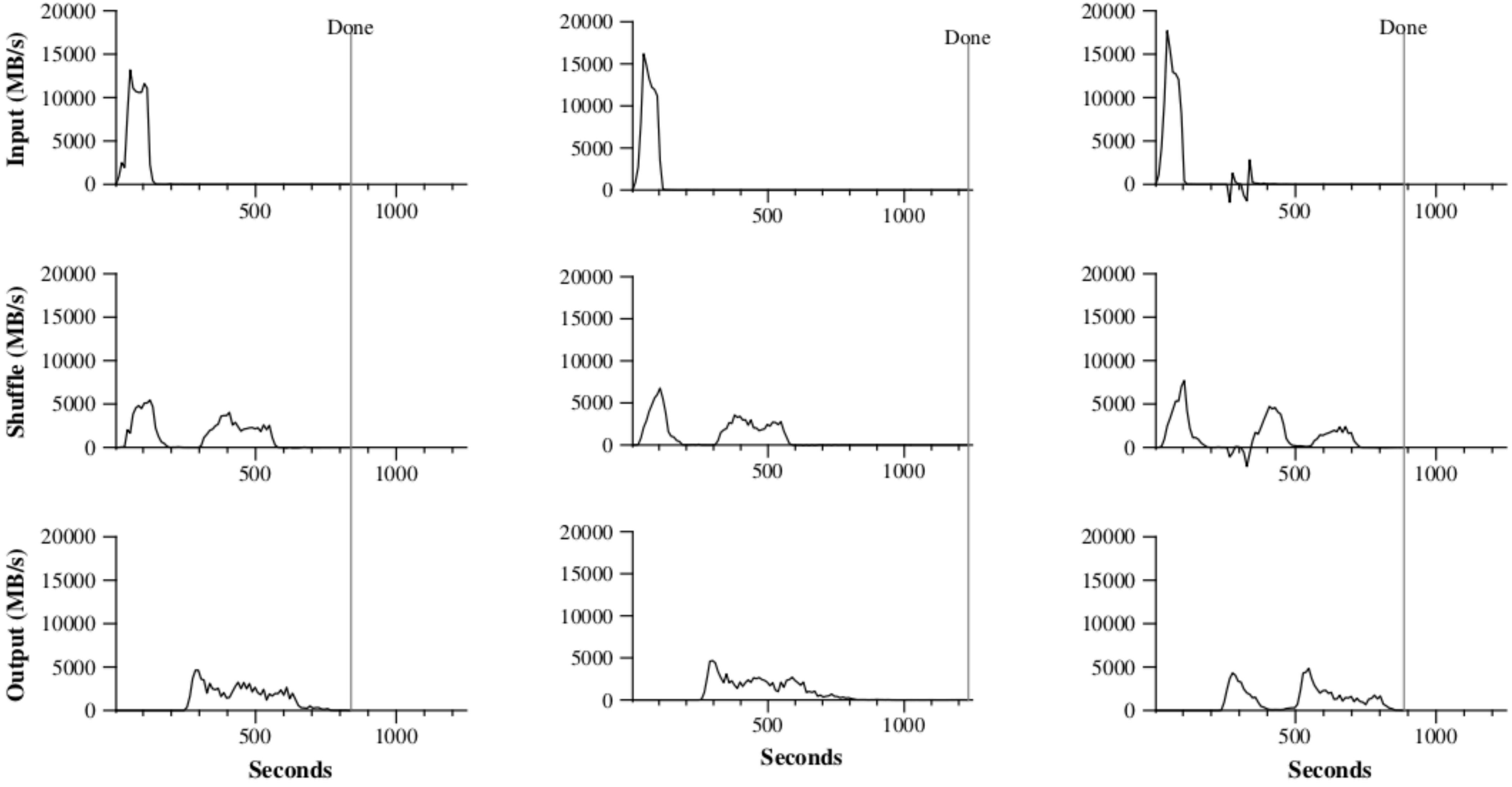


Figure 2: Data transfer rate over time

Performance Measure on Grep program
(Searching for a particular pattern in one terabyte of data).

Performance

input data split: 64MB * 15000
output file: 4000



(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

Performance Measure on Sorting one terabyte of data.

Conclusion

Conclusion

- An programming model easy to use

  - hiding the details of parallelization, fault-tolerance, locality optimization.

- Large variety of problems are easily expressible.

  - Google web search service, data mining…

- Computation scaling on large clusters.

  - efficient to use machine resources.

THNAKS