Homework 6

CS550 (V01)

Amit Nikam (A20470263)
anikam@hawk.iit.edu
Fall 2020

---

1 **Ans:** In application-aware I/O data access is application dependent. In application-aware I/O optimization we combine SSD technology with Parallel File Systems. A data service is plugged in as normal but with smart selective SSD as cache. We optimize this via application-aware.

2 **Ans:** Parallel File systems are used to store and provide data that is usually extremely large and can't fit in memory and is stored persistently. It is designed for parallelism while maintaining high performance and also provides globally shared name-spaces. Due to it high-performance design it is used in complex computational science tasks where data access is a huge challenge.

3 **Ans:** In file-per-process each application task creates a separate file and writes to only that file while in single-file access only single file is accessed by all applications. Thus lock contention on file systems are avoided in file-per-process but shared-file increase usability. Major issues with file-per-process is that it may create too many files as processes might be many and this complicates the system making it impossible to restart application with different number of tasks, while this issue is not faced in single-file, performance might be hindered due to lock contentions.

4 **Ans:** Since logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence, these sequence of bytes are stored in a set of objects on PFS servers and are mapped by the Metadata associated with the respective file. This mapping is usually decided at the time of file creation and is often round-robin distribution of fixed extent size. Since space is allocated on demand, unwritten parts in the logical file do not consume disk space. A static file chunk mapping from a logical file to objects allows clients to easily calculate server(s) to contact for specific regions eliminating need to interact with a metadata server on each I/O operation.

5 **Ans:** For most times PFS use locks to manage concurrent access to files. These locks are handled by the distributed lock managers which grant or revoke locks to or from clients respectively. This adds extra complexity to the system since extra latency is added and the DLM ends up being a single point of failure. For this clients can be configured to just not write to overlapping regions.

6 **Ans:** Hadoop was introduced as a open-source software framework for storage and processing of large-scale data on clusters of commodity hardware. This ability to run on commodity hardware made it useful. More so, the hadoop software framework provided a good reliability into each hardware by assuming that failure is expected and implementing a few fail-safe measures like data distribution and replication. This sort of solution was not provided by existing solutions at the time and thus hadoop gained more popularity.

7 **Ans:** MapReduce is a method by which a task is distributed across multiple nodes, where each node processes data stores on that node. It has two phases, Map and Reduce. Its advantage is that it is automatically parallelized and distributed, and

also provides a clean abstraction for developers.

8 **Ans:** Since MapReduce sorted the list even in word count, the pseudocode would be the same, just that the output would be key/value paid of sorted values and frequency of that value. Run the below pseudocode in sequence.

Pseudocode:

```
 global_list = [ ]

func Mapper(number_column):
    int_count = [ ]
    for each row in number_columns:                        //for each integer
        int_count =  append( (number_column[row], 1) )     //Count integer once
    return int_count                  //return integers list with 1 count for each

 func concat_int(int_count):            //add all integer 1's list to global list
      global_list.append(int_count)         //after global list is updated each time
      global_list.sort( global_list[:][0] )     //sort itself as per integer values within

 func Reducer(global_list):             //finally reduce once the global_list is complete
      output = [ ]
      for each row in global_list:              //do this for each int in sorted global_list
           if global_list[row] in output[:][0]:          //if integer already in the output list
                output[ global_list[row] ][1] = +1              //increment its count by 1
           else:
                output.append( (global_list[row], 1) )      //else add first entry in output
      return output                                //return sorted reduced list of integers
```

9 **Ans:**

HDFS	PVFS
It runs on commodity hardware	Requires specialized, well planned and organized hardware for performance.
Is more useful for cases with "write-once, read-many-times"	Is useful for cases where high-data reading/writing is required. (eg: Scientific Computations)
often stores entire objects (files) on a single storage node	distribute data across multiple storage nodes
Throughput is more important	Low Latency is important (high-performance)
often run on architectures where the storage is co-located with the	often run on architectures where storage is physically separated from the

application	compute system
take on fault tolerance	runs on enterprise shared storage

Some similarity between HDFS and PVFS are that they single namespace, so once the metadata is available, intelligent client can find location of file blocks directly.

10 **Ans:** The data is distributed at the time of creation in the HDFS. The data is replicated usually three times and divided into blocks of 64MBs of data which can have empty spaces, and are stored across different data nodes. The location of these blocks in the data nodes is stored in the name node. The name node is a single point of failure and is responsible for maintaining the distributed nature of HDFS. The operations are usually written to the data node where they are locally run and the reply is send to the client.

To make the system faster and more reliable I would split the file into partitions such that read times are minimized using ML. And would implement a secondary name node that saves snapshot of the name node from time to time, so that if the main name node fails, it can start from the last snapshot and look into logs after the snap to configure self.

11 **Ans:** A name node gets a heart beat from the data nodes at each three seconds. If a beat is not received at the name node it detects the data node as failed. It then picks a new Data Node for new replicas and then balances disk usage and communication traffic to DataNode. The data is replicated on the new Data Node. This way even if the fault takes place it is quickly recovered from automatically.

12 **Ans:** HDFS are ideally geared for loosely coupled, distributed applications where the applications are on the same node as machine. Usually just the code to be executed is migrated and run on the data node and the output is returned. Worst-case scenario would be one where the entire data is read or being returned.

13 **Ans:** In this algorithm the applications are server one at a time. All I/O requests are serviced within an acceptable period to avoid starvation and also to improve performance. I/O is coordinated and scheduled in time windows based on the application ids. Within the window, small application ids get first preference. In case of the same id, whichever has early issued-time is served first.

14 **Ans:** Yes in-fact it will almost be the same, since the message is coming from Alice with the $K_{A,B}$ instead of one after another.

15 **Ans:** That's because KDC sends an encrypted message as a reply to Alice which only Alice can decrypt if it's truly him. Thus the KDC doesn't need to know for sure if it's Alice because even if it's not, the message is not readable by the third party.

16 **Ans:** Alice can authenticate Bob by sending Bob a challenge, let's say C. Bob will return his challenge response as signature encrypted with Bob's own key. Upon decrypting and verifying the signature with Bob's key, Alice will know that it is Bob indeed.