# MapReduce

Marek Adamczyk
24 XI 2010


LIFE

# Example
# Counting word occurrences

Input document:

NameList and its content is:

“Jim Shahram Betty Jim Shahram Jim Shahram”

Desired output:

- Jim: 3
- Shahram: 3
- Betty: 1

# How?

```
Map(String doc_name, String doc_content)
  //doc_name, e.g. NameList
  //doc_content, e.g. ”Jim Shahram ...”
  For each word w in value
   EmitIntermediate(w, ”1”);
```

Map (NameList, ”Jim Shahram Betty ...”) emits:
[Jim, 1], [Shahram, 1], [Betty, 1], ...

# How?

```
Reduce(String key, Iterator values)
  // key is a word
  // values is a list of counts
  Int result = 0;
  For each v in values
    result += ParseInt(v);
  Emit(AsString(result));
```

Reduce(”Jim”, ”1 1 1”) emits ”3”

# Other examples: Distributed Grep

- Map function emits a line if it matches a supplied pattern.
- Reduce function is an identity function that copies the supplied intermediate data to the output.

# Other examples: Count of URL accesses

- Map function processes logs of web page requests and outputs <URL, 1>
- Reduce function adds together all values for the same URL, emitting <URL, total count> pairs.

# Other examples: Reverse Web-Link graph

- e.g. all URLs with reference to http://dblab.usc.edu
- Map function outputs <tgt, src> for each link to a tgt in a page named src
- Reduce concatenates the list of all src URLS associated with a given tgt URL and emits the pair: <tgt, list(src)>

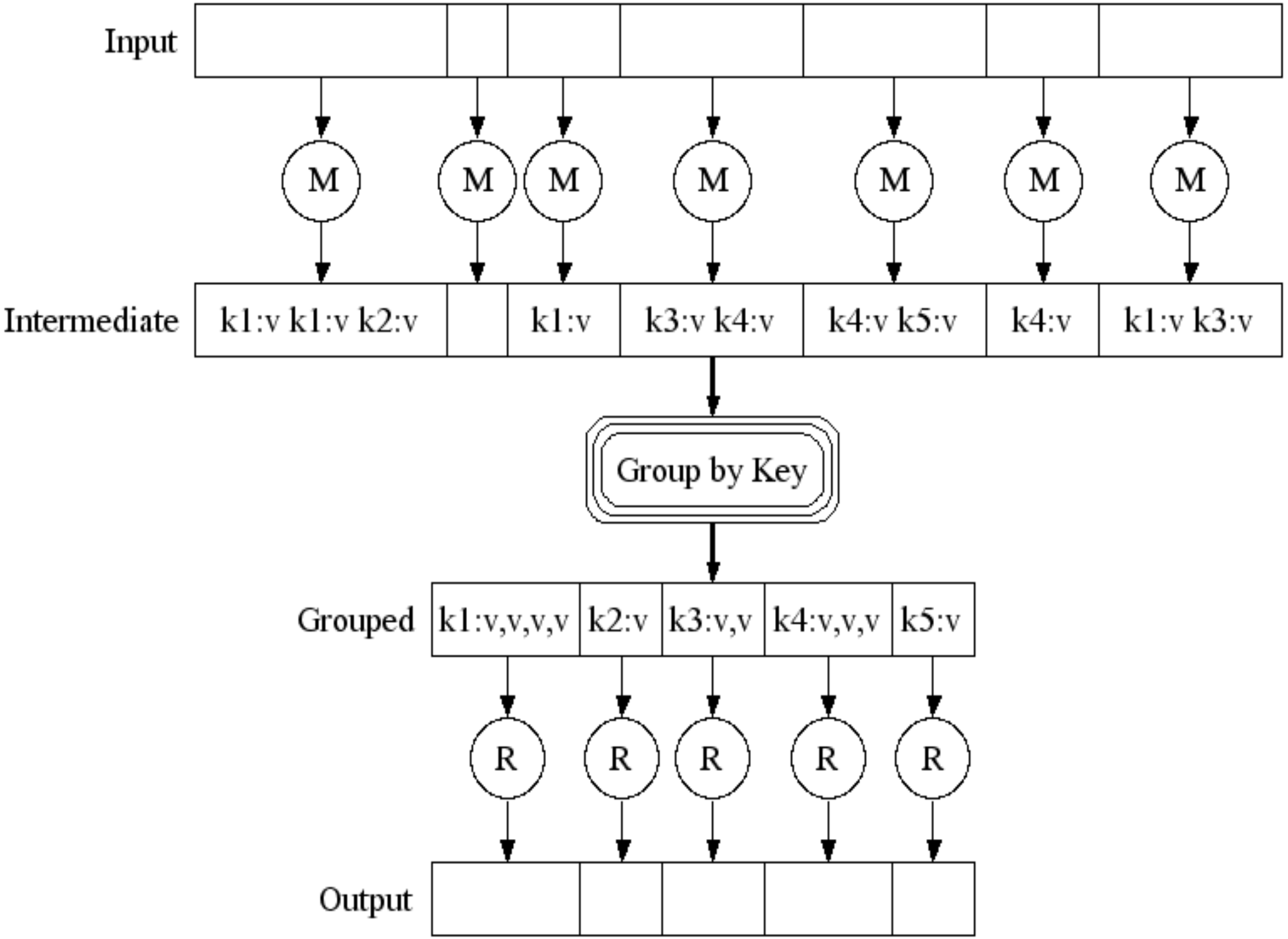
# Other examples: Inverted Index

- e.g. all URLs with 585 as a word
- Map function parses each document, emitting a sequence of <word, doc_ID>
- Reduce accepts all pairs for a given word, sorts the corresponding doc_IDs and emits a <word, list(doc_ID)> pair
- All output pairs form a simple inverted index

# MapReduce

- M(Input) → {[K1, V1], [K2, V2], ... }
  - M(”Jim Shahram Betty Jim Shahram Jim Shahram”)

    →{[”Jim”, ”1”], [”Jim”, ”1”], [”Jim”, ”1”], [”Shahram”, ”1”], [”Shahram”, ”1”], [”Shahram”, ”1”], [”Betty”, ”1”] }
- [”Jim”,”1 1 1”], [”Shahram”,”1 1 1”], [”Betty”,”1”]
- R(Ki, ValueSet) → [Ki, Reduce(ValueSet)]
  - R(”Jim”, ”1 1 1”) → [”Jim”, ”3”]


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

# MapReduce

- Programs written in functional style
- Automatically parallelized and executed on a large cluster of commodity machines.
- The run-time system takes care of the details of
  - partitioning the input data,
  - scheduling the program’s execution across a set of machines,
  - handling machine failures,
  - and managing the required inter-machine communication.
- This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system

# Implementation: Word Frequency

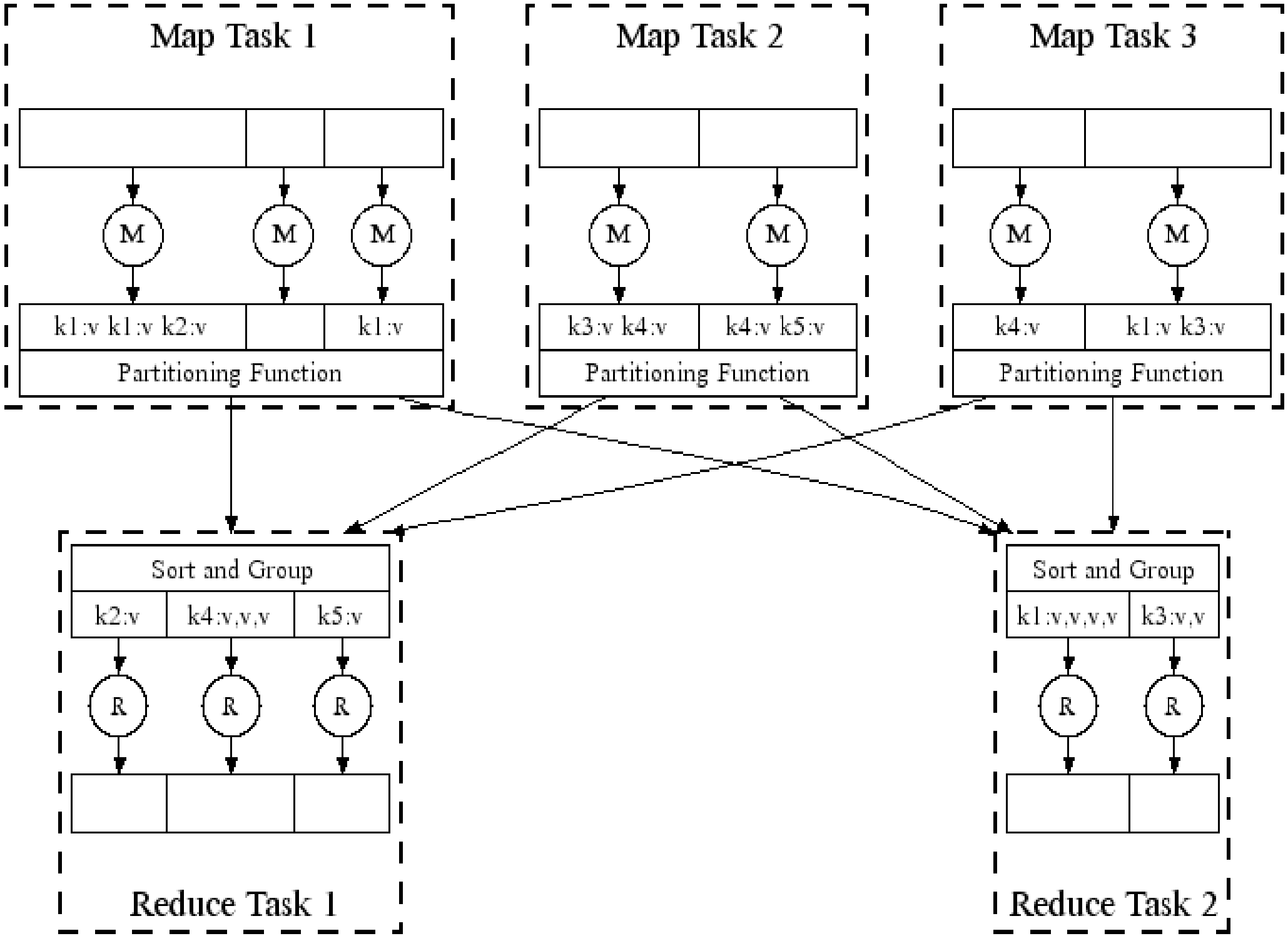
```
int main(int argc, char** argv) {
   ParseCommandLineFlags(argc, argv);

   MapReduceSpecification spec;
   // Store list of input files into "spec"

   for (int i = 1; i < argc; i++) {
      MapReduceInput* input = spec.add_input();
      input->set_format("text");
      input->set_filepattern(argv[i]);
      input->set_mapper_class("WordCounter");
   }
```

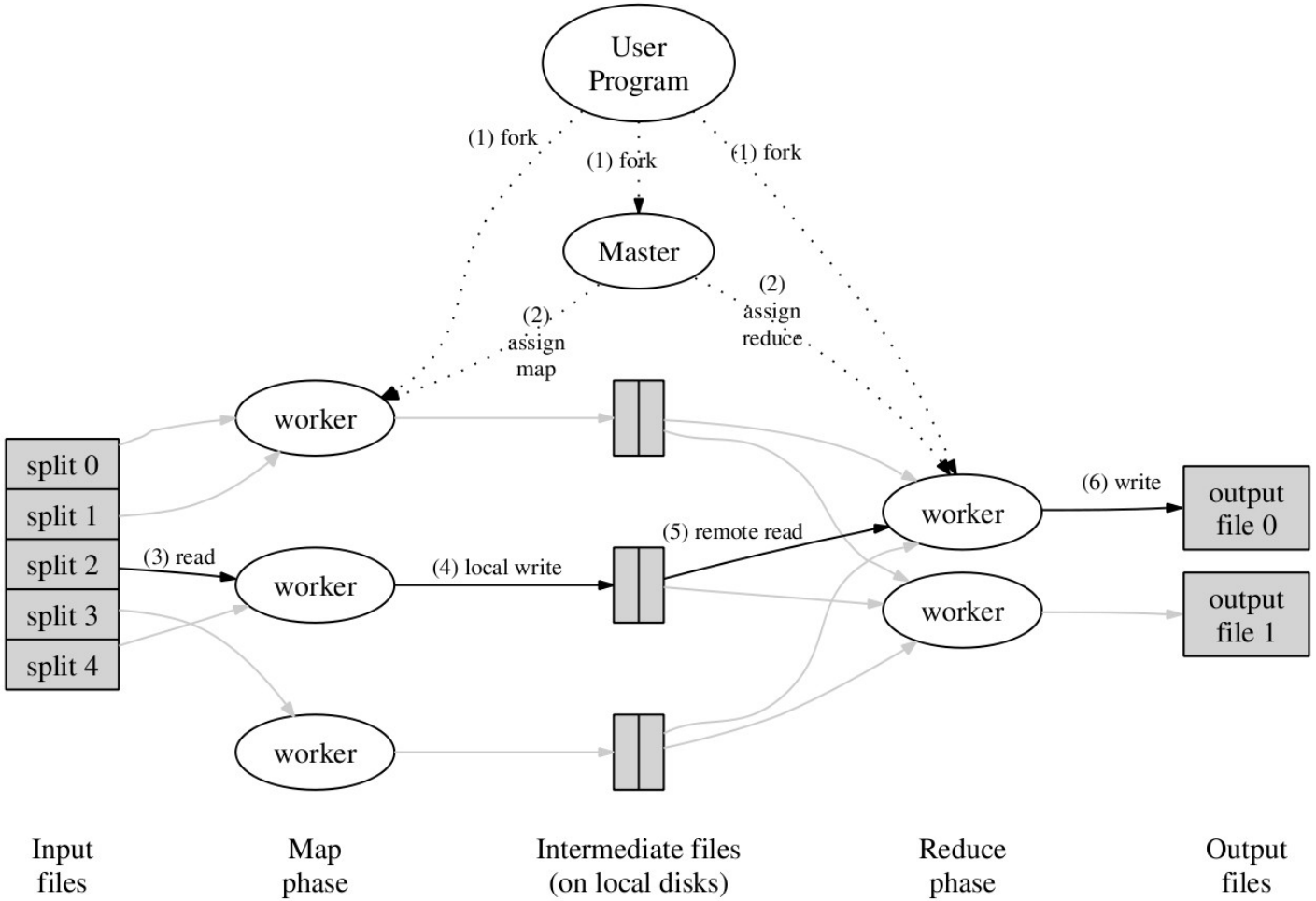
# Implementation: Word Frequency

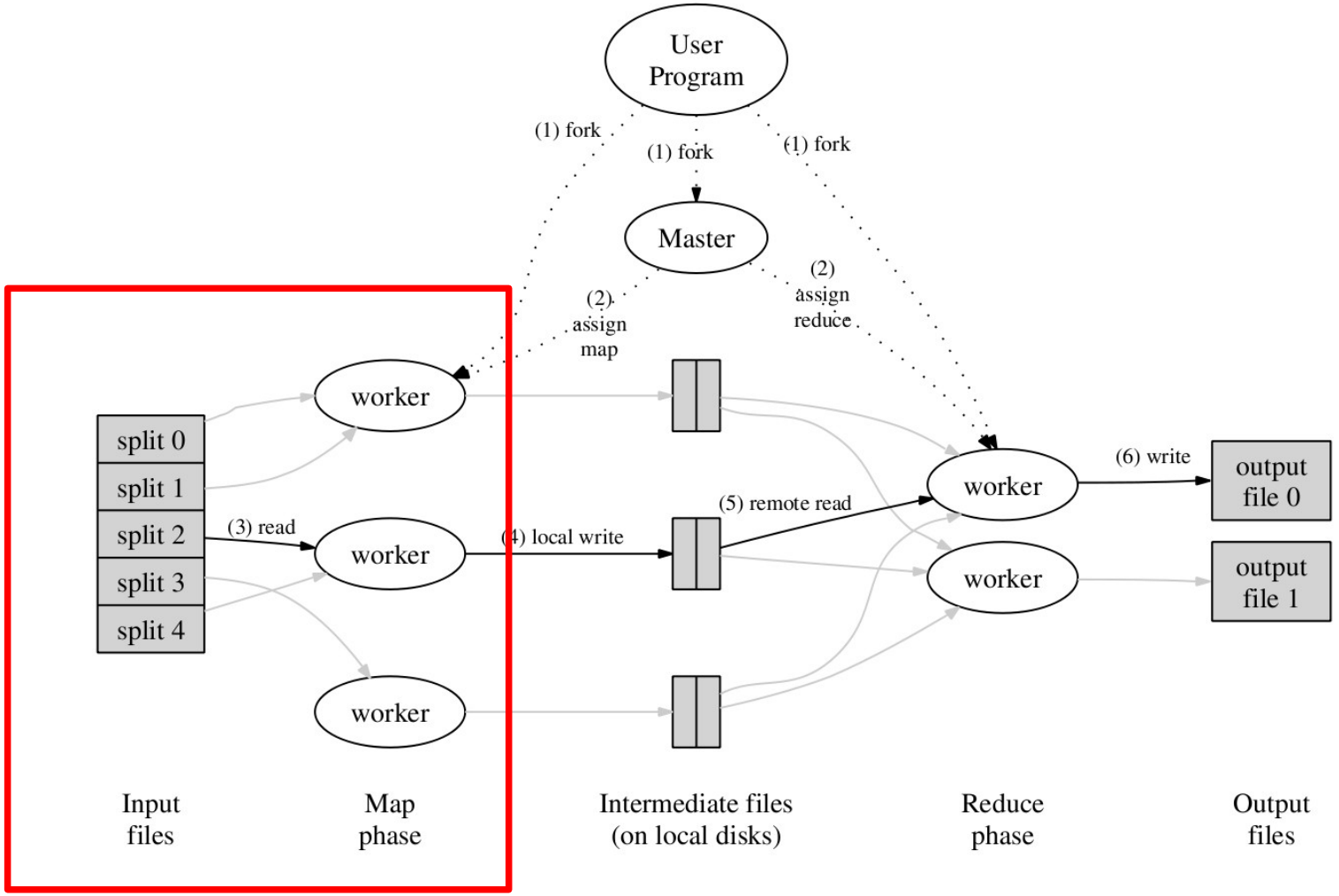
```
// Specify the output files:
//    /gfs/test/freq-00000-of-00100
//    /gfs/test/freq-00001-of-00100
//    ...

MapReduceOutput* out = spec.output();
out->set_filebase("/gfs/test/freq");
out->set_num_tasks(100);
out->set_format("text");
out->set_reducer_class("Adder");
```

# Implementation: Word Frequency

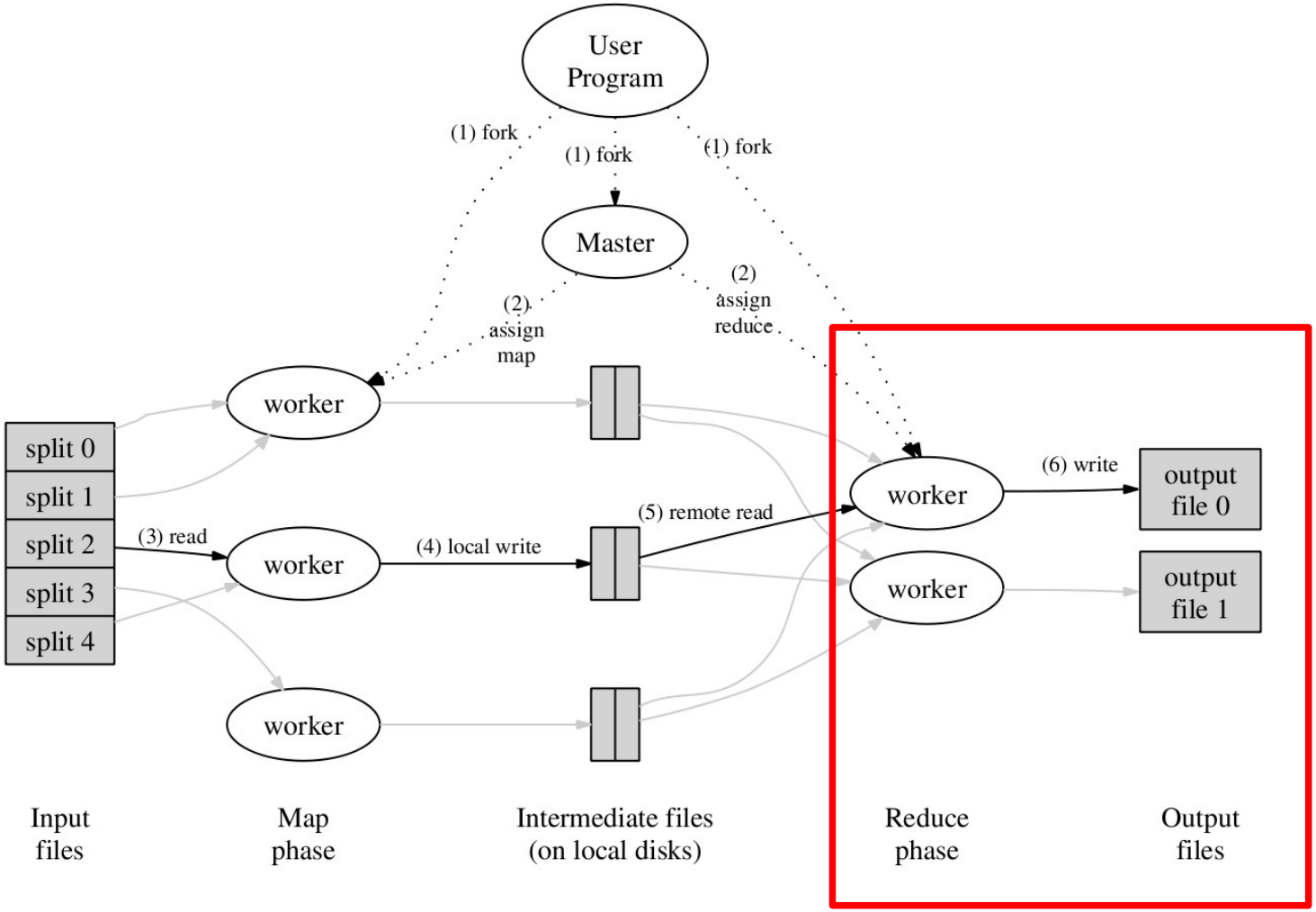
```
  // Tuning parameters: use at most 2000
  // machines and 100 MB of memory per task

  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);

  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  return 0;
}
```


Map Task 1
Map Task 2
Map Task 3
M
M
M
M
M
M
M
M
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Partitioning Function
Partitioning Function
Partitioning Function
Sort and Group
k2:v
k4:v,v,v
k5:v
Sort and Group
k1:v,v,v,v
k3:v,v
R
R
R
R
R
Reduce Task 1
Reduce Task 2

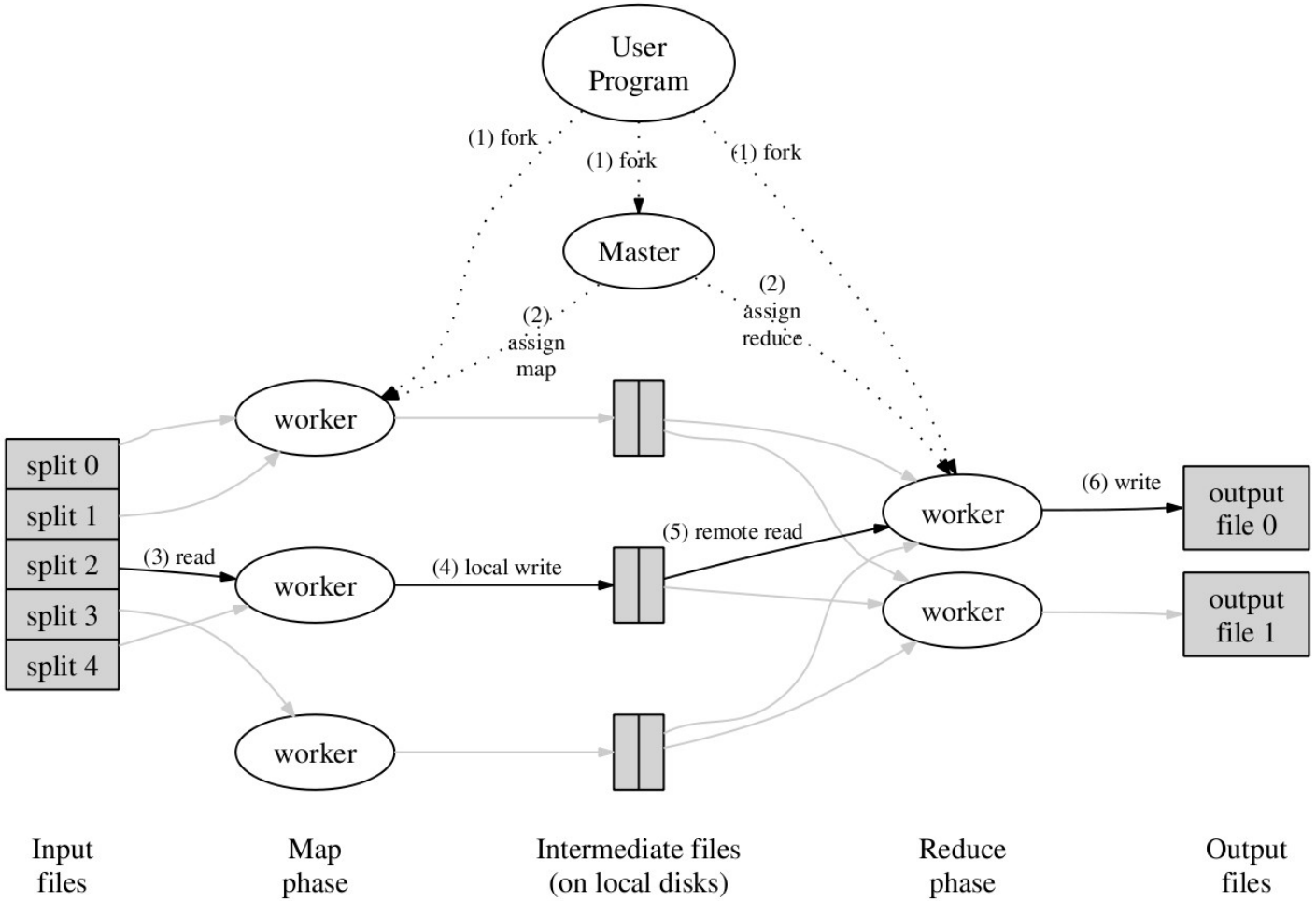

User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files



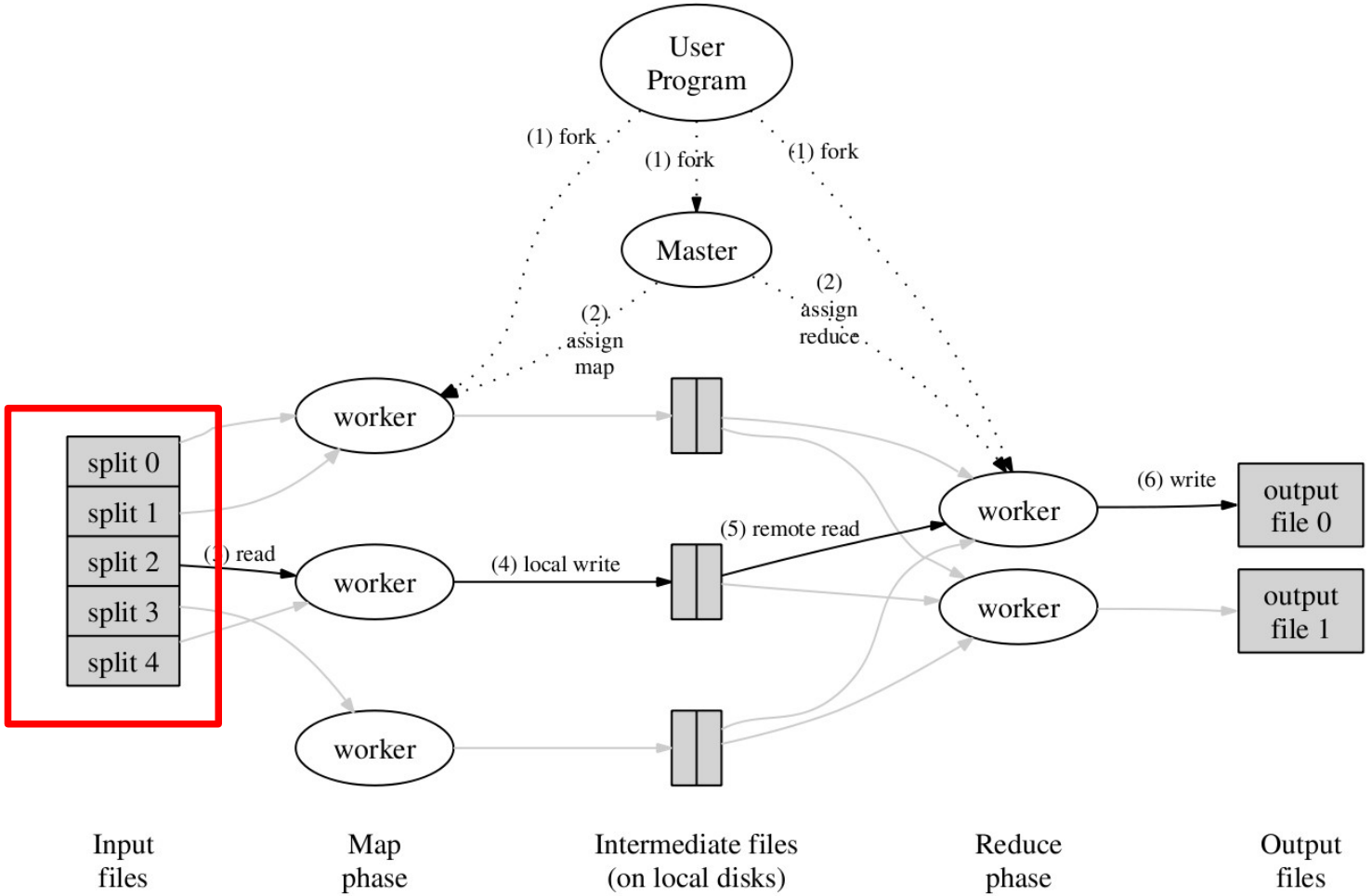

The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines.



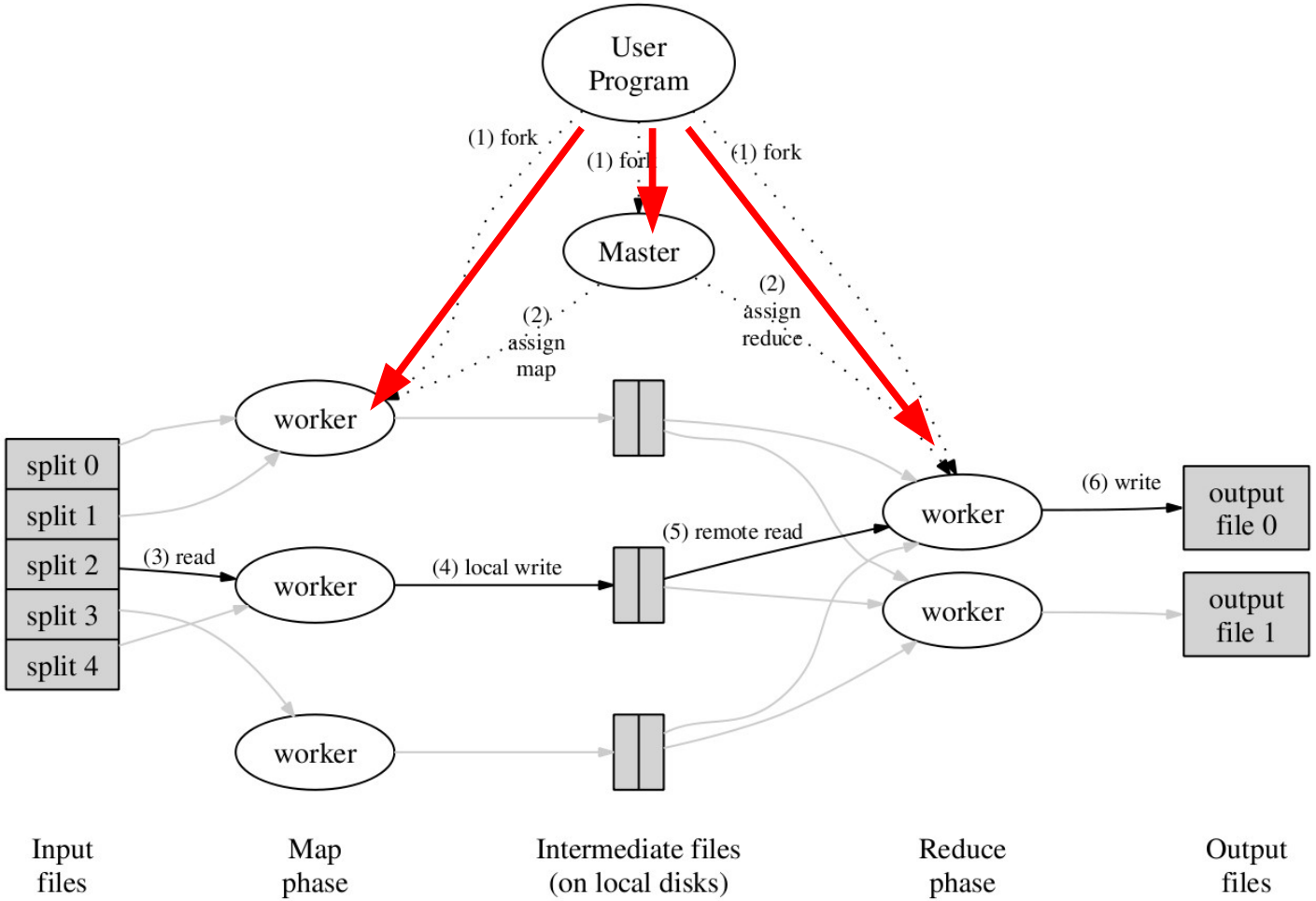

Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g. hash(key) mod R). The number of partitions (R) and the partitioning function are specified by the user.



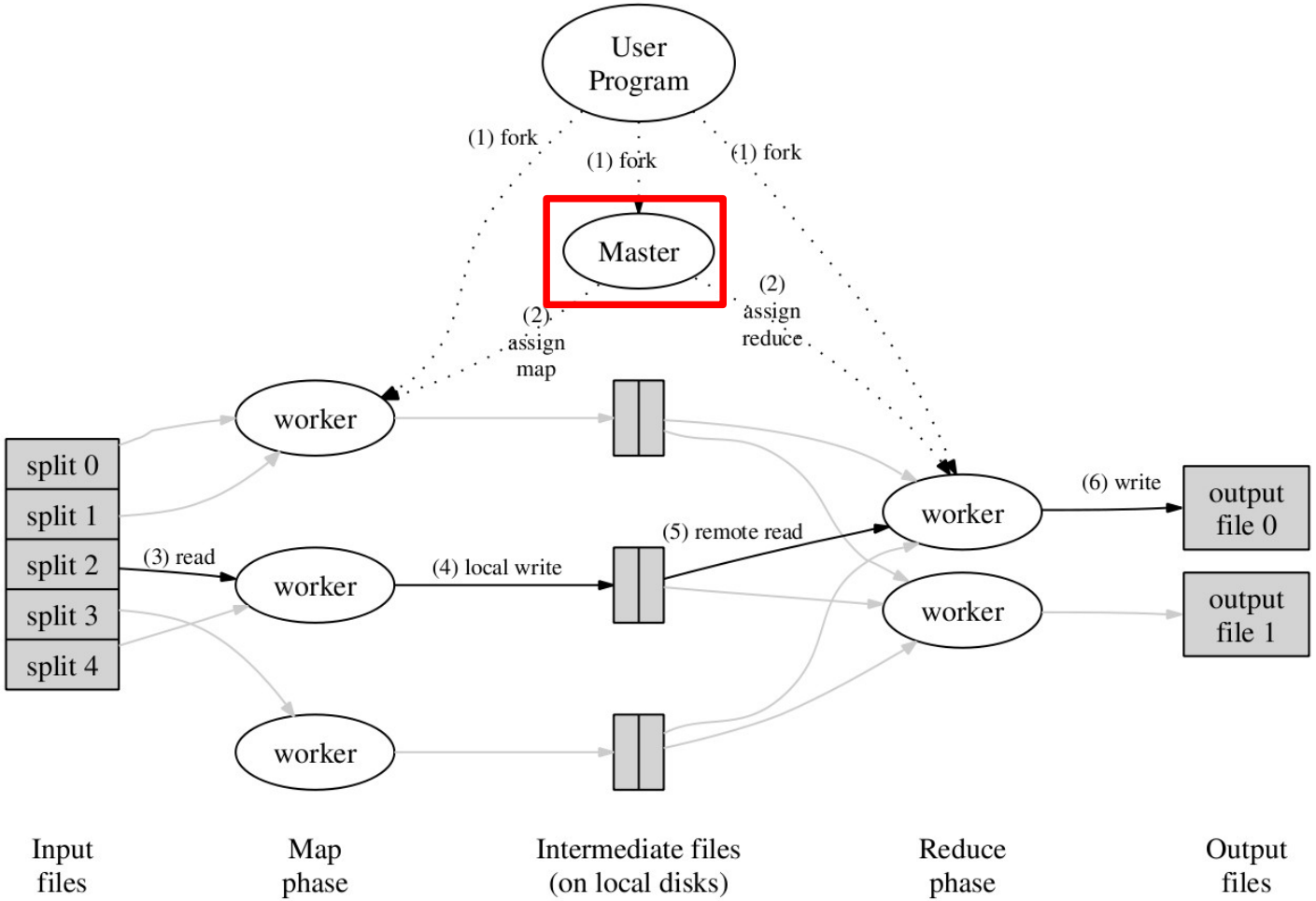

# MapReduce function call



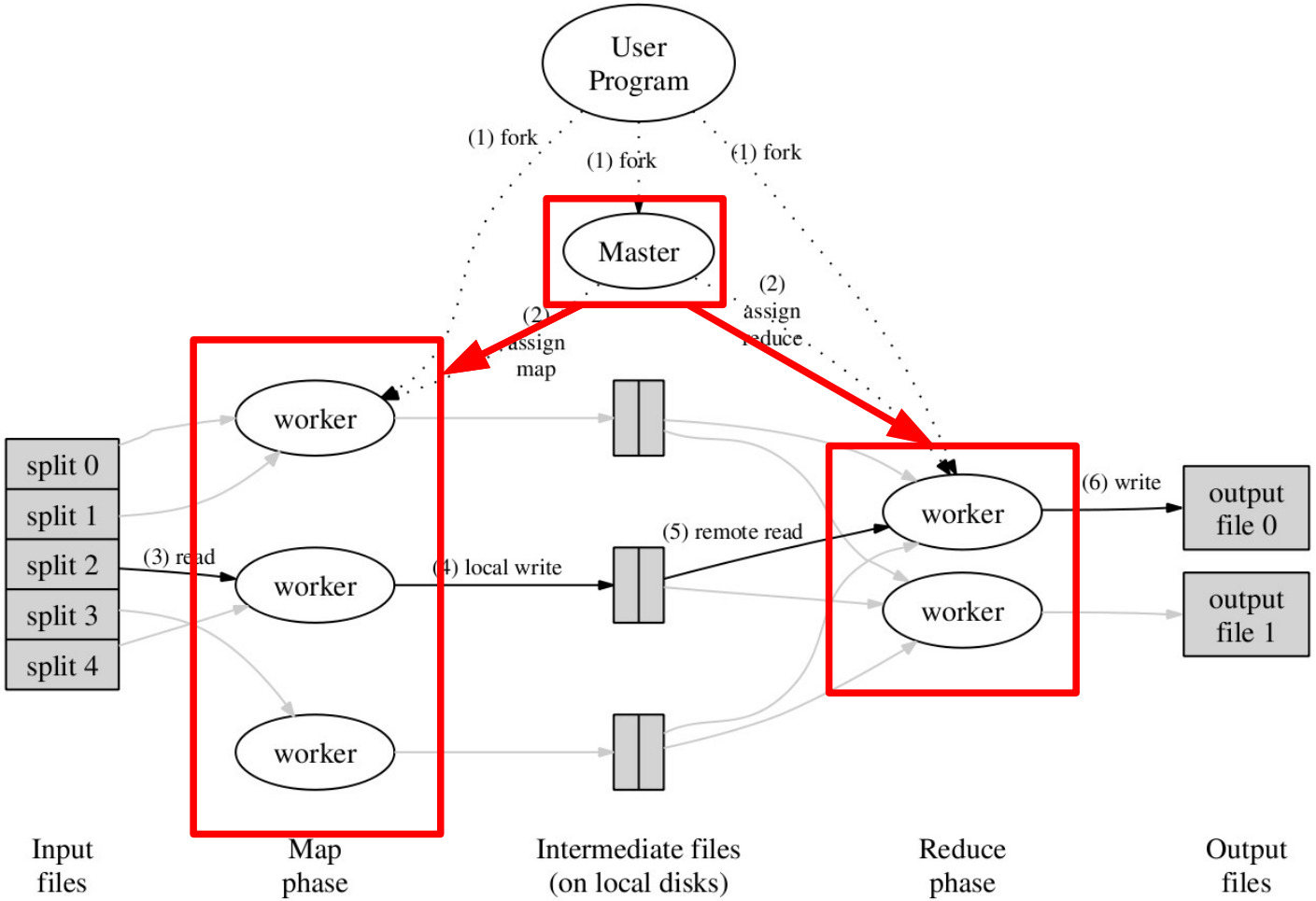

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.



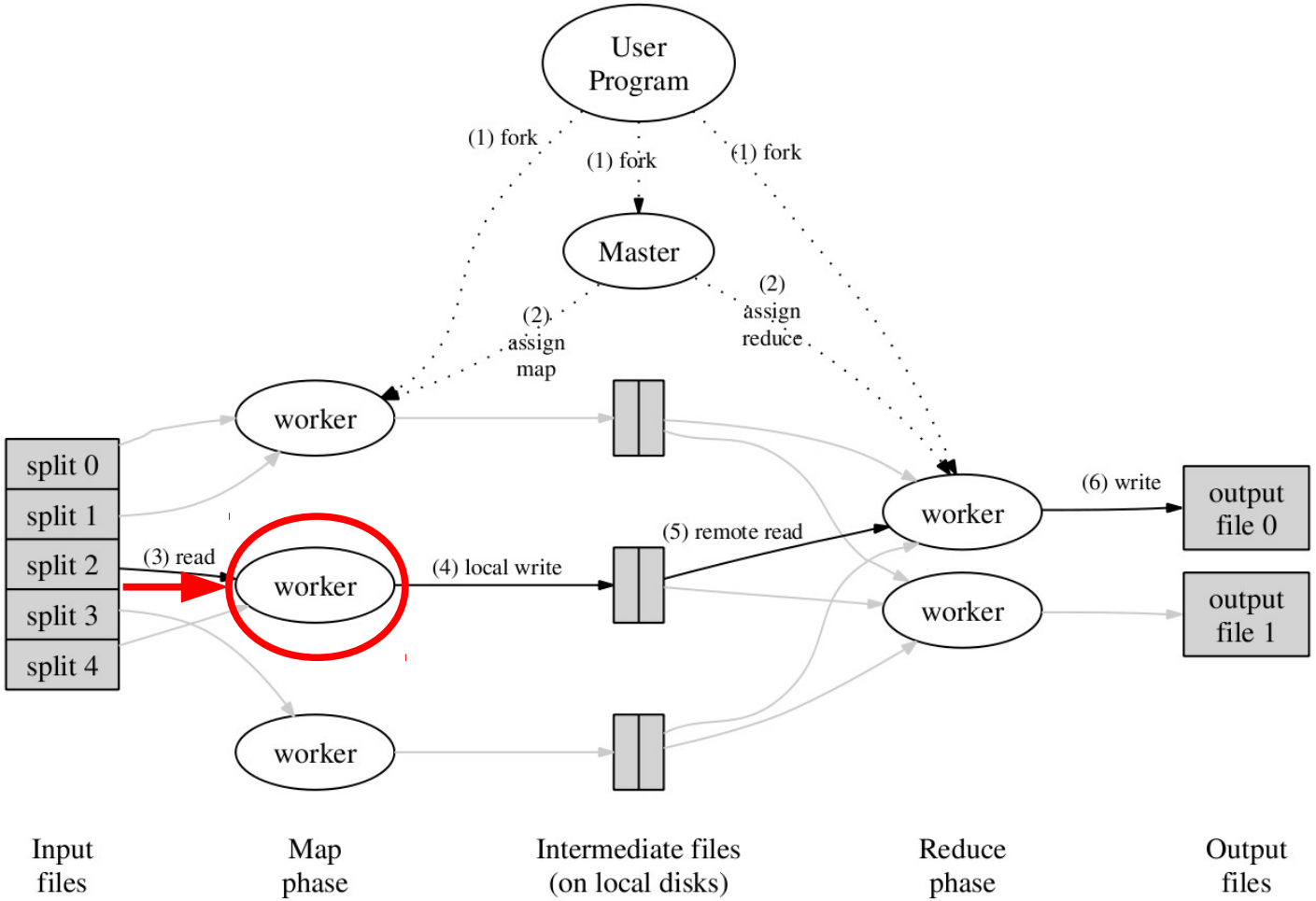

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.



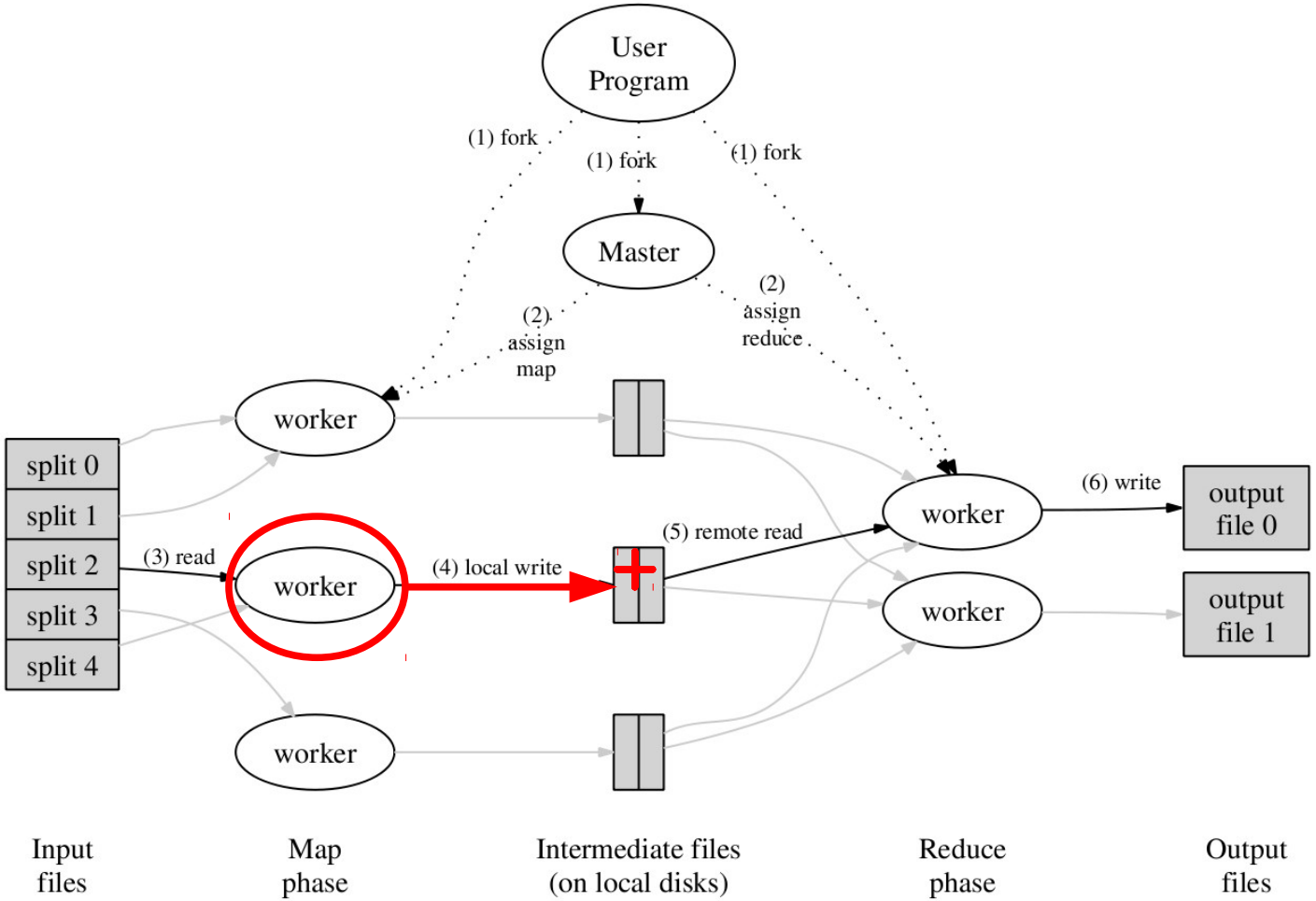

2. One of the copies of the program is special – the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.



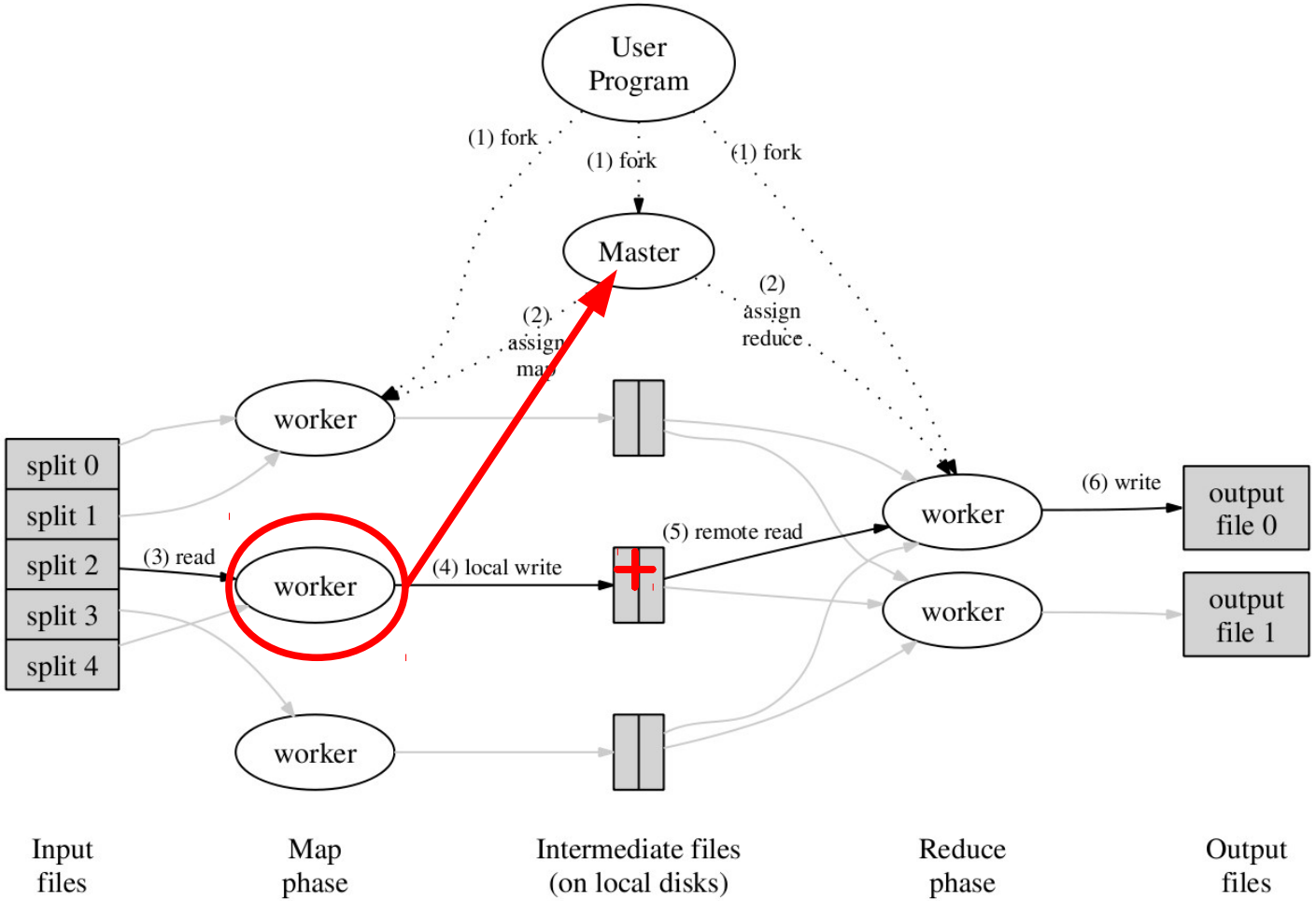

2. One of the copies of the program is special – the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.



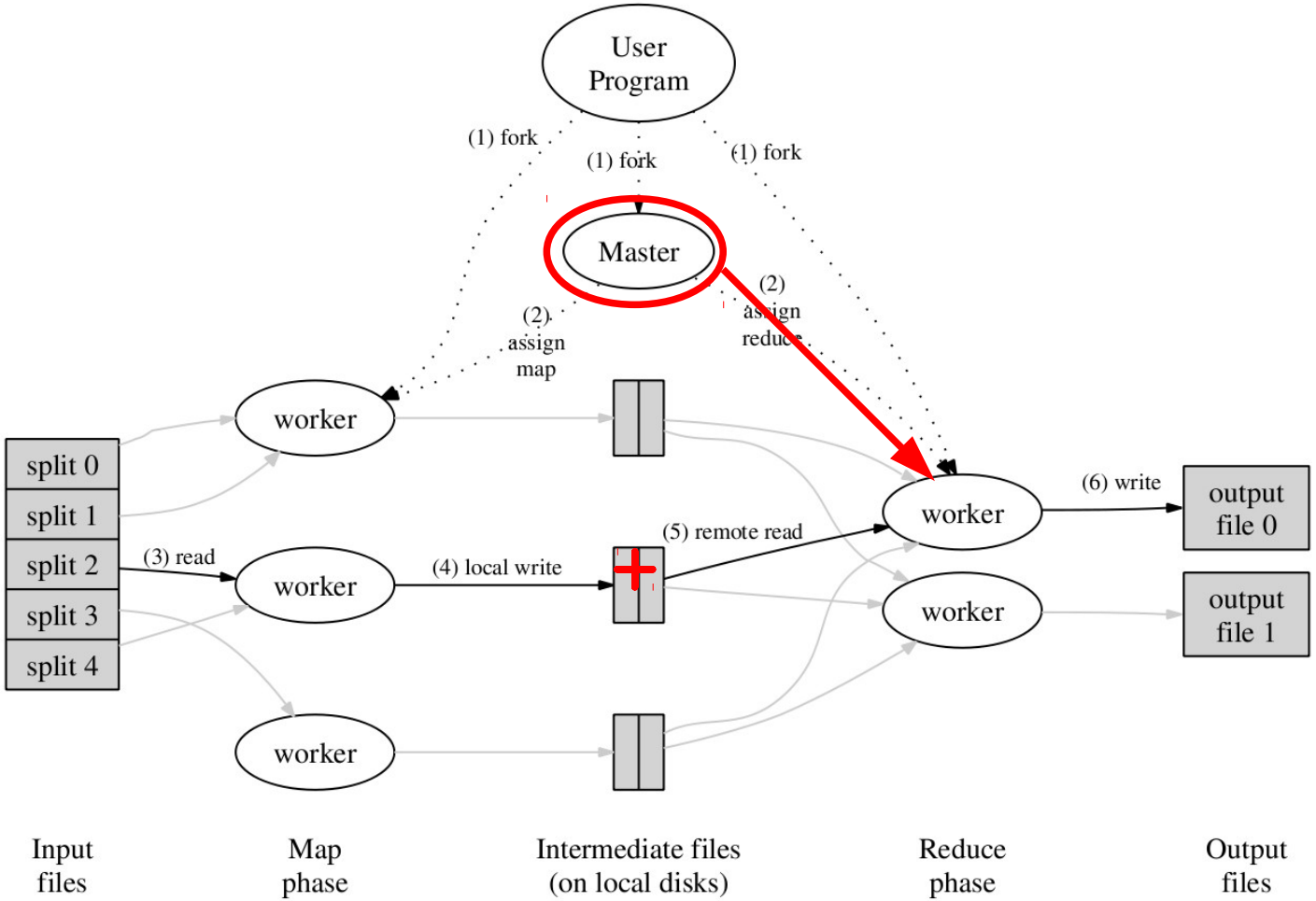

3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.



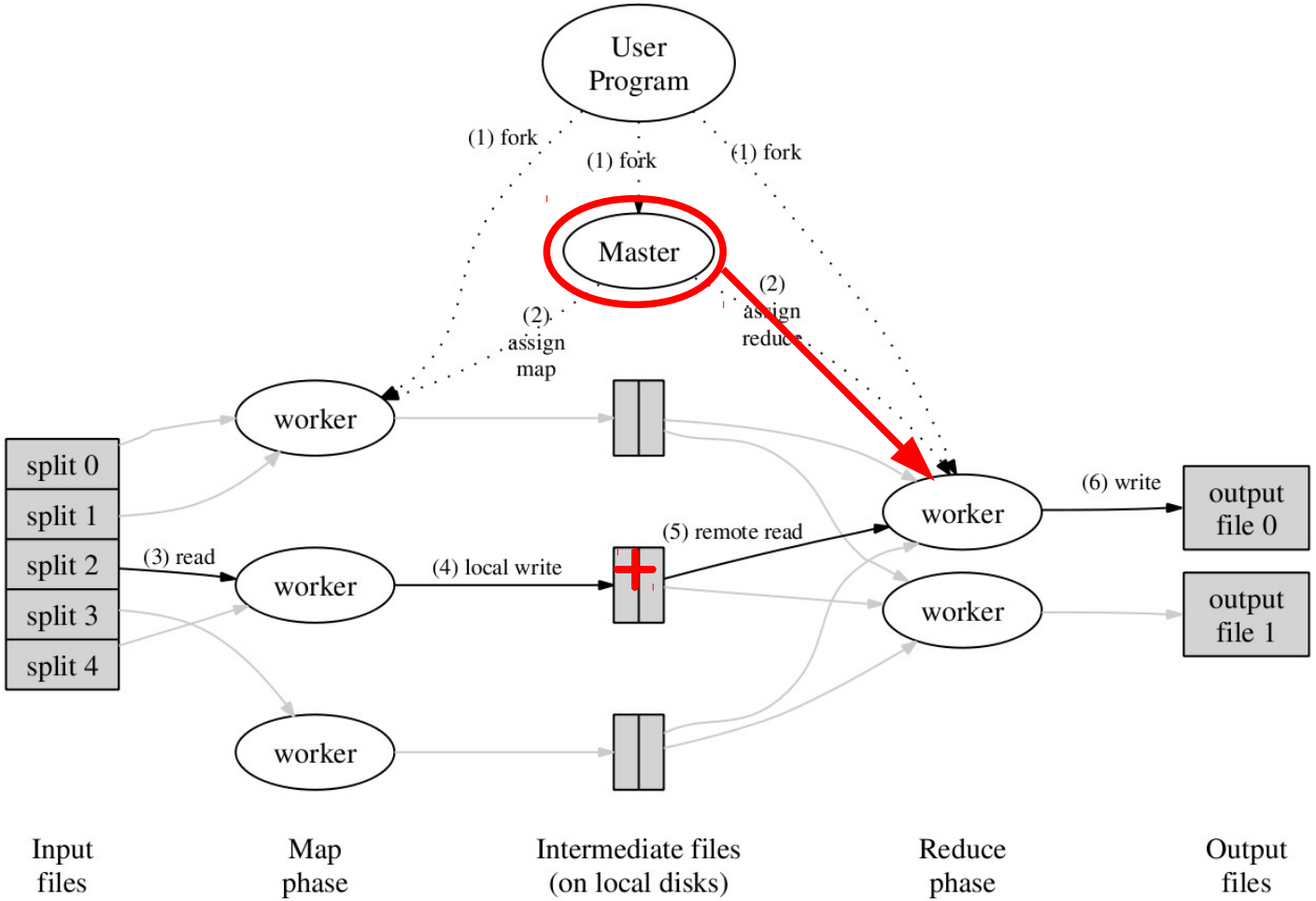

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.



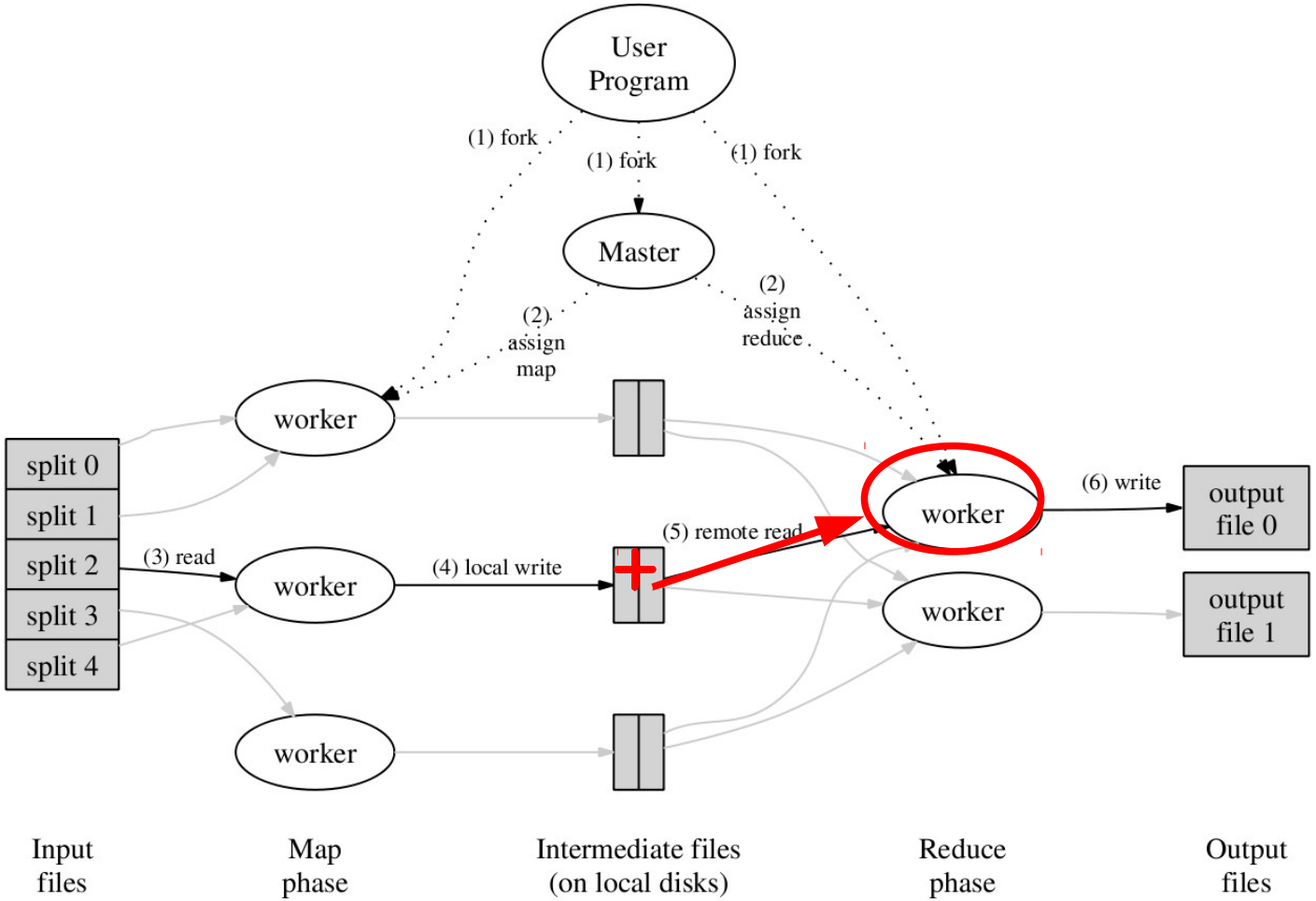

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.



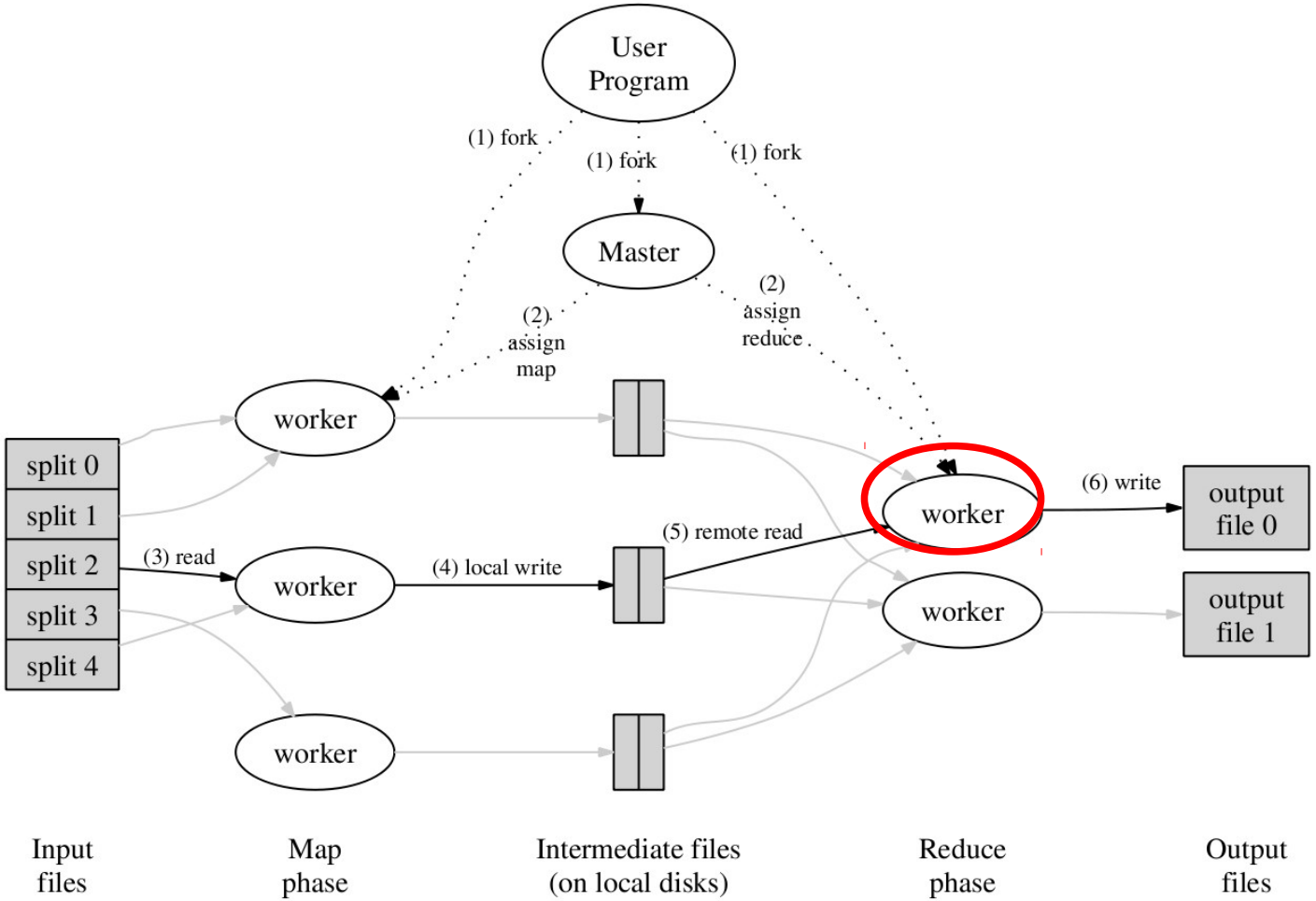

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.



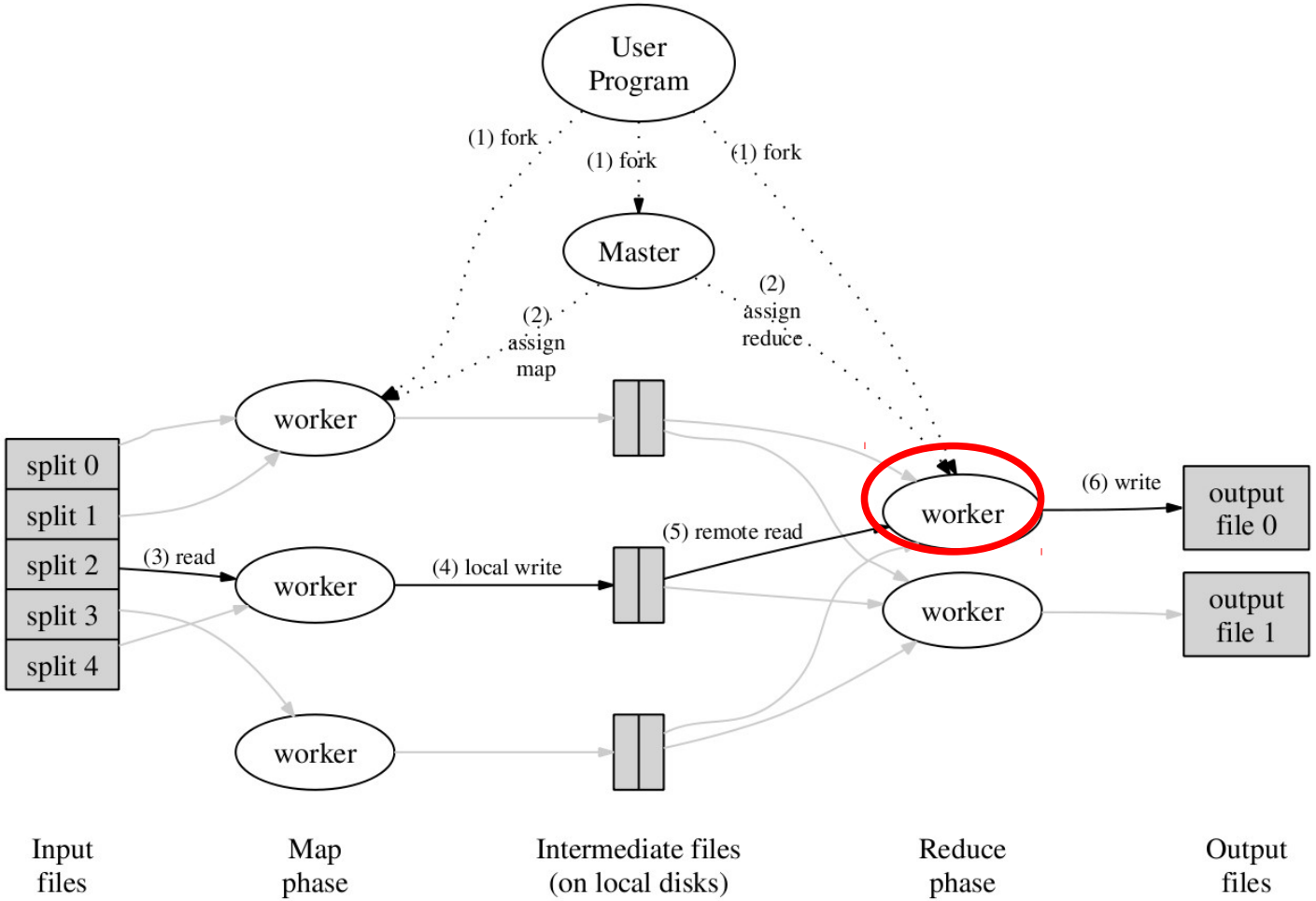

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers.



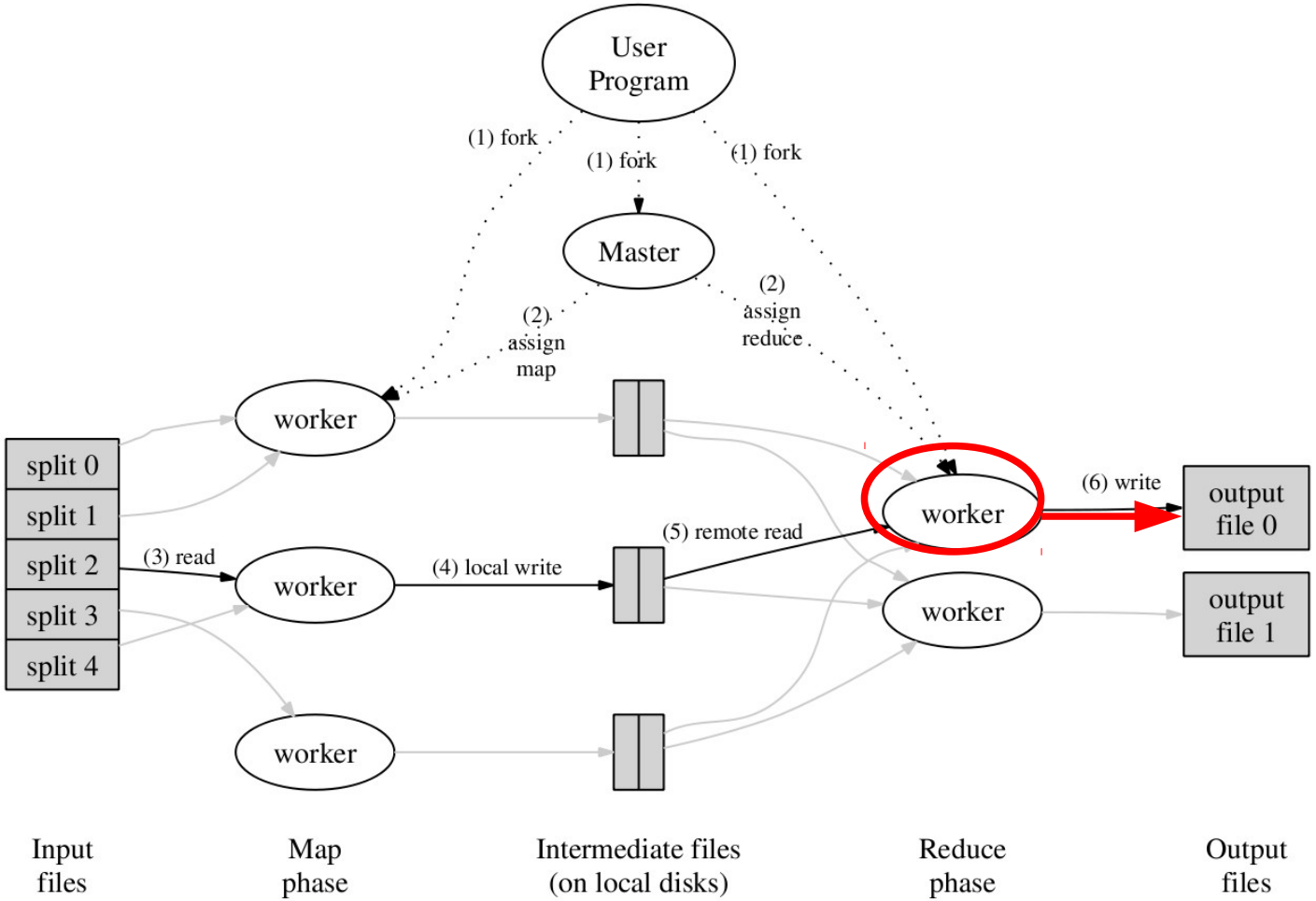

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers.



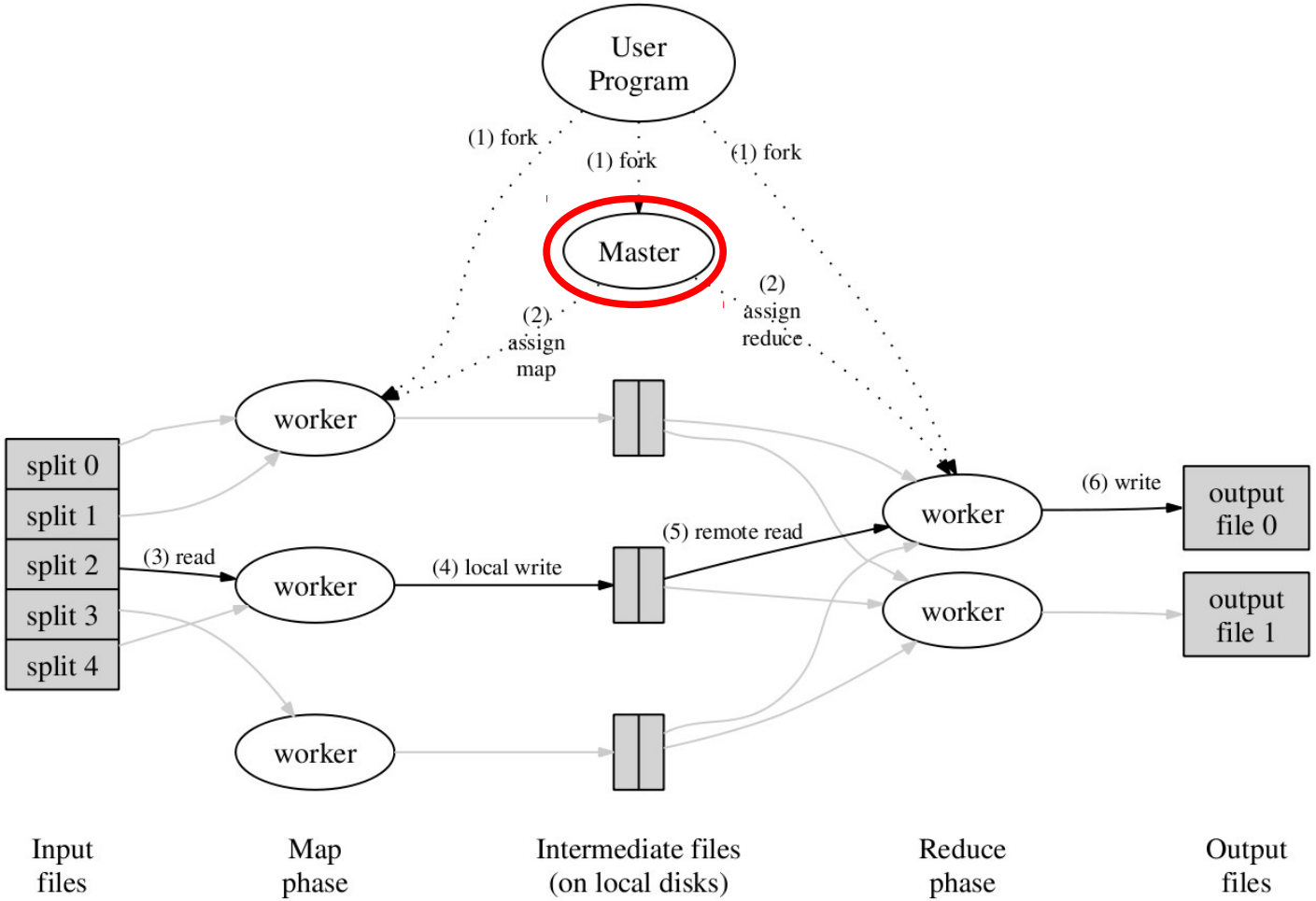

5. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.




6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.




6. The output of the Reduce function is appended to a final output file for this reduce partition.




7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

# Example execution

Often MapReduce computations with

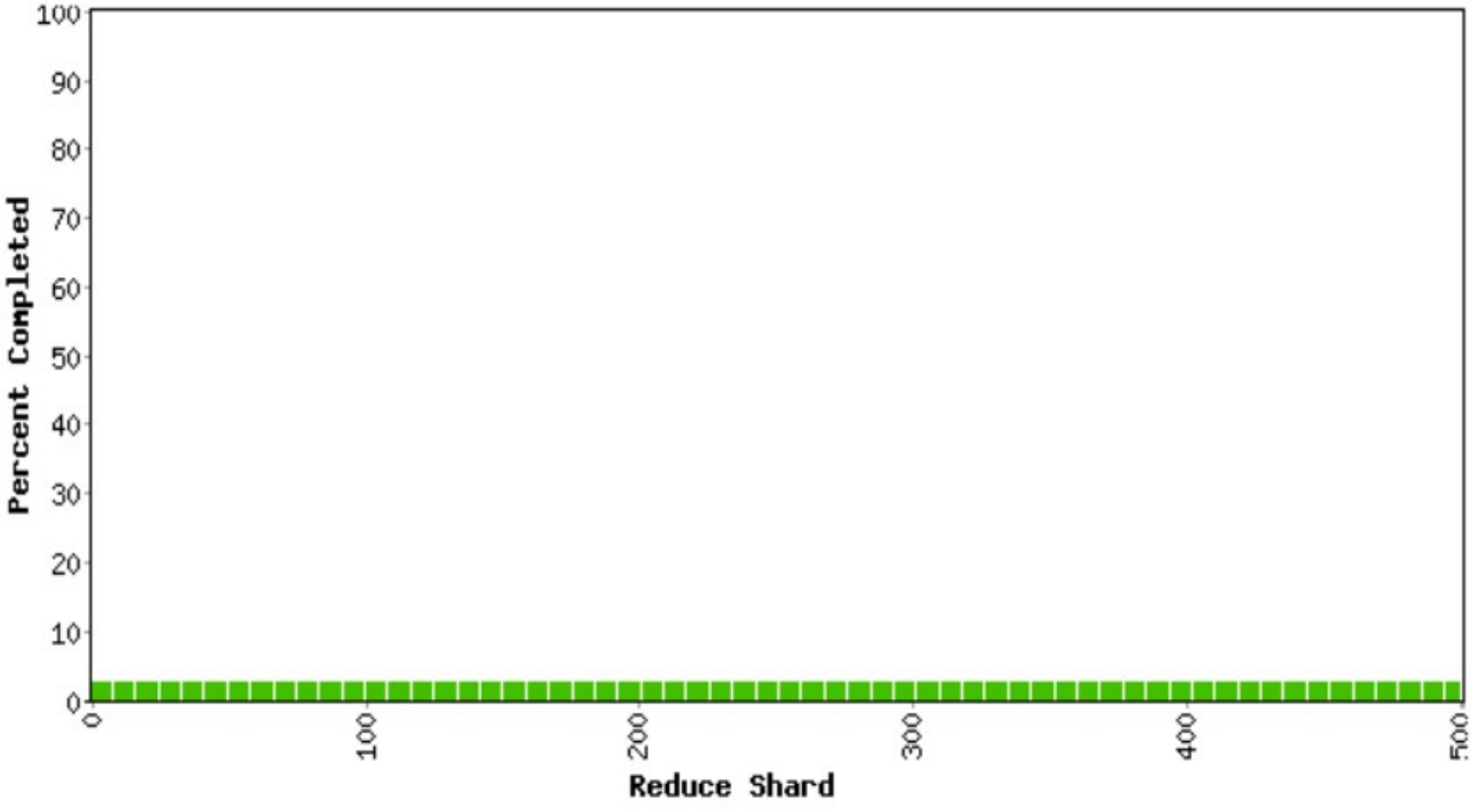
- M = 200, 000
- R = 5, 000
- using 2,000 worker machines.

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 0 | 323 | 878934.6 | 1314.4 | 717.0 |
| Shuffle | 500 | 0 | 323 | 717.0 | 0.0 | 0.0 |
| Reduce | 500 | 0 | 0 | 0.0 | 0.0 | 0.0 |




Counters

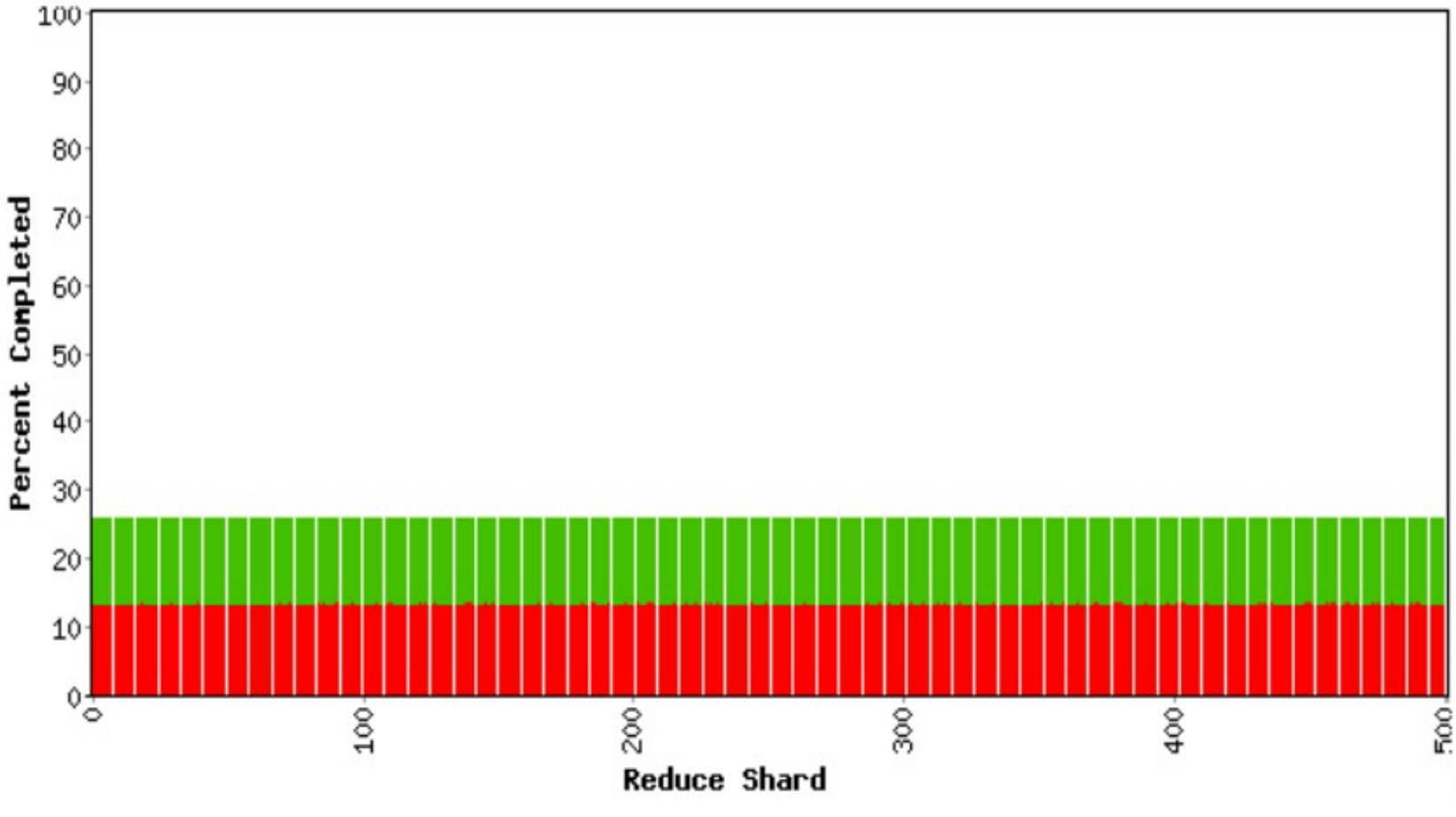
| Variable | Minute |
|---|---|
| Mapped (MB/s) | 72.5 |
| Shuffle (MB/s) | 0.0 |
| Output (MB/s) | 0.0 |
| doc-index-hits | 145825686 |
| docs-indexed | 506631 |
| dups-in-index-merge | 0 |
| mr-operator-calls | 508192 |
| mr-operator-outputs | 506631 |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 1857 | 1707 | 878934.6 | 191995.8 | 113936.6 |
| Shuffle | 500 | 0 | 500 | 113936.6 | 57113.7 | 57113.7 |
| Reduce | 500 | 0 | 0 | 57113.7 | 0.0 | 0.0 |




Counters

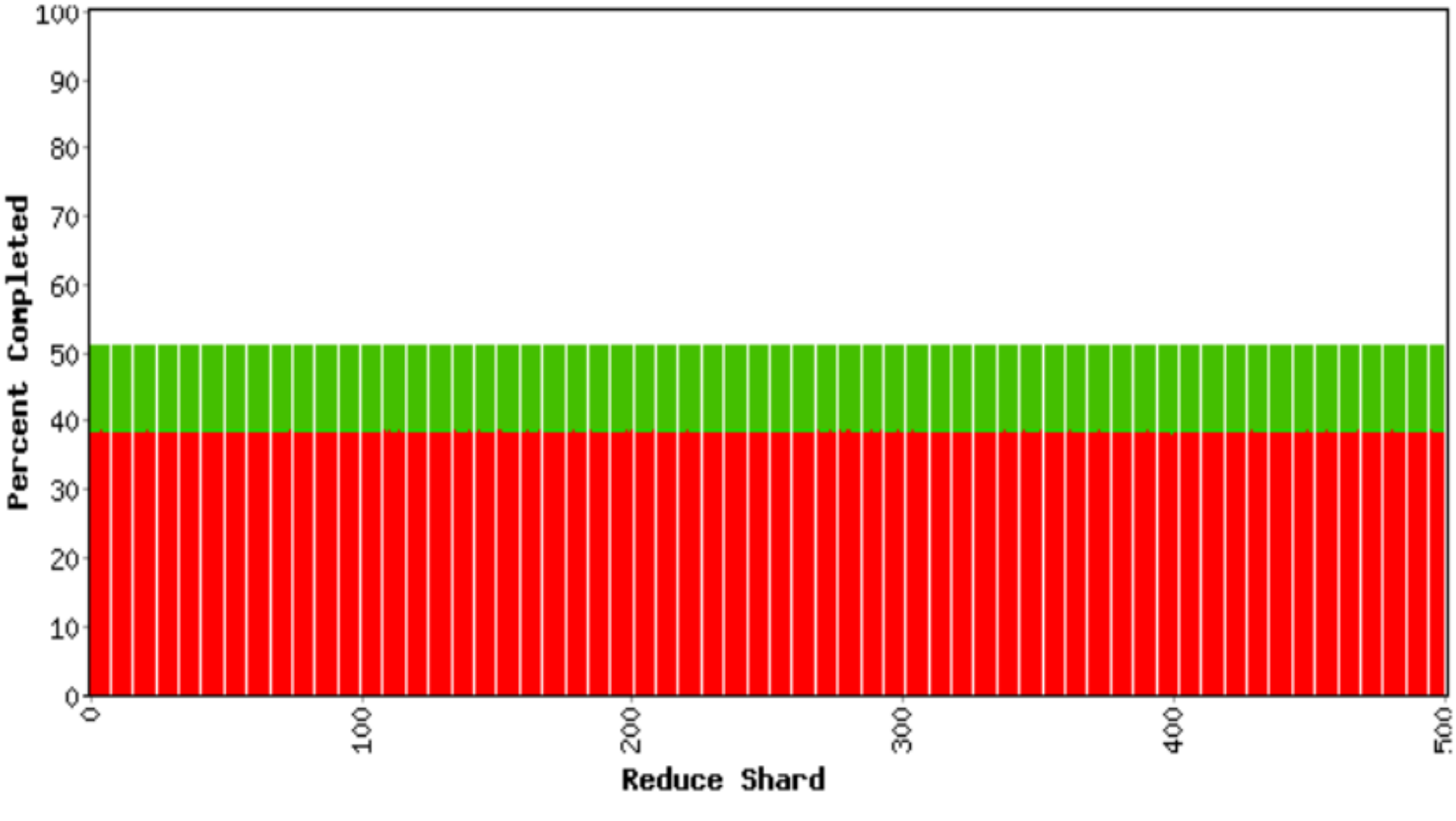
| Variable | Minute |
|---|---|
| Mapped (MB/s) | 699.1 |
| Shuffle (MB/s) | 349.5 |
| Output (MB/s) | 0.0 |
| doc-index-hits | 5004411944 |
| docs-indexed | 17290135 |
| dups-in-index-merge | 0 |
| mr-operator-calls | 17331371 |
| mr-operator-outputs | 17290135 |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 5354 | 1707 | 878934.6 | 406020.1 | 241058.2 |
| Shuffle | 500 | 0 | 500 | 241058.2 | 196362.5 | 196362.5 |
| Reduce | 500 | 0 | 0 | 196362.5 | 0.0 | 0.0 |




Counters

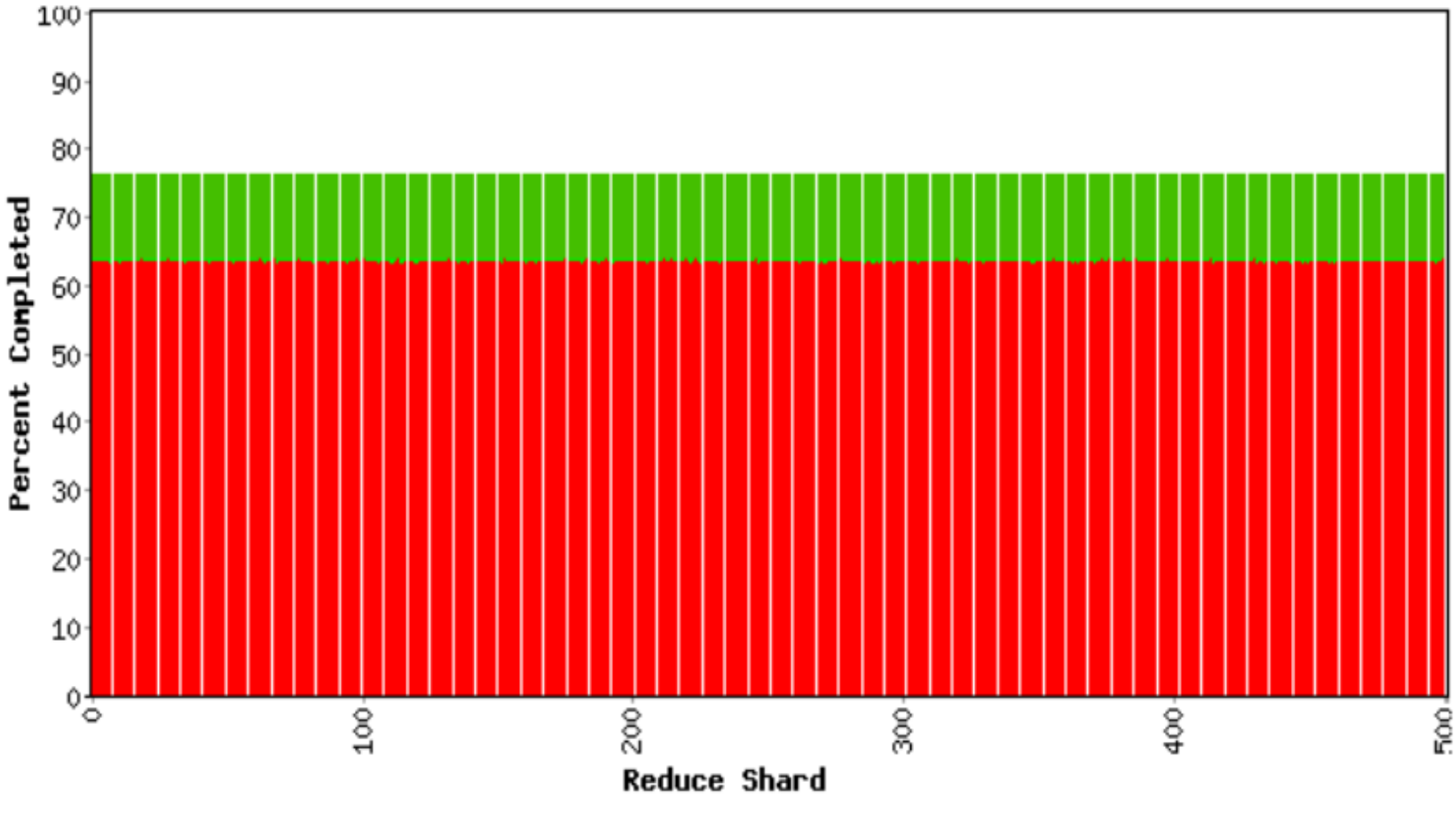
| Variable | Minute |
|---|---|
| Mapped (MB/s) | 704.4 |
| Shuffle (MB/s) | 371.9 |
| Output (MB/s) | 0.0 |
| doc-index-hits | 5000364228 |
| docs-indexed | 17300709 |
| dups-in-index-merge | 0 |
| mr-operator-calls | 17342493 |
| mr-operator-outputs | 17300709 |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 8841 | 1707 | 878934.6 | 621608.5 | 369459.8 |
| Shuffle | 500 | 0 | 500 | 369459.8 | 326986.8 | 326986.8 |
| Reduce | 500 | 0 | 0 | 326986.8 | 0.0 | 0.0 |




Counters

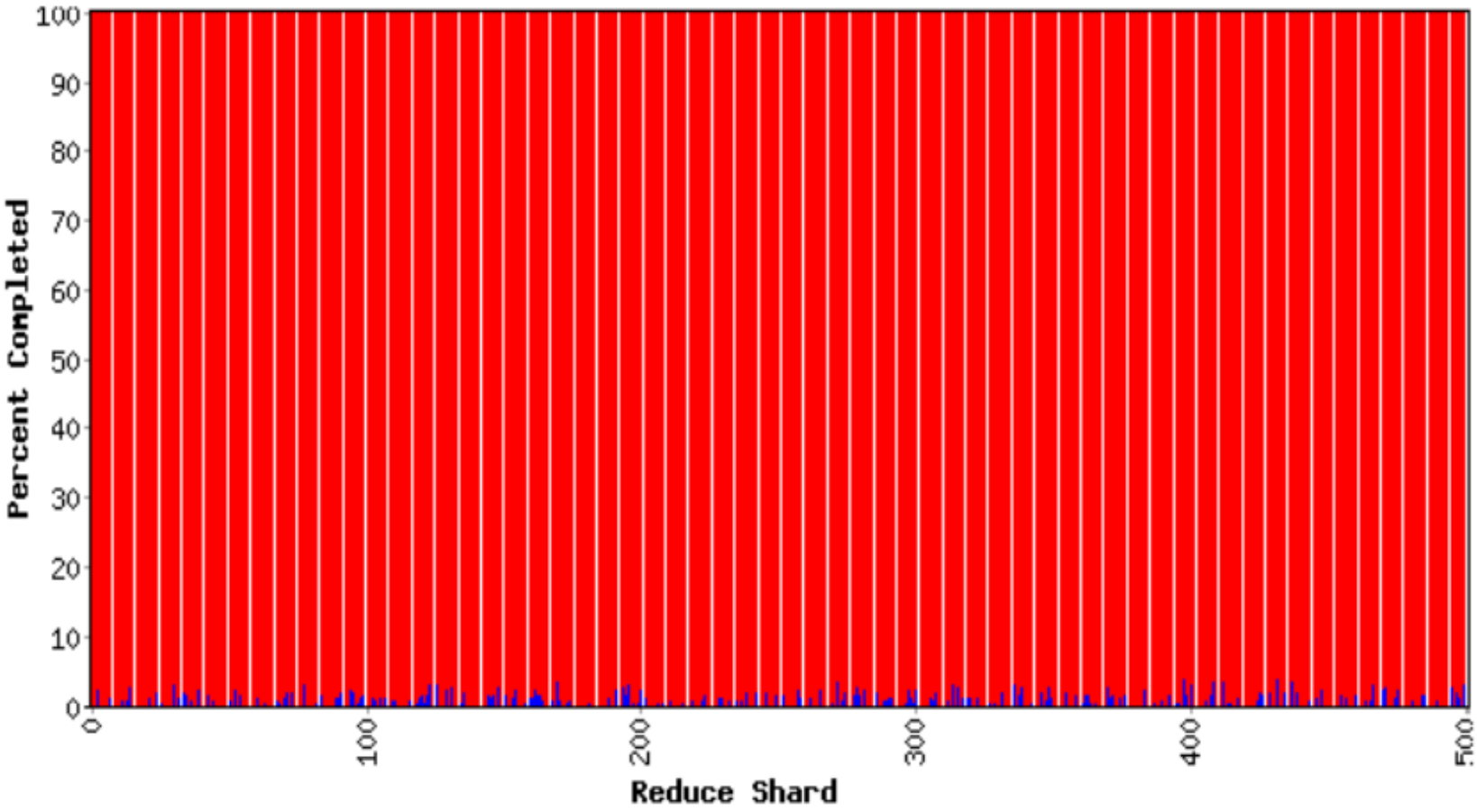
| Variable | Minute |
|---|---|
| Mapped (MB/s) | 706.5 |
| Shuffle (MB/s) | 419.2 |
| Output (MB/s) | 0.0 |
| doc-index-hits | 4982870667 |
| docs-indexed | 17229926 |
| dups-in-index-merge | 0 |
| mr-operator-calls | 17272056 |
| mr-operator-outputs | 17229926 |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 195 | 305 | 523499.2 | 523389.6 | 523389.6 |
| Reduce | 500 | 0 | 195 | 523389.6 | 2685.2 | 2742.6 |




Counters

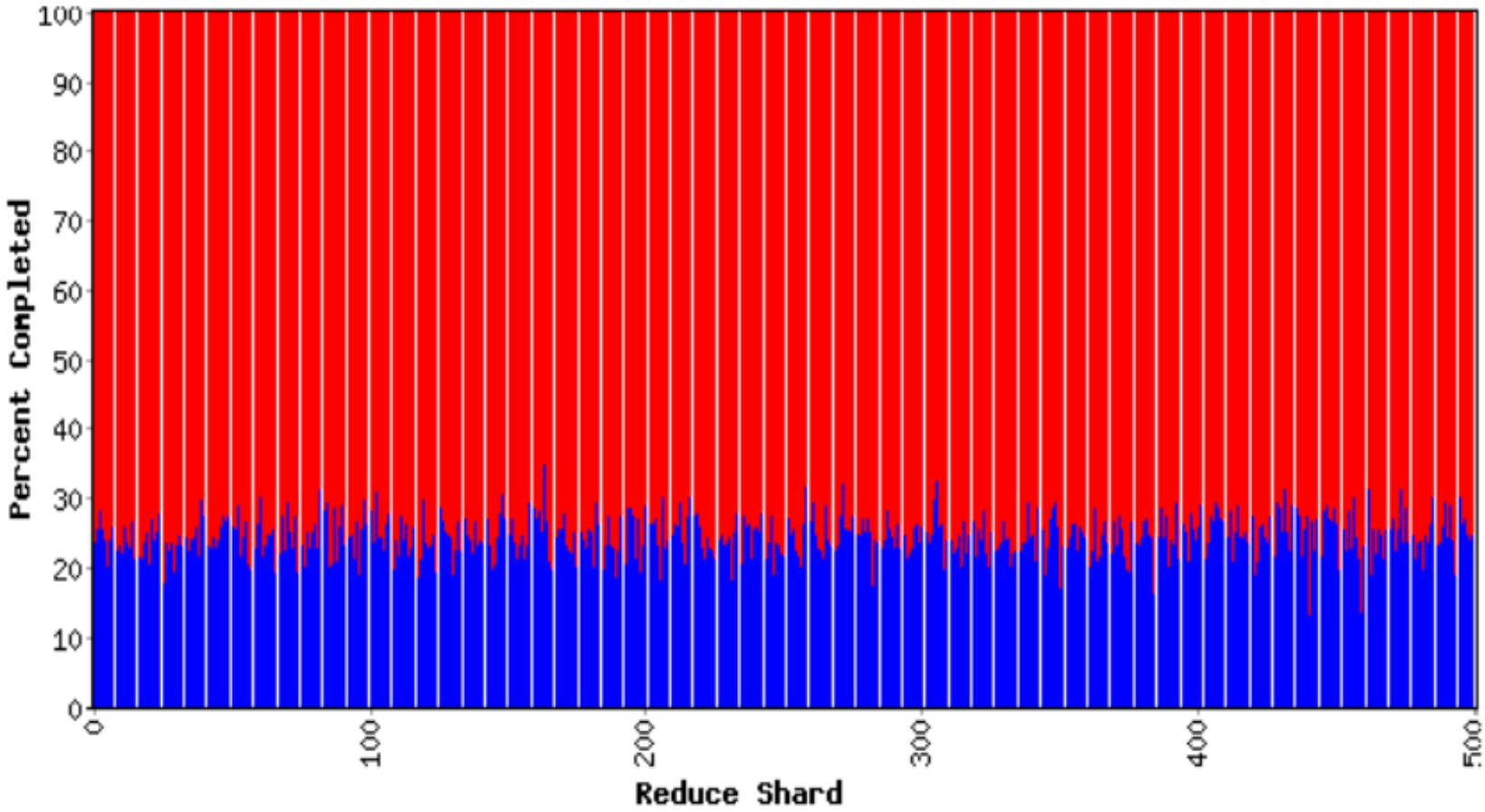
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.3 | |
| Shuffle (MB/s) | 0.5 | |
| Output (MB/s) | 45.7 | |
| doc-index-hits | 2313178 | 105 |
| docs-indexed | 7936 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 1954105 | |
| mr-merge-outputs | 1954105 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 523499.5 | 523499.5 |
| Reduce | 500 | 0 | 500 | 523499.5 | 133837.8 | 136929.6 |




Counters

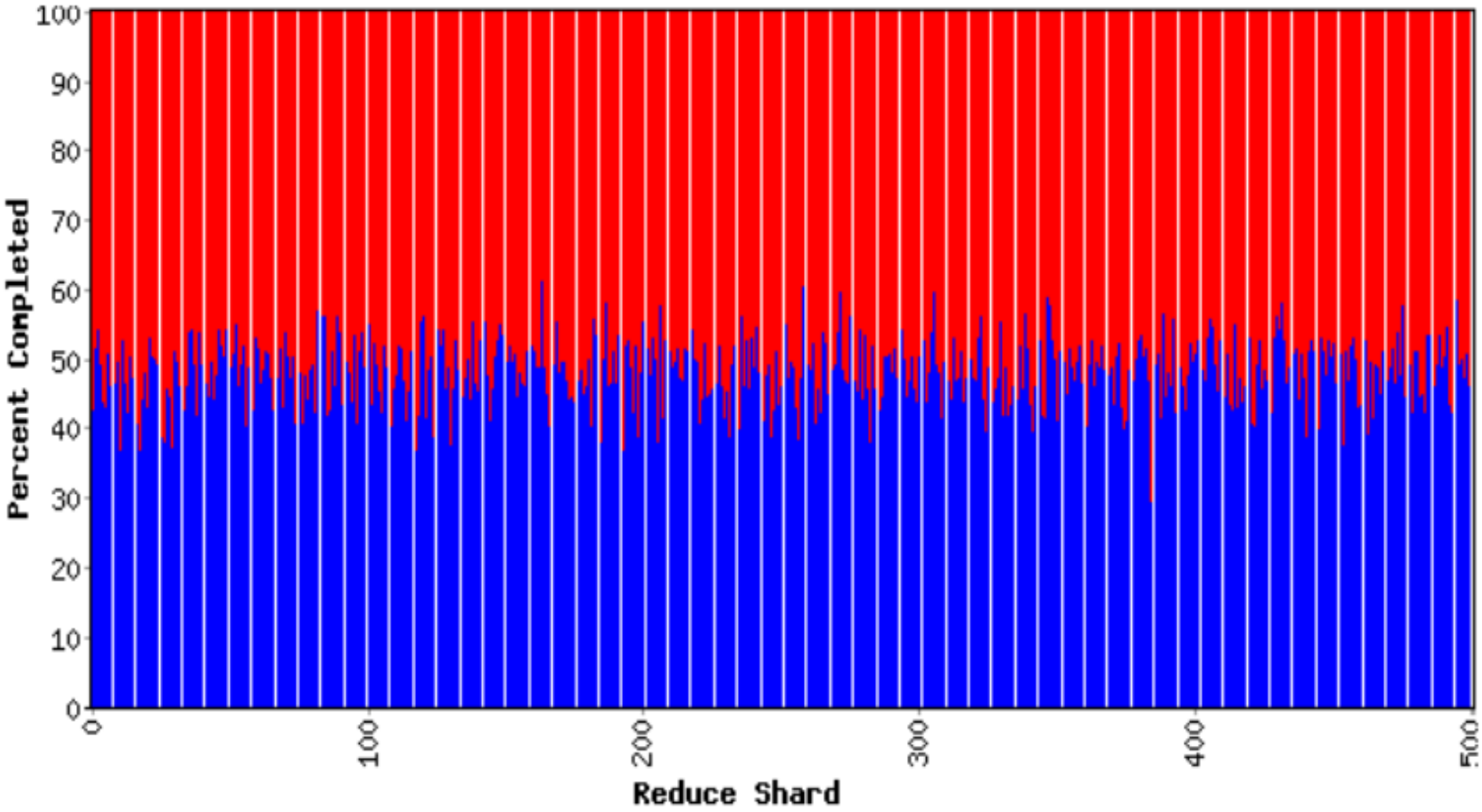
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.1 | |
| Output (MB/s) | 1238.8 | |
| doc-index-hits | 0 | 1 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 51738599 | |
| mr-merge-outputs | 51738599 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 523499.5 | 523499.5 |
| Reduce | 500 | 0 | 500 | 523499.5 | 263283.3 | 269351.2 |




Counters

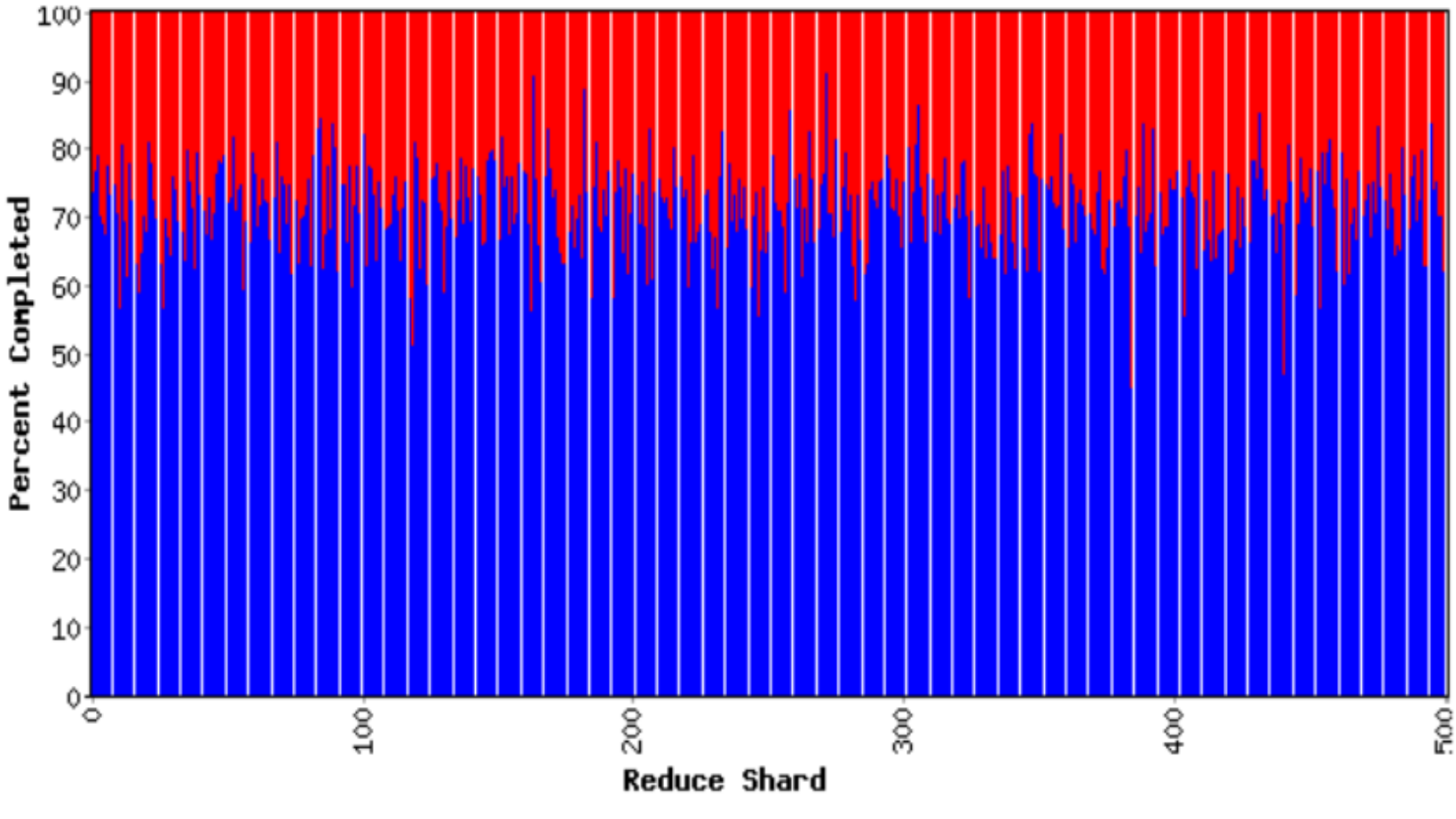
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.0 | |
| Output (MB/s) | 1225.1 | |
| doc-index-hits | 0 | 1 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 51842100 | |
| mr-merge-outputs | 51842100 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 523499.5 | 523499.5 |
| Reduce | 500 | 0 | 500 | 523499.5 | 390447.6 | 399457.2 |




Counters

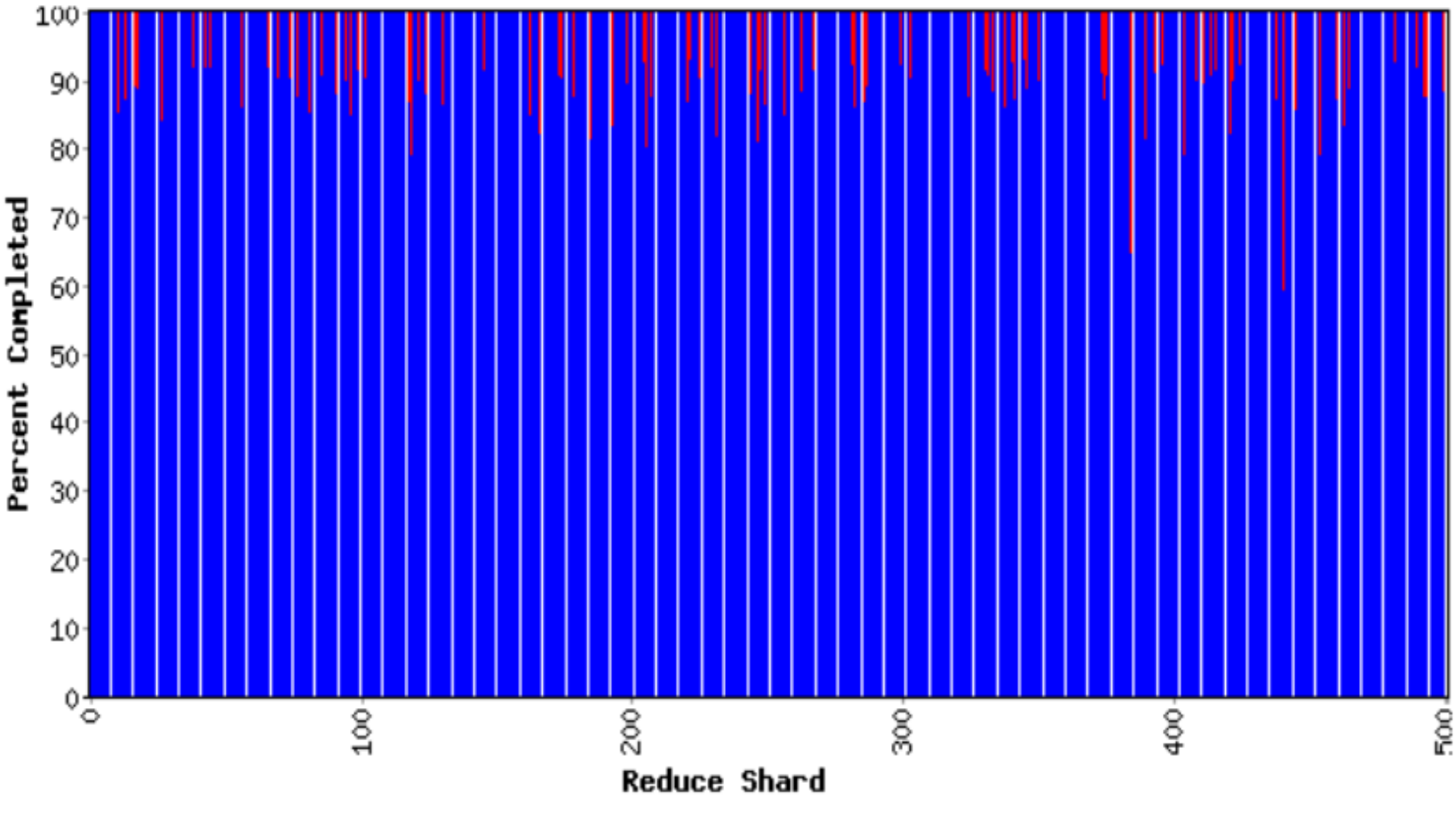
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.0 | |
| Output (MB/s) | 1222.0 | |
| doc-index-hits | 0 | 1 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 51640600 | |
| mr-merge-outputs | 51640600 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 520468.6 | 520468.6 |
| Reduce | 500 | 406 | 94 | 520468.6 | 512265.2 | 514373.3 |




Counters

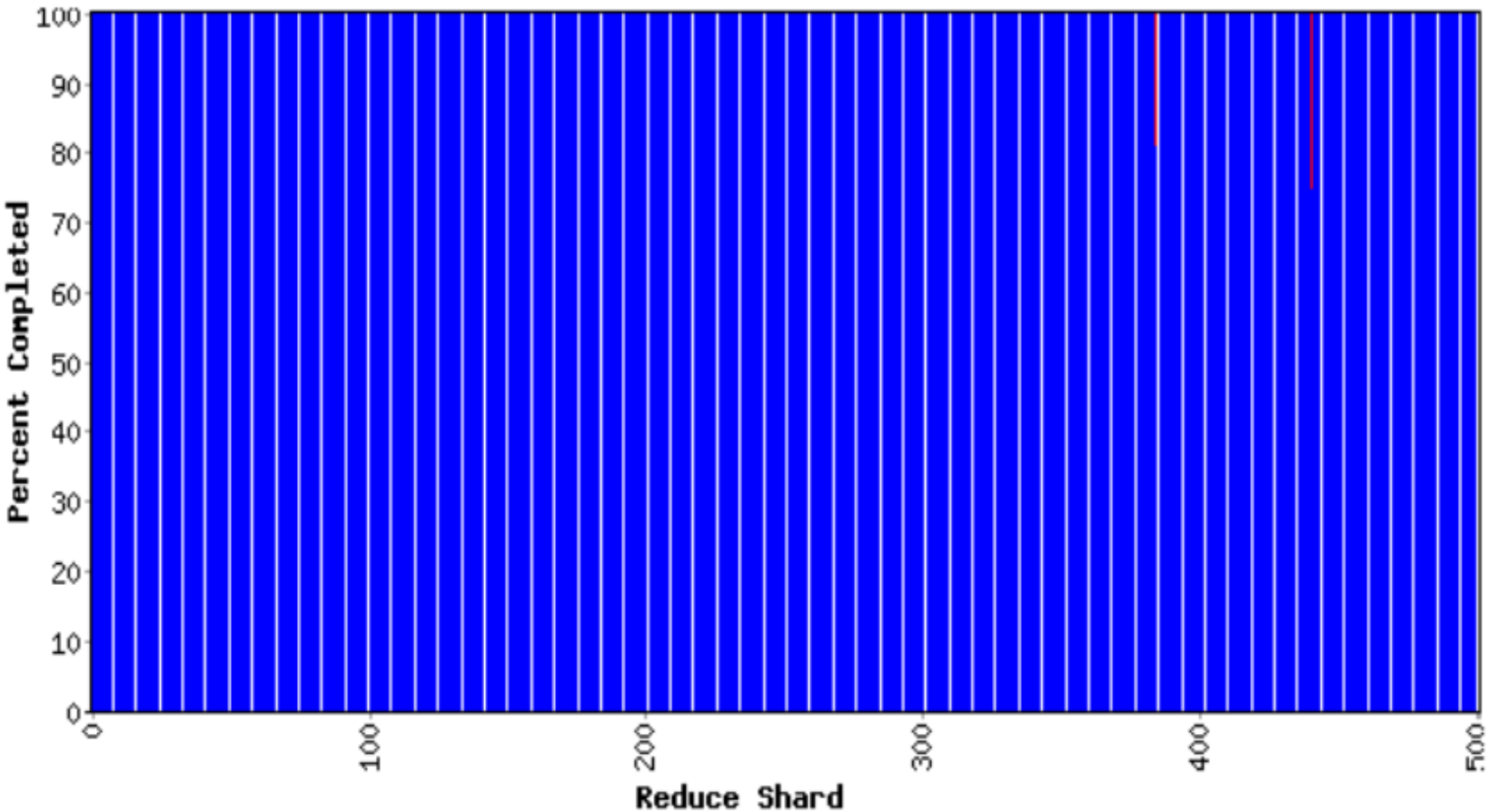
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.0 | |
| Output (MB/s) | 849.5 | |
| doc-index-hits | 0 | 1 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 35083350 | |
| mr-merge-outputs | 35083350 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 519781.8 | 519781.8 |
| Reduce | 500 | 498 | 2 | 519781.8 | 519394.7 | 519440.7 |




Counters

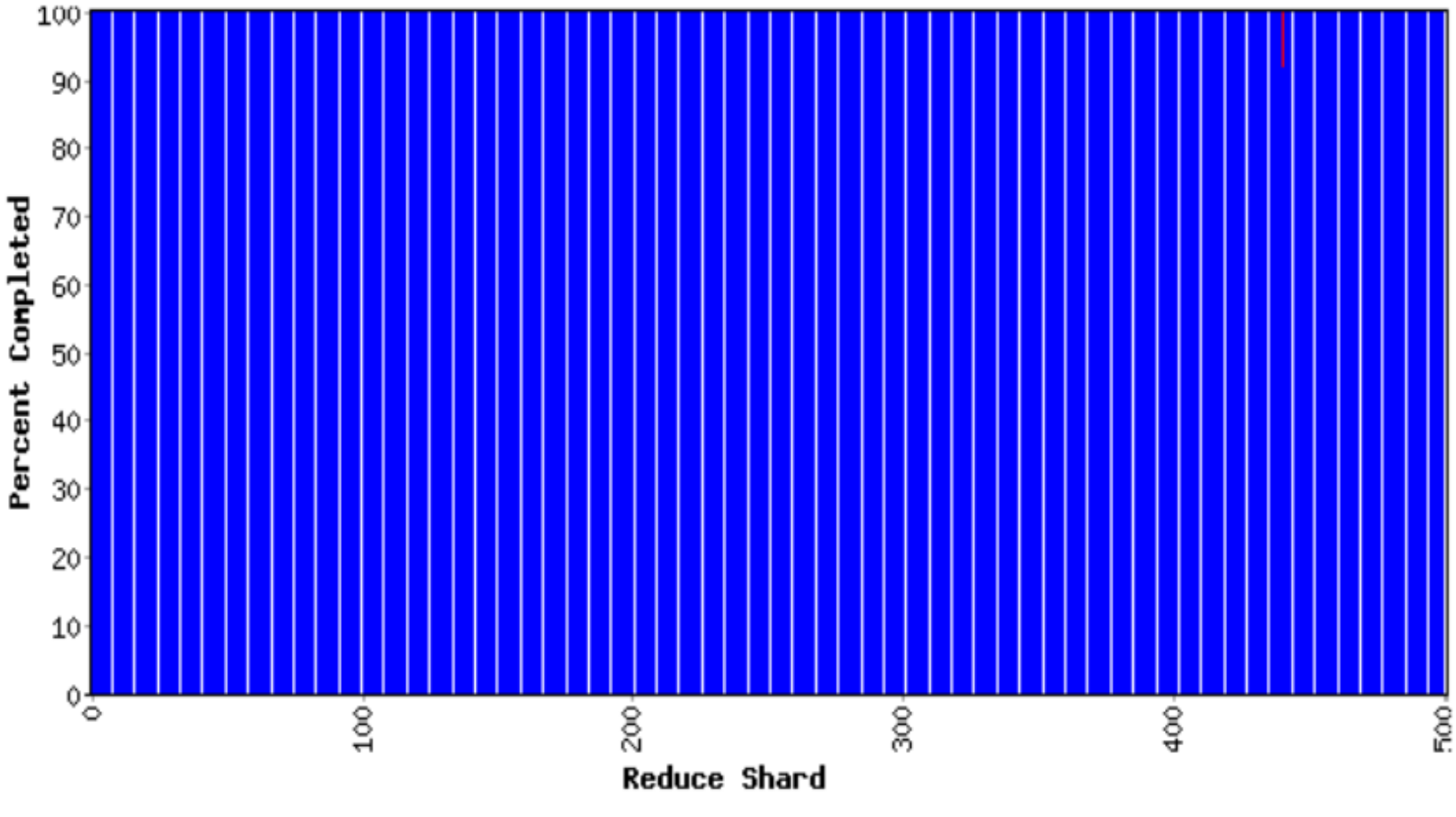
| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.0 | |
| Output (MB/s) | 9.4 | |
| doc-index-hits | 0 | 1056 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 394792 | |
| mr-merge-outputs | 394792 | |

# MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

| Type | Shards | Done | Active | Input(MB) | Done(MB) | Output(MB) |
|---|---|---|---|---|---|---|
| Map | 13853 | 13853 | 0 | 878934.6 | 878934.6 | 523499.2 |
| Shuffle | 500 | 500 | 0 | 523499.2 | 519774.3 | 519774.3 |
| Reduce | 500 | 499 | 1 | 519774.3 | 519735.2 | 519764.0 |




Counters

| Variable | Minute | |
|---|---|---|
| Mapped (MB/s) | 0.0 | |
| Shuffle (MB/s) | 0.0 | |
| Output (MB/s) | 1.9 | |
| doc-index-hits | 0 | 105 |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 73442 | |
| mr-merge-outputs | 73442 | |

# Fault Tolerance

Failures of workers are very likely

# Worker Failure

- The master pings every worker periodically.
- If no response is received from a worker in a certain amount of time, the master marks the worker as failed.
- Any map tasks completed by the worker are reset back to their initial idle state, and therefore become eligible for scheduling on other workers.
- Similarly, any map task or reduce task in progress on a failed worker is also reset to idle and becomes eligible for rescheduling.

# Worker Failure

- Completed map tasks are re-executed on a failure because their output is stored on the local disk(s) of the failed machine and is therefore inaccessible.
- Completed reduce tasks do not need to be re-executed since their output is stored in a global file system.
- When a map task is executed first by worker A and then later executed by worker B (because A failed), all workers executing reduce tasks are notified of the re-execution. Any reduce task that has not already read the data from worker A will read the data from worker B.

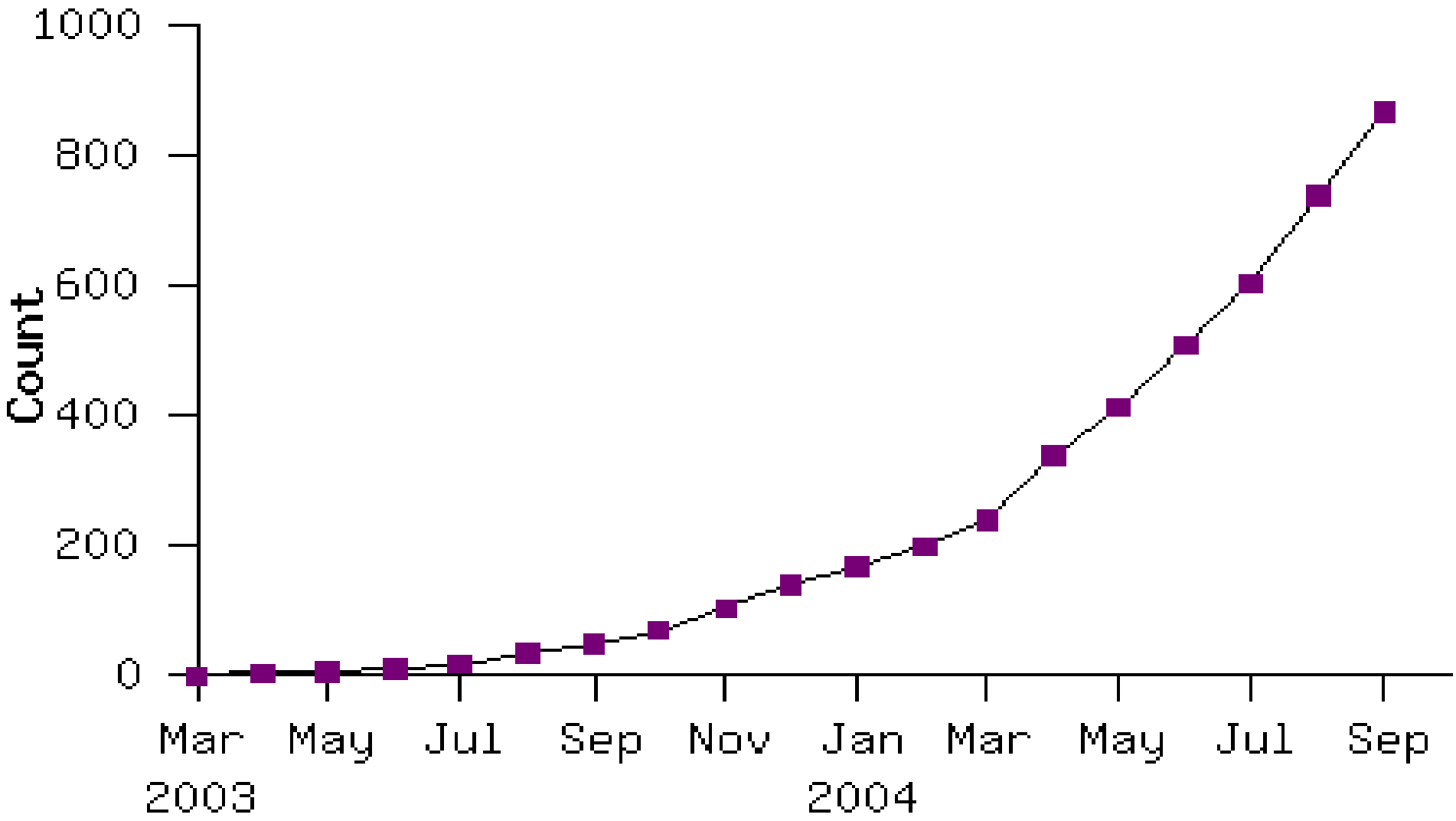
# Master Failure

- Easy to make the master write periodic checkpoints of the master data structures
- However, failure of a master is unlikely
  - Restart whole MapReduce

# Refinements

- Backup Tasks
- Input and Output Types
- Locality (GFS)
- Partitioning
  - `hash(Hostname(urlkey)) mod R`
- Combiner function

# MapReduce Programs In Google Source Tree



References:

http://labs.google.com/papers/mapreduce-osdi04.pdf
http://labs.google.com/papers/mapreduce-osdi04-slides/