PROGRAMMING ASSIGNMENT 4

BUILDING A DISTRIBUTED, REPLICATED, AND FAULT TOLERANT FILE SYSTEM

VERSION 1.0

DUE DATE: Wednesday, November 8th, 2017 @ 5:00 pm

OBJECTIVE

The objective of this assignment is to build a distributed, replicated, and failure-resilient file system. This assignment has several sub-items associated with it.

There are 3 programs that you need to develop.

1. Chunk Server responsible for managing file chunks. There will be one instance of the chunk server running on each machine.
2. A controller node for managing information about chunk servers and chunks within the system. There will be only 1 instance of the controller node.
3. Client which is responsible for storing, retrieving, and updating files in the system

Similar to other assignments, this must be implemented in Java and there are steep deductions for using 3rd party libraries.

Grading: This assignment will account for **10 points** towards your cumulative course grade. There are several components to this assignment, and the points breakdown is listed in the remainder of the text. This assignment is meant to be done individually. The scoring process will involve a one-to-one interview session of approximately 20 minutes where you will demonstrate all the required functionality based on the inputs that will be provided to you. The slots for these interview sessions will be posted a few days prior to the submission deadline.

The lowest score that you can get for this assignment is 0: deductions will not result in a negative score.

Fragment and Distribute

In this file system, portions (or ***chunks***) of a file are dispersed on a set of available machines. There are multiple chunk servers in the system: on each machine there can be at most one chunk server that is responsible for managing chunks belonging to different files. A chunk server stores these chunks on its local disk (in most cases, this will be /tmp).

Every file that will be stored in this file system will be split into 64KB chunks. These chunks need to be distributed on a set of available chunk servers. Each 64KB chunk keeps 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. Individual chunks will be stored as regular files on the host file system.

File writes/reads will be done via the chunk servers that hold portions of the file. The chunk server adds integrity information to individual chunks before writing them 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 (the integrity information is not sent).

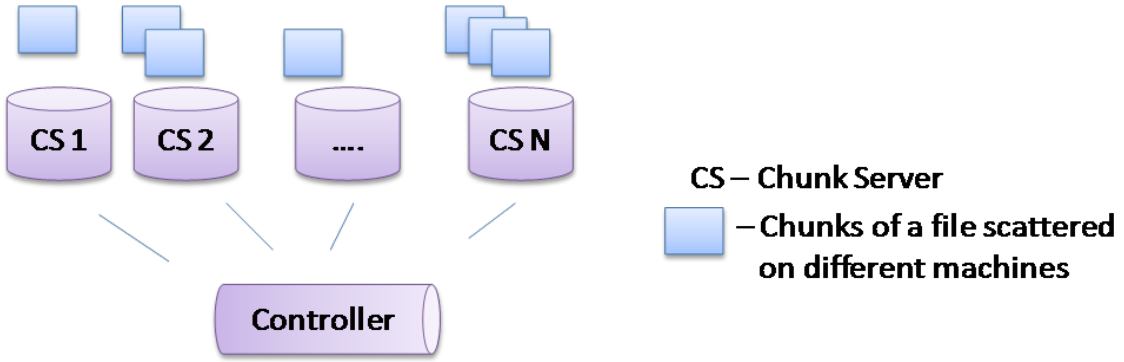

Figure 1: A file will be split into chunks and dispersed on multiple machines

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 associated with the chunk: this additional information should not be encoded in the filename; this includes –

- Versioning Information: Multiple writes to the chunk will increment the version number associated with the chunk.
- Sequencing Information: There will be a sequence number associated with each chunk.
- File name: The file that the chunk is a part of
- Timestamp: The time that it was last updated.

Chunk Server and the Controller Node

Each chunk server will maintain a list of the files that it manages. For each file, the chunk server will maintain information about the chunks that it holds.

There will be one controller node in the system. This node is responsible for tracking information about the chunks held by various chunk servers in the system. It achieves this via heartbeats that are periodically exchanged between the controller and chunk servers. The controller is also responsible for tracking *live* chunk servers in the system. The controller does not store anything on disk, all information about the chunk servers and the chunks that they hold are maintained in memory.

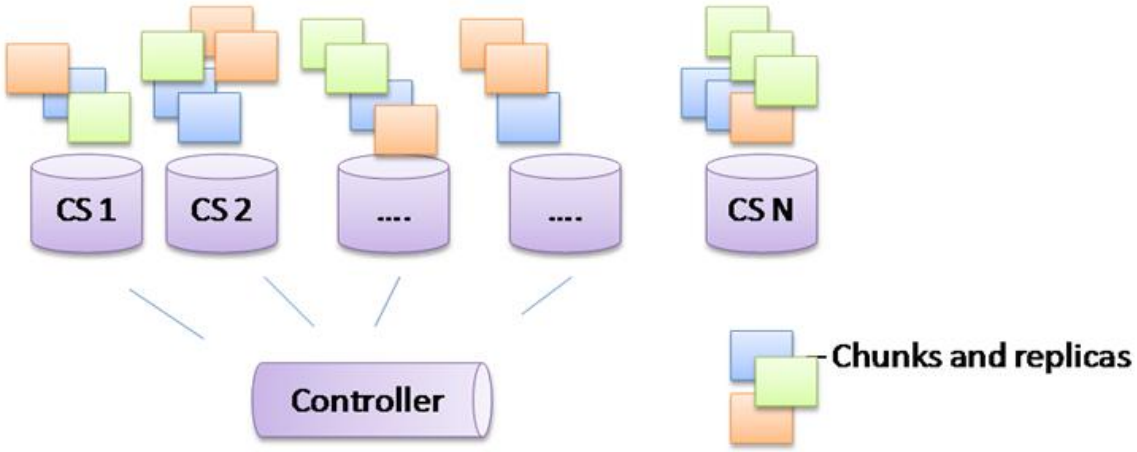

Figure 2: Distribution of chunks of a file and their corresponding replicas

Heartbeats

The Controller Node will run on a preset host/port. A chunk server will regularly send heartbeats to the controller node. These heartbeats will be split into two

1. A major heartbeat every 5 minutes
2. A Minor heartbeat every 30 seconds

At the 5 minute mark ONLY the major heartbeat should be sent out.

The major heartbeat will include metadata information about ALL the chunks maintained at the chunk server. The minor heartbeat will include information about any newly added chunks. Additionally, when a chunk server detects file corruption, it will report this to the Controller Node.

All heartbeats will include information about the total number of chunks and free-space available at the chunk server. Free space information should be one of the metrics used for distribution of chunks on the set of available commodity machines.

The Controller will also send heartbeats to the chunk servers to detect failures.

Replication of files

Each file should have a replication level of 3; this means that every chunk within the file should be replicated at least 3 times. 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. Rather than write to each chunk server directly, if there are 3 chunk servers **A**, **B** and **C** that were returned by the controller, the client will only write to chunk server **A**, which is responsible for forwarding the chunk to **B**, which in turn is responsible for forwarding it **C**. Propagation chunks in this fashion has the advantage of utilizing the bandwidths more efficiently. After the first 64KB chunk of a file 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. *A given chunk server cannot hold more than one replica of a given chunk.*

Chunk data will be sent to the chunk servers and not the controller. The controller is only responsible for pointing the client to the chunk servers: chunk data *should not* flow through the controller.

Disperse a file on a set of available chunks servers (1 point)

You will take a file and ensure the storage of chunks of this file on different chunk servers. Each chunk of the file should be replicated 3 times. This chunk should be available on the local disk (/tmp) of the chunk server.

Deductions

1. If you use the controller to forward chunk data to the chunk servers **(-1 point)**
2. If more than 1 replica of a chunk is stored at the same chunk server **(-1 point)**

Reading a previously stored file (1 point)

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.

Deductions

1. If you use the controller to forward chunk data from the chunk servers **(-1 point)**
2. If more than 1 replica of a chunk is accessed at the same time. A given read should result in only 1 copy of a chunk being accessed. **(-1 point)**

Tampering with chunks (2 points)

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.

Deductions

1. If you use the controller to detect corruptions of a chunk replica (2 points)

Error Correction (2 points)

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.

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. Error detections will be performed outside the heartbeat control message scheme. The control flow is through the Controller, but the data flow is between the chunk servers.

Coping with failures of chunk servers (2 points)

We will terminate one/more of the chunk servers. In response to detection of failures of the chunk servers, the Controller should contact chunk servers that hold legitimate copies of the affected chunks and have them send these chunks to designated chunk servers. Note: The control flow is through the Controller, but the data flow is between the chunk servers.
The metadata maintained at the Controller is updated to reflect this. How are reads handled during this failure?

Coping with the failure of the Controller (2 points)

The Controller process is terminated, and after some time it is restarted. The Controller receives minor/major heartbeats from the chunk servers. In response to receiving a minor update from a chunk server, the (recovering) Controller must contact the chunk server and request a major heartbeat. The controller SHOULD NOT maintain any persistent information (i.e. it should not write to its stable storage) about the locations of the chunk servers.

Third-party libraries and restrictions:

You are allowed to use a 3rd party library for the SHA1 hash function. You can also use networking capabilities provided in the language of your choice. You are not allowed to download *any* other code from *anywhere* on the Internet. You are also not allowed to use RPC or distributed object frameworks to develop this functionality (there is a **8 point deduction** for this). You can discuss the project with your peers at the architectural level, but the project implementation is an individual effort.

Testing Scenario

You will be asked to launch between 10-20 processes possibly on different machines. The port number on which your chunk server runs should be configurable. There will be only 1 chunk server per machine.

Submission deadline:

Please submit the source codes for your project by 5:00 pm on the due date. Please submit a zip file containing your source codes and a Readme.txt using **CANVAS**. We will rely on the honor system: please do not make any modifications to the codebase after the submission deadline has elapsed. There will be steep deductions for making modifications to the source code after you have submitted it.

> *Nota Bene*: Please do not e-mail the source codes to the Professor or the GTA – there will be a **3 point** deduction for doing this.