Mattia Danese
Homework #4
CS118 - Cloud Computing
Professor Sambasivan

Question 1
The Metadata Server, also known as the Master, has several crucial roles in GFS. First, it stores all system metadata: namespaces, access control information, and file path to physical Chunksever location mappings. While all types of metadata are important and serve their own purpose, the most impactful metadata stored by the Metadata Server is the mappings from file paths to physical locations because this allows clients to then directly contact Chunkservers instead of querying the Master for every file read and write. In turn, this sidesteps a clear opportunity for the Master to become a bottleneck and greatly aids in the scalability and concurrent usage of GFS. Storing these mappings also allows for the state of the entire filesystem to be known which is extremely helpful in the case of a node, Master or Chunkserver, failing. That being said, storing namespaces and access control information also has its own benefits of knowing specific properties of each file (i.e. number of replicas) and each file's permissions, respectively. Additionally, the Metadata Server is responsible for controlling system-wide activities, such as Chunk lease management, garbage collection, and Chunk migrations. These activities encompass the mutation of a file, post file-deletion procedures, and moving (replicas of) files across Chunks and Chunkservers.

Question 2

The reason why GFS is traditionally thought of as a poor design choice for workloads that use many small files instead of very large ones comes down to the size of each GFS Chunk. Seeing as the size of each Chunk is relatively big (64 MB), small(er) files might be spread out across just a few Chunks or even just one Chunk. If such a file has high demand (i.e. frequent reads and writes), then the few (or one) Chunks that contain the file may become hotspots (i.e. frequently and simultaneously queried to the point of being overloaded). This issue has mostly gone unaddressed, seeing as the vast majority of files stored by GFS are very large, though a potential solution is to simply replicate small files with high demand more than usual. This would then allow for the intense demand of those files to be spread across more Chunkservers, which would decrease the probability of any one Chunkserver becoming a hotspot, at the expense of storage space. Clearly, this is not a scalable solution if the vast majority of files in the workload are small because the available storage space would have to increase by 100% (or at least a percentage very close to 100%) for every additional replica needed (assuming the majority of original files would need additional replicas). Furthermore, if the Chunk size is rather big and the workload is comprised of many small files, then it is fairly likely that the Chunk size will be bigger than the majority of file sizes. In such a case, when a Client requests a particular file and instead receives the entire Chunk the desired file is located in, network and Client resources would be unnecessarily wasted in order to now store the entire Chunk. As such, I do agree that the *design choice* of GFS is not optimal for workloads that use many small files instead of very large ones.

Question 3

Suggesting to use the GFS system when very low latency reads and writes are desired is not a good idea because, as stated in the final assumption of the GFS paper, GFS is tailored to prioritize high sustained bandwidth at the expense of low latency[1]. Thus, design choices of GFS were made in accordance with this assumption. Such design choices are: pushing data linearly across linked Chunkservers rather than distributing data across some other topology (i.e. like a tree) and the chunk replica placement policy (i.e. spreading replicas across machines *and* racks). Additionally, the authors of GFS admit that the speed of writes to files is subpar, which they have observed to increase client latency though claim is not an issue as the effect on the aggregate write bandwidth is rather minimal. That being said, some measures have been taken to decrease latency across GFS; such as, forwarding data to the "closest" machine that has not already received it and pipelining data across TCP connections. However, these efforts were made to solely decrease latency and prevent high-latency links, they were not made to bring latency down to as low as it can be. Seeing as GFS is clearly optimized for high bandwidth and not low latency, suggesting the GFS architecture for a system where low-latency is desired is not a good idea.

[1] https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

Extra Credit

What is the difference between high-bandwidth and low-latency? When would one be preferable over the other? Name a real-world example or service (i.e. Netflix or Online Gaming) where you would want each.