Efficiency Database and Design Guide

Deliverable No:	D4.4
Deliverable Title:	Efficiency Database and Design Guide
Deliverable Publish Date:	13 October 2023
Project Title:	SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID:	H2020-JTI-EuroHPC-2019-1
Project No:	956213
Project Duration:	36 months
Project Start Date:	1 April 2021
Contact:	sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS*	Graphcore	Norway

*until M21

CONTENTS

1 Introduction 1
1.1 Objectives of this Deliverable 1
1.2 Deviations and Counter Measures 1
1.3 Resources 1

2 Review of SuperTwin and Recent Changes 3

3 Design Towards Performance Database 7
3.1 BenchmarkInterface 7
3.2 ObservationInterface 8
3.3 Abstraction Layer 9

4 Performance Database 10
4.1 Contributions by Each Partner 12
4.2 Deviations (if Any) 12
4.3 History of Changes 14

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme-Scale Computing and Data-Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High-Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time, it is challenging to achieve high performance when performing sparse computations. SparCity delivers a coherent collection of innovative algorithms and tools for enabling high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications.
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

The objective of this deliverable is to provide a technical overview of the research activities carried out..

1.2 DEVIATIONS AND COUNTER MEASURES

1.3 RESOURCES

The following is the list of websites that host software and data repositories that have so far been developed for the SparCity framework. It is expected that these websites will later receive a more coherent organization.

- Deliverables that are public are available in the project website

  http://sparcity.eu

- Source code developed in this project is available at the project github repository https://github.com/sparcityeu
- A repository of sparse problem instances https://datasets.simula.no/sparcity

2 REVIEW OF SUPERTWIN AND RECENT CHANGES

SUPERTWIN relies on a comprehensive Knowledge Base (KB, previously named SuperTwin Description or STD in earlier deliverables) and linked-data capabilities. SuperTwin design is refactored towards including a global SUPERFDB. In this deliverable, we will examine data structures and methods that allow the creation of a SUPERFDB that seamlessly keeps and interacts with the performance data collected from different hosts at different times and settings. SuperTwin is implemented using the parameter-object design pattern; the Knowledge Base is used by each SUPERTWIN function as a parameter, including all SUPERFDB functionalities. It is dynamic, evolving over time to capture and link additional telemetry and metadata as they become available. Moreover, the KB has modular classes that encapsulate the collected information from a time window. This allows the twin to continue its operations in a live fashion without procedural changes and comprehend the factors influencing system performance in real-time. An example KB is shown in Fig. 1.

SuperTwin's distinguishing capabilities lie in its ability to manage and utilize an extensive array of performance metrics, ensuring robust monitoring and analysis of system behaviour.

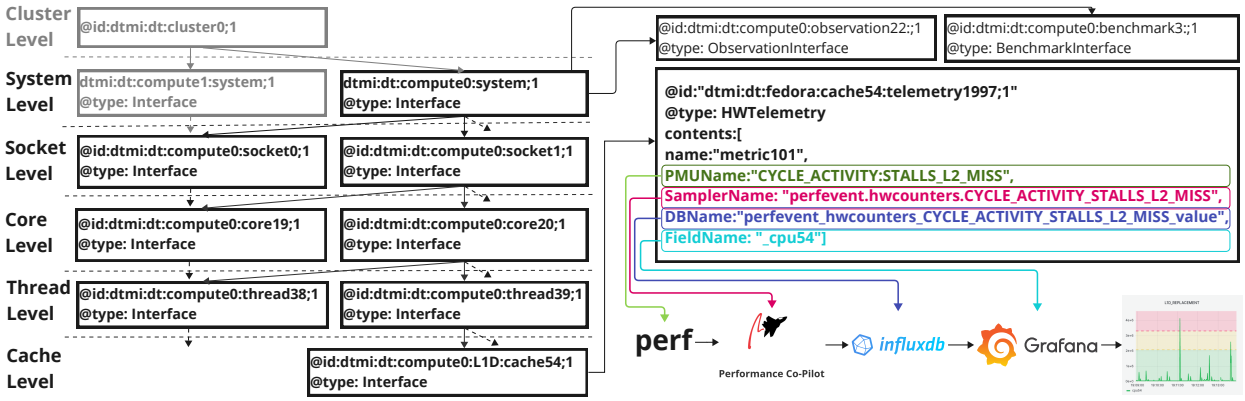


Figure 1 *Knowledge Base of* SUPERTWIN.

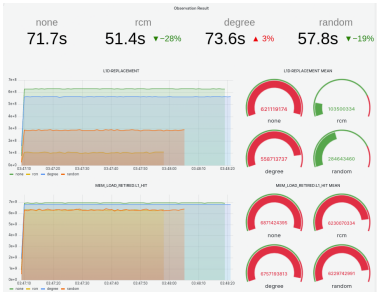
Capturing the target system and its component hierarchy, the KB can be parsed to acquire any information from topology to database parameters. There are two types of metrics to be sampled from an HPC system. The first type is *SWTelemetry*, i.e., software and system state-related metrics such as the number of processes, CPU, and memory load. A pre-selected set of these metrics is set to be *always sampled* with a low frequency, however, they are completely configurable and can be reconfigured by the user at any time. The second type is *HWTelemetry*, sampled from PMUs during kernel executions with high frequency. Sampling different metrics with varying frequencies yields a need for metadata associated with the host system's metadata. While time-series databases are tailored for telemetry data, they cannot keep much (linked) metadata. On the contrary, managing time-series data via a document database is impractical.[1] For this reason, SUPERTWIN's KB uses two types of databases with links between them. To this end, while InfluxDB stores the sampled SWTelemetry and HWTelemetry, MongoDB stores the knowledge

[1] Friedemann. "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing". *MATEC Web of Conferences* 304 (2019), p. 04006. DOI: 10.1051/matecconf/201930404006; Katarina Milenković. "Enabling Knowledge Management in Complex Industrial Processes Using Semantic Web Technology". English. *Proceedings of the 2019 International Conference on Theory and Applications in the Knowledge Economy*. 2019 International Conference on Theory and Applications in the Knowledge Economy, TAKE 2019 ; Conference date: 03-07-2019 Through 05-01-2020. 2019. URL: https://www.take-conference2019.com/.

base as JSON-LD extended with entries for each computation. To associate the computations with telemetry, pointers to InfluxDB are used to recall corresponding metrics.

We consider the KB with a tree-structured organization and doing this enables fully automated performance monitoring, anomaly detection and dashboards with meticulously selected metrics, tailoring various *views*. These views, namely (a) *Focus View*, (b) *Level View*, and (c) *Subtree View*, allow for a dynamic and versatile performance data exploration. Multiple views enable fine- and coarse-grain investigations into the component and system performance. Overall, SuperTwin can visualize data from different components and systems in tandem allowing for comprehensive analysis and comparison, further enriched by the inclusion of various views using Grafana visualization tool.

(a) *Focus view for an individual cache.* ***Will be supported by*** SuPerfDB



(b) *Level view for processes* ***Supported by*** SuPerfDB

(c) *Level view for procs/sockets* ***Supported by*** SuPerfDB

(d) *Subtree view for a node.* ***Not supported by*** SuPerfDB

Figure 2 *Sample dashboards, automatically generated by* SuperTwin.

- **The focus** (i.e., component) view offers a dashboard that visualizes active metrics from a single component, e.g., a socket, core, thread, network, disk, or process, providing a focused lens on individual element performance. This view can be extended to focus on the path from the root (whole system) to the focused component to investigate the root cause of anomalous behaviors or performance drawbacks. That is the path navigating from a component perspective to a more generalized system perspective is analyzed, aiding in tracing and isolating performance issues. An example focus-view dashboard is given in Fig. 2(a) for an individual cache.

- **The level** (i.e., type) view generates a dashboard that visualizes multiple instances from the same type, such as a group of threads, disks and even processes. This view allows the *isolation* of a single type, which corresponds to a level in the KB tree, treating them individually or in comparison to components within the same or a different system, whereas the linked-data capabilities enable the automatic visualization of component performance across different machines. For instance, the level-view dashboards for different processes running SpMV (each with a different reordering of the same matrix) and on two sockets running the SpMV code with two different orderings on the same matrix are given in Fig. 2(b) and Fig. 2(c), respectively.

- **The subtree** (i.e., (sub)system) view seeks to *zoom* into performance events, starting from an arbitrary node and extending to all connected leaf nodes, moving from a general perspective to a more specific one, i.e., from a single socket to all cores/caches. The detail intensifies as the path moves from the root (subsystem) to the leaf (components at the bottom of the KB hierarchy), facilitating a deeper dive into specific performance events and data. An example subtree-view dashboard for a single server is given in Fig. 2(d).

KB Lifecycle: The knowledge base is not a static object. It captures more about the system it represents as time passes by attaching new entries. To initialize the KB, SuperTwin uses its *probing tool*. To comprehensively capture the structural details of a system, including component specifications, inter/intra-relationships, and their associated performance metrics, a detailed probing is required. SuperTwin targets each hardware component that can be monitored, produce metrics or affect the overall system performance. Furthermore, it captures their relationships in a lightweight and adaptable fashion. SuperTwin's probing relies on widely available Linux tools to gather data. The system, network, and memory information are collected via `lshw`. The CPU, memory/cache topology metadata are collected by parsing `likwid-topology` from `likwid` tools[2] and `cpuid` instruction. When available, disk info is probed from `/sys/block/*/device` and `SMART`[3] utility. PMU information is collected with `libpfm4` library, which can recognize model-specific registers and their events of virtually every x86 and ARM processor available on the market. Upon probing available PMU metrics via `libpfm4`, and software telemetry via PCP, are filtered and mapped with the components.

In the initial KB, every single component that performs computation, communication, or I/O is represented with an `Interface`. Furthermore, each relationship among these components is encoded into these interfaces with a `Relationship`. The available metrics for the components are filtered and encoded as SWTelemetry and HWTelemetry. This makes precisely pinned executions and automated queries possible.

[2]Thomas Röhl et al. "LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses". *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 781–784. DOI: 10.1109/CLUSTER.2017.115.

[3]The Smartmontools Team. *Smartmontools*. Accessed on 5th October 2023. URL: https://www.smartmontools.org/.

SuperTwin is designed to run on a *host* that can be different than the *target* system. The host runs the SuperTwin daemon as well as the tools with heavy workloads, e.g., InfluxDB, MongoDB, and Grafana. The target only runs the PCP samplers and reports telemetry to the host when requested. In Figure 3, step ⓪ reads the environment variables such as the IP addresses of InfluxDB and MongoDB instances and Grafana token to the SuperTwin daemon. In step ①, the probing module is copied to the target system to generate a JSON file containing the system information which, in ②, is copied back to the host to generate the KB. The information collected from all the tools, components, and third-party tools SuperTwin manages is fused for KB generation. Once the KB is generated, it is inserted into MongoDB in step ③. Step ③ re-occurs every time KB changes or SuperTwin is restarted. When this phase is completed, the framework becomes fully functional using only this data structure.

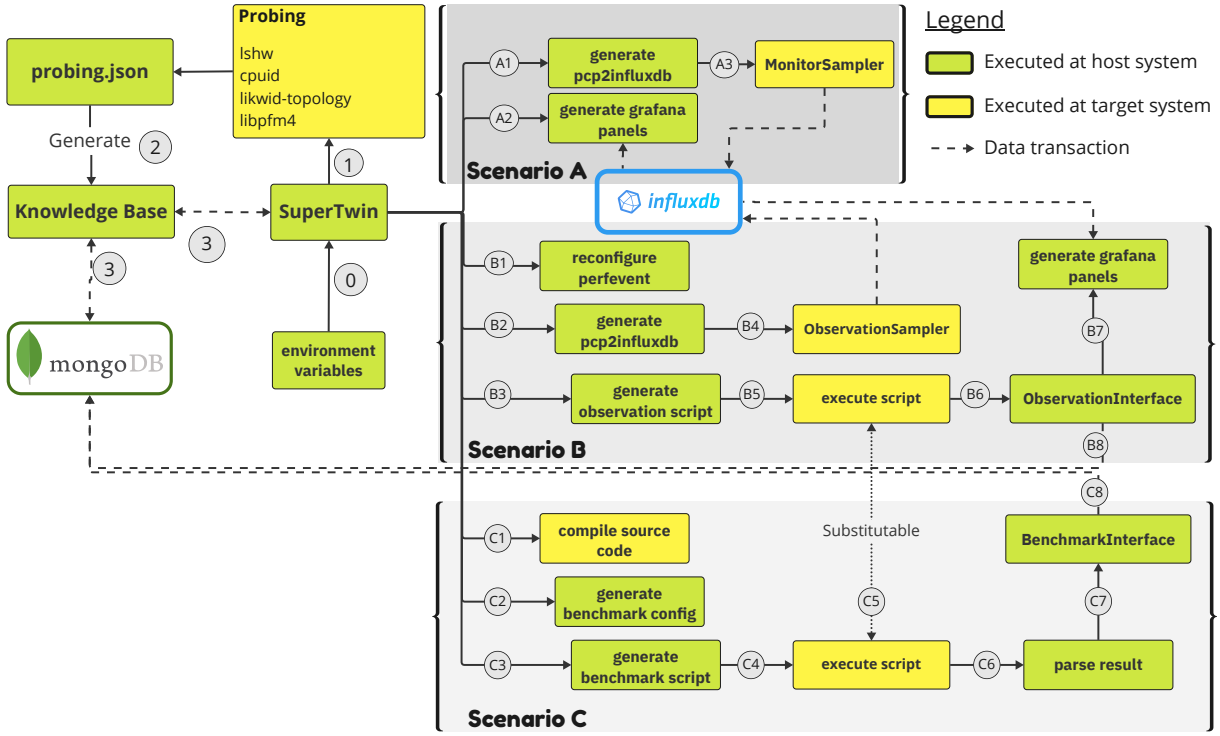


Figure 3 *Three scenarios within the* SuperTwin *framework*

In Figure 3, t SuperTwin scenarios are shown; the first is sampling software emitted metrics to monitor system state (Scenario A), and the other is capturing the hardware performance events during kernel execution. In step (A1), using KB, SuperTwin configures the PCP collectors and samples system-related metrics, such as CPU and memory usage, NUMA-related events, and energy spent. In (A3), a sampler on the target is requested for this telemetry. Since the query parameters are already encoded in KB, steps (A1) and (A2) can happen at the same time. That is the dashboards are already generated on the host when the target starts reporting.

In Scenario B, SuperTwin samples hardware events reported from the PMUs. In this case, it focuses on an execution on the target and the components on which the execution takes place. Therefore, SuperTwin requests an executable and its command-line parameters. Once these are provided, the PMUs are configured to report the requested metrics in step (B1). That is, SuperTwin configures the sampler in the same way as step (A1). After the PMUs are configured, it generates a script to run the requested kernel on the target system. This script bounds the threads

to the cores using one of the *balanced*, *compact*, *numa balanced*, *numa compact* strategies based on the probed target system topology. Then it samples performance events, executes the script to run a kernel on a target and stops the sampling as the kernel is halted. An `ObservationInterface` is generated to encode the execution metadata, collected metrics and the unique observation ID associated with the time-series data in InfluxDB. In step (B8), the `ObservationInterface` is appended to the system's KB. This `ObservationInterface` entry is later used to recall the performance data for visualization or analysis purposes. For example, using the generated KB and run configuration module, an execution that will run on 4 threads on each socket that does not share an L1 cache could be launched; similarly, after the execution of a run, performance metrics from threads that share same L2 cache with a given thread could be queried.

In Scenario C, SuperTwin automatically executes a selected benchmark with selected parameters at the target system. Using the system info that is encoded within KB, SuperTwin first copies the source code of a selected benchmark to the target system at step (C1). Then, based on the information in KB, SuperTwin first compiles the benchmarks on the target system using the benchmark's preferred compiler if it exists, e.g., `icc` or `gcc`. At step (C2), the configuration files the benchmark requires are generated by using the system information (cache sizes, CPU frequency) or parameters (number of threads, problem size) set. At step (C3), a bash script to execute the benchmark on the target system is generated. By default, the generated scripts cover a set of threads ranging from 1 to the number of threads as powers of 2, placed either on a single NUMA node or distributed evenly across NUMA nodes. Additionally, threads are pinned to specific cores and particular sockets. At this point, the user can select to sample PMU and system telemetry as this script is an observation script. After the execution of the benchmark, SuperTwin parses the results and creates a `BenchmarkInterface` with the corresponding `BenchmarkResult` at step (C7). Then similar to step (B8) the newly generated `BenchmarkInterface` is appended to the KB. If optional performance event sampling is selected at step (C5), an `ObservationInterface` in accordance with generated `BenchmarkInterface` is appended to the KB. The aim is to assist architecture research by determining performance events affected by system specifications such as cache architecture, topology and memory bandwidths.

3 DESIGN TOWARDS PERFORMANCE DATABASE

To keep the KB dynamic and continuously link the system components to performance data, SuperTwin uses `Interfaces` and attaches their instances (i.e., entries) to KB. For instance, as mentioned above, processes are monitored via per-process kernel metrics. JSON-LD interfaces are serialized with given parameters into a run-time object.

3.1 BENCHMARKINTERFACE

`BenchmarkInterface`, and `BenchmarkResult` as a helper class, are designed to record benchmark results. SuperTwin is able to perform *Cache Aware Roofline Model* (CARM), *STREAM*[4] and *High Performance Conjugate Gradient*[5] (HPCG) benchmarks homogeneously using the `BenchmarkInterface`.

`BenchmarkInterface` entries are designed to be able to access benchmark results in a structured manner. Being appended to the KB -along with an associated `ObservationInterface`- provides the ability to analyze the relation between the benchmark results and the system speci-

[4]John McCalpin. "Memory bandwidth and machine balance in high performance computers". *IEEE Technical Committee on Computer Architecture Newsletter* (1995), pp. 19–25.

[5]Jack Dongarra and Michael A Heroux. "Toward a new metric for ranking high performance computing systems". *Sandia Report, SAND2013-4744* 312 (2013), p. 150.

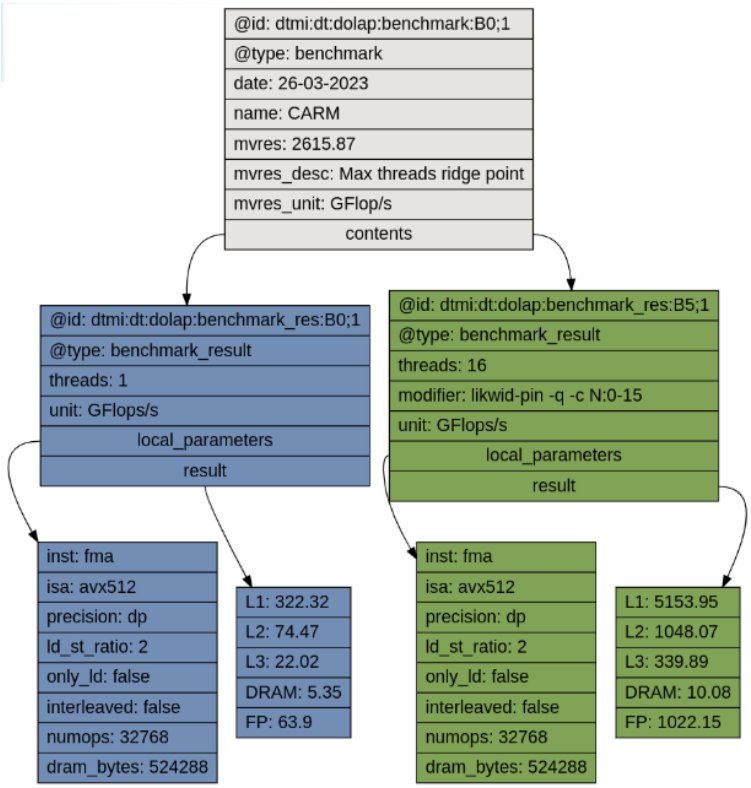


Figure 4 *An example* `BenchmarkInterface` *entry encoding two different settings for CARM benchmark. This entry is later used to generate performance models.*

fication. Moreover, any modifier, such as pinning and NUMA placement or first touch policies, are encoded together with the results. This design is implemented to enable smart placement of executions to the target systems via machine learning models later.

```
{
  "@type": "ObservationInterface",
  "@id": "278e26c2-3fd3-45e4-862b-5646dc9e7aa0",
  "displayName": "rcm_rma10_mt",
  "time": 48.667,
  "command": "./spmv -f rma10.mtx -o rcm -t 4",
  "modifier": "likwid-pin -q -c S0:0-1@S1:0-1",
  "no_threads": 4,
  "involved_threads": [0,1,22,23],
  "sampled_sw_metrics": ["kernel.percpu.cpu.idle", "mem.numa.alloc.hit", "mem.numa.alloc.
      miss"],
  "sampled_hw_metrics": ["RAPL_ENERGY_PKG", "INSTRUCTION_RETIRED", "FP_ARITH:SCALAR_DOUBLE",
       "MEM_LOAD_RETIRED:L1_HIT"],
  "dashboard": "http://localhost:3000/d/-PiOFZEVz/pmus-278e26c2-3fd3-45e4-862b-5646dc9e7aa0?
      time=1681499308500&time.window=17000"
}
```

Listing 1: An example `ObservationInterface` entry which is used to retrieve sampled metrics. A report is generated on the fly and added to the entry before appending to KB.

3.2 OBSERVATIONINTERFACE

`ObservationInterface` entries encode sampled hardware performance events and system metrics, executed commands, generated affinity, time, and other relevant metadata. Using the encoded parameters in KB, queries can be generated to automatically retrieve data through these

entries. A basic `ObservationInterface` entry is shown in Listing 1. The queries automatically generated by SuperTwin to analyze the `BenchmarkEntry` in Listing 1 are given in Listing 2.

```
1 SELECT "_cpu0", "_cpu1", "_cpu22", "_cpu23" FROM "kernel_percpu_cpu_idle" WHERE tag="278
      e26c2-3fd3-45e4-862b-5646dc9e7aa0"
2 SELECT "_node0", "_node1" FROM "mem_numa_alloc_hit" WHERE tag="278e26c2-3fd3-45e4-862b-5646
      dc9e7aa0"
3 SELECT "_cpu0", "_cpu1", "_cpu22", "_cpu23" FROM "
      perfevent_hwcounters_fp_arith_scalar_double" WHERE tag="278e26c2-3fd3-45e4-862b-5646
      dc9e7aa0"
4 SELECT "_node0", "_node1" FROM "perfevent_hwcounters_RAPL_ENERGY_PKG" WHERE tag="278e26c2-3
      fd3-45e4-862b-5646dc9e7aa0"
```

Listing 2: Queries automatically generated by SuperTwin for the `ObservationInterface` entry given in Listing 1.

3.3 ABSTRACTION LAYER

To perform its actions and to effectively monitor PMU events on diverse target systems, each hosting CPUs across various vendors and micro-architectures, SuperTwin leverages an *Abstraction Layer*. The monitoring units and their reported events can significantly vary among different micro-architectures and from vendor to vendor. For instance, Intel has four general-purpose programmable counters/per-core to count performance events (eight if it is not shared with a second thread in the core), whereas AMD has two internal counters, one for each sampling flag. Intel provides 62 sub-events corresponding to 12 events, each accompanied by mask values. Similarly, AMD offers support for events similar to Intel. As an example, similarities and differences of events for Intel Cascade and AMD Zen3 are listed in Table 1. A detailed comparison between Intel and AMD PMUs can be found in.[6]

Event	**Intel Cascade**	**AMD Zen3**
Energy	RAPL_ENERGY_PKG RAPL_ENERGY_DRAM	RAPL_ENERGY_PKG RAPL_ENERGY_DRAM
Retired Inst.	INSTRUCTIONS_RETIRED	RETIRED_INSTRUCTIONS
Tot. Mem. Op.	MEM_INST_RETIRED:ALL_LOADS + MEM_INST_RETIRED:ALL_STORES	LS_DISPATCH:STORE_DISPATCH+ LS_DISPATCH:LD_DISPATCH
L3 Hit	**Not Supported**	LONGEST_LAT_CACHE:MISS + LONGEST_LAT_CACHE:RETIRED

Table 1 *Intel vs. AMD PMU events: the same, similar, different, and exclusive names for the same generic event, respectively.*

To facilitate the monitoring of PMU events in a platform-agnostic manner, an abstraction layer is implemented for SuperTwin. This layer effectively maps generic event names to concealed hardware-specific PMU event names, enhancing the system's versatility and ease of use. We have established a set of common events, such as L1_CACHE_DATA_MISS, FP_DIV_RETIRED, and RAPL_ENERGY_PKG, that are *assumed to be* supported by all the commodity CPUs. The rest of the events are left to the user's discretion. For further flexibility and scalability, SuperTwin utilizes configuration files to establish a straightforward mapping of common events to corresponding hardware events. The structure of a configuration file is as follows:

[6] Muhammad Aditya Sasongko et al. "Precise Event Sampling on AMD Versus Intel: Quantitative and Qualitative Comparison". *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1594–1608. ISSN: 1558-2183. DOI: 10.1109/TPDS.2023.3257105.

```
[pmu_name | alias]
<generic_event>:<hardware_event_1> [op]
[op] : ((+|-|*|/)(<hw_event> | <const>)) [op]
```

Following the pattern delineated, it is possible to generate a configuration file for "any" hardware by specifying the events intended for monitoring. Upon registering the desired configuration files within SuperTwin, the application proceeds to configure the PCP of the target system using the registered configuration files when needed. Additionally, users can access event information in a CPU agnostic manner within the program using `pmu_util.get(...)` method. As an example;

```
>pmu_utils.get(HW_PMU_NAME, COMMON_EVENT_NAME)
>pmu_utils.get("skl", "TOTAL_MEMORY_OPERATIONS")
>[
  "MEM_INST_RETIRED:ALL_LOADS",
  "+",
  "MEM_INST_RETIRED:ALL_STORES"
 ]
```

Although this example belongs to a recent Intel CPU, SuperTwin's configuration mapping via its abstraction layer offers versatility. Users can create mapping files for a wide range of CPUs, including Intel, AMD, PowerPC, ARM, and others, as long as they are supported by the *libpfm4* library which is the core library that enables PCP to monitor PMU events in CPUs. As SuperTwin configures PCP on the target, it creates empty and *zero-overhead* dashboards on Grafana, which are simply JSON files. Last, but not least, the abstraction layer seamlessly generates the formulas for the events the user is interested in. This changes from vendor to vendor as well as for every architecture even when the events are the same. An abstraction layer is necessary in modern tools to handle this diversity for performance profiling.

4 PERFORMANCE DATABASE

For long-term data management, thanks to its modular design and presented data structures, SuperTwin operates a *global performance database*, SuPerfDB. Unlike local instances, SuPerfDB employs cloud instances of MongoDB and InfluxDB. With a global performance database, SuperTwin aims to accumulate performance metrics from a wide array of systems to enhance architectural research and train robust machine learning models, particularly leveraging Large Language Models (LLMs), which can exploit the rich metadata collected to be trained as an assistant for performance engineering. The users of SuperTwin have the option to report their performance telemetry readings and the system's knowledge base to the performance database, alongside their local instances.

In SuPerfDB, the `ObservationInterface` of SuperTwin evolves into two versions within the performance database context: `TSObservationInterface` and `AGGObservationInterface`, where the latter statistically summarizes data using various aggregations, e.g., min, max, mean, to manage high data volumes. The users require a local SuperTwin instance to access SuPerfDB, visualize performance data, and automatically generate dashboards and reports. Without SuperTwin, they can only download selected data for ML training. Future adaptations may include appending source code and binary executables to the collected metadata, facilitating the training of models that can optimize code and predict performance and potential inefficiencies.

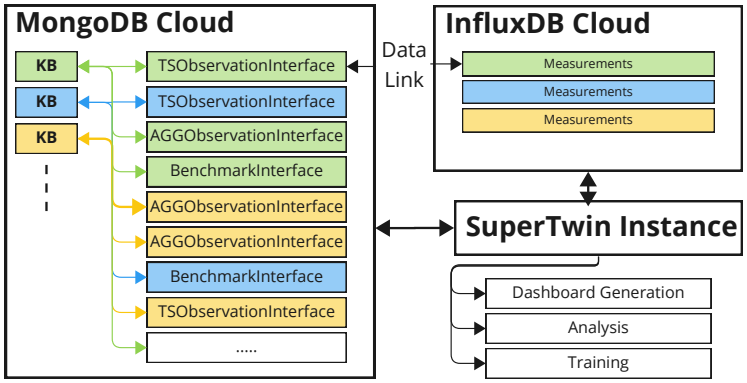


Figure 5 *Overview of* SuPerfDB

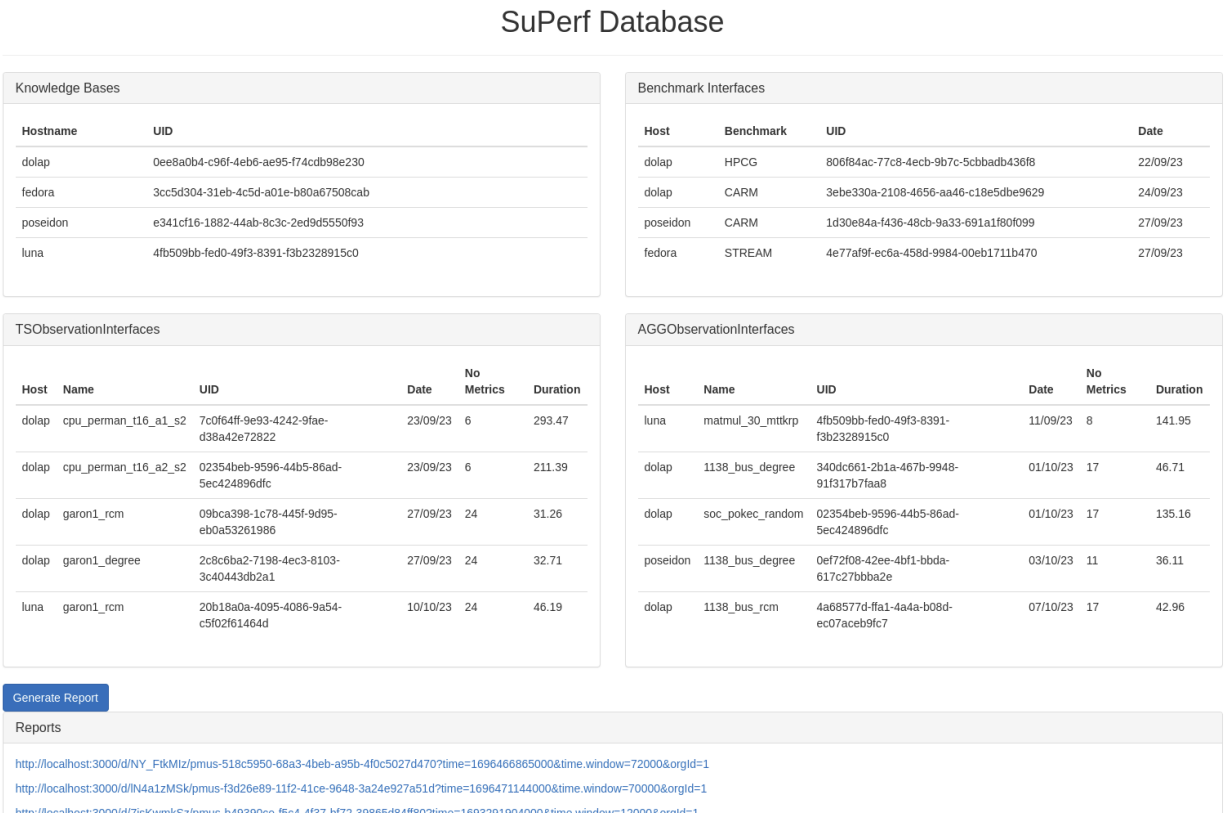


Figure 6 *Example usage of* SuPerfDB *web application: Knowledge bases and other classes are grouped together. When entries are selected within the same panel, a Level view dashboard is generated. A comparison of mutual HWTelemetry and SWTelemetry is shown in the dashboard, for the HWTelemetry, the abstraction layer for the comparison of equivalent metrics is leveraged. Tailored visualizations for* `BenchmarkInterface` *entries and Focus view from different hosts are currently under development.*

4.1 CONTRIBUTIONS BY EACH PARTNER

SU, KU, and INEC contributed to the design of the performance database. The detailed design is a product of equal contributions by three partners. SU implemented the main components of the database.

4.2 DEVIATIONS (IF ANY)

There are no deviations.

REFERENCES

Dongarra, Jack and Michael A Heroux. "Toward a new metric for ranking high performance computing systems". *Sandia Report, SAND2013-4744* 312 (2013), p. 150.

Friedemann. "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing". *MATEC Web of Conferences* 304 (2019), p. 04006. DOI: 10.1051/matecconf/201930404006.

McCalpin, John. "Memory bandwidth and machine balance in high performance computers". *IEEE Technical Committee on Computer Architecture Newsletter* (1995), pp. 19–25.

Milenković, Katarina. "Enabling Knowledge Management in Complex Industrial Processes Using Semantic Web Technology". English. *Proceedings of the 2019 International Conference on Theory and Applications in the Knowledge Economy*. 2019 International Conference on Theory and Applications in the Knowledge Economy, TAKE 2019 ; Conference date: 03-07-2019 Through 05-01-2020. 2019. URL: https://www.take-conference2019.com/.

Röhl, Thomas et al. "LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses". *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 781–784. DOI: 10.1109/CLUSTER.2017.115.

Sasongko, Muhammad Aditya et al. "Precise Event Sampling on AMD Versus Intel: Quantitative and Qualitative Comparison". *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1594–1608. ISSN: 1558-2183. DOI: 10.1109/TPDS.2023.3257105.

Team, The Smartmontools. *Smartmontools*. Accessed on 5th October 2023. URL: https://www.smartmontools.org/.

4.3 HISTORY OF CHANGES

Version	Author(s)	Date	Comment
0.1	Fatih Taşyaran	10.10.2023	First draft
0.2	Kamer Kaya	13.10.2023	Final version

Table 2 *Document History of Changes*