

ENS 492 – Graduation Project (Implementation)

Progress Report II

Real-Time Supercomputer Monitoring

Ali Eren Ak, Deniz Batu, Muhammed Orhun Gale

Supervisor: Kamer Kaya



November 27, 2022

1 Project Summary

Increasing complexity in the High-Performance Computing (HPC) clusters entails various bottlenecks in these clusters both software and hardware-wise. Therefore, detecting and overcoming bottlenecks in order to increase the performance and energy efficiency of these parallel systems is an important research and practice topic nowadays. [KSV⁺21] [TRQR21] [BG09] To be able to tackle bottleneck problems, the first requirement is being able to detect bottlenecks in HPC clusters. Bottleneck detection in HPC clusters can be achieved via detailed monitoring of the cluster and interpreting the gathered data utilizing hardware counters and software events. However, to monitor a cluster in such an exhaustive manner, one needs to utilize complex hardware and software process investigation tools. Also, in order to enable reasoning of monitored metrics, complex tools should harmoniously convey data to a well-defined structure to comprehensively display remarking events. [IRM⁺20] [PPG20] [HBK05] [DWKN20].

A digital twin is a synchronized computerized model of a physical device which has active communication with the physical device in order to create a real-time image of the physical device. Conceptually, creating a digital twin of a HPC cluster would enable real time monitoring in a more structured manner and ease the analytic process. Also, structured data can be used for prediction and simulation purposes. [Che17] [KMM20]

Considering these, in our project "Real-Time Supercomputer Monitoring", we designed and developed a framework which assembles various complex hardware and software investigation tools and gather the data produced by these tools within a set of rules which describes a Digital Twin Description Language (DTDL) to enable real-time monitoring of HPC clusters. It has a system probing design which allows it to investigate parallel architectures with complex memory hierarchies in detail and detect internal communication. By using these, the tool collects cluster data such as cache line transfers in a systematic manner to enable vendors and customers to detect hardware, kernel, and user-program bottlenecks, provide them with performance and efficiency insights and perform prediction-based simulations of various hardware and or software scenarios by using the models that are trained via the

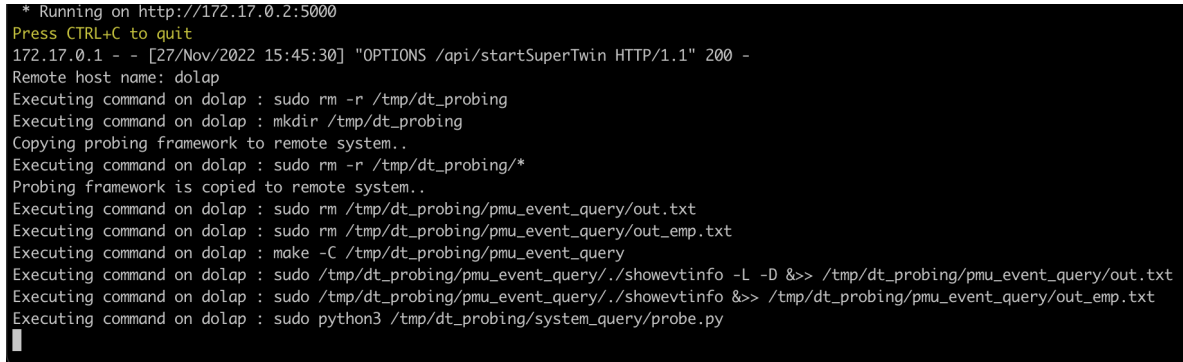
cluster’s digital twin all without creating significant overhead.

To achieve these, the tool initially probes the structure of the monitored system and create a Digital Twin Description (DTD) of it in a NoSQL database via using the DTDL which is specified. After that, the real-time monitoring phase starts by using the monitoring daemons. Daemons store the monitored system’s data into a time-series database and the monitoring system pipes the data to a visualization tool structurally by performing queries in accordance with the DTD. Also, in order to make it easy to use, this tool has an integrated web application.

2 Scientific/Technical Developments

In the Progress Report I, the following three tasks is given to be completed until the Progress Report II: digital twin implementation, structured database design and user interface development. Digital twin implementation task stands for the development of monitoring and Digital Twin Description (DTD) production features of the tool, structured database design task concerns about the real-time data transfer and data storage and the user interface development task is for the development of a web application for enabling efficient use of the tool. In the time between Progress Report I and II, general requirements that are provided by these three tasks are successfully completed and the tool is currently functional however, there still exist parts to be added in order to make the tool more convenient for project aims.

As the initial step of the development, modules that will be developed are identified to make the process more clear and efficient. The first identified module is the ”Probing and Benchmarks module” which is responsible to probe the hardware and software capabilities of the system by using various tools. In this module, the output of the probing and benchmarks are used to create the DTD of the system that is going to be monitored. The second module is ”Data Gathering module” which collects the system data by using various tools and stores in a time-series database. The third module is the ”Visualization module” which gets the time-series data from the database by queries which are created according to the DTD and visualizes them for real-time monitoring. As the last module, the ”Learning and Reasoning module” is identified to perform bottleneck detection and simulations. Also, it is decided to develop the core functionalities with Python.



```
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [27/Nov/2022 15:45:30] "OPTIONS /api/startSuperTwin HTTP/1.1" 200 -
Remote host name: dolap
Executing command on dolap : sudo rm -r /tmp/dt_probing
Executing command on dolap : mkdir /tmp/dt_probing
Copying probing framework to remote system..
Executing command on dolap : sudo rm -r /tmp/dt_probing/*
Probing framework is copied to remote system..
Executing command on dolap : sudo rm /tmp/dt_probing/pmu_event_query/out.txt
Executing command on dolap : sudo rm /tmp/dt_probing/pmu_event_query/out_emp.txt
Executing command on dolap : make -C /tmp/dt_probing/pmu_event_query
Executing command on dolap : sudo /tmp/dt_probing/pmu_event_query/./showevtinfo -L -D &>> /tmp/dt_probing/pmu_event_query/out.txt
Executing command on dolap : sudo /tmp/dt_probing/pmu_event_query/./showevtinfo &>> /tmp/dt_probing/pmu_event_query/out_emp.txt
Executing command on dolap : sudo python3 /tmp/dt_probing/system_query/probe.py
```

Figure 1: Digital twin instantiation via using the web application

The Probing and Benchmarks module is developed to utilize various hardware and software probing tools namely, perf, likwid, lshw, libpfm4, cpuid, pciutils and ethtool in order to subtract the system structure of the monitored system. By using the DTDL that we described in the Progress Report I, a Python script generates DTD recursively via using the outputs of these probing tools. Also, in order to profile the system, various benchmarks that would reveal the utilization of certain aspects of the system such as STREAM and HPCG are conducted on the system and added to the DTD structure. [1][2]

```

Remote probing is done..
Creating a new digital twin with id: 2bb28f62-f99e-4603-932b-aa4385ed19cb
Collection id: 63838642938892d328a60734
pcp2influxdb configuration: pcp_dolap_monitor.conf generated
A daemon with pid: 941 is started monitoring dolap
STREAM Benchmark thread set: [1, 2, 4, 8, 16, 22, 32, 44, 64, 88]
STREAM benchmark script generated..
STREAM benchmark result added to Digital Twin
Using database 'dolap_main' and tags 'tag=_monitor'.
Sending 35 metrics to InfluxDB at http://host.docker.internal:8086 every 1.0 sec...
(Ctrl-C to stop)
HPCG Benchmark thread set: [1, 2, 4, 8, 16, 22, 32, 44, 64, 88]
HPCG benchmark script, with params nx: 104 ny: 104 nz: 104 time: 60 is generated..
HPCG benchmark result added to Digital Twin
CARM config generated..
ADCARM Benchmark thread set: [1, 2, 4, 8, 16, 22, 32, 44, 64, 88]
adCARM benchmark script generated..
CARM benchmark result added to Digital Twin
Twin state is registered to db..
172.17.0.1 - - [27/Nov/2022 15:46:17] "POST /api/startSuperTwin HTTP/1.1" 200 -
172.17.0.1 - - [27/Nov/2022 15:46:22] "GET /api/setDB HTTP/1.1" 200 -
172.17.0.1 - - [27/Nov/2022 15:46:24] "GET /api/getMetrics/monitoring HTTP/1.1" 200 -

```

Figure 2: Daemon generation and benchmark execution

```

1  _id: ObjectId('63838642938892d328a60734')
2  uid: "2bb28f62-f99e-4603-932b-aa4385ed19cb"
3  address: "10.36.54.195"
4  hostname: "dolap"
5  date: "27-11-2022"
6  > twin_description: Object
7    influxdb_name: "dolap_main"
8    influxdb_tag: "_monitor"
9    monitor_pid: 982
10   prob_file: "probing_dolap.json"
11   roofline_dashboard: "to be added"
12   monitoring_dashboard: "http://host.docker.internal/d/wa83tmD4z/dolap-monitor_1"
13 > twin_state: Object

```

ObjectId
String
String
String
String
Object
String
String
Int32
String
String
String
Object

Figure 3: Digital twin description (DTD) of Dolap server

```

> dtmi:dt:dolap:thread6;1: Object
> dtmi:dt:dolap:thread50;1: Object
> dtmi:dt:dolap:core7;1: Object
  @type: "Interface"
  @id: "dtmi:dt:dolap:core7;1"
  @context: "dtmi:dtdl:context;2"
  > contents: Array
    > 0: Object
      @type: "Relationship"
      @id: "dtmi:dt:dolap:core7:contains23;1"
      name: "contains23"
      target: "dtmi:dt:dolap:thread7;1"
    > 1: Object
      @type: "Relationship"
      @id: "dtmi:dt:dolap:core7:contains24;1"
      name: "contains24"
      target: "dtmi:dt:dolap:thread51;1"
      displayName: "core7"
> dtmi:dt:dolap:thread7;1: Object
> dtmi:dt:dolap:thread51;1: Object
  @type: "Interface"
  @id: "dtmi:dt:dolap:thread51;1"
  @context: "dtmi:dtdl:context;2"
  > contents: Array
    > 0: Object
    > 1: Object
    > 2: Object
    > 3: Object

```

Object
Object
Object
String
String
String
Array
Object
String
String
String
String
Object
Object
String
String
String
Array
Object
Object
Object
Object
Object

Figure 4: A frame of core and thread DTD of the Dolap

Here note that the monitoring machine (local) is a remote system which connects remotely to the monitored (remote) HPC cluster to perform probing and benchmarks. To do that, the tool uses SSH utilities of Python. After the probing and benchmarks are done, the tool pulls the created DTD to the local machine and stores it as an entry on a specifically created local MongoDB database and a collection in it. [1]

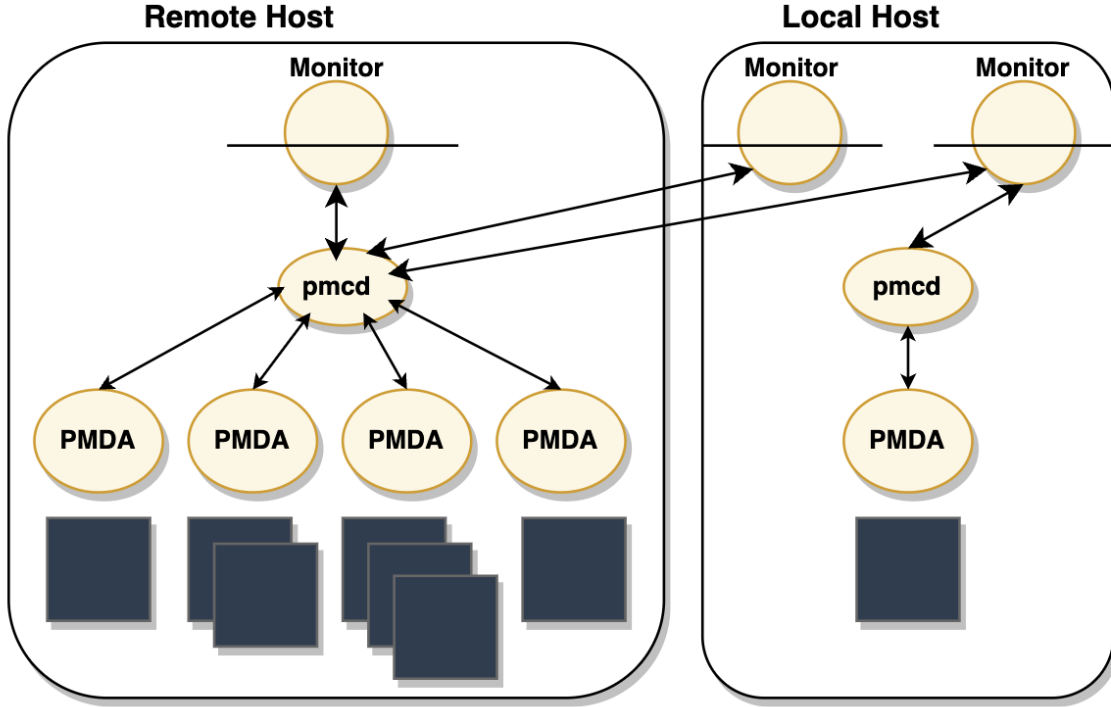


Figure 5: [PCP] Remote Collector

The Data Gathering module mainly collects data by employing the Performance Co-Pilot (PCP) tool's Performance Metrics Domain Agents (PMDA) which are utilized by daemons which are named as Performance Metrics Collection Daemons (PMCD). PMDAs that are used in the tool are as follows: perfevent PMDA, proc PMDA, linux PMDA and lmsensors PMDA. Note that there are available 75 PMDAs currently therefore, more PMDA can be added to this stack for increasing the monitoring complexity. [3]

Besides the integrated data gathering tools, ComDetective was one of the tools that were aimed to be integrated into the tool's technology stack for further data acquisition capability. ComDetective works by sampling selected cache access events by PMUs(Performance Monitoring Unit) to create a matrix that shows the inter-thread communications.[SCAU19] When a thread accesses another thread's cache, the corresponding matrix entry gets incremented. The matrix can then be visualized for observers to visually understand the communications and used as an input for optimization programs. When threads access each other's cache, a slowdown happens. Albeit it is expected that threads inevitably access each other's cache, minimizing this event will allow speed-ups. The project aimed to directly add the ComDetective tool into the digital twin monitoring tool. ComDetective is built on HPC Toolkit.

HPC Toolkit is a tool collection that allows comprehensive analysis and monitoring techniques to be used on an HPC structure[ABF⁺10]. The ComDetective paper was investigated to understand the reasoning and the algorithm behind the data collection technique to figure out whether the tool could be directly integrated. It was realized that adding the HPC Toolkit to the monitoring tool in order to build the ComDetective would create unacceptable overhead with regard to the slow-down

range set by the project. Furthermore, many of the functionalities that are used by the ComDetective through HPC Toolkit are already available on the monitoring tool. Thus, instead of integrating ComDetective, a program that replicates its functionality will be written. To this end, there has been ongoing communication between the ComDetective developers and our team to aid in understanding the technology and come up with a way to reconstruct it.

Another tool planned to integrate into the SuperTwin is Reuse Tracker [SCMU21]. It focuses on reuse distance, a widely used metric that measures data locality. Data locality is a significant point for HPC clusters since the capacity of high-speed links between separate storages is less than the bandwidth of the compute nodes [GFZ12]. Thus, better data locality increases computation power and efficiency in HPC clusters. Reuse Tracker leverages debug registers and hardware counters of performance monitoring units, which are already focus points in this project. The integration plan consists of integrating the existing Reuse Tracker into the Super Twin to monitor specific computing jobs and measure data locality in HPC clusters.

Attributes	[BH05]	[WLC19]	ReuseTracker
Time Overhead	1.4x<	2.03x	2.9x
Space Overhead	-	2.8x	2.8x
Accuracy	-	90%	92%
Method	PMU	PMU	PMU
Open Source	-	YES	YES

Table 1: Available Reuse Distance monitoring tools

Among other available reuse distance analysis tools [BH05], [WLC19], Reuse Tracker is a lightweight and valuable monitoring tool for the Super Twin with only 2.9X time overhead and 2.8X space overhead. However, it is also built on HPC Toolkit as ComDetective and comes with many software dependencies and unnecessary functionalities such as source code line attribution. Source code line attribution is another valuable information we do not need in Super Twin. It gives a code line from the monitored script to investigate the job more deeply. Since we aim to make Super Twin an accessible, lightweight, and fast monitoring tool, we decided not to include HPC Toolkit dependence in our monitoring daemons. Instead, it makes more sense only to gather dependencies that calculate reuse distance. To do that, we only need to configure debug registers and hardware counters of performance monitoring units to collect the reuse distance of specific runs in HPC clusters.

Since the aim is to integrate Reuse Tracker without HPC Toolkit, it comes with challenges. Extracting the tracker from the HPC Toolkit codebase is under development. Since Reuse Tracker has two central parts of calculating use distances with PMUs and trapping reuses with debug registers, development is divided into two parts accordingly. Our team is currently working on extracting the use distances of running jobs, which is an essential part of Reuse Tracker.

Currently, use distances can be extracted from the given instructions [12]. `PERF_SAMPLE_IP`, the sample instruction point monitored by the tool, and `PERF_SAMPLE_ADDR`, the address of the data point, will be used to calculate reuse distances with debug registers. The next step is to implement debug registers and integrate current scripts into the Super Twin’s daemons.

In order to start the Data Gathering module, one must specify the metrics that will be monitored among all possible monitoring metrics that are extracted by probing. After specifying the metrics and starting monitoring, the data which is collected by PMCDs is firstly transferred to the local machine from the remote HPC cluster and then stored in an InfluxDB database in the form of time-series data.

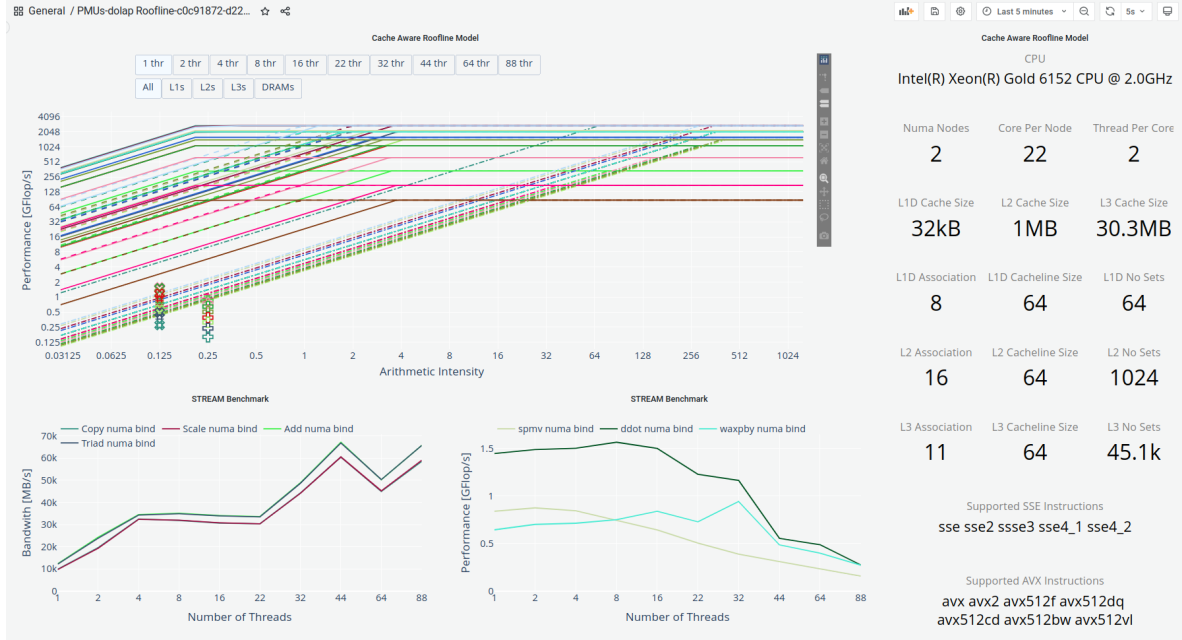


Figure 6: Dolap's Roofline Dashboard

Visualization module, via querying the time-series database, retrieves the data and displays each monitored metric in real-time using Grafana Dashboards. Note that, as adding more monitoring metrics is possible in case of need, adding Grafana figures is also viable too.[6][7]



Figure 7: Monitoring dashboard of the Dolap which is generated with some available metrics

The tool can be used as a console application by running the "main.py" in the source folder. However, in most cases, the tool finds more than 1000 monitoring metrics and for a user, it is hard to

identify and select metrics to monitor. Therefore for the tool, a web application is implemented with React.js and Flask to improve user experience. [8]

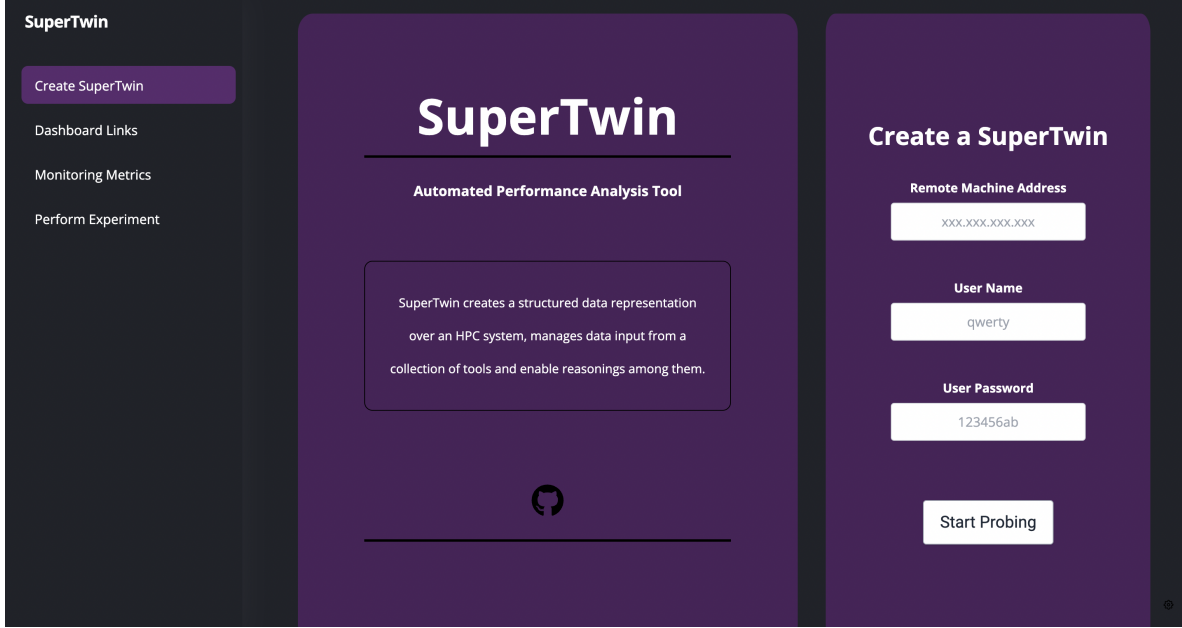


Figure 8: Index page of the web application of the tool

Web application offers users to specify a remote machine to probe, select monitoring metrics among the extracted metrics, trace the data gathering daemon, select experiment metrics and perform experiments and reach created monitoring dashboards. [9][10][11] Fundamentally, what web application does is creating a tool object instance via using user inputs from the front-end and presenting various outputs of the tool such as extracted monitoring metrics to a user. The web application is able to send commands to the remote machine to perform experiments (for example stressing a CPU with "sudo stress -cpu 8" command) and receive information from it. [11] Also, in order to make the tool and web application easy to deploy and use on different settings, it is containerized via using Docker.

3 Encountered Problems

Since the tool that is aimed to be developed in this project requires the integration of different methods and tools, throughout the development process The most exhaustive challenge while implementing the tool was dealing with the continuously increasing code complexity and the number of dependencies. As more part of the modules are implemented, it became harder to connect new parts with the old ones without changing the structure. In order to deal with this problem, the code is refactored to make it more object-oriented. Also, another blocking problem throughout the development process was the operating system compatibility issues that were caused by the tools that are employed. Most of the probing and monitoring daemons that are utilized in the tool only work on Linux operating system while as group members we had macOS and Windows operating systems. To deal with this issue, Docker containers which enables using the tool in different operating systems are created. Containerization of the tool also helps manage dependencies such as installing and initializing PMDAs.

For ComDetective, there have been efforts in understanding the paper and installing the tool on the twin. This proved challenging and time-consuming for the following reasons: Both the tools and the twin is built on Linux operating system and installing the HPC Toolkit to build the ComDetective turned out to be a complex process. Project team communicated both with itself and the team that

built the ComDetective to achieve progress. The paper itself also turned out to contain complex details beyond the scope of the integration task which took an effort to research before the realization. With the tests done after the installations and discussions with the ComDetective team, it was decided that the integration was not going to take place and ComDetective technology will be replicated for the requirements and structure of the twin instead. Arriving at the final decision of reconstructing the technology took 8 weeks. In the remaining weeks, the tool’s full functionality will be added to the twin. The cache listening function of the tool will take 4 weeks, matrix creation and visualization will take one week each. There are no deviations from the schedule.

The most challenging part of the Reuse Tracker is extracting the exact tool from the large codebase, HPC Toolkit. The developer of Reuse Tracker directly integrates their tool to the top of the existing HPC Toolkit structure. Since we do not want to add more dependence to our monitoring daemon, the team is struggling with removing this dependence. In addition, the codebase has low-level instructions for hardware registers, which made us learn new low-level Linux APIs and hardware-level APIs. Since those APIs mainly depend on the hardware version, they come with different problems in detecting and understanding the tool. And unfortunately, the issues that we encountered were primarily unique to the tool, and there were not many available resources to get help. With the help of Reuse Tracker developers, our team extracted the tool from the codebase and got helpful sample scripts about Reuse Tracker itself.

4 Tasks To Be Completed Until Final Report

4.1 Slurm Data Extraction

Currently, the tool is able to create DTD and monitor only one machine. In order to scale up the monitoring to the cluster degree, it is required to develop an approach that the tool can manage multiple digital twins and gather their data. In order to do that in an HPC cluster, the monitoring system can be scaled via using Slurm Workload Manager. The main aim of this task is creating a digital twin monitoring instance on every single node of the cluster and an inclusive digital twin monitoring instance for the head node to assemble the cluster together.

4.2 ComDetective Reconstruction

Signal processing is required to sample cache events by using PMUs for given configurations in order to detect inter-thread communications without creating significant overhead. This is the most crucial part of the task. The retrieved samples will be processed and added to a communication matrix that holds the number of inter-thread cache accesses. The matrix will be visualized like a heat map to allow observers to easily analyze communications.

4.3 Reuse Tracker Integration

Currently, our tool is able to extract use distances from the given instructions. Hereafter, it is necessary to configure the current refactored reuse tracker to monitor external jobs. In addition to that, we need to learn and implement the trap to detect reuses in debug registers. Finally, it is required to configure signal handlers to get the file descriptor provided by hardware counters to store use distances and reuse distances to a local file. Our analysis tool will use this file, and reuse distances will be shown as a new metric in the Super Twin.

4.4 Machine Learning and Anomaly Detection

Our team is focused on a metric gathering that will feature machine learning features. After that, the pipeline will be developed to detect anomalies and predict the performances of HPC clusters. For an accurate pipeline, we must apply feature selection and/or dimension reduction to our data. Since gathered metrics are various, this step is crucial for our machine learning model. In addition, the data we are dealing with is time series data with an extensive amount of feature count, and there is no labeled data. Thus, our team will consider active-learning, human-in-the-loop setting, and

self-supervised algorithms to leverage existing monitoring tools with state-of-the-art machine learning model.



5 Appendix

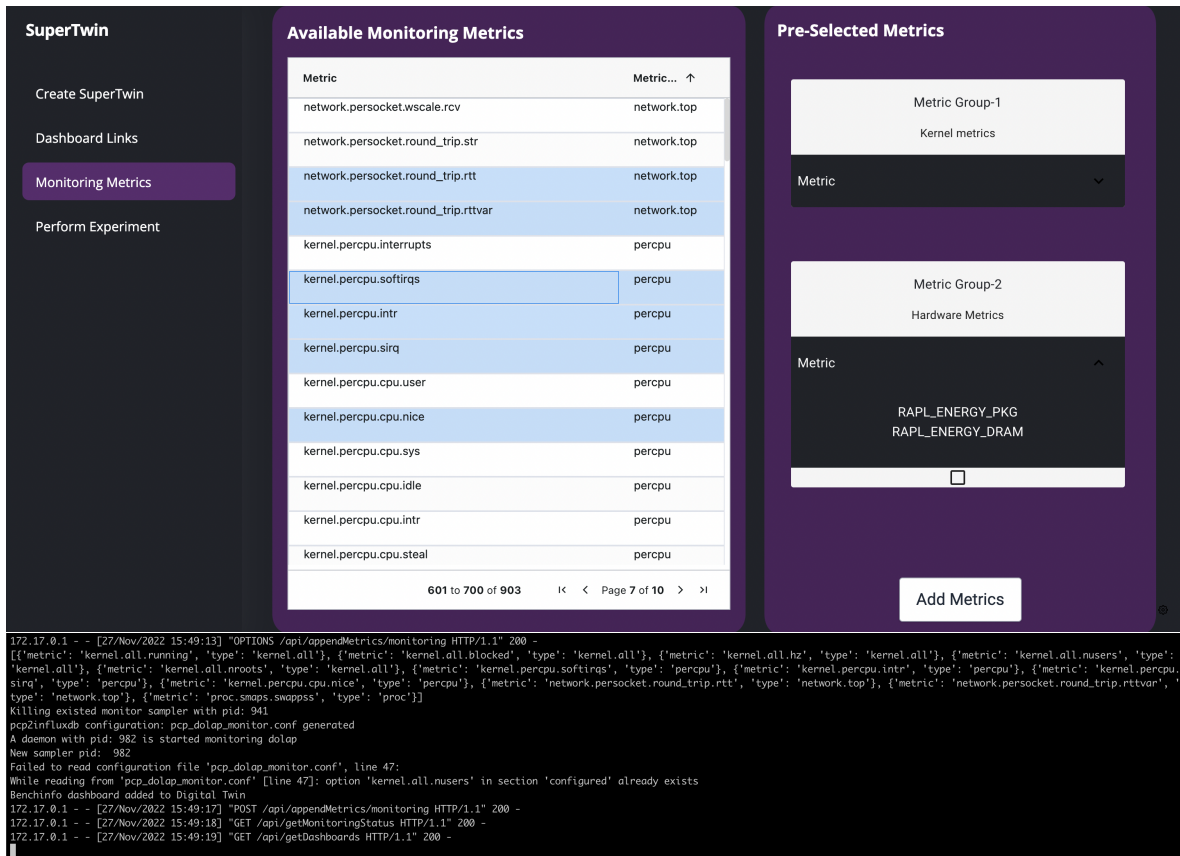


Figure 9: Monitoring metric selection and dashboard generation on web application

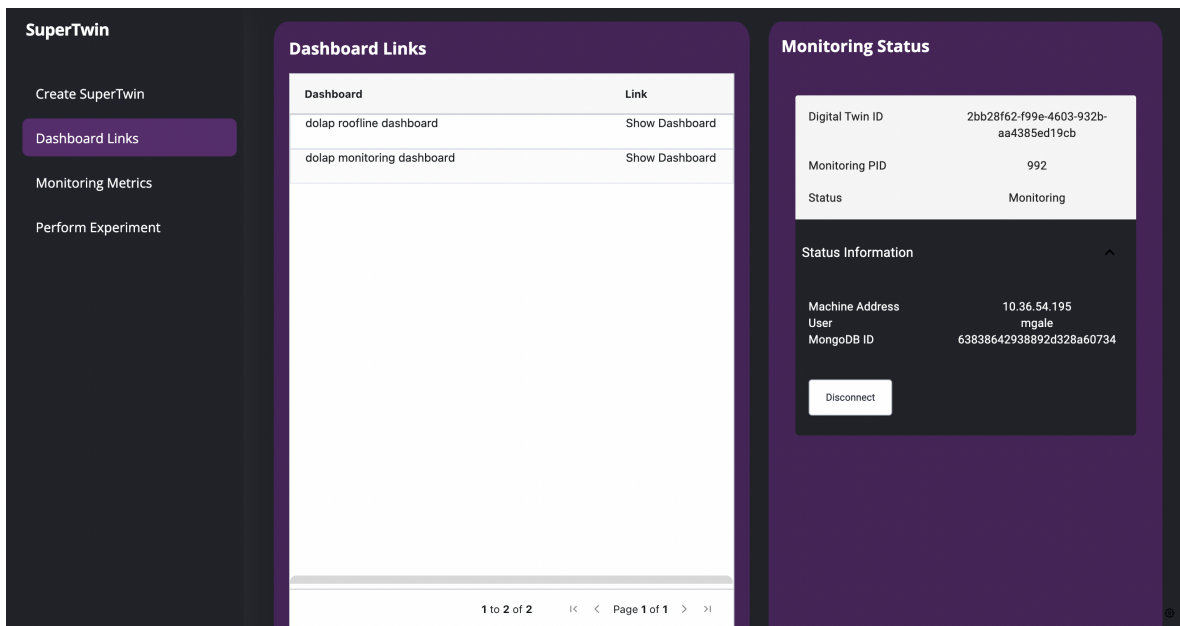


Figure 10: Dashboard links and monitoring status screen

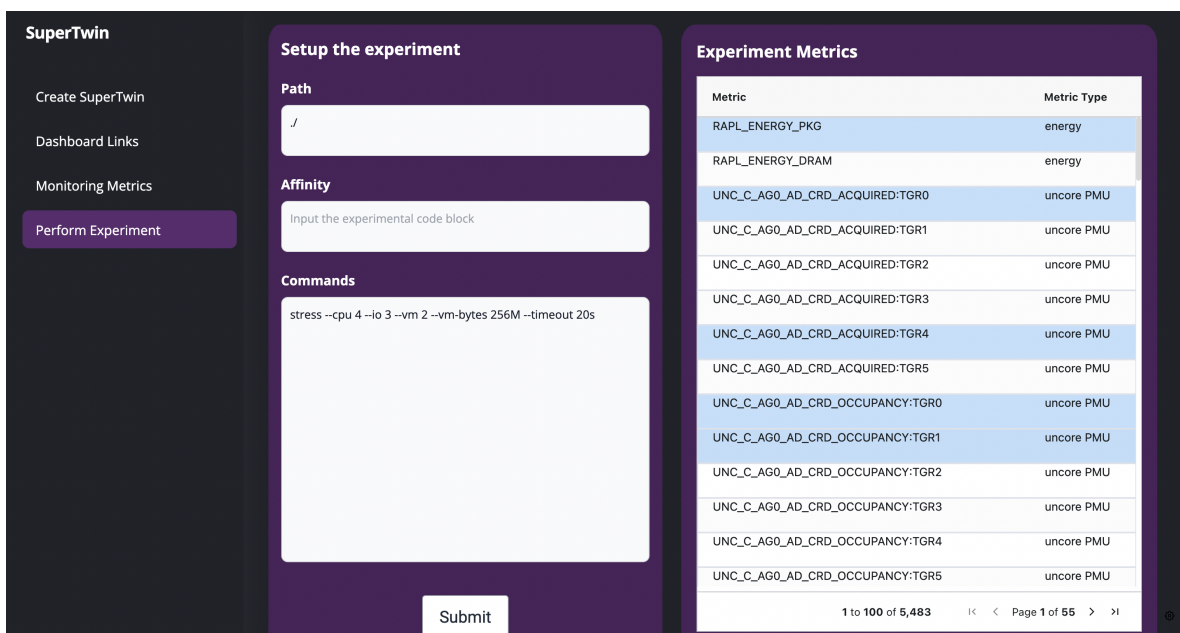


Figure 11: Perform experiment screen

```

PERF_RECORD_SAMPLE [4009], MISC=16386 (PERF_RECORD_MISC_USER,PERF_RECORD_MISC_EXACT_IP ), Size=32
  PERF_SAMPLE_IP, IP: 556bb983f8e8
  PERF_SAMPLE_ADDR, addr: 7fffc90247c8
  PERF_SAMPLE_WEIGHT, Weight: 0
this is perf_mmap_read
Head: 429984 Prev_head=429920
64 new bytes
PERF_RECORD_THROTTLE, MISC=0 (PERF_RECORD_MISC_CPUMODE_UNKNOWN), Size=32
  Time: 6935754261668994
  ID: 28024852
  Stream ID: 28024852
PERF_RECORD_SAMPLE [4009], MISC=16386 (PERF_RECORD_MISC_USER,PERF_RECORD_MISC_EXACT_IP ), Size=32
  PERF_SAMPLE_IP, IP: 556bb983f8e8
  PERF_SAMPLE_ADDR, addr: 7fffc90247c8
  PERF_SAMPLE_WEIGHT, Weight: 0

```

Figure 12: Hardware Counters of Performance Monitoring Units for Use Distances

References

- [ABF⁺10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: Tools for performance analysis of optimized parallel programs <http://hpctoolkit.org>. *Concurr. Comput.: Pract. Exper.*, 22(6):685–701, apr 2010.
- [BG09] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. Association for Computing Machinery.
- [BH05] Erik Berg and Erik Hagersten. Fast data-locality profiling of native execution. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, page 169–180, New York, NY, USA, 2005. Association for Computing Machinery.
- [Che17] Yubao Chen. Integrated and intelligent manufacturing: Perspectives and enablers. *Engineering*, 3(5):588–595, 2017.

- [DWKN20] Robert Dietrich, Frank Winkler, Andreas Knüpfer, and Wolfgang Nagel. Pika: Center-wide and job-aware cluster monitoring. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 424–432, 2020.
- [GFZ12] Zhenhua Guo, Geoffrey Fox, and Mo Zhou. Investigation of data locality in mapreduce. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 419–426. IEEE, 2012.
- [HBK05] Ben Huang, Michael Bauer, and Michael Katchabaw. Network performance in distributed hpc clusters. volume 2, pages 546–549, 01 2005.
- [IRM⁺20] Mihailo Isakov, Eliakin del Rosario, Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert B. Ross, and Michel A. Kinsy. Hpc i/o throughput bottleneck analysis with explainable local models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’20. IEEE Press, 2020.
- [KMM20] Maninder Jeet Kaur, Ved P. Mishra, and Piyush Maheshwari. *The Convergence of Digital Twin, IoT, and Machine Learning: Transforming Data into Action*, pages 3–17. Springer International Publishing, Cham, 2020.
- [KSV⁺21] Awais Khan, Hyogi Sim, Sudharshan S. Vazhkudai, Ali R. Butt, and Youngjae Kim. An analysis of system balance and architectural trends based on top500 supercomputers. In *The International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2021*, page 11–22, New York, NY, USA, 2021. Association for Computing Machinery.
- [PCP] Programming Performance Co-Pilot process structure for distributed operation. <https://pcp.readthedocs.io/en/5.2.3/PG/ProgrammingPcp.html>. Accessed: 2022-11-27.
- [PPG20] Ivy Peng, Roger Pearce, and Maya Gokhale. On the memory underutilization: Exploring disaggregated memory on hpc systems. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 183–190, 2020.
- [SCAU19] Muhammad Aditya Sasongko, Milind Chabbi, Palwisha Akhtar, and Didem Unat. Comdetective: A lightweight communication detection tool for threads. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’19, New York, NY, USA, 2019. Association for Computing Machinery.
- [SCMU21] Muhammad Aditya Sasongko, Milind Chabbi, Mandana Bagheri Marzijarani, and Didem Unat. Reusetracker: Fast yet accurate multicore reuse distance analyzer. *ACM Transactions on Architecture and Code Optimization (TACO)*, 19(1):1–25, 2021.
- [TRQR21] Erfan Bank Tavakoli, Michael Riera, Masudul Hassan Quraishi, and Fengbo Ren. Fspgmm: An opencl-based hpc framework for accelerating general sparse matrix-matrix multiplication on fpgas, 2021.
- [WLC19] Qingsen Wang, Xu Liu, and Milind Chabbi. Featherlight reuse-distance measurement. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 440–453, 2019.