Report

Solution Design

The given solution consists of the implementation of the client file. Given the already existing server, only 2 message types were used - *filemsg* and *datamsg*.

For single messages, *datamsg* has been used. Due to the already existing implementation of the server, the functionality of fetching the first 1000 entries of each file has been implemented using this message type too. In reality, 2000 messages are exchanged between the client and the server in that case.

For the file transfer, the buffer length either defaults to 256 or uses a custom value. The client sends *filemsg* messages while the server responds with sections of the file. This is highly performant, depending greatly on the size of the given buffer.

As the server is initialized in the background by the client, the used buffer length to the client is passed as an argument to the server, for convenience.

Performance Analysis

To analyze the performance of the given solution, some benchmarks have been performed, which give some insight into the approach.

For exchanging a single data point, the performance is negligible, with the operation occurring very fast for any entry of any file - averaging 260ms on my machine. Most of the time is actually spent initializing the server.

However, such is not the case for fetching the first 1000 entries, with an average duration of 6.8s. Although the transferred data is not that large ((20 bytes + 8 bytes) * 2000 = 56KB) it takes a long time to transfer. This is due to the synchronization mechanism used, where only 1 message is sent at a time, together with the small size of the message, which incurs low throughput.

Transferring files is, on the other hand, very efficient and fast, given an acceptable size for the underlying data buffer. The larger, the better. For correctness, reads from the named pipe were done repeatedly until the entire data had been transferred, given that the local machine was limited to 8KB.

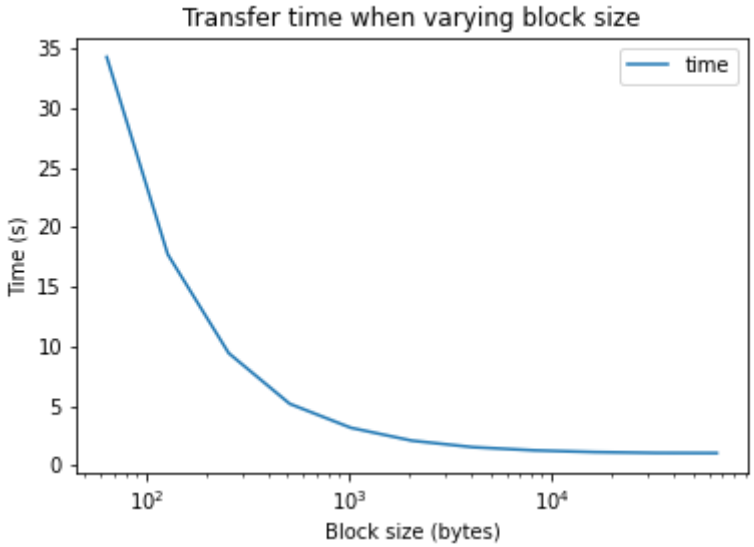

Fig 1. Transfer time according to the buffer block size. A file of size 25MB was used.

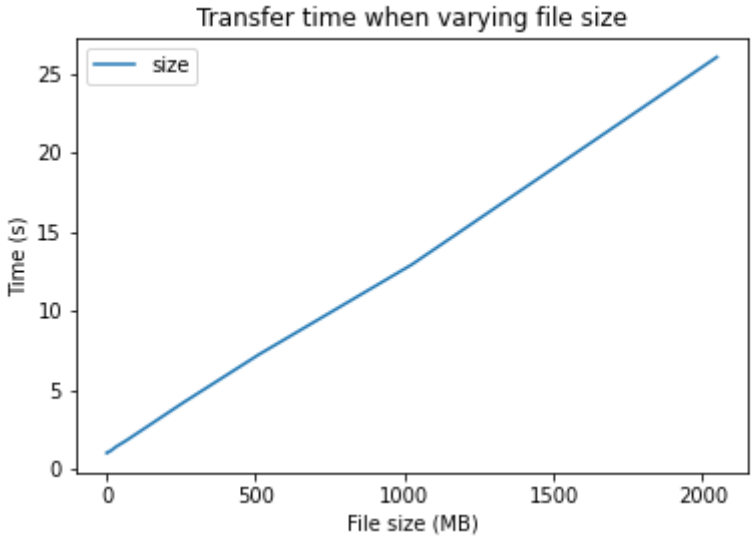

Fig 2. Transfer time according to the size of a file. A block of size 8KB was used.

As visible in Fig 1, a very small buffer size incurs in a large overhead - similarly to what happens on the 1000 entries fetch situation. Many small messages are transferred, and that together with the synchronization between the client and the host causes a lot of wasted performance. For larger values, the performance eventually flatlines, and no benefits are seen from increasing the block size.

In terms of throughput when varying the block size, it is easy to see that there is a linear relationship between the file size and the time it takes to transfer the file, achieving around 80MB/s on the testing machine. There is still room for improvement, given that the I/O is not entirely saturated.