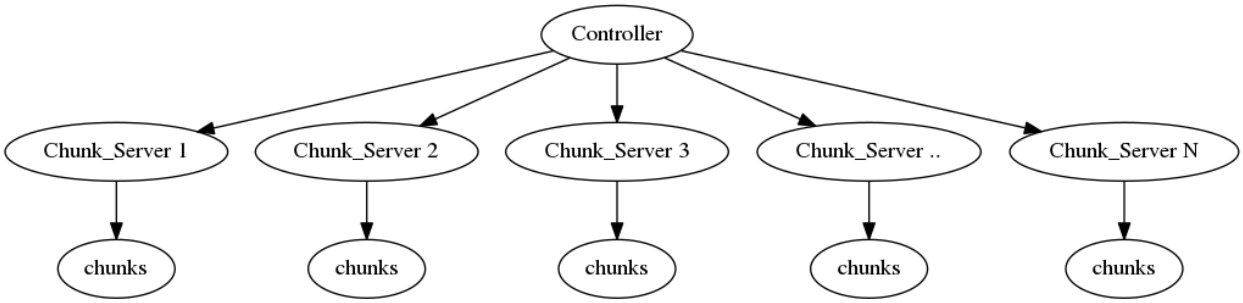
Network Centric Computing Assignment 3

Distributed and Fault Tolerant File System

Deadline : 22nd April

This assignment should be done individually. You can do this assignment in either python or java. The objective of this assignment is to build a distributed, replicated, and failure-resilient file system.



Fragmentation and Distribution [50]

This assignment has several sub-items associated with it.Every file will be split into 64KB chunks. These chunks need to be distributed on a set of available chunk servers, M. Each 64KB chunk keep track of its own integrity, by maintaining checksums for 8KB slices of the chunk. The message digest algorithm to be used for computing this checksum is SHA -1: this returns a 160 -bit digest for a set of bytes. Chunks will be stored as regular files on the host file system.

File writes/reads should be done via the chunk server. The chunk server adds integrity information to the chunks before writing it to disk. Reads done by the chunk server will check for integrity of the chunk slices and will send only the content to the client. Each chunk being stored to a file needs to have metadata associated with it. If the file name is /user/bob/experiment/SimFile.data , chunk 2 of this file will be stored by a chunk server as /user/bob/experiment/SimFile.data_chunk2 . This is an example of the metadata being encoded in the name of the file. There will be other metadata (such as versioning, checksums) which you will typically store in the file. Manage a file that is being actively appended to. Manage 64 KB chunks and disperse it over a set of available chunk servers.

- Make sure that you are able to reconstruct this file from retrieving chunks in the right sequence.
- Manage random writes to the file.
- Manage concurrent writes to the file managed

During the testing process, you will have to read the file that was previously scattered over a set of chunk servers . For reading each 64 KB chunk, the client will contact the Controller and retrieve information about the chunk server that holds the chunk. Assuming there were no failures, the file read should match the file that was dispersed. Next, we will go to an individual chunk file managed by your File System and tamper this by modifying the content of the file. This may be deleting /adding a line or a word to the file: this is done outside the purview of your chunk server . This should cause the file read to report a data corruption, and the specific chunk (and slice within it) that was corrupted.

Replication [30]

Each file should have 3 copies. When a client contacts the Controller node to write a file, the Controller will return a list of 3 chunk servers to which a chunk (64KB) can be written. The client then contacts these chunk servers to store the file. After the first 64KB chunk has been written, the client (this should be managed transparently by your API) contacts the Controller to write the next chunk and repeat the process . [Test Scenario] This test will check to see that a given read result s in only 1 copy of a chunk being accessed.

Failure Resilience [20]

If it is detected that a slice of a chunk is corrupted, contact other valid replicas of this chunk and perform error correction for the chunk slice. The control flow is through the Controller, but the data flow is between the chunk servers. The contents of one of your chunks will be tampered with. A subsequent read of the file should detect this corruption and initiate a fix of this chunk slice.
Also both the controller and the chunk server failure and unexpected termination should be incorporated.