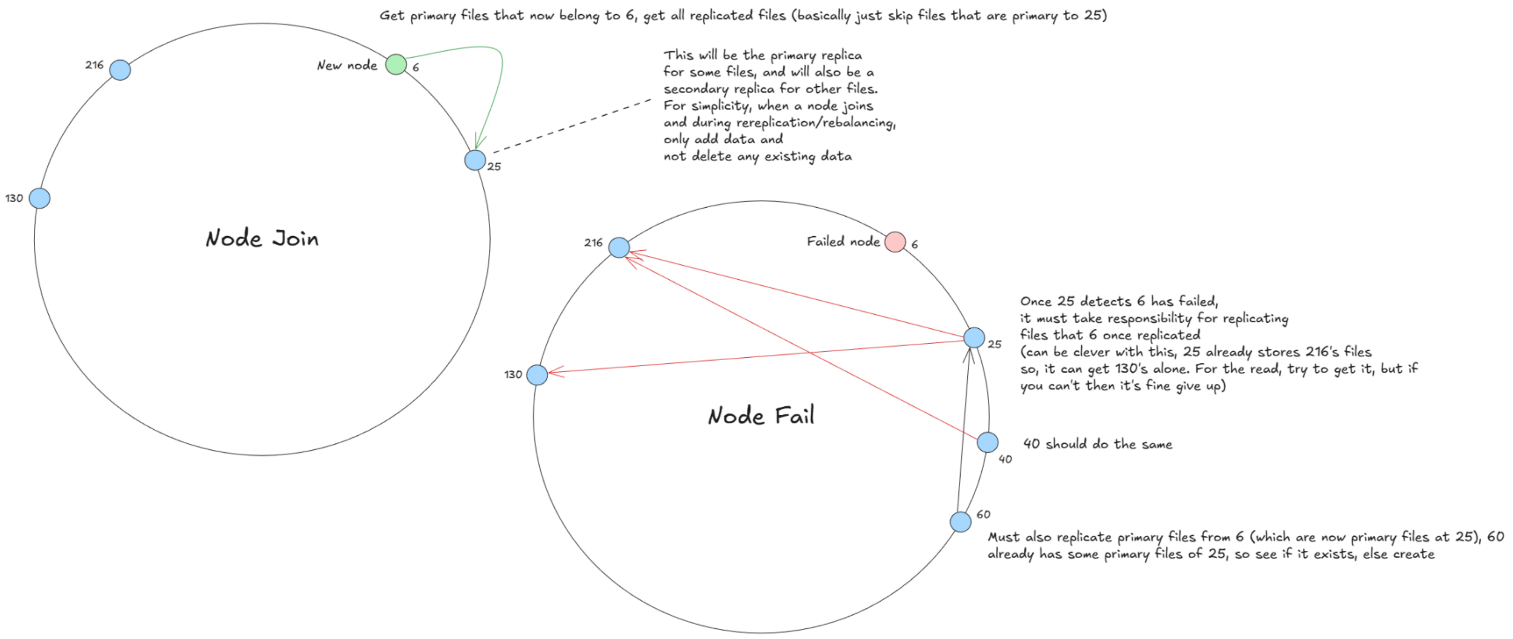
Akil Karthikeyan Anandan (anandan3), Akshat Gupta (akshatg4)

MP3 HyDFS Report

a) Design Overview

Our HyDFS system is built on top of the gossip failure detector (gossip round = 1s, K = 3, Tfail = 5s, Tcleanup = 5s). To satisfy the requirements, we use a replication factor = 3, write quorum of 3, read quorum of 1. We use consistent hashing (SHA1, 64 bits) for a ring of size of 2^64. We use UDP for control messages and TCP for file transfers (we convert file content to base64 for transfer). We can support up to 2 simultaneous failures because at least 1 replica will be alive. On node join, we have the joining node get files that it has to replicate from its successor. On node failure, when the failing node's successors detect that it has failed, they take responsibility for re-replicating files. We store files as chunks (each chunk is either content from a file create or a file append). Each chunk has a unique id (computed by the node that uploads it) that is common across all replicas. Each node's process stores metadata about all the files that it contains and also the order of chunks for each file. Actual file data is stored in the node's local file system under /hydfs directory. This (unique id for each chunk and write-read quorum intersection) ensures that we get per-client append ordering and read-my-writes for successful (write quorum = 3 reached) file appends. There is a background merge thread that runs every 30s (eventual consistency). For each file stored in the node, the thread gets chunk order from the primary replica (ensures order is the same across replicas) and enforces the same order by changing metadata. The thread also creates missing replicas if any and deletes replicas that are not needed.



Get primary files that now belong to 6, get all replicated files (basically just skip files that are primary to 25)

This will be the primary replica
for some files, and will also be a
secondary replica for other files.
For simplicity, when a node joins
and during rereplication/rebalancing,
only add data and
not delete any existing data

Once 25 detects 6 has failed,
it must take responsibility for replicating
files that 6 once replicated
(can be clever with this, 25 already stores 216's files
so, it can get 130's alone. For the read, try to get it, but if
you can't then it's fine give up)

40 should do the same

Must also replicate primary files from 6 (which are now primary files at 25), 60
already has some primary files of 25, so see if it exists, else create

For writes (create/append), we write to all 3 replicas for the file and return success only if all 3 writes have been ACKed. For reads, we issue simultaneous reads to all 3 replicas and return immediately as soon as we get the file from one replica.

b) Past MP Use

Used gossip failure detector from MP2 and grep from MP1 for looking through file logs.

c) LLM Use

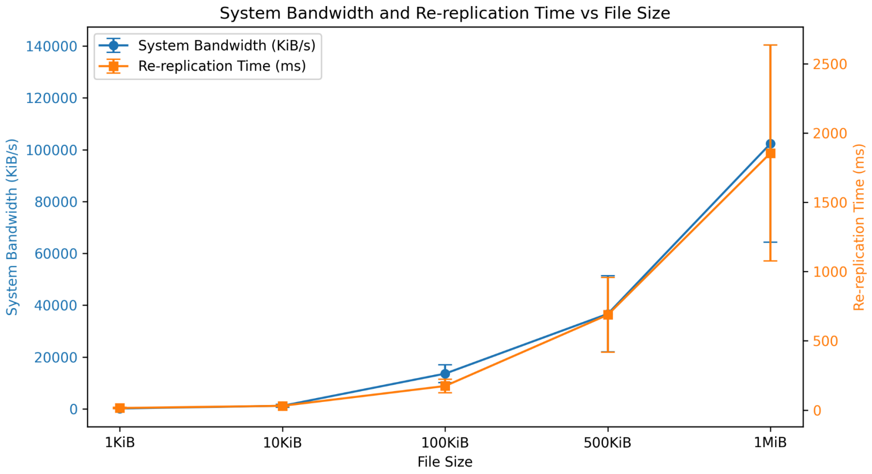
Used LLMs for discussing design (wasn't really helpful) and for Go syntax (very useful) and python code to generate plots (very useful).

https://chatgpt.com/share/69116be2-cbd4-8010-98c3-eeb28b991003
https://chatgpt.com/share/69116b96-73ec-8010-a1bd-11d10622638e
https://chatgpt.com/share/69116c10-f90c-8010-a088-1e0bd08d5b7f

d) Measurements

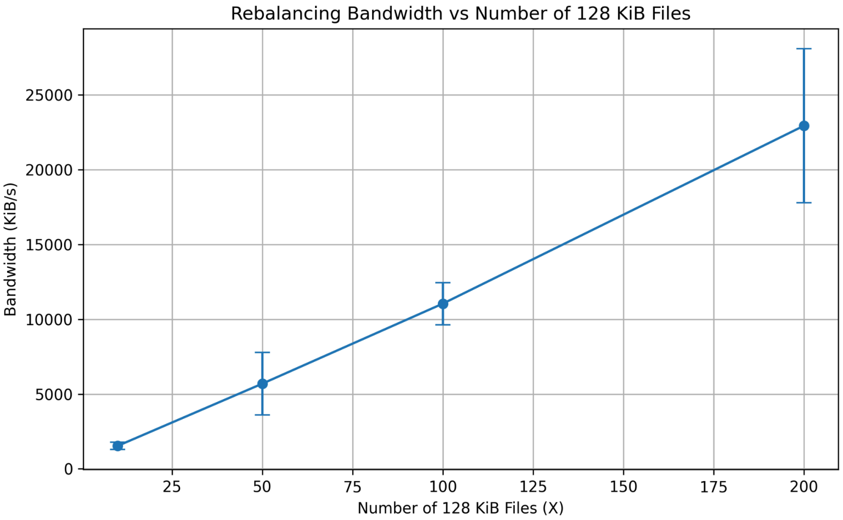
For each point in the following graphs, we took 3 measurements and plotted the mean with the standard deviation.

(i) Overheads



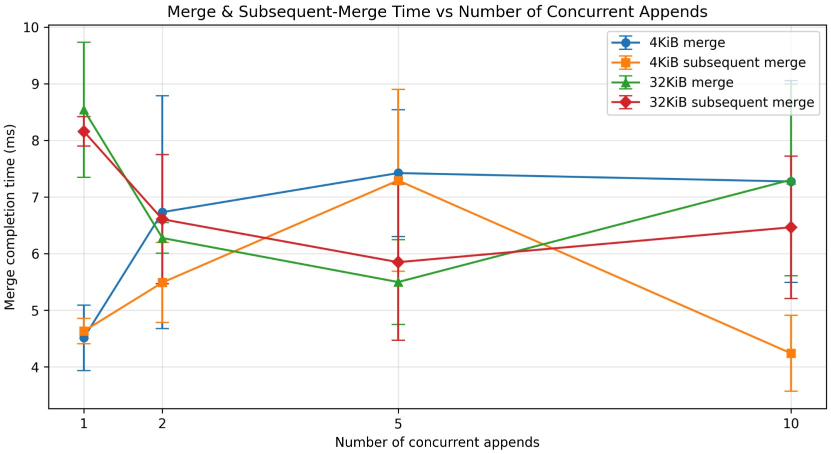
For this experiment we started HyDFS with 10 VMs, preloaded it with 100 files of size X and crashed the HyDFS process in VM10. For re-replication after this failure, as file size (100 files each of size X in HyDFS) of files in HyDFS increases, both re-replication time and system-wide bandwidth increase. This is because the volume of data sent over the network and the processing time increase with increasing file size.

(ii) Rebalancing overheads



For this experiment we started HyDFS with 5 VMs, preloaded it with X 128 KiB files and joined VM 6. Again, when the number of files increased, the system bandwidth also increased because the volume of data sent over the network increases with more number of files in the system.

(iii, iv) Merge performance and Subsequent-merge performance



For this experiment we started HyDFS with 10 VMs, created a file of size 128KiB in it and performed X concurrent appends. For merge performance, we measured time for first merge after these appends to complete. For subsequent-merge performance, we measured time for merge after a merge was already done to complete. We did this both for appends of size 4KiB and also 32KiB. The results do not show anything significant changing when the number of concurrent appends change, merge/subsequent-merge happens or when append size (4KiB/32KiB) changes. During merge, we only get chunk ordering from the primary replica and make sure that local ordering is the same as ordering in primary. That's it! Only when a file/chunk is missing, we get actual file content from primary and add it locally. But, file/chunks are rarely ever missing because our write quorum = number of replicas = 3.