Distributed Systems Assignment – 3

Cs17b034

1. Read Google's MapReduce:
   https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
   Write a two to three page report on the same. Avoid copying sentences from the paper.

   Why MapReduce:
   MapReduce was developed by google, To provide better search results it required to index all web documents and to count the URL access frequency etc. which are simple operations but it requires to go through entire documents that are present with google. Since the data is large and stored on different nodes in the network. If the operation is executed on single node and data is brought to it through network. it becomes inefficient.
   Let us assume downlink speed of a node to be 5MB/sec.
   And there are around 1billion documents. With each document taking around 10KB.
   Therefore total size of data is 10TB.
   Then it takes for a node to compute the operation is 2 million seconds which is around 23 days.
   Instead of bringing data to the node we can take operations to the data. This is the idea of MapReduce.
   In Google, data is stored on hundreds of thousands of commodity machines, which are generally a dual core x86 processor running Linux, with 2-4 GB memory and connected using commodity network hardware typically with 100Mb/sec machine level.
   Since there are lot of machines involved there is a high chance that a node fails.
   There for to do operation on a cluster of nodes it should take care of fault tolerance, parallelization, Load balancing and data distribution.
   Which makes the operations more complex.

   What is MapReduce:
   Mapreduce is a Library.
   It provides interface to do operations on cluster.
   It handles fault tolerance, Data distribution, Load balancing, parallelization.
   It brings computation close to data.
   Stores replicas of files for reliability and durability.
   Follows googles data manipulation model.
   Uses googles Global Distributed file system.

   How to use MapReduce:
   MapReduce is a programming model and have an associated implementation.

   MapReduce programming model:
   Computations take input as Key, Value pairs and produces outputs as alist of key,value pairs.
   Computations can be done by two operations.

Map operation:
Takes a key , value pair as input.
Produces a list of key,value pairs as output.
Key k1,value v1 → Map → {key k2, Value v2}.
Her k1 belongs to input domain and k2 belongs to output domain.
K2,v2 are intermediate pairs.

Here runtime aggregate all pairs which contains same key and converts them into a key and list of values.
{k,v1} → k,{v}

Reduce operation:
Takes a key and list of associated values and produces a key and list of values but in general one or zero values.
Key k1, list of value v1 → Reduce → Key k1, list of value v2
Here both keys belongs to the same output domain.

Example Application of MapReduce Programming Model:
Consider the problem of counting the number of occurrences of each word in a large collection of documents.
Solution:

```
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));
```

How to implement MapReduce:
MapReduce interface can be implemented for different environments.
Google implemented it for its infrastructure environment.

Implementation details:
Step 1:
MapReduce Library in the user program is invoked by user.

Then library splits the input files into M pieces(typically each piece takes around 16 to 64 MB).
Then it starts many copies of the program on a cluster of machines.

Step 2:
One copy among them is the master and the rest of them are workers.
Master assigns work to the workers.
M – map tasks and R – reduce tasks.
Picks an idle worker and assigns either a map task or a reduce task.

Step 3:
Worker who got assigned map task reads the content of the corresponding split.
Parses key, value pairs from input and passes each pair to the user defined Map function.
Intermediate key value pairs produced are buffered in the memory.

Step 4:
Periodically, buffered pairs are written to the local disk. Which is partitioned into R regions. Locations of the buffered pairs on the local disk is passed back to the master.
Master is responsible for forwarding the location information to reduce workers.

Step 5:
Reduce worker when gets notified by master about location, it uses RPC to read buffered data. After Reading all intermediate data, it will sort them by keys so it can group them by keys. If memory is not sufficient it uses external sort.

Step 6:
Reduce worker iterates over the sorted intermediate data. And for each unique key it passes that key and its list of values to the Reduce operation which is defined by user. Here list of values is passed as an iterable.
Appends the output of the reduce operation to the final output file which is global.

Step 7:
When all map reduce tasks are completed, master wakes up users program at this point MapReduce call in the user program returns.

Fault Tolerance in MapReduce:

When master fails. The computation is failed and user gets notified based on which user can decide whether to redo the computation or not.

Worker failure is detected by master by pinging worker periodically. If a worker fails to respond in a time frame master assumes the worker as failed.

When a map worker fails
- Map tasks in-progress at failed worker are set to idle for rescheduling.
- Map tasks completed at failed worker are reset to idle.

Output of map is stored on local disk. If the worker goes done after task completion.

Still the local files are inaccesable and requires to redo and reduce workers gets notified regarding the same.

When Reduce worker is failed

In progress tasks are reset to idle.

We can do refinements to the programming model like

1. Backup tasks.
2. Combiner functions.
3. Partitioning function.

Results:

In GREP program, looking for a particular pattern on 1 TB of data with

M = 15000 and R = 1

Data transfer rate increase as more machines are assigned and peaks at 30 GB/s.

Data transfer rate drops as map tasks complete.

Took around 150 seconds to complete the task[including a minute for startup].