ECE568 HW4: Scalability Report

Xiaoyu Wang, Yitao Shi

Implementation Approach:

In this project, we utilized the Boost.Asio library in C++ to implement an asynchronous thread pool. First, we created a thread pool using Boost.Thread and create worker threads that run the io_service's run() method. After that, we defined a function that handles incoming connections using a boost::asio::ip::tcp::acceptor object. This function created a new boost::asio::ip::tcp::socket object for each incoming connection and start an asynchronous read operation on the socket.

With the thread pool, we can easily distribute and manage threads for each incoming client without creating new ones like what we did in Homework 2. This approach can significantly enhance the overall performance of our server.

Scalability Test:

To test the scalability of our server, we implemented a Python script that can automatically send randomly generated requests in a certain amount (including both create and transaction requests) to the server and receive the responses sent back from the server. The script will also record the total run time to finish the send-receive process, with which we can calculate both the latency and throughput to evaluate the performance. The latency is defined as the time required for a request to execute, and the throughput is the number of requests that can be passed in a given time period.

In the scalability experiment, the script was set to send 500 requests in each run, and we tested it for 10 times on two kinds of VMs, where VM1 has 4 cores and 8GB RAM, and VM2 has 2 cores and 8GB RAM. With the statistics obtained in the experiment, we can draw some comparison plots to visualize the performance of both VMs.

Experiment Result:

Total number of requests in each experiment is 500.
Avg latency of VM1 is: 125.345 seconds
Avg throughput of VM1 is: 3.99 requests/second
Avg latency of VM2 is: 239.654 seconds
Avg throughput of VM2 is: 2.08 requests/second

	# of Cores	**RAM**
VM#1	4	8GB
VM#2	2	8GB

Table 1. System Information of Two VMs

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
VM1	123.56	124.66	121.39	117.68	129.80	126.32	132.17	128.69	120.44	128.74
VM2	240.75	245.67	249.35	227.98	232.72	238.12	243.19	239.02	243.01	238.72

Table 2. Total Latency (s) of Two VMs in Ten Experiments

	1	2	3	4	5	6	7	8	9	10
VM1	4.05	4.01	4.12	4.25	3.85	3.96	3.78	3.89	4.15	3.88
VM2	2.08	2.04	2.02	2.19	2.15	2.10	2.06	2.09	2.06	2.09

Table 3. Throughput (#/s) of Two VMs in Ten Experiments

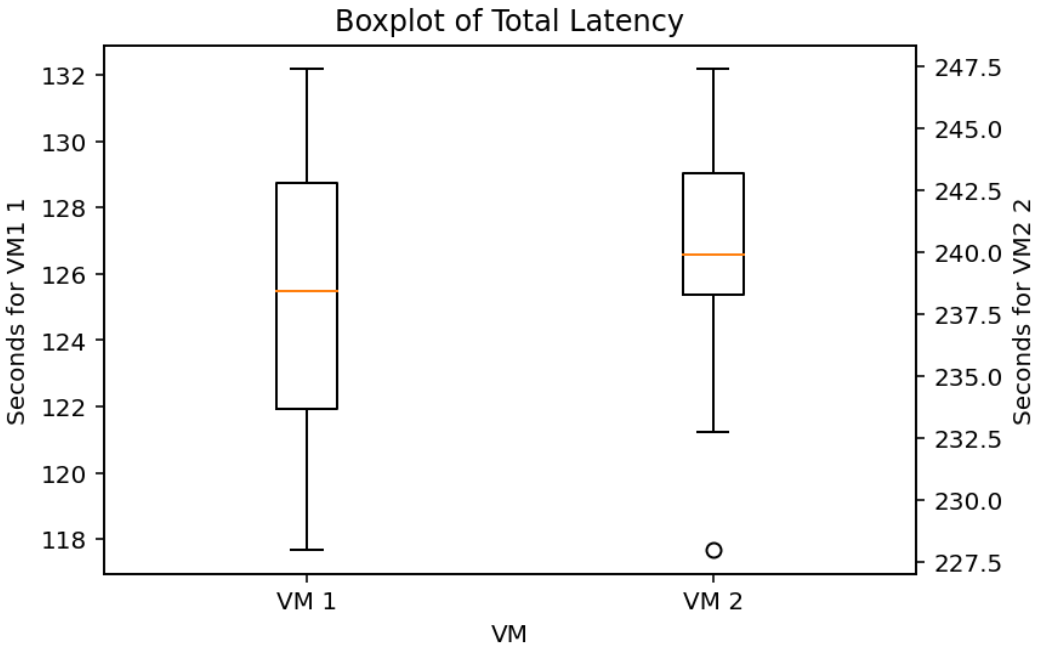

Fig 1. Boxplot of Total Latency of Two VMs

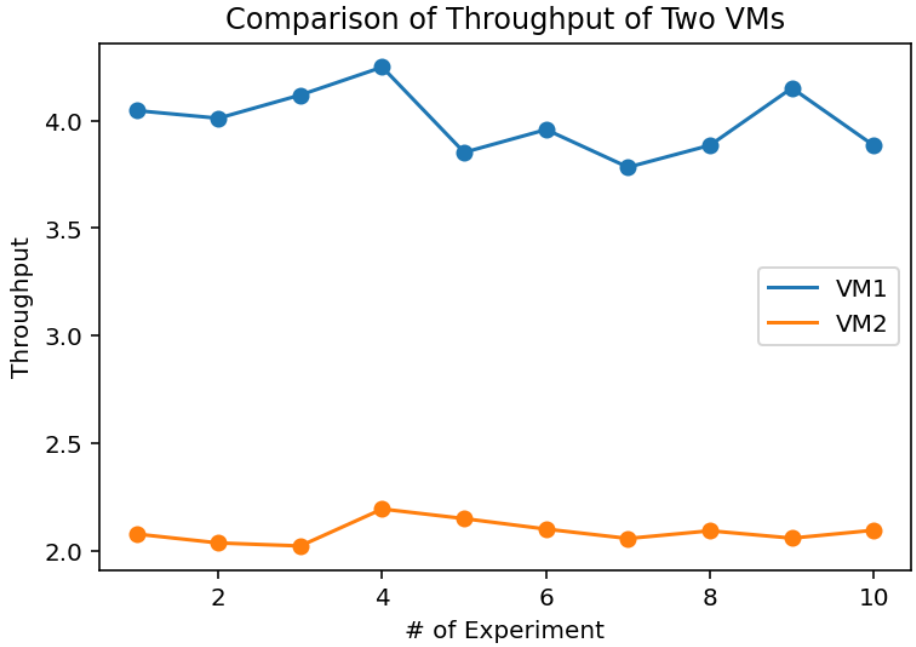

Fig 2. Comparison of Throughput of Two VMs

Summary:
With the statistics of two VMs, we can see that although the two VMs have the same size of RAM, but VM1 has 4 cores while VM2 only has 2 cores. Moreover, the performance of VM1 is nearly two times of VM2, which is proportionally to the ratio of their number of cores. To sum up, since this project handles requests with multiple threads, the system with more cores will generally yield better performance.