CSE 291 Homework #2

Spring 2016 (Kesden)

Data-Intensive Scalable Computing (DISC)

1. HDFS is implemented as a user-level file system vs an in-kernel file-system. (a) What is the advantage of this in the context of Hadoop?

First, it is safer to run in user mode than in kernel mode. If the user-mode HDFS crashes, then other unrelated applications will not be affected.

Second, it is more portable for HDFS to use the local filesystem provided by the operating system. The filesystem may be FAT32 in one host, and EXT4 on the other. HDFS doesn’t have to know about the implementation of local filesystem.

1. The output of a Mapper is written into the local filesystem instead of the global filesystem. Why? Your answer should explain both why writing into the global file system would be undesirable as well as why it would be of minimal benefit.

The result of mapping usually is not the final result. Usually the result of mapping will be processed locally before being merged with data from other hosts. So writing it to the global filesystem is not necessary. However, writing it to the global filesystem can cause heavy communication (called “shuffling”) which takes a lot of unnecessary time.

1. Why does Hadoop sort records en route to a Reducer? How would it affect things if these records were processed by the Reducer in the order in which they were received from the various Mappers?
2. Human prefer to read data in sorted order.
3. Sometimes the records are to be searched using techniques like binary sort, which benefits from sorted data.
4. During the reduce step, a very common operation is to reduce-by-key. If data are sorted by key, then it will be more efficient because data with same key are close to each other, and are very likely to be on the same host.

As an interesting aside, since the keys are sorted en route to a reducer, it is possible to write a massively distributed sort via MapReduce. And, this is pretty cool.

1. How is the failure of a Mapper or Reduce managed?

If a Worker fails, it is marked as bad, and its work is rescheduled to another Worker. Furthermore, any Reduce that was scheduled to get results from the old Worker is told to begin getting them from the new Worker, instead.

If a mapper dies, since the related data result is stored on its local filesystem, the result is lost. So the new worker should recompute it from scratch.

In contrast, if a reducer dies, since the related data result is stored on the global filesystem, it is still available.

1. In a typical Map-Reduce graph algorithm, what data structure is used to represent the graph? Why?

Adjacency list.

Because using the information of very few nodes’ adjacency list, one worker can

* + Perform local computations in mapper
  + Pass along partial results via outlinks, keyed by destination node
  + Perform aggregation in reducer on inlinks to a node

And they are parallel!

1. In a typical Map-Reduce graph algorithm, how many Map-Reduce phases are typically necessary before the graph can be traversed? Why?

Processor Allocation and Migration

1. If processor allocation is optimal, is it possible that migration will subsequently improve system performance? If not, why not? If so, how?
2. Why are periodic broadcast advertisements often considered to be a poor way of communicating information about resource availability? What is the risk?
3. Please explain two commonly used alternatives to the advertisements mentioned above and the relative costs and benefits.

Distributed File Systems

1. In class we observed that AFS and NFS manage consistency differently. AFS issues callbacks upon updates. NFS validates the client cache periodically.
2. Do either of these mechanisms eliminate the window of vulnerability? If so, how? If not, is possible to eliminate the window of vulnerability? Why or why not?
3. Which mechanism will result in less network traffic in the event that many dozens of clients have the same file open for high-frequency random-access reads?

Security

1. Consider *Onion Routing* and the case of a compromised router. In this worst case, will it know the source of the message, the destination of the message, both? Why?
2. Consider *Onion Routing,* why is the path chosen in advance by an agent of the client, rather than the network hop-by-hop?
3. Kerberos enables a client to communicate credentials to a server. What guarantees that the server will be able to trust these credentials? (Covering this on Monday)
4. Kerberos uses *symmetric/secret key* cryptogrophy, rather than *asymmetric/public key* cryptopgraphy. Why? (Covering this on Monday)