Distributed Systems / Fall 2022
21.11.2022
Tapio Salonen, Johannes Lares, Heikki Nenonen

Design Plan: Distributed file service running on Kubernetes cluster on CSC server

Our plan is to set up a distributed file service. The service is run on CSC (Finnish IT center for science) servers. The user interface will work through a website made using React and Node.js. The list of current files is shown on the website and the user has the ability to download one of them or to upload a new file using TCP protocol. The initial setup will be one master node and two worker nodes under one Kubernetes cluster. The master makes all the decisions and thus we do not have consensus between the nodes. Naming and node discovery is handled by the Kubernetes. We do not have to hard code the IPs into our code. The node synchronization and consistency with each other is done using a message queue, RabbitMQ. The files are stored on only one node initially and the nodes communicate and load the file in the database. Each node has a filelist of the existing files in the systems and their locations, which is shared and updated through messages between nodes. The file storage itself is implemented using PostgreSQL.

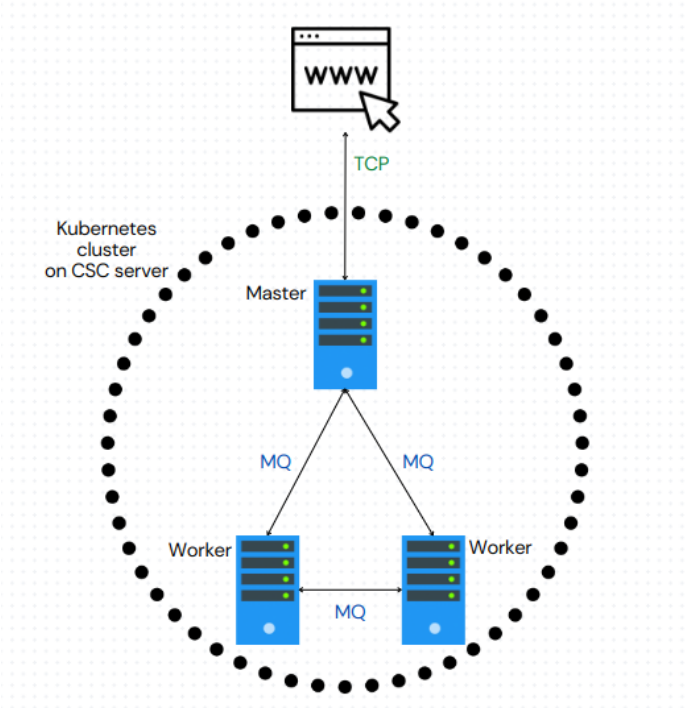

Figure 1. Structure of our distributed file system.

Our system setup follows the structure presented above in Figure 1. Master node works as a load balancer and runs the website. When a user wants to upload a file, the master node first decides which worker node it is stored on. Master node does not store any files. The file is then uploaded using TCP protocol to the worker node and after a successful upload (into Postgre) its filelist is updated. The updated file list is then shared between other nodes using a message

queue (RabbitMQ). The filelist contains the following information: filename, file size, nodes it is located on. Other nodes after updating the file list know that they have a new file to load and its location. The file transfer between nodes is executed through TCP. The updated filelist is then presented through the website.
The user can now download a file from the system by selecting the wanted file from the filelist presented on the website. The request is sent through TCP to the master node running the UI. The master node checks from its filelist where the file is located and sends the download request through a message queue to the node having the file. If multiple nodes have the file, it decides the worker to serve the client based on traffic etc. The worker node then sends the file using TCP.

The fault-tolerance is weak in the initial version, if the master goes down the whole service goes. If a worker node goes down all files on it become unreachable, but the rest of the files on other nodes continue working normally. This is why we will implement file sharing between nodes to increase fault-tolerance via replication. The filename is saved on filelist only after successful upload, which makes sure only valid files are available for download.

Our system provides good scalability for adding more worker nodes. This allows the increase of processing power, networking capacity and storage space in an online system. Our initial setup does not support the addition of new masters.

If after the initial system setup and we still have time left, we will add a third worker node and implement file replication so that each file is stored on two nodes at every given time. This improves scalability, because then not every node has every file.