AASSIST - ONE STOP INTERNET BANKING

TEAM ONEMAN

PROJECT DESCRIPTION:

AAssist is a one stop internet banking solution wherein you can conduct transactions with any registered bank from a common mobile application interface. AAssist provides you with a virtual aassistant ***Susan*** who can make any transaction and you just have to speak to her. She can make complex transactions like fund transfers from multiple internet banking subscriptions of multiple banks without even you having to remember the credentials for each bank as you just need to get you subscription registered by Susan once and she would remember it forever.

Its a single app replacement for all the different banking apps of different banks having totally diverse Interfaces.The internet banking interface of different banks have elements and features which you might be hardly interested in and so it becomes more difficult for finding the actual feature you need but now you can just ask Susan to do it and she would.

Simple !!!

IMPLEMENTATION AND NOVELTY:

The entire system consists of four components.

1) AAssist Server powered by Intel (TBB)
2) AAssist Mobile Application (Android)
3) Mysql Database Server
4) AWS EC2 C5 Instance

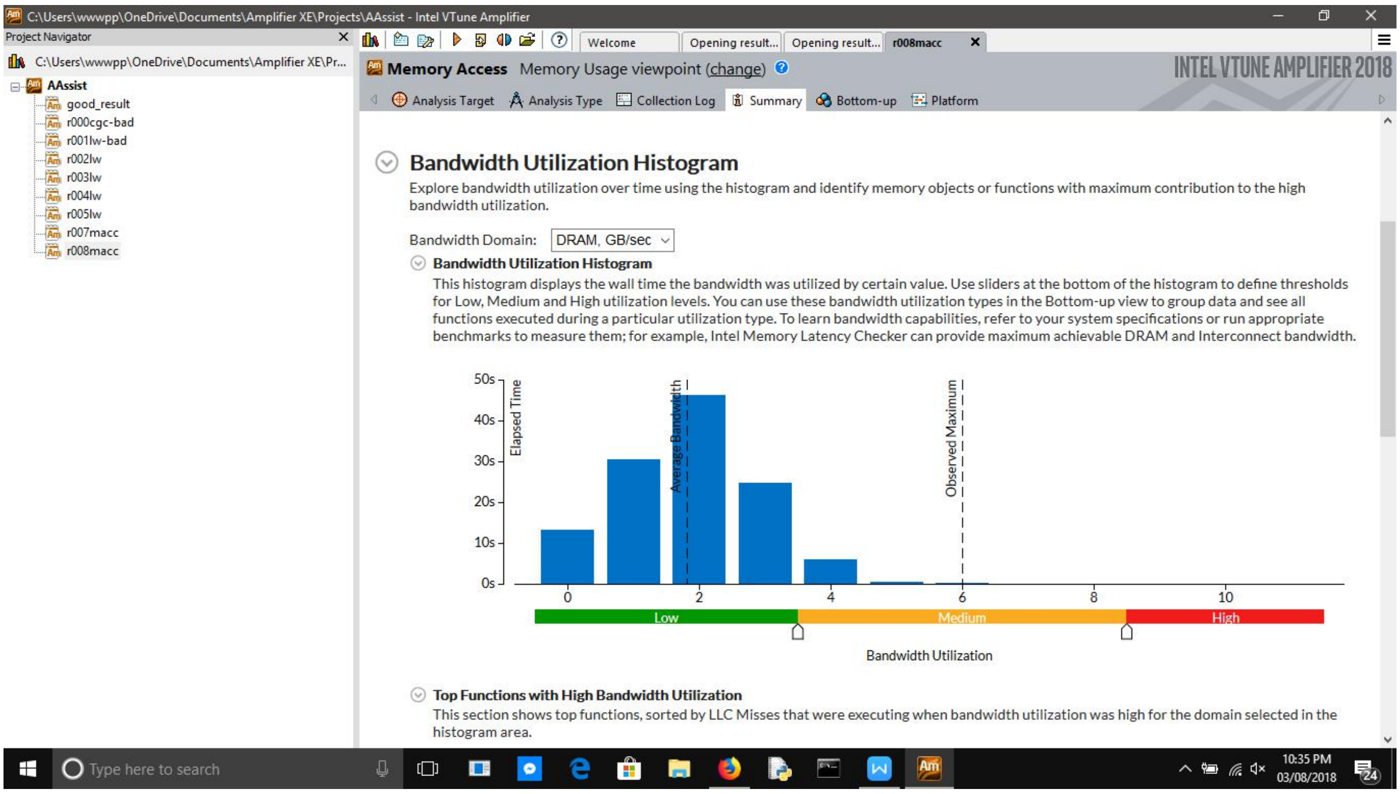
AASSIST SERVER :

The server is written in python powered by Tornado. It is a parallel processing enabled server which uses process and thread pools. The web handlers are run asynchronously in the background so that at any time the server is not in a blocking state.

The system itself consists of blocking long database operations and thus would stop execution, increasing response time and slowing everything. Two stop such a situation all blocking tasks are executed on thread pools in parallel and thus increasing speed. The increase in speed is proved later in this report.

AAssist Mobile Application:
This is the actual UI where the user can interact with AAssist server. “Susan” powered by Dialogflow for speech synthesis. The App has been developed so neatly that adding or removing a new feature will be as simple as adding or removing an element from an array and training the voice recognition model and we can release a new update. The proof of this system can be seen in file “botlogic.java” in the android app source. Susan can also answer generic questions which are not of any internet banking command context. Between any transaction you can also change the current transaction context. For example: In the middle of a fund transfer you can start checking your balance and then again ask Susan to make a fund transfer and continue from where you left.

CODE OPTIMIZATION:
Every reusable code was encapsulated in modules and classes so that everything could be put together fast enough. The user driven details could be edited as fast as just editing the config.yml file. File I/O has been reduces as much as possible so that the server is never stuck in a deadlock causing race conditions. The blocking operations were submitted to thread pools for parallel execution asynchronously. The memory access bandwidth utilization of the app can be show by the following output from Intel vTune Amplifier.



Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain: DRAM, GB/sec

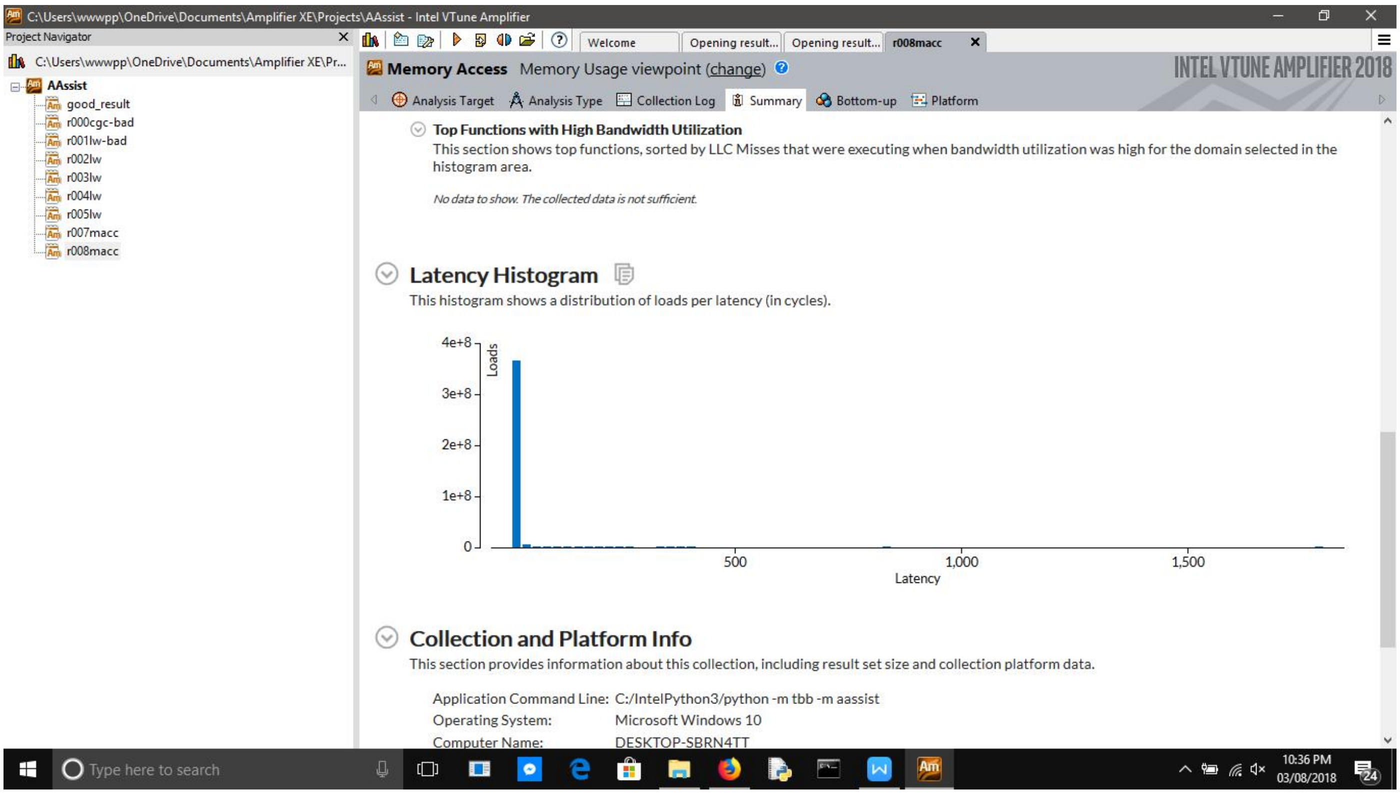
Bandwidth Utilization Histogram

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.

Top Functions with High Bandwidth Utilization

This section shows top functions, sorted by LLC Misses that were executing when bandwidth utilization was high for the domain selected in the histogram area.

BANDWIDTH UTILIZATION HISTOGRAM


C:\Users\wwwpp\OneDrive\Documents\Amplifier XE\Projects\AAssist - Intel VTune Amplifier
Project Navigator
Memory Access Memory Usage viewpoint (change)
INTEL VTUNE AMPLIFIER 2018
Loads
Latency
Type here to search
10:36 PM 03/08/2018

Top Functions with High Bandwidth Utilization

This section shows top functions, sorted by LLC Misses that were executing when bandwidth utilization was high for the domain selected in the histogram area.

No data to show. The collected data is not sufficient.

Latency Histogram

This histogram shows a distribution of loads per latency (in cycles).

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

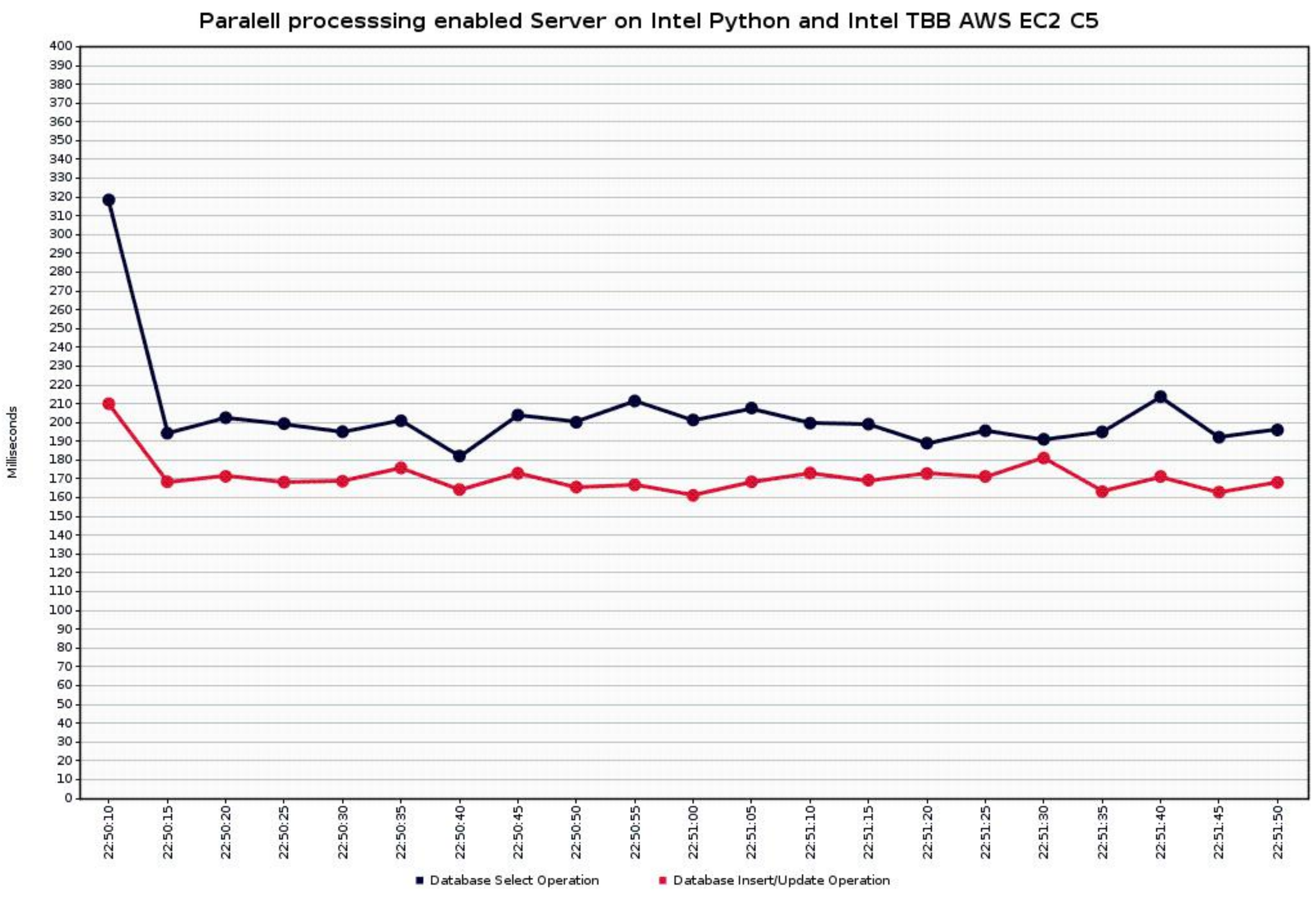
Application Command Line: C:/IntelPython3/python -m tbb -m aassist
Operating System: Microsoft Windows 10
Computer Name: DESKTOP-SBRN4TT

LATENCY HISTOGRAM

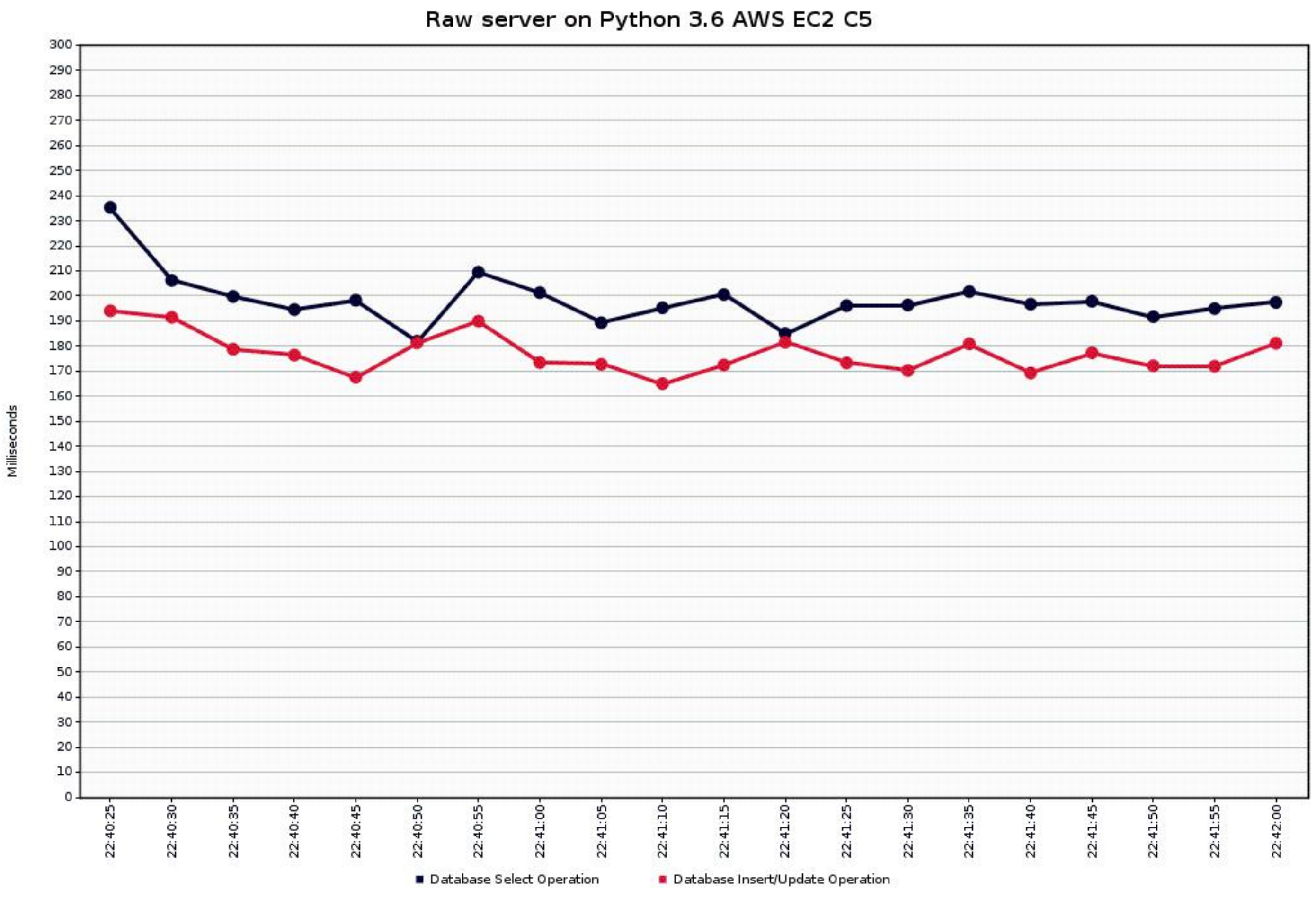
PERFORMANCE NUMBERS:

To make sure that we squeeze out the maximum performance from the processors using Intel TBB and thread pools.The performance test were conducted by **Apache Jmeter** to send out requests to the server and test its response time on AWS EC2 C5.
Number of Connections : 100
Request Types: Balance Enquiry and Change Default Account
Number of times to run test : 10

The Database select operations where more or less same and gave a performance increase of *nearly 1%* on an average . But **huge** performance improvement was seen in update/insert Database operations through Tornado Server on Intel Python with Intel TBB had response time of **average 169ms** where the raw server had a response time of **average 176ms**. A performance improvement of **3.97% nearly 4%** which is a valuable increase in performance. Further we can see that the Intel powered server maintains a constant curve/line where as the raw server does not stay constant.
The analysis data is show below.



Paralell processsing enabled Server on Intel Python and Intel TBB AWS EC2 C5
Milliseconds
Database Select Operation
Database Insert/Update Operation

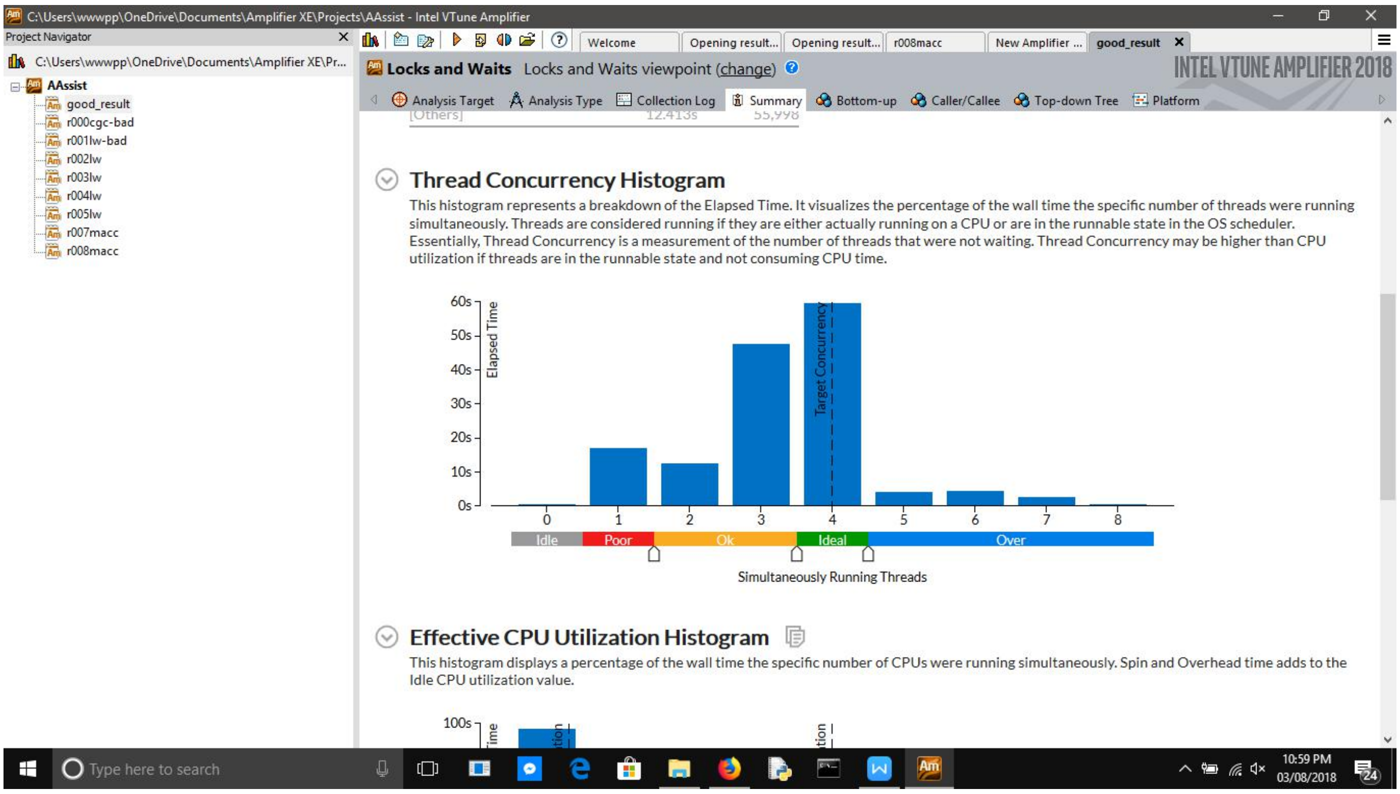


Raw server on Python 3.6 AWS EC2 C5
Milliseconds
Database Select Operation
Database Insert/Update Operation

STEPS FOR IMPROVEMENT : PERFORMANCE & CPU UTILIZATION :
When I initially ran my raw server on Intel Python first I saw that I could get my system to work effortlessly and everything was working fine but..

I brought up my server under high load test in Apache Jmeter and saw that it gave me an average response time of 240ms . Then I analyzed it using Intel vTune Amplifier for Locks and Waits . I could then find out that the database operations - connecting, querying, inserting and updating as well as commits, rollbacks and closing operations were taking most of the execution time. Then I though of deploying it using Thread Pools and submitted the blocking operations to them for getting it done asynchronously. When I again analyzed the app I found that that the select queries were the only once taking time because the thread had to wait for the result to be returned other than that everything was way better. But still I could not get Thread concurrency higher. Then I ran my python code with -TBB flag which gave me a higher thread concurrency (Result attached below) and subsequent reduction in oversubscription.

NOTE: Analysis was conducted by Intel vTune for exactly 148.811 secs Equivalent to Apache Jmeter test of 100 concurrent users and 2 different tests , looped 10 times.



Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU utilization if threads are in the runnable state and not consuming CPU time.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

But still I did not get a high CPU concurrency. Then I ran the server on a process pool with one process per core and then I could get the high CPU concurrency.

APPLICATION CORRECTNESS/VALIDITY:

Every possible measure has been taken to improve the overall performance of the entire system. Tools such as **Intel Distribution for Python, Intel TBB, Intel vTune Amplifier and Apache Jmeter** have been used to analyze the performance of the system and improve benchmarks and overall response times and any possible error that might stop such an app.