Application of Virtual Client for Azure Hardware Qualification

Anna Mary Mathew
anna.mathew@microsoft.com

Bryan DeYoung
brdeyo@microsoft.com

Michael Chhor
michaelchhor@microsoft.com

Sharjil Khan
sharjilkhan@microsoft.com

***Abstract*—Virtual Client (VC) is an open-source platform intended to evaluate system performance by running industry available benchmarks. VC helps to standardize the workflow for validation and qualification, compare relevant system performance generation over generation, and correlate workload performance and system telemetry for results review.**

Keywords—Hardware, Qualification, Virtual Client

I. Introduction

With Azure server hardware systems, there is a pervasive need to run many tests of different kinds on every new platform produced. The first focus is to ensure there are no fundamental issues with the platform and that it can function to the design specifications. The next focus is to measure the performance of the system using a repeatable methodology that allows for verification that the system is producing generation-over-generation improvements as expected. In the end, there can be hundreds of tests to run on each and every system, and the work involved is a daunting task to expect someone (or some team) to produce manually. In fact, it would require an army of people to conduct server hardware qualifications at the magnitude inherent in commercial cloud server design and development. Automation is essential to making scalable processes possible.

Virtual Client is an open-source platform, designed by Microsoft, that enables users/teams to run a large set of tests quickly and efficiently on any hardware server system while capturing telemetry structured for analysis and decision-making. Virtual Client, or "VC" for short, has a set of 'onboarded' or supported public and industry standard workloads that can be executed today. As with many things in today's world, VC is a living and growing platform and is constantly improving and adding support for more workloads and more hardware systems. This paper talks about how Virtual Client was used as part of the validation strategy to evaluate and validate hardware designed for Azure datacenter.

This document will not go into finer details of the Virtual Client platform as it relates to design, implementation, execution, and debugging of hardware platforms. Please see the References section for more information regarding Virtual Client.

A. Problem Statement

As newer AI models and benchmarks are released, there is a growing need to standardize qualification efforts and automate new workloads into existing test automation harnesses. Combining this with the ever-growing number of hardware platforms developed for AI space, the task of qualifying the systems for production use becomes an incredible task. The following sections will describe the learnings from the process of onboarding new workloads to the Virtual Client platform and using them to support AI server hardware system validations that resulted in standardizations. The remainder of this document discusses the work and process that facilitated the opportunities.

II. Use of Virtual Client for Hardware Evaluation

Virtual Client (VC) is an open-source platform that can easily adapt to a variety of hardware, software, and operating system configurations with little to no change. With support for Windows and Linux environments, VC simplifies workload execution. Simplifications include aspects such as the installation of dependencies, the installation of workload tooling, setting operating system settings and scaling resource usage for workloads to the system on which it is running. This simplifies the need to dig into the many details required by the workload itself (e.g. understanding how to build and compile tools/applications in Linux) just to get it running to begin with. Using a single command line to execute the workload, VC will install required dependencies, install the workload toolset, and then execute the workload using repeatable steps. As the workload is running, VC will collect results along with information about the system itself and will upload structured telemetry for this information into a centralized cloud storage location. This enables the user/team to take advantage of "big data" processing resources available publicly in the Azure cloud. This in turn makes it very easy to analyze data over a large sample size (e.g. many distinct server hardware systems) removing the need to task a person with logging into each system to capture the log files and results. Additionally, the ability to execute predictable steps is useful for replicating repeatable results across different machines or for reproducing specific behaviors when debugging particular issues.

Virtual Client is executed using a single command line per workload. On one hand, VC can be executed by a user on a single server running workloads or tests or to help repro/debug a specific issue. On the other hand, VC allows for easy integration into higher-level orchestration or automation systems. Integration into automation improves the breadth and depth of testing that can be completed within a given timeframe on a set of server hardware systems by minimizing setup and downtime between tests as well as human resources required to manage the operations. The VC command line offers parameters that define workload(s) to execute. However, the user can additionally specify a variety of 'monitors' to run in the background that collect telemetry from the system around the workload it is

executing. This enriches the understanding of the overall system behavior while being exercised by the workload. For example, this telemetry data can be used to identify performance or reliability trends and/or undesirable behaviors on the server hardware exposed by the operations/influence of the workload enabling users to identify appropriate actions.

The use of Virtual Client is not limited to a select few types of environments. It is useful across smaller application domains such as benchtop testing and in development environments where there might only be a few server nodes or a few racks worth of server nodes such as with early design-phase SKU qualifications. It operates just as well in larger-scale application domains where there may be many clusters of server hardware to validate. Suffice it to say, the data and telemetry gathered by VC across the range of application domains and scale provides ample opportunities to identify performance trends and flaws with the accuracy required to meet cloud business prediction models.

Another point to consider is that Virtual Client minimizes the challenges and distinctions of evaluating systems in bare metal versus virtualized environments. VC is designed to operate seamlessly in both environments and is additionally designed to scale appropriately to the resources (e.g. CPU, memory, disk size) available on those systems. This is very useful because it allows the user to perform initial hardware evaluations in the bare metal environment simplifying the process of testing a server without intermingling the hypervisor layer and to compare those results with the runs from the virtualized environment illuminating the influence of the hypervisor. In most cases, bare metal testing provides the truest view of the server hardware and its relative performance and is expected to achieve the highest performance levels. Testing in virtualized environments, however, produces outcomes that are closer to the customer experience (i.e. the commercial cloud production use case). The performance is not expected to be as high as with bare metal execution due to a more complex software stack but provides a great view of what can be legitimately offered to customers.

A. Workloads in Virtual Client

Virtual Client provides a wide range of workloads for hardware validation and system performance measurement. There are many Windows/Linux based standardized workloads that can run on both x86 and ARM based architectures which enable comprehensive testing and performance comparison across CPU, Memory, Network, Disks etc. There are several new Machine Learning/AI workloads such as MLPerf and SuperBechmark with detailed performance metrics that allow engineers to evaluate training/inference performance between systems. Table 1 shows some examples of workloads that are currently available on VC. Using single line commands to launch each of these workloads makes it easy to automate execution using scripts or other test orchestration software. Coverage across many hardware areas and use cases can be achieved by selecting a few of the workloads from VC. More workloads can be added to VC to target specific areas based on platform requirements. The ability to then use these newly added workloads across other projects ensures maximum re-usability.

Table 1: Examples of Workloads Offered by Virtual Client Out of the Box

Hardware Coverage	**Workloads**
CPU Performance	CoreMark, Prime95, HPLinpack
Memory	LMbench, Memcached, Redis
Network	NTttcp, CPS, Latte
Disk I/O	DiskSpd, Flexible I/O Tester(FIO)
Fault Tolerance	Stress-Ng, StressAppTest
Compression	7Zip, Gzip
Database Performance	PostGreSQL
Java Performance	SPECjvm
High Performance Compute	HPCG, NAS Parallel
Power Usage	SPECpower
Machine Learning / AI	MLPerf, NVLPerf, SuperBenchmark

Note: Adopted from Microsoft SCHIE CRC OCP 2023 presentation [1]

B. Measuring Performance

Virtual Client offers standardized methods to collect system information, telemetry and performance data regardless of the workload being executed. These data points give a detailed snapshot of the state of the system while a workload is being executed for detailed analysis later. For example, “performance counters are captured on Windows systems every 1 second and are aggregated/averaged out on 10-minute intervals by default. This allows the Virtual Client to have a large number of samples over the interval of time when calculating averages. This in turn increases the accuracy and validity of the performance measurements” [5]. In addition to performance measures, it is possible to capture system health information for CPU, Memory, PCI devices and GPUs which help debug issues and identify unintended bottlenecks. VC is also able to utilize Nvidia SMI utility to provide GPU health information at all times including aggregates for ECC errors, instantaneous GPU power draw and PCIe topology information.[5]

There are pre-defined Monitor profiles that can be used that allow the user to pre-select a group of metrics they want to collect.

“--profile=MONITORS-GPU-NVIDIA.json”

For example, using the above “MONITORS-GPU-NVIDIA” profile will automatically collect all Nvidia related performance counters using nvidia-smi utility without the user having to explicitly install any packages.

In addition to all the platform level information, additional data can also be collected by the various benchmarks in the form of workload outputs. For example, while running SuperBenchmark various bandwidth measurements can be captured. Table 2 describes some

bandwidth data for memory copy between CPU and GPU that can be captured and aggregated. Many other relevant data can be captured based on the workload and the scenarios being tested.

Table 2: Example of Memory Copy Bandwidth Data Available Using SuperBenchmark

Name	Unit	Description
cpu_to_gpu[0-9]+_by_gpu[0-9]+_using_(sm|dma)_under_numa[0-9]+_bw	bandwidth (GB/s)	The bandwidth reading from all NUMA nodes' host memory using DMA engine or GPU SM by all GPUs.
gpu[0-9]+_to_cpu_by_gpu[0-9]+_using_(sm|dma)_under_numa[0-9]+_bw	bandwidth (GB/s)	The bandwidth writing to all NUMA nodes' host memory using DMA engine or GPU SM by all GPUs.
gpu[0-9]+_to_gpu[0-9]+_by_gpu[0-9]+_using_(sm|dma)_under_numa[0-9]+_bw	bandwidth (GB/s)	The bandwidth reading from or writing to all GPUs using DMA engine or GPU SM by all GPUs with peer communication enabled.

Note: From Virtual Client Superbenchmark documentation [6]

C. *Reviewing Results with Azure Data Explorer*

All the system information, telemetry, results, and even the individual test steps executed by Virtual Client are captured and uploaded to Azure Data Explorer (ADE), a "big data" service that is publicly available and that allows for the exploration and analysis of large volumes of data using the Kusto Query Language (KQL). This service will allow users to query and analyze the information/telemetry captured by the Virtual Client at a large scale. This information will be structured specific to the server hardware systems and workloads/tests executed allowing users to analyze the outcomes across different areas of validation requirements (e.g. reliability, performance). Some examples of the types of analysis data that is available in Azure Data Explorer include:

- Workload output data to understand execution progress and results.
- Context informational data to best describe the state of the system during the execution of a workload/test and that is useful for identifying breaches in expected operating conditions.
- Step-by-step commands executed by VC to understand if there is a failure within the individual workload/test toolsets or the platform runtime itself.

From Azure Data Explorer, users can directly access and view the data using queries based on the Kusto Query Language (KQL). Kusto Query Language is a robust and powerful language for data science and analytics needs. Experienced users can manipulate and post-process the data right across giant-sized data sets with relative ease to derive high value insights. The data can even be downloaded and saved offline for additional post-process and analysis with other tools the user may want to use.

Users can often start with simple queries that show the data in "raw" form as tabular views with rows of information. 'Raw' form data represents the individual, discreet data points captured by Virtual Client that can be used to derive precise intelligence. This raw data forms the basis for more advanced queries and views of the data where it is aggregated, expanded and further contextualized into a rich view of information that enables decision making. Figure 1 is an example of "raw" form data and Figure 2 shows the query used to gather the data. Note both the options available and the simplicity inherent in the Kusto Query Language. In this example, the data is collected from an open source AI/ML workload called SuperBenchmark that is often used to evaluate AI/ML systems with specialized GPU components.

ScenarioName	ScenarioStartT...	ScenarioEndTime	MetricCategori...	MetricName	MetricValue
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	cpu-memory-bw-latency/mem_max_bandwidth_stream-triad_like_bw	747229.92
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_gpu1_write_by_sm_bw	250.263
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_gpu1_write_by_dma_bw	265.725
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_gpu1_read_by_sm_bw	250.945
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_gpu1_read_by_dma_bw	263.734
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_sm_under_numa0_bw	53.6358
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_dma_under_numa0_bw	57.6796
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_sm_under_numa0_bw	54.2635
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_dma_under_numa0_bw	58.0663
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu0_write_by_sm_bw	250.245
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu0_write_by_dma_bw	265.797
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu0_read_by_sm_bw	250.871
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu0_read_by_dma_bw	263.902
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu1_by_sm_bw	1747.26
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_gpu1_by_dma_bw	1415.69
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu0_by_sm_under_numa1_bw	53.634
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu0_by_dma_under_numa1_bw	57.6816
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_cpu_by_sm_under_numa1_bw	54.2644
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_cpu_by_dma_under_numa1_bw	58.0702
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_sm_under_numa1_bw	53.6334
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_dma_under_numa1_bw	57.6808
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_sm_under_numa1_bw	54.2618
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_dma_under_numa1_bw	58.0648
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu0_by_sm_under_numa2_bw	53.6386
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu0_by_dma_under_numa2_bw	57.6807
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_cpu_by_sm_under_numa2_bw	54.2642
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu0_to_cpu_by_dma_under_numa2_bw	58.0656
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_sm_under_numa2_bw	53.6368
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/cpu_to_gpu1_by_dma_under_numa2_bw	57.6809
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_sm_under_numa2_bw	54.2646
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	gpu-copy-bw:perf/gpu1_to_cpu_by_dma_under_numa2_bw	58.0689
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-base/fp16_train_throughput	943.8245...
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-base/fp8_hybrid_train_step_ti...	62.87826...
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-base/fp8_hybrid_train_throug...	509.8149...
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-large/return_code:0	0
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-large/return_code:1	0
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	model-benchmarks:bert/pytorch-bert-large/fp32_train_step_time	152.6580...
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_1024_busbw	0.11
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_1024_algbw	0.11
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_1024_time	9.17
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_2048_busbw	0.24
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_2048_algbw	0.24
SuperBenchmark	2023-12-06 21:2...	2023-12-07 00:14:1...	2xH100.yaml	nccl-bw:nvlink/allreduce_2048_time	8.38

Figure 1: Raw Data View of ADE Query

```
let profile = "PERF-GPU-SUPERBENCH.json";
let dateRangeBegin = ago(360d);
let dateRangeStart = datetime(2023-12-07);
let dateRangeEnd = datetime(2023-12-09);
cluster("azurecrcworkloads.westus2.kusto.windows.net").database("WorkloadPerformance").Metrics
| where Timestamp >= dateRangeStart and Timestamp <= dateRangeEnd and ProfileName == profile
| where ToolName == "SuperBenchmark" and MetricCategorization == "2xH100.yaml"
| where ClientId startswith "684,"
| extend Parameters = todynamic(CustomDimensions.executionProfileParameters)
| extend Metadata = todynamic(CustomDimensions.metadata)
| extend ExperimentName = tostring(Parameters.experimentName)
| extend Revision = tostring(Metadata.revision)
```

Figure 2: Sample ADE Query for Raw Data

In contrast, more experienced KQL query users can manipulate the data within ADE to generate views that are easier to analyze and present a clearer understanding of the data. Figure 3 shows an example of partially processed data and Figure 4 shows the KQL query used to generate the table.

ToolName	MetricName	count_	Minimum	Average	50th	75th	90th	Maximum
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_2_0_lat	702	202.8	203.9	202.8	202.9	202.9	207.1
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_2_1_lat	702	202.2	203.3	202.2	202.2	202.2	207
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_2_2_lat	702	106.4	106.9	106.4	106.4	106.4	107.4
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_2_3_lat	702	116.1	116.9	116.1	116.1	116.1	121.1
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_3_0_lat	702	202.3	203	202.3	202.3	202.3	208.6
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_3_1_lat	702	202.7	203.5	202.7	202.7	202.7	207
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_3_2_lat	702	116.1	116.5	116.1	116.1	116.1	119.4
SuperBenchmark	cpu-memory-bw-latency/mem_latency_matrix_numa_3_3_lat	702	105.9	106.6	105.9	105.9	105.9	107.3
SuperBenchmark	cpu-memory-bw-latency/mem_max_bandwidth_1_1_reads-writes_bw	702	745424.3	750094	745529.2	745753.2	745837.8	751531.6
SuperBenchmark	cpu-memory-bw-latency/mem_max_bandwidth_2_1_reads-writes_bw	702	769133	771680.8	769308.1	769468.8	769524.4	775307.2
SuperBenchmark	cpu-memory-bw-latency/mem_max_bandwidth_3_1_reads-writes_bw	702	762120.6	763603.8	762397.5	762414.8	762425.8	766001.2
SuperBenchmark	cpu-memory-bw-latency/mem_max_bandwidth_all_reads_bw	702	797813	800984.1	797919.3	798032.9	798182.1	806242.8
SuperBenchmark	cpu-memory-bw-latency/mem_max_bandwidth_stream-triad_like_bw	702	744942.8	746265.4	744993.5	745035.8	745051.9	748667
SuperBenchmark	cpu-memory-bw-latency/return_code	702	0	0	0	0	0	0
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_1024_flops:0	702	938.4	949.9	940.7	941.9	942.3	959.2
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_1024_flops:1	702	937.9	950.7	941.9	942.9	943.1	960.1
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_3072_flops:0	702	300.5	1224.8	1214.3	1214.5	1215	1238
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_3072_flops:1	702	308.1	1226.2	1217.6	1217.7	1219	1237.6
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_4096_flops:0	702	377.3	1344.1	1316.9	1317.4	1317.7	1367.3
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_1024_4096_flops:1	702	388.5	1338.3	1311.5	1313	1313.3	1363.4
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_3072_1024_flops:0	702	299.4	1220.6	1212.8	1213.2	1213.3	1233.9
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_3072_1024_flops:1	702	307.6	1221.5	1213.2	1213.5	1213.5	1233.3
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_4096_1024_flops:0	702	371.2	1242.4	1231.2	1231.9	1232.3	1252.5
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_1260B_4096_1024_flops:1	702	380.2	1244	1235.7	1236.7	1237	1253.2
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_16384_16384_16384_flops:0	702	433.4	1187.1	1157.9	1159	1160	1211.1
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_16384_16384_16384_flops:1	702	446.6	1183	1159.1	1161.6	1162.1	1211.2
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_4096_4096_4096_flops:0	702	351.1	1482.6	1466.5	1471.5	1471.9	1498.6
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_4096_4096_4096_flops:1	702	360	1484	1472.6	1474.2	1474.6	1499.9
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_8192_8192_8192_flops:0	702	411.5	1275.6	1182.4	1185.6	1185.6	1411
SuperBenchmark	cublaslt-gemm/fp8e4m3_0_8192_8192_8192_flops:1	702	421.4	1271.6	1180.2	1185	1185.1	1410.7
SuperBenchmark	cublaslt-gemm/return_code:0	702	0	0	0	0	0	0
SuperBenchmark	cublaslt-gemm/return_code:1	702	0	0	0	0	0	0
SuperBenchmark	gemm-flops/bf16_tc_flops:0	700	333307	341651.9	333856	334077	334085	348627
SuperBenchmark	gemm-flops/bf16_tc_flops:1	700	329397	339806.2	330085	330257	330368	347446
SuperBenchmark	gemm-flops/fp16_flops:0	700	54287.1	54646	54292	54310.7	54319.2	54664.1
SuperBenchmark	gemm-flops/fp16_flops:1	700	53649.8	54614.6	53667.2	53694.7	53701.8	54664
SuperBenchmark	gemm-flops/fp16_tc_flops:0	700	355837	363805	356158	356216	356253	372826
SuperBenchmark	gemm-flops/fp16_tc_flops:1	700	351484	362071.9	352813	352932	352975	371736
SuperBenchmark	gemm-flops/fp32_flops:0	700	39175.6	40123	39181.2	39185.1	39186	40946.3
SuperBenchmark	gemm-flops/fp32_flops:1	700	38761.4	39927.2	38851	38873.4	38879.1	40880.5
SuperBenchmark	gemm-flops/fp64_flops:0	700	20257.9	20641.6	20277.5	20288.3	20291.4	20698.5
SuperBenchmark	gemm-flops/fp64_flops:1	700	20028.8	20588.4	20057.8	20067.1	20070.8	20698.7
SuperBenchmark	gemm-flops/fp64_tc_flops:0	700	29211.7	29291.1	29217.8	29225.8	29227	29299.2

Figure 3: Summarized Data View of ADE Query

```
let profile = "PERF-GPU-SUPERBENCH.json";
let dateRangeBegin = ago(360d);
let dateRangeStart = datetime(2023-12-07);
let dateRangeEnd = datetime(2023-12-09);
cluster("azurecrcworkloads.westus2.kusto.windows.net").database("WorkloadPerformance").Metrics
| where Timestamp >= dateRangeStart and Timestamp <= dateRangeEnd and ProfileName == profile
| where ToolName == "SuperBenchmark" and MetricCategorization == "2xH100.yaml"
| where ClientId startswith "684_"
| extend Parameters = todynamic(CustomDimensions.executionProfileParameters)
| extend Metadata = todynamic(CustomDimensions.metadata)
| extend ExperimentName = tostring(Parameters.experimentName)
| extend Revision = tostring(Metadata.revision)
| summarize count(), Minimum = round(min(MetricValue),1), Average = round(avg(MetricValue),1), 50th = round(percentile(MetricValue, 0.5),
1), 75th = round(percentile(MetricValue, 0.75),1), 90th = round(percentile(MetricValue, 0.9),1), Maximum = round(max(MetricValue),1) by
ToolName, MetricName
| order by ToolName asc, MetricName asc
```

Figure 4: Sample ADE Query for Summarized Data View

This example data was also captured during the execution of the SuperBenchmark workload. In fact, it shows 700 distinct data points captured for each metric (e.g. 'MetricName') emitted by the SuperBenchmark workload. The KQL example shows how "raw" form data can be easily aggregated into a summary/rollup view. For example, the query aggregates the metrics into various buckets such as minimum, maximum, average, and specific percentiles (e.g. P50, P90) for easier analysis and digestion of the data. This is especially useful when comparing the aggregated results of individual systems with other systems or even against aggregations of the lab as a whole.

III. Onboarding of AI Workloads

Virtual Client is a fast-evolving platform that must constantly adapt to the needs of new server hardware platforms. One of the major advantages of VC is the ability to onboard new workloads (or monitors) to the platform in a relatively efficient manner and in a way that ensures consistency with all other workloads. When a workload is onboarded to VC, the fundamental goal is to dramatically simplify what it would take to execute the workload in an effective manner on a cloud server system. Workload onboarding requirements vary from one workload to another; however, to do so almost always requires a deep level of understanding of the workload (e.g. research involved). To onboard the workload might require understanding where to find the workload toolsets, identifying which operating systems or architectures it can run on and determining how to run the workload effectively on the system. It might require knowledge for interacting with Git repos and how to build/compile the workload from those repos. There might be an extensive set of options on the command line available for controlling the behaviors of the workload. There are an endless number of challenges for running workloads on cloud server systems that require some level of expertise to be developed each time before the workload can be usefully onboarded. The VC team of engineers and open-source community do a great deal of the heavy lifting here to make it possible by leveraging the expertise of hardware, system and software engineers across Microsoft and the industry. The VC engineering team additionally scrutinizes the workloads that are chosen for onboarding to ensure those selected offer high return on investment for most of the server hardware system scenarios that matter for the Azure cloud business. In essence, workloads selected must be demonstrably useful throughout the lifetime of the hardware from design to production maintenance and sustainment. When the workload is onboarded, it will come with a default profile (a "recipe") that comes with expertise built-in and designed to run on most relevant server hardware systems. The user does not need to have any expertise whatsoever in order to run the workload profile/recipe.

The concept of a recipe is a particularly important aspect for server hardware validation. Employing a recipe increases the reproducibility of a workload scenario and is especially useful for surfacing bug/defect behaviors. This is also important when running the workload multiple times on a system and across different sets of systems where fidelity across the systems is important. Virtual Client profiles/recipes are designed with this in mind, and the user or automation can execute Virtual Client with the confidence that the workload will be run in the exact same way each time. Virtual Client abstracts away many complexities for modern day server hardware validations. A simple command line is all that is required.

The following are examples of workloads that were onboarded in a collaboration between Microsoft system engineering teams and the Virtual Client engineering team. These workloads were onboarded and used to qualify Nvidia and AMD GPU-integrated AI/ML server platforms:

- Workloads for Nvidia H100 PCIe and HGX-based platforms:
  - SuperBenchmark
  - GPU Stream
  - Nvidia DCGMI
- Workloads for AMD v620 based platforms
  - Furmark
  - SGEMM
  - UBM Perf

IV. Improvements to Virtual Client

Over the course of qualifying cloud server hardware systems at-scale, the teams involved inevitably find opportunities to make improvements to process, to automation and to the hardware systems themselves. Sometimes the automation finds new defects that were missed during design phases or coverage for areas not previously explored. These situations often lead to direct

improvements to the automation. In fact, Microsoft's use of the Virtual Client during the qualification of the H100 and v620 systems noted above resulted in several contributions to the overall improvement of the VC platform…a virtuous cycle. The onboarding of several workloads for the GPU and AI space (mentioned earlier) is a great example. Gaps in coverage were identified early on and filled. The workloads created the capabilities for testing Nvidia and AMD GPUs at to fill a fast-growing need with today's AI/ML server platform qualifications.

Additionally, improvements were identified for providing enhanced instructions to the Virtual Client to make debugging and triage requirements easier to request. These new features were exposed as command line parameters to give the user control over the behaviors (see the Command Line Parameters section). For example, the "--log-to-file" parameter enables users to specify that log files should be produced in addition to telemetry. This in turn sets the foundation to have those log files uploaded to a central file store for reference removing the need to login to any system.

V. LEARNINGS

The "--log-to-file" feature has proven to be very useful when developing and debugging test execution in lab scenarios. Since it creates a log on the filesystem, the user can immediately view this log to identify behaviors and develop next steps. The creation of the log file is near instantaneous once VC has completed running a workload scenario. In contrast, there is a finite amount of delay before the data is available in cloud storage. This feature has really enabled real-time feedback to test out a user's latest changes.

There is a large amount of data generated by a single VC test execution. This is due to capturing not only the performance metrics of the workload, but also the system telemetry data while the workload is running. It is very beneficial to have data post-processing to make the consumption and understanding of the data more manageable. Such post processing mechanisms make it easier to assess outcomes (and even graph the data) over time to highlight outliers that are beyond the specification range or to compare metrics between like systems.

VI. DEBUG METHODOLOGY AND RESULTS

Debugging the resulting behavior or state of the workload is a must with any validation effort. With VC, there are two main ways this can be achieved. One way is to use a service called Azure Data Explorer and the second is to use the "--log-to-file" to generate on system logs.

A. Azure Data Explorer

As highlighted earlier, the Kusto Query Language can additionally be used to gather telemetry on errors that happen during workload execution. In Figure 5 and Figure 6, the query looks for any errors that may have occurred when running the SuperBenchmark workload.

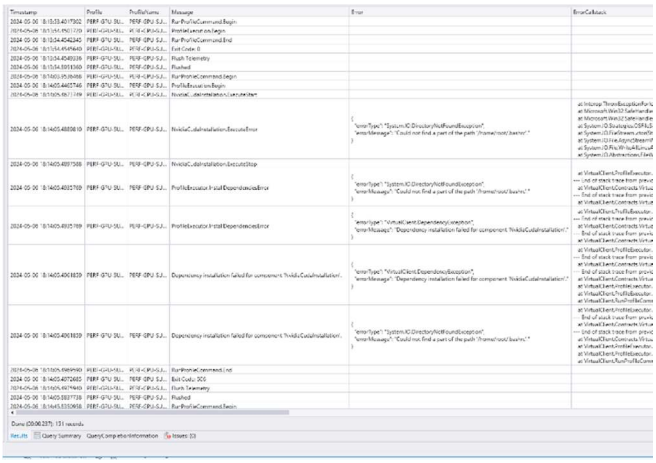

Figure 5: Same Query for Errors During Workload Execution

```
let profile = "PERF-GPU-SUPERBENCH.json";
let dateRangeBegin = ago(360d);
cluster("azurecrcworkloads.westus2.kusto.windows.net").database("WorkloadDiagnostics").Traces_Dev01
| where Timestamp >= dateRangeBegin and ProfileName == profile and SeverityLevel > 1 // severity levels: 1 = Info, 2: Warning, 3: Error, 4: Critical
| extend Errors = todynamic(CustomDimensions.error)
| extend ErrorCallstack = tostring(CustomDimensions.errorCallstack)
| mv-expand Error = Errors
| project Timestamp, Profile, ProfileName, Message, Error, ErrorCallstack, SeverityLevel, AppHost, AppVersion, CustomDimensions
| order by Timestamp asc
```

Figure 6: Sample Query for Errors

In the example, the errors and callstack are separated into separate fields. This makes it easy to identify what the error was and where the execution failed.

B. Command Line Parameters

There are several command line parameters the user can invoke at the time of execution. There are a few parameters that can aid debug when the system exhibits failure or undesired behavior.

"--log-to-file"

The "--log-to-file" parameter will cause VC to generate on system logs during execution. This is helpful because the logs will be on the system and the user will not need to use other means (such as ADE) to analyze the data. Not all of the data gathered is able to be logged into the system.

"--debug"

This parameter is useful when executing VC via the command line. Including this flag will cause VC to print additional debug info to the console.

"--fail-fast"

The "--fail-fast" parameter will cause VC to terminate once it encounters an error of any severity. Combining this flag with "--debug" can be a means to determine where the test is failing.

C. Results

Virtual client monitors were used to track the critical metrics while running critical AI workloads. In the work done key metric related to bandwidth, temperature profiles were collected for large language models.

The example used in section 2.C shows how VC collected the workload outputs and how they can be summarized for a particular view. In this case, the min, max, and specific percentiles were shown for the different metrics collected.

This type of view is very helpful to evaluate the performance of the hardware. In one sense, it shows the engineer how well the hardware can perform based on the minimum and maximum values achieved for a given metric. On the other hand, it allows the engineer to see how well the hardware sample size is performing as a single entity. The output data can be queried to show the precision of the hardware.

By slightly modifying the query, the engineer can view the system telemetry gathered while the workload was running. In a similar fashion to the workload output, the telemetry can be bucketed and viewed to confirm the similarity of system behavior.

VII. Acknowledgements

This work leverages the Virtual Client platform, which was designed and developed by the Cloud Readiness Criteria ("CRC") within Microsoft.

VIII. References

[1] Open Compute Project, "Virtual Client library for Azure - presented by Microsoft," *YouTube*, Nov. 01, 2023. https://www.youtube.com/watch?v=VhmFkazCQeM

[2] "Virtual Client Platform | Virtual Client Platform," microsoft.github.io. https://microsoft.github.io/VirtualClient

[3] shsagir, "Kusto Query Language (KQL) overview- Azure Data Explorer," learn.microsoft.com. https://learn.microsoft.com/en-us/azure/data-explorer/kusto/query/

[4] shsagir, "What is Azure Data Explorer?," learn.microsoft.com. https://learn.microsoft.com/en-us/azure/data-explorer/data-explorer-overview

[5] "Performance Counter Metrics | Virtual Client Platform," microsoft.github.io. https://microsoft.github.io/VirtualClient/docs/monitors/0100-perf-counter-metrics

[6] "SuperBenchmark | Virtual Client Platform," microsoft.github.io. https://microsoft.github.io/VirtualClient/docs/workloads/superbenchmark

[7] Munteanu, V. Debusschere, S. Bergeon and S. Bacha, "Efficiency metrics for qualification of datacenters in terms of useful workload," 2013 IEEE Grenoble Conference, Grenoble, France, 2013, pp. 1-6, doi: 10.1109/PTC.2013.6652470.

[8] D. Diamantopoulos and C. Hagleitner, "HelmGemm: Managing GPUs and FPGAs for Transprecision GEMM Workloads in Containerized Environments," 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 71-74, doi: 10.1109/ASAP.2019.00-27.

[9] L. Chen, S. Patel, H. Shen and Z. Zhou, "Profiling and Understanding Virtualization Overhead in Cloud," 2015 44th International Conference on Parallel Processing, Beijing, China, 2015, pp. 31-40, doi: 10.1109/ICPP.2015.12.

[10] H. M. Helal and R. E. Ahmed, "Performance evaluation of datacenter network topologies with link failures," 2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), Sharjah, United Arab Emirates, 2017, pp. 1-5, doi: 10.1109/ICMSAO.2017.7934898.

[11] M. Bala and Devanand, "Performance evaluation of cloud datacenters using various green computing tactics," 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2015, pp. 956-961.

[12] M. Emani et al., "A Comprehensive Evaluation of Novel AI Accelerators for Deep Learning Workloads," 2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Dallas, TX, USA, 2022, pp. 13-25, doi: 10.1109/PMBS56514.2022.00007.

[13] Z. Quan, X. Chen and Y. Han, "AIC-Bench: Workload Selection Methodology for Benchmarking AI Chips," 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Hainan, China, 2022, pp. 687-694, doi: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00117.

[14] Y. Wang et al., "Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training," 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, 2020, pp. 744-751, doi: 10.1109/CCGrid49817.2020.00-15.

[15] Z. Jiang et al., "HPC AI500 V2.0: The Methodology, Tools, and Metrics for Benchmarking HPC AI Systems," 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 2021, pp. 47-58, doi: 10.1109/Cluster48925.2021.00022.

[16] M. Plagge et al., "ATHENA: Enabling Codesign for Next-Generation AI/ML Architectures," 2022 IEEE International Conference on Rebooting Computing (ICRC), San Francisco, CA, USA, 2022, pp. 13-23, doi: 10.1109/ICRC57508.2022.00016.