Bryan Martin

Xin Xu

Nicholas Castorina

**CSE 451 Project 3 Write Up**

(Note: graphs are attached at the end of the report.)

For both the synchronous and asynchronous case we divided the files up into chunks and processed the chunks sequentially, giving each thread a shared list to pull from. We realized that this single list may not have been the best decision after doing our performance analysis.

**Synchronous Case:**

With large files we found that more threads led to worse performance. This is probably because all the threads are trying to write or read from the same file at the same time and the file system needs to serialize some of those accesses. More threads also leads to more overhead switching between them. What would have been better is to give each thread their own chunk list and try to spread the files out between them. It may have also been slowed down with more threads because more reads and writes were being interrupted before all of the desired number of bytes were read or written, leading to an increased number of calls.

With a bunch of smaller files we can see some performance improvement with having two threads rather than one, especially with larger buffer sizes. After about five threads the performance decreases as with large files. This is probably because if the buffer can hold all or most of a file, the threads will be spread out between the files.

Having a larger buffer size almost always increases the throughput of the copy, independent of the number of threads used. Since the threads run independently, doing more work at one time is better.

When writing to or from a network drive, the throughput is much slower that over a local drive. But the throughput is proportional to a local drive given the number of threads and buffer size.

**Asynchronous Case:**

The number of buffers does not have much of an effect on the throughput. This is probably because the reads and writes are pretty much sequential anyways so increasing the number of sequential things will not affect it much. Also since it does not use threads, there is no overhead for context switching. However, increasing the number of buffers does decrease the throughput slightly, possibly due to having more read and write jobs sitting around waiting to do something.

When writing to and from the local drive, if the buffer size increases when copying large files, the throughput is lower than when using a smaller buffer. This is not seen when copying small files or any files over the network. Over the network it is better to do one big job rather than many smaller jobs because the latency is the limiting factor.

**Comparing the Two:**

Our asynchronous case generally performed better, with bigger buffer sizes and when copying over the network. Over the network this may have something to do with the way overlapped reading and writing is implemented.

The asynchronous case was less affected by the number of threads or buffers. This is probably due in part to the overhead of using threads and how the two different versions of read and write are implemented.

**In General:**

Reading from a network drive is faster than writing to it. This is probably because waiting for the round trip of executing a write then receiving a completion message is slower than just reading data received. Also network drive can send more data than you requested so subsequent reads can execute much faster due to locality.

**Graphs:**