Michael Free
19 October 2013
Lab 3 – Download Accelerator Report

Downloading in Parallel

The Project

For lab 3, I created a downloadAccelerator.py python program that would download a given url and store it in the current directory. As a user, one could input how many threads that my downloadAccelerator would use. If more than 1 thread was to be used, my program would split up the bytes to read and do so in parallel. Then, the program would output to the terminal the [url] [threads] [bytes] [seconds].

The Experiments

To find out the significance of parallel downloading, I ran a series of experiments. These were provided in the experiments.py file from Professor Zappala. They were done over a Wi-Fi connection once, and then through an Ethernet connection using BYU's internet.

The python script would download a file 10 times using a certain amount of threads. It would then download the same file 10 times using another amount of threads. We tested 1, 2, 3, 5, and 10 threads.

It was important to run the test 10 times per amount of threads, just in case of error. And it was important to test this on differing Internet connection speeds, as that could give us more insight into the importance of parallel downloading. The files could be divided into 3 categories: small, medium, and large.

Small

The small file was a pdf from a website: http://ilab.cs.byu.edu/zappala/files/design-philosophy-sigcomm-88.pdf. It consisted of 1,180,016 bytes. As can clearly be seen in Figure 1 and Figure 2, there was a significant change in download speed going from 1 thread to 2. This happened for both the Wi-Fi and Ethernet connections. Interestingly, it took a little longer with 3 threads than 2 when downloading over a Wi-Fi connection; however, when done through the Ethernet, each incrementation of the number of threads helped speed up the download.

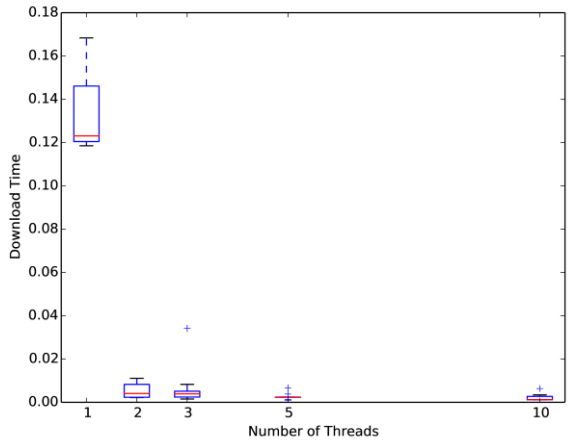

Figure 1 – Small file Ethernet Results

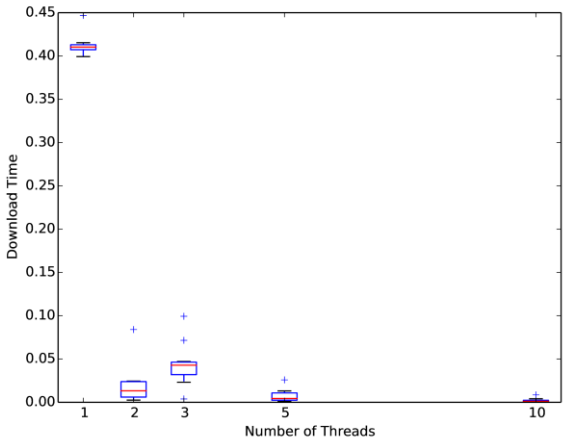

Figure 2 – Small file Wi-Fi Results

Medium

The medium file was an mp3 from a website: http://ilab.cs.byu.edu/zappala/files/Delta-Rae-Morning-Comes-Live.mp3. It consisted of 10,785,918 bytes. Both the Wi-Fi and Ethernet showed great improvement of download speed moving from 1 thread to 2. As seen in Figure 3, it nearly drops by ¼ the download time. With more and more threads, the download time became slightly quicker and quicker.

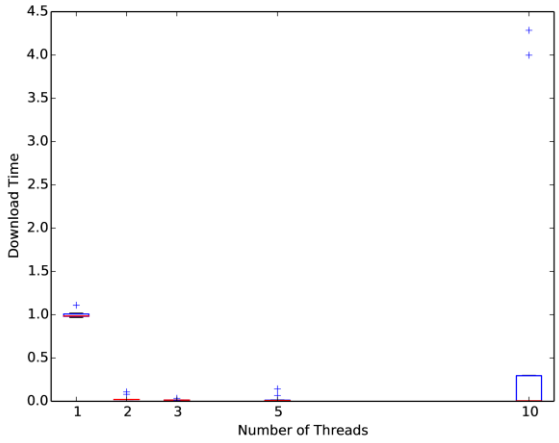

Figure 3 – Medium file Ethernet Results

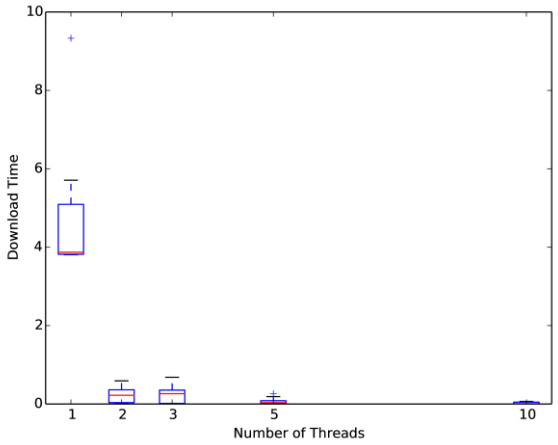

Figure 4 – Medium file Wi-Fi Results

Large

The large file was another pdf from a website: http://ilab.cs.byu.edu/zappala/files/poster.pdf. It consisted of 95,692,170 bytes, nearly 100 MB. As with the previous sizes, the download speed drastically improved when switching from 1 thread to 2. Both Figures 5 and 6 show that the download speed of 2 threads is nearly 1/7th the time it takes to download the file using 1 thread. An interesting find for me was that it made another significant improvement when going from 2 to 3 threads over Wi-Fi. It cut the time by 1/5th. However, there was no significant jump from 2 to 3 threads on the Ethernet side.

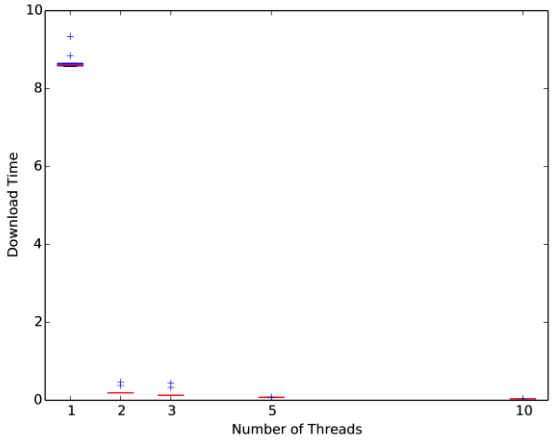

Figure 5 – Large file Ethernet Results

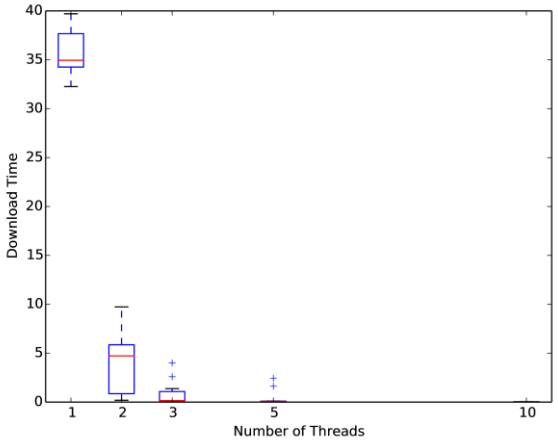

Figure 6 – Large file Wi-Fi Results

Conclusions – benefits and trade offs

It doesn't take too long to come to the conclusion that download speed increases with more threads. So, a clear benefit from parallel downloading would be the speed of the download. One would think, "Why would I ever decide not to download a file with multiple threads?"

Although speed is very desirable, reliability is another thing to consider. The more threads you have, the more chances there are for errors to occur when connecting to or downloading from a server. If even one of your 10 threads gets an error, the whole file is corrupted because it is missing some information. Therefore, the programmer should weigh the advantages of speed with the disadvantage of the risk with multiple threads.

After conducting this experiment, I would conclude that a good amount of threads to use would be at least 2. There was such a significant difference in each case (small, medium, and large) that 2 should definitely be the lowest amount to choose from. I think that the benefits outweigh the risks.

If one considers doing more than 2 threads, they should also consider the size of the file to be downloaded, which thankfully can be discovered in the headers – before having to download the file to find out how big it was. Perhaps some situations where you would want more than 2 threads would be for larger sized files. As was mentioned above, there was a significant difference between the time to download a file with 3 threads as opposed to 2 with the large file. It cut the time by 1/5th, which is quite significant.

In conclusion, I would say that parallel downloading is very desirable. It can allow one to download files much quicker than 1 thread can do. However, one should also be cautious of how many threads they use. With more threads, the download speed may be slightly faster, but does that outweigh the risks that are associated with having multiple threads? These are all questions and thoughts a programmer should think about when writing code to download a file.