CS553 – Cloud Computing

Programming Assignment 1

Performance Evaluation

Karan Jeet Singh; Kanwal Preet Singh; Anushka Dhar
9-23-2014

Performance Evaluation

1 Objective

This document compares the benchmarking results obtained for the below mentioned system, testing its CPU, GPU, Memory, Disk, and Network. Variable parameters have been used to test the system as effectively as possible.

2 System Specifications

The benchmarking programs were run on a local virtual machine created VM Player running Ubuntu 14.0.1, with virtual machine having 4GB ram, 2 cores, and 20 GB hard disk space.

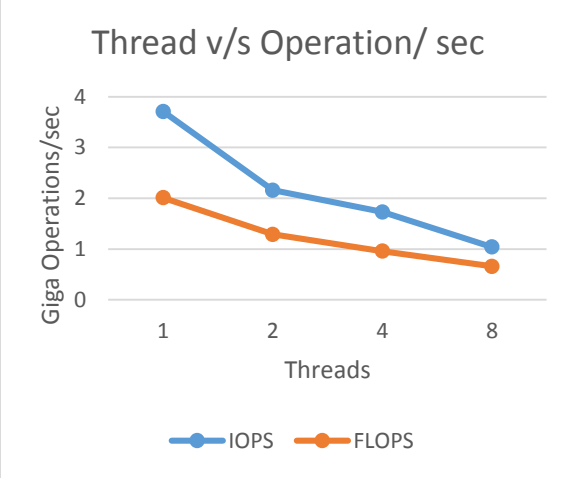
The host system host following configuration: i5 2nd generation CPU @ 2.5GHz, 8GB DDR3 RAM @ 1333MHz, and an NVidia GT630M 2GB GPU.

3 CPU

The CPU benchmarking is done in respect to calculating integral operations, and float point operations done by CPU per second. We are feeding a certain number of operations to the CPU limited number of times, then the time taken to do so is measured and is used to calculate the IOPS and FLOPS of the system. The results are obtained by varying the number of threads the program runs on. The trend of doing so is measured and displayed via the graph. We are running the test on 1, 2, 4, and 8 threads to get more accurate results and also see what configuration works best on the current system. The **theoretical** performance was recorded at 20 GFLOPS, of which we could achieve only 2GFLOPS. **LINPACK** gave the peak performance rating of 17.49 GFLOPS. The results of our program vary so much because of lack of assembly level code at our end and also we didn't run complex computations which usually give a much better result.

3.1 Effect of Threads on the operations done by CPU

We are running the CPU benchmark code for varied number of threads ranging from 1, 2, 4, till 8, and compare the results of same. We can see that the count of FLOPS and IOPS almost jumps to half when the threads are increased from 1 to 2, which is because the system running the test is configured to have a single core processor, which means 2 threads cannot run concurrently resulting in doing double work in same amount of time, which basically means that the performance decreases by almost a factor of 2. After that we increase the thread count to 4 and 8, we can see that the result pretty much stagnates after 2 threads. This is because the number of cores provided to the system is 1, so running any more number of threads would cause system to keep the next thread idle unless the previous ones are done with their execution. We created the below graph from the results of three consecutive runs, the **standard deviation** we achieved was not more than 0.12Giga Operations per second



4 GPU

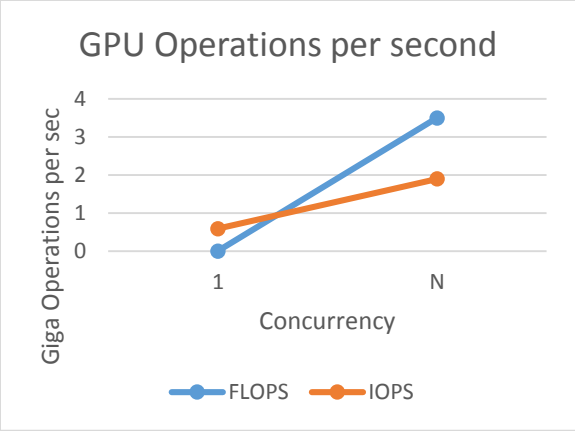
The GPU benchmarking is a two part test, firstly we are trying to calculate the float point operations, as well as integral point operations done within a second, and then the second test is to check the throughput and latency of GPU memory. The FLOPS and IOPS for GPU are calculated by running a simple operation through all the cores of GPU and then dividing it by the time taken to do so. The number of threads per core are not varied, but rather, no. of cores used are varied.

The second experiment runs the memcpy command on GPU memory and copies the data from one stream to another depending upon the block size specified. The block sizes being provided are 1B, 1KB, and 1MB.

We ran all of the tests three times and made the graphs using average of them, the standard deviation achieved have been given just above the graphs.

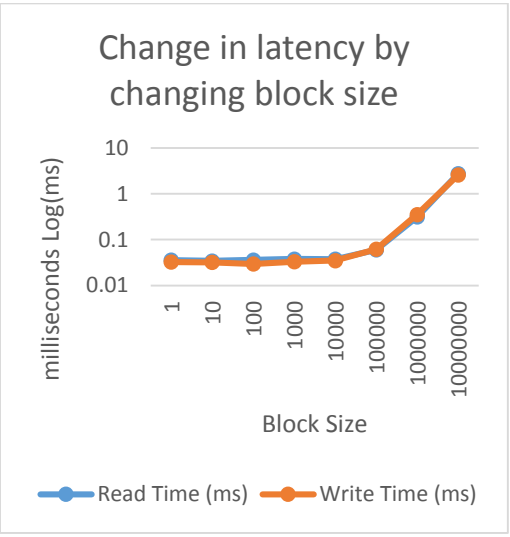
4.1 Effect of Cores on performance

GPU is a SIMD device which makes it optimal for processing large amount of data with the same instruction .The same can be inferred from the graphs which clearly depicts that as the no. of threads are increased the performance improves. The performance is proportional to the amount of parallelism of the system under observance.



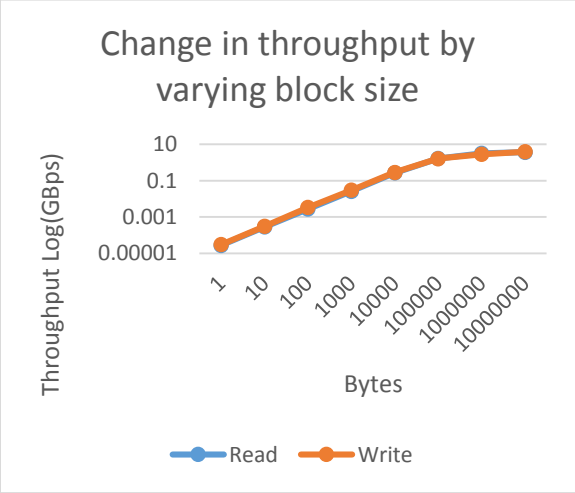
4.2 Change in Latency by Changing Block size

The latency for reading and writing data sequentially and randomly in memory varies in direct correlation with the size of block of data that is used. The latency increases as the block size does, which means that larger the block size, more time it takes to read or write it. When building a real world system, we need to take in effect this balance of throughput and latency achieved by varying the block size. In the latency specific environments where size of data is small but the speed is of the essence, the latency for same can be taken into account. We achieved a standard deviation of not more than 0.9 ms in this test.



4.3 Change in Throughput on changing Block Size

Throughput for reading and writing into the memory can be seen in direct correlation with the size of block which is written or read. Larger the block size, the better throughput we get. Even though this holds true for most of the block sizes, but we can see via the graph below that the performance stagnates after a certain size. The optimum performance is obtained at 100KB block size. The standard deviation achieved was not more than 0.1 GBps.



5 Memory

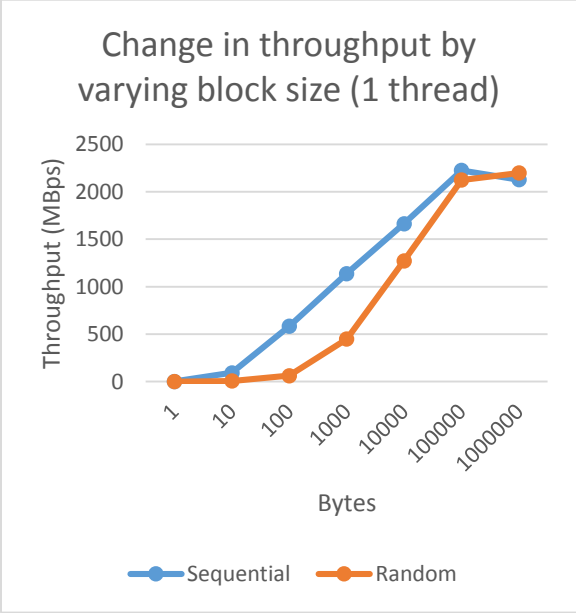
The memory benchmarking aims at calculating the latency and throughput of writing and reading data of variable size to the memory, either sequentially or randomly. This is achieved by running a read+write operation on the memory multiple times by varying the size of block read and written, the dexterity of accessing data, i.e. sequentially or randomly, and also by varying the threads used to perform the action. The block size written and read from the memory is varied from 1B to 1MB, and the number of threads are varied from 1, 2, 4, till 8.

We ran all of the tests three times and made the graphs using average of them, the standard deviation achieved have been given just above the graphs.

The **theoretical** peak performance calculated is 85GBps, and the **stream benchmarking tool** revealed a peak performance of 30GBps. Our tool was able to reach a peak performance of 23GBps.

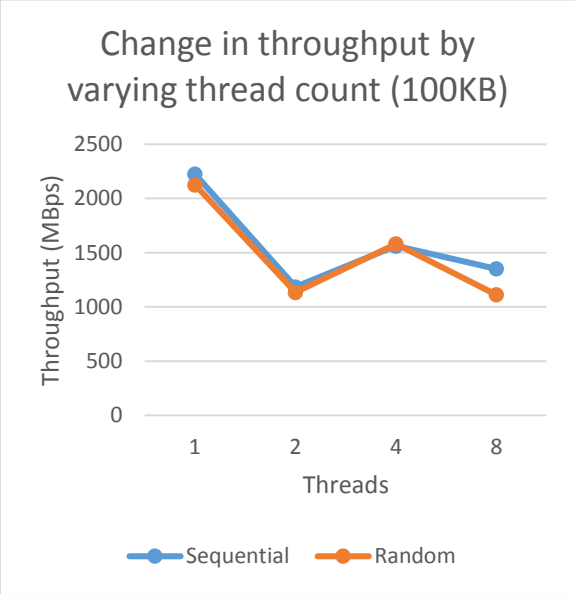
5.1 Change in Throughput by varying Block Size

Throughput for reading and writing into the memory can be seen in direct correlation with the size of block which is written or read. Larger the block size, the better throughput we get. Even though this holds true for most of the block sizes, but we can see via the graph below that the performance stagnates after a certain size. The optimum performance is obtained at 100KB block size. The standard deviation achieved was not more than 10.1 MBps.



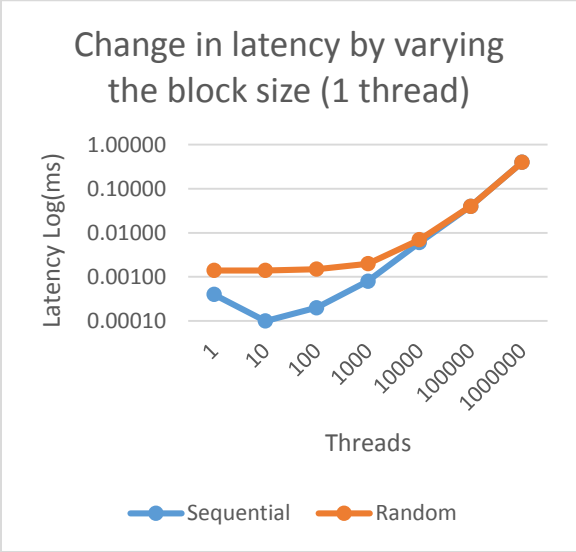
5.2 Change in Throughput by Changing the number of Threads (100 KB)

Throughput varies by the change in number of threads as well. Since, the machine in use was a single core, we could see that the performance actually dropped when the thread count was increased from 1 to 2 and so on. The throughput of memory didn't vary a lot even when the method of accessing data was changed from sequential to random. The output was equally quick in both the cases. The standard deviation achieved was not more than 9.1 MBps.



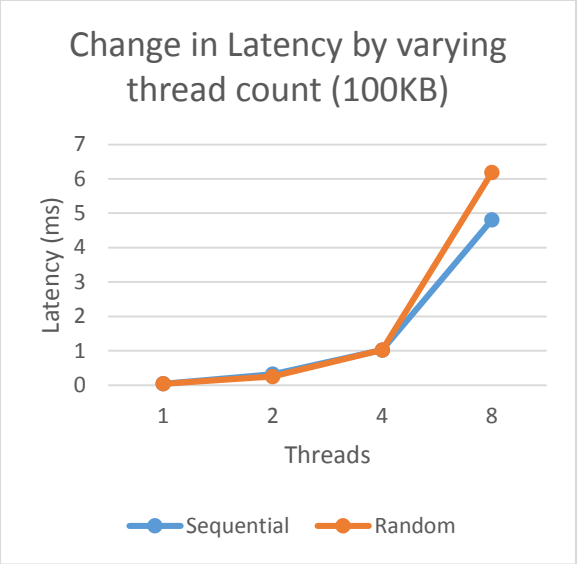
5.3 Change in Latency by Varying Block Size

The latency for reading and writing data sequentially and randomly in memory varies in direct correlation with the size of block of data that is used. The latency increases as the block size does, which means that larger the block size, more time it takes to read or write it. When building a real world system, we need to take in effect this balance of throughput and latency achieved by varying the block size. In the latency specific environments where size of data is small but the speed is of the essence, the latency for same can be taken into account. We achieved a standard deviation of not more than 0.319 ms in this test.



5.4 Change in Latency by varying the number of Threads (10KB)

The latency of the system in use gave optimal results with when only one thread was used, this is because our system was a single core machine. As we increased the threads from 1 to 8, we could see the latency shoot up drastically. This further showed as that peak performance of our system could be achieved only at lower number of threads, and on increased number of threads, the system could not cope well. We achieved a standard deviation of not more than 0.231 ms in this test.



6 DISK

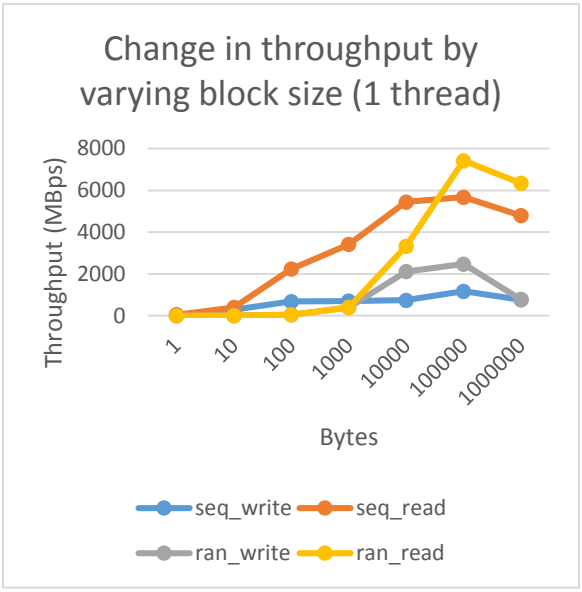
The disk benchmarking aims at calculating latency and throughput of the disk by writing and reading files to the disk. The program writes and then reads number of different files depending upon the buffer size that is used to read and write the file, the number of threads the program runs on, and the dexterity of data being written and read, on whether it is done randomly, or sequentially. The buffer size is varied from 1B till 1MB, and number of threads are varied from 1, 2, 4, till 8.

We ran all of the tests three times and made the graphs using average of them, the standard deviation achieved have been given just above the graphs.

The **theoretical** peak performance is 6GBps and the peak performance revealed by **iozone** tools was 0.6 GBps for writing and 5.8 GBps for reading. Our tool was able to record

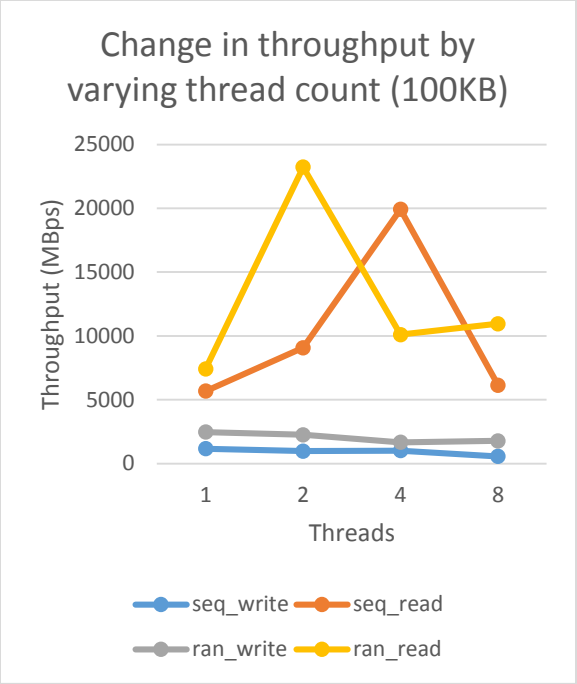
6.1 CHANGE IN THROUGHPUT BY VARYING BLOCK SIZE

Throughput for reading and writing into the disk can be seen in direct correlation with the size of block which is written or read. Larger the block size, the better throughput we get. Even though this holds true for most of the block sizes, but we can see via the graph below that the performance stagnates after a certain size. The optimum performance is obtained at 100KB block size. The standard deviation achieved was not more than 63 MBps for read and 12MBps for write.



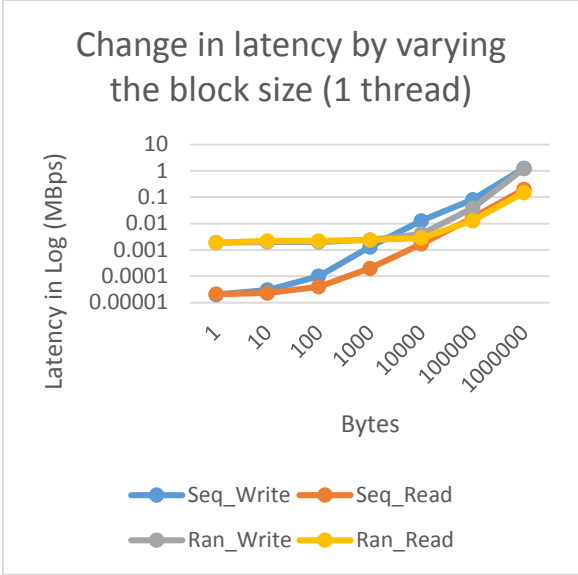
6.2 CHANGE IN THROUGHPUT BY VARYING THE NUMBER OF THREADS

Throughput varies by the change in number of threads as well. The write throughput didn't vary much with change in number of threads for both sequential and random methods of writing, but we could see drastic rise and fall in the read methods. The system could give a peak performance at 2 and 4 thread count for random and sequential methods respectively. This is because a single file was accessed by multiple number of threads and multiple switching threads could traverse the file more quickly than one thread traversing along. The standard deviation achieved was not more than 189 MBps for read and 32.2 MBps for writing



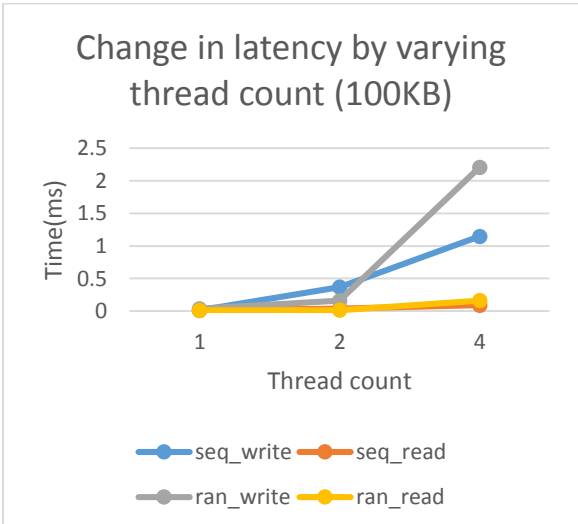
6.3 Change in Latency on varying the block size

The latency for reading and writing data sequentially and randomly in disk varies in direct correlation with the size of block of data that is used. The latency increases as the block size does, which means that larger the block size, more time it takes to read or write it. When building a real world system, we need to take in effect this balance of throughput and latency achieved by varying the block size. In the latency specific environments where size of data is small but the speed is of the essence, the latency for same can be taken into account. We achieved a standard deviation of not more than 3.2 ms in this test.



6.4 Change in latency on changing the number of threads

The latency of the system in use gave optimal results with when only one thread was used, this is because our system was a single core machine. As we increased the threads from 1 to 8, we could see the latency shoot up drastically. This further showed as that peak performance of our system could be achieved only at lower number of threads, and on increased number of threads, the system could not cope well. We achieved a standard deviation of not more than 0.419 ms in this test.



7 NETWORK

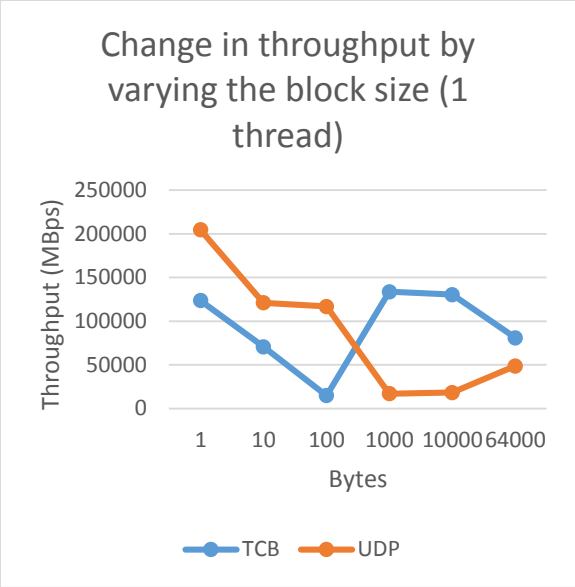
Network benchmarking is done by sending a block of data through a call back loop and then noting the time taken to do so. The data packet can be sent via both TCP, as well as UDP streams. We can also vary the block size sent via the network, as well as the threads used to do so. We are varying the block size from 1B to 64KB, and varying the thread count from 1, 2, 4, till 8.

We ran all of the tests three times and made the graphs using average of them, the standard deviation achieved have been given just above the graphs.

The theoretical peak performance of callback is the same as that of ram, which in our case was 85 GBps. The performance noted by **iperf** was 36 GBps for TCP and 42.5 GBps for UDP. Our tool was able to capture 15 GBps for TCP and 21 GBps for UDP.

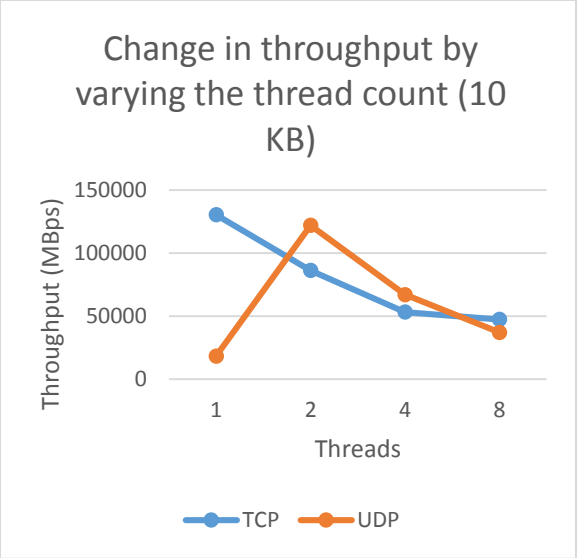
7.1 CHANGE IN THROUGHPUT BY VARYING THE BLOCK SIZE

The trend of throughput depends a lot upon the protocol being used, the performance of TCP increases when the block size is increased from 100B to 1KB, and then the performance doesn't vary a lot when the block size is increased further. In case of UDP, the peak performance is achieved at 1B block size, after that the performance drops continuously till 1KB is reached. This tells us that the protocol of sending data is very important when designing a system. Standard deviation was at around 0.5GBps for both TCP and UDP.



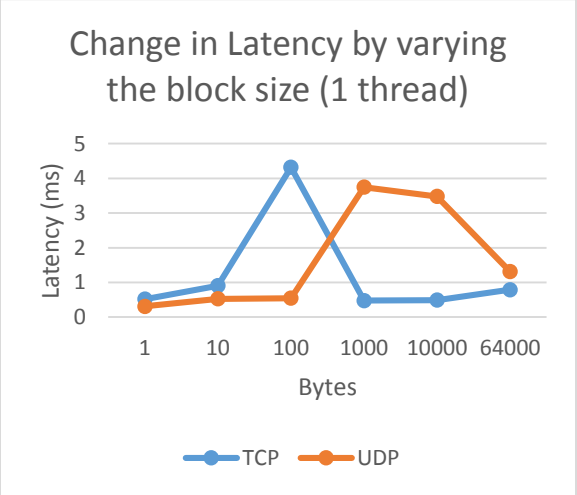
7.2 CHANGE IN THROUGHPUT BY VARYING THE THREAD COUNT

Again we see that the performance of system varies a lot by changing the protocol used and the number of threads on which data is sent. The TCP revealed a steady performance, peaking at 1 thread and then slowly deteriorating when the thread count is increased gradually, but in case of UDP, the difference between the performance achieved at 1 thread is immense when 2 threads are used, after which it drops when the count is increased. We achieved a standard deviation of 0.21 GBps in this test.



7.3 Change in Latency by varying the block size

The trend of latency depends a lot upon the protocol being used, the performance of TCP increases when the block size is increased from 100B to 1KB, and then the performance doesn't vary a lot when the block size is increased further. In case of UDP, the peak performance is achieved at 1B to 100B block size, after that the performance drops suddenly when 1KB is reached. This tells us that the protocol of sending data is very important when designing a system. We achieved a standard deviation of not more than 0.23 ms in this test.



7.4 Change in Latency by varying the thread count

Again we see that the performance of system varies a lot by changing the protocol used and the number of threads on which data is sent. The TCP revealed a steady performance, peaking at 1 thread and then slowly deteriorating when the thread count is increased gradually, but in case of UDP, the difference between the performance achieved at 1 thread actually improves when 2 threads are used, after which it drops when the count is increased. We achieved a standard deviation of not more than 0.31 ms in this test.