Mini-Project

This project is focused on understanding, implementing and comparing performance of a distributed storage system using different schemes for data redundancy: replication and erasure coding. Use the tools and ideas from the lectures and labs to predict the expected behaviour and reason about the observed behaviour in your system.

You are expected to work in a team to design and implement a distributed storage system using Python 3 and run it on the provided Raspberry Pi Stack. Every team member should participate in all stages of the project, e.g. specification, system design, implementation, testing and measurements. The storage system has to run on the Raspberry Pi Stack mini datacenter, using all four devices.

Exam: Each team member will take the oral exam individually. You must bring the Pi Stack of your team, briefly present the storage system and demonstrate how it works. You will need to defend your team's design decisions and answer questions about implementation details. You will also need to show the results of measurement tasks and reason about the expected and observed performance.

Measurements

- Measure the following metrics using files of different sizes: 10kB, 100kB, 1MB, 10MB, and 100MB
- Process each file size at least 100 times, remove the outliers (top and bottom 5%) and report the measured times as a histogram, clearly marking the average.
- Metrics to measure:
  - Time from receiving a file to successful generation of redundancy (replicas or coded fragments).
  - Processing time for encoding/decoding in the case of erasure coding.
  - Access time measured at the client, from requesting a file to receiving the last byte.

Report: The report for the mini-project should be of no more than 10 pages [5 to 10 good pages are expected - more is not always better].

Exercise 1: Interface to the Outside World ....

Your storage system should provide an interface to external clients. It can be either a standard REST API over HTTP that you can test with Postman or a simple web application, or a ZeroMQ socket API that is called by a client application. You are encouraged to use the techniques (and code) from the Labs.

There are only two mandatory functions your storage system has to support: store and retrieve a file.

Pointers:

- Keep the API and system simple to start with
- Consider using Flask for a REST API
- Consider using Protocol Buffers for a ZeroMQ-based interface
- Consider having a lead node that provides the API and coordinates the process

Exercise 2: Replication ....

(a) (Design & Development) Develop a strategy that can generate k full replicas of a file and places them in different nodes (Raspberry Pi devices), with a general k, from the lead node, i.e., the lead sends file directly to each of the other nodes. Selection of the nodes is random.

(b) (Design & Development) Develop a strategy that can generate k full replicas of a file and places them in different nodes (Raspberry Pi devices), with a general k, also using random placement strategy but where the lead node delegates to other nodes the generation of the next replica (similar to what Hadoop HDFS does).

(c) (Analysis) Calculate the time to generate all replicas of both schemes if the connection from one node to another is R bps and a maximum outgoing/incoming data rate of R bps.

(d) (Analysis) Calculate the time for the lead node (the one originally receiving the file) from finishing its replication task in both strategies. When is the lead node freed earlier to take in another request?

(e) (Analysis) Calculate the time accessing the file when requested?

(f) (Measurements) Compare the two approaches when ingesting different file sizes and compare to the analysis. Consider the cases of $k = 2, 3, 4$ and the two time metrics: time to generate full replicas, time for the lead node to complete its tasks, and access time. Explain the results and discuss the advantages of each approach.

Exercise 3: Erasure Coded Storage ....

(a) (Design & Development) Develop a coded strategy that tolerates $l = 1, 2$ node losses in order to later compare to the case of $k = 2, 3$ replicas above, respectively. Consider that the lead node performs the encoding/decoding of the data, for simplicity. Suggestion: consider splitting the file in two equal-sized fragments in both cases

(b) (Design & Development) Consider that the case where the lead node randomly selects another node to carry out the encoding/decoding of the file.

(c) (Analysis) Calculate the time to generate redundancy of both schemes, if the connection from one node to another is R bps and a maximum outgoing/incoming data rate of R bps. Consider also that encoding/decoding can be carried out at R_{enc} and R_{dec} bps, respectively. Compare the case of $l = 1$ and $l = 2$ and meditate on the behaviour when more than 4 nodes are available.

(d) (Analysis) Calculate the time for the lead node (the one originally receiving the file) from finishing its task in both strategies. When is the lead node freed earlier to take in another request?

(e) (Analysis) Calculate the time accessing the file when requested?

(f) (Measurements) Compare the two coded approaches when ingesting different file sizes and compare to the analysis. Consider the cases of $l = 1, 2$ and the two time metrics: time to generate full replicas, time for the lead node to complete its tasks, encoding/decoding time, and overall access time at time of retrieval of data. Explain the results and discuss the advantages of each approach.

Exercise 4: Cross Comparison ....

(a) Compare the measurements $l = 1$ of the coded case to the case of $k = 2$ replicas without coding and explain the results. Consider the different variations of each coded scheme and reason about the implications in larger systems.

(b) Compare the measurements $l = 2$ of the coded case to the case of $k = 3$ replicas without coding and explain the results. Consider the different variations of each coded scheme and reason about the implications in larger systems.