Quiz on GFS

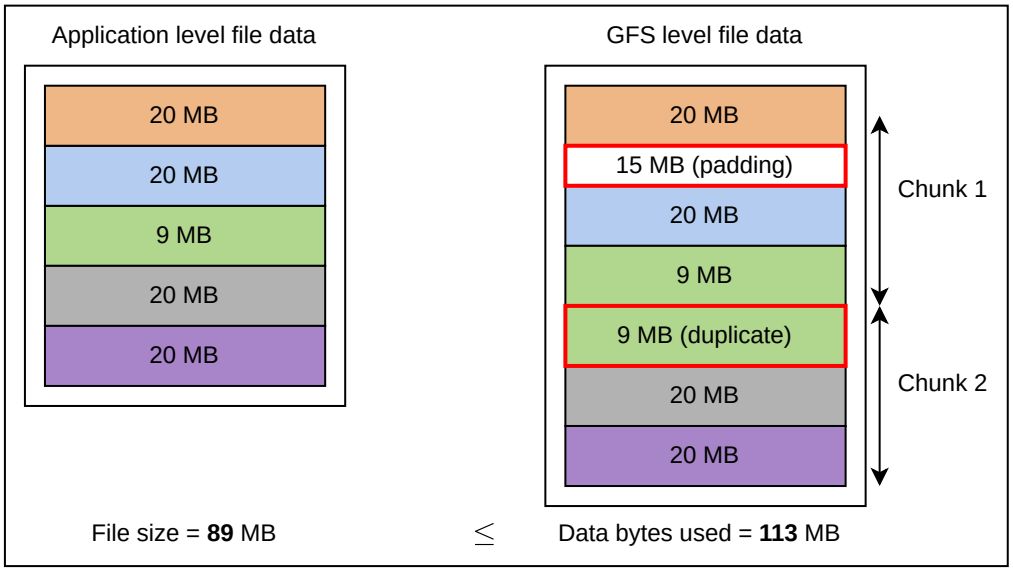
Test your understanding of concepts related to the design of Google File System via a quiz.

Question 5

How can a client find the right chunk in the presence of padding?

Hide Answer

Because of padding and record duplicates, the application-level file size would be less or equal to the bytes the system has occupied to store that file. To read a particular data byte, the clients have to find the chunk number that contains that data byte. Since there are paddings and record duplicates, we may not land on the right chunk just by dividing the data byte offset with chunk size.



If a client is reading a file sequentially from start to end, then it has to iterate over all the chunks; there is no need to find a specific chunk. The clients identify the padding and record duplicates while reading all the chunks, discard the padding and record duplicates and return the actual

data bytes to the end user.

> The padding and record duplicates can be identified using checksums and special markings that are stored with the data.

If a client starts sequential reading from a random offset, or if it's a small random read, then the client needs to know the right chunk/s containing the requested data byte. The client doesn't know how much padding or recorded duplicates are present in the file chunks; it can't just find the right chunk in the first place, nor can it start its search for the right chunk starting from a random estimated chunk. The client has to iterate over all the chunks from the start until it finds the chunk with the requested data byte, which is very costly if we do it on each read. It is on the applications how they tackle these problems on their side.

An application might put hints while writing the records that help later readers approximate the application-level byte index.

We encourage you to brainstorm other ways to find out the application-level byte index efficiently when the underlying file might be mutating concurrently.

Question 1

Do all the chunks of a file have a lease associated with them all the time?

Hide Answer

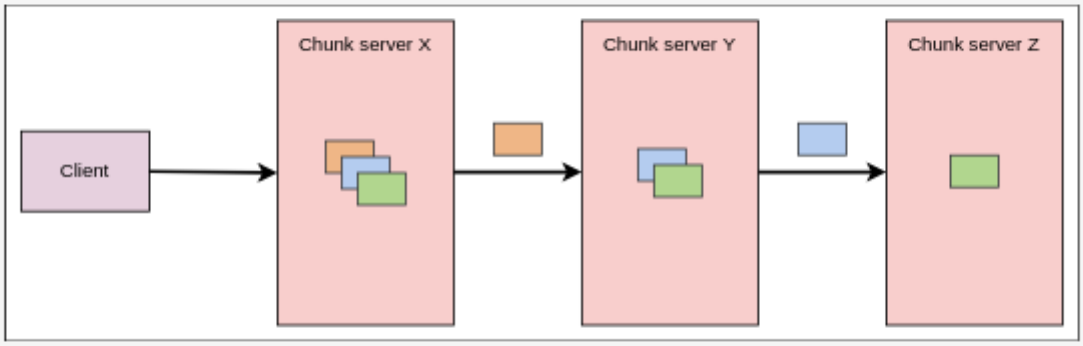
No, not all the time. If there are no mutations to be performed on a chunk, there is no need to associate a lease with it. When a client requests a mutation on that chunk, the manager will give the lease to that chunk for a limited time. If the mutations are in progress and the lease time is going to expire, the extension can be requested from the manager.

Question 2

Why is there a pipelined way of data flow in writes?

Hide Answer

Pipelining helps the system utilize each machine's network bandwidth. If one machine, the primary replica, has to forward the data to all secondaries, the outbound bandwidth of the primary replica will be divided among multiple secondaries, and the secondary replica's outbound bandwidth wouldn't be utilized hence wasted. All the burden will be on the primary replica, and it will take time to send data to all replicas. In comparison, if one replica forwards the data to the second replica, the second replica forwards it to the third, and so on, then each machine's outgoing bandwidth will be utilized. By pipelining the data transmission over a TCP connection, we will be able to send the data to all replicas as fast as possible. Once a replica receives some data, it forwards the data immediately to its closest replica on the network, as shown in the following illustration. The closest replica forwards the data to its next closest replica that hasn't received the data yet, and so on. It helps avoid network bottlenecks and minimize latency.



The orange, blue, and green boxes in the above image represent different parts of the total data that is being sent to a chunkserver.

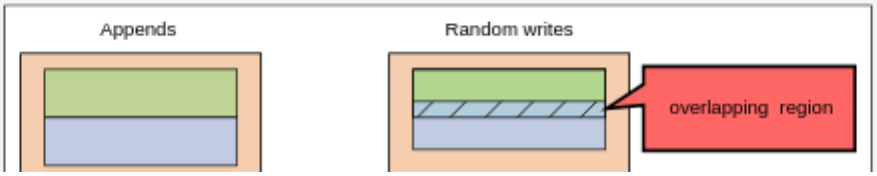
> The network distance between two machines can be measured accurately and is often a function of a typical topology.

Question 3

Why are appends always defined while concurrent random writes might not?

Hide Answer

For appends, the system chooses the offset at which the data has to be written so one append operation can't overwrite or overlap the other append operation's data, and the region will always be defined. In random writes, the client specifies the offset. So, the concurrent random write operations can write data to the same or overlapping file regions, resulting in mixed data from multiple writes producing undefined regions.



Question 4

What if a failed manager comes back up while a new one is initiated? How will such a situation be managed?

Hide Answer

There should be only one manager at a time because the GFS architecture doesn't support distributed metadata models. There are no mechanisms to deal with multiple manager problems like inconsistencies, and so on. If the manager fails at one machine, a new instance of the manager is started on another machine. The clients are redirected to the new manager using the canonical name of the manager, which is a DNS alias that is updated as soon as the manager is restarted on a new machine.

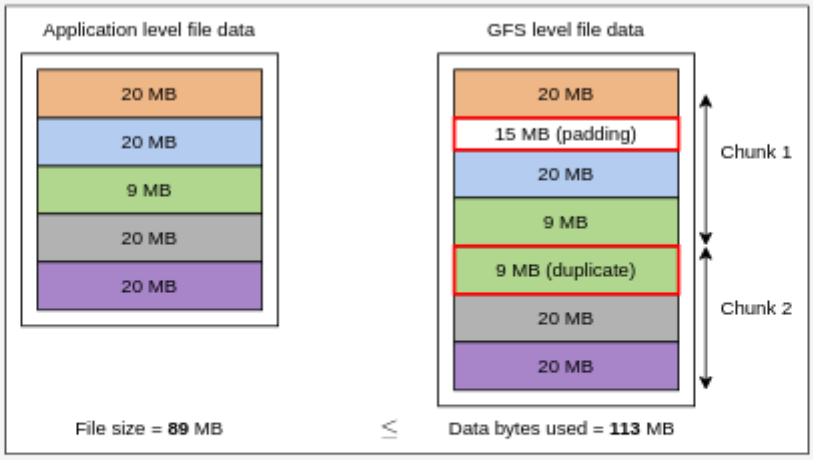
The problem occurs when the failed manager comes back, and some clients have access to it. We have a new manager now, and it would be wrong if some clients accessed the old manager and others accessed the new one. The operations performed on the old manager wouldn't be reflected in the new one, and vice versa. Such a scenario causes metadata inconsistencies. We got into the complexities due to which we have chosen a single manager in our design. So, to avoid blocking clients from being directed to different managers simultaneously, we can use a locking mechanism such as Chubby. For a manager to start acting as a primary manager, it must exclusively hold a lock; there can only be one entity that can hold that lock at a time.

Question 5

How can a client find the right chunk in the presence of padding?

Hide Answer

Because of padding and record duplicates, the application-level file size would be less or equal to the bytes the system has occupied to store that file. To read a particular data byte, the clients have to find the chunk number that contains that data byte. Since there are paddings and record duplicates, we may not land on the right chunk just by dividing the data byte offset with chunk size.



If a client is reading a file sequentially from start to end, then it has to iterate over all the chunks; there is no need to find a specific chunk. The clients identify the padding and record duplicates while reading all the chunks, discard the padding and record duplicates and return the actual data bytes to the end user.

> The padding and record duplicates can be identified using checksums and special markings that are stored with the data.

If a client starts sequential reading from a random offset, or if it's a small random read, then the client needs to know the right chunk/s containing the requested data byte. The client doesn't know how much padding or recorded duplicates are present in the file chunks; it can't just find the right chunk in the first place, nor can it start its search for the right chunk starting from a random estimated chunk. The client has to iterate over all the chunks from the start until it finds the chunk with the requested data byte, which is very costly if we do it on each read. It is on the applications how they tackle these problems on their side.