

Usage Statistics for a Scientific Testbed

Marcel Kornegger, Kilian Holzinger*, Sebastian Gallenmüller*

**Chair of Network Architectures and Services*

School of Computation, Information and Technology, Technical University of Munich, Germany

Email: marcel.kornegger@tum.de, holzinger@net.in.tum.de, gallenmu@net.in.tum.de

Abstract—The plain orchestrating service (pos) has been developed to manage and orchestrate experiments on various scientific testbeds. While pos provides a structured experiment workflow and automation features to enhance reproducibility, it currently lacks the capability to analyze historical testbed usage. For testbed operators, it would be particularly useful to know to what extent individual nodes are being utilized.

To address this limitation, we present the design and implementation of a dedicated usage statistics system for pos. This system leverages the existing calendar-based reservation system to generate approximated node utilization statistics based on historical booking data. The primary purpose of the tool is the generation of usage statistics. In addition, it enables users to further filter and aggregate these statistics based on various criteria, such as access groups and time ranges.

We demonstrate the applicability of our tool through a case study on the *Blockchain* testbed, showcasing its ability to provide insights into resource usage.

Index Terms—usage statistics, testbeds, pos

1. Introduction

Scientific testbeds play a crucial role in advancing research in computer networking and distributed systems by providing researchers with access to infrastructure to create reproducible experiments [1]. For this purpose, the plain orchestrating service (pos) has been developed at the *Chair of Network Architectures and Services* to manage and orchestrate experiments on various testbeds [2]. Pos enables a structured experiment workflow with fully scripted experiments and support for automated collection and publication of experiment artifacts [2]. By design, this approach reduces manual work and minimizes sources of nondeterminism. Therefore, pos enhances the reproducibility and repeatability of experiments.

Pos currently lacks the capability to analyze historical testbed usage. In particular, pos does not offer statistics on how test nodes are utilized over time. However, this information would be beneficial for testbed operators. For example, in heterogeneous testbeds with different types of nodes, such as GPU-equipped machines, usage statistics can help to identify popular resources and usage patterns. These insights help operators make informed decisions regarding resource allocation and future infrastructure investments.

In this paper, we present the design and implementation of a usage statistics system for the plain orches-

trating service to address this issue. Our proposed system leverages the existing calendar-based reservation system of pos to approximate node utilization statistics based on historical booking data. The implemented tool allows users to filter and aggregate usage statistics based on various criteria, such as access groups and time ranges. In addition, we demonstrate the applicability of our tool through a case study on a testbed of the *Chair of Network Architectures and Services*, showcasing its ability to offer resource usage insights.

The remainder of this paper is structured as follows. In Section 2, we investigate related work on usage statistics for scientific testbeds. Section 3 provides background information on the plain orchestrating service (pos). In Section 4, we present the design of our usage statistics system for pos, followed by implementation details in Section 5. We demonstrate our implementation in a case study in Section 6 and conclude the paper with an outlook on future work in Section 7.

2. Related Work

To contextualize our work, we review related scientific testbeds and their approach to monitoring and publishing resource utilization. Our analysis is based on publicly available information, as we do not have access to internal usage statistics or monitoring systems of these testbeds.

Several scientific testbeds for research in computer networking and distributed systems have been developed, e.g., Emulab [3], CloudLab [4], Grid'5000 [5], Chameleon [6], FABRIC [7], PlanetLab [8] and Fed4FIRE+ [9]. These differ in their targeted research domain, provided resources, architecture, and user communities [1], [10]. Available usage statistics of academic testbeds typically fall into two categories: (i) on-demand statistics available through a user interface [11]–[14], and (ii) published statistics that appear in papers or reports but are not accessible in real time [4], [15], [16]. On-demand statistics are either generated upon request or continuously updated, thereby providing a real-time view of current and historical resource utilization.

Emulab offers node usage statistics publicly via a web interface [11]. This includes graphs indicating the average number of free nodes, clustered by different node types, with historical data over both short- and long-term periods. CloudLab provides basic live statistics about the number of active experiments and available nodes on its website [12]. Historical usage statistics were published in the past but are not available on demand [4]. Grid'5000 has a web interface for usage statistics. However, it is not publicly

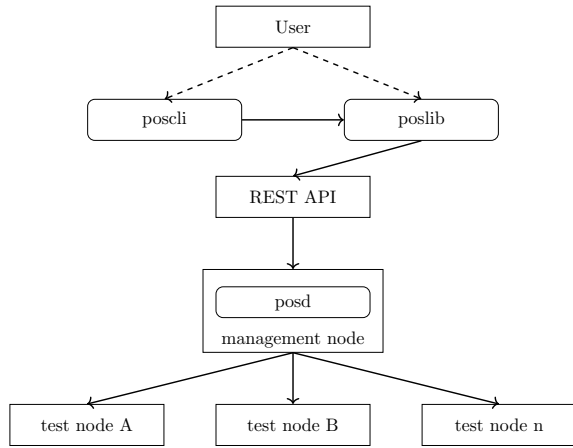


Figure 1: Overview of pos testbed architecture.

available [13]. FABRIC offers a dashboard providing an overview of the currently available resources but does not include any historical usage data [14]. Fed4FIRE+ published usage statistics about users and experiments in its EU project deliverables [15], but did not offer a usage interface at the federation level. Chameleon has published statistics, such as available nodes and used node-hours over time, but does not provide on-demand statistics publicly [16].

3. Background

The testbeds of the *Chair of Network Architectures and Services* are powered by the plain orchestrating service (pos). In the following, we give a brief overview of pos, its workflows, and its architecture, mainly based on the documentation in [17].

Figure 1 shows an overview of the architecture of a pos-based testbed. A testbed consists of multiple physical machines, called test nodes, which are connected to a management node via a dedicated management network. The management node runs the pos daemon (posd), which is responsible for managing the test nodes, orchestrating experiments, and collecting experiment data.

Typical experiment workflow. In order to run an experiment on a pos-based testbed, a user must first create a calendar entry that reserves the required test nodes for a specific time period. The calendar system is an optional feature, but it is used in all testbeds of the chair. Within this time period, an allocation has to be created to gain access to the reserved test nodes. Then, an experiment can be conducted by configuring image, variables and scripts for the test nodes. After the experiment has finished, the user can optionally free the allocation. Users may intentionally leave the allocation active, allowing the experiment to run on a best-effort basis until someone else frees it.

Daemon. The pos daemon is the core component of a pos-based testbed. The daemon maintains the state of the testbed and manages test nodes, calendar bookings, allocations, users and more. The daemon exposes a REST API to allow users to interact with the testbed.

Database. The daemon stores persistent state of the testbed in its PostgreSQL database. This includes test nodes, users, calendar, allocations and other relevant in-

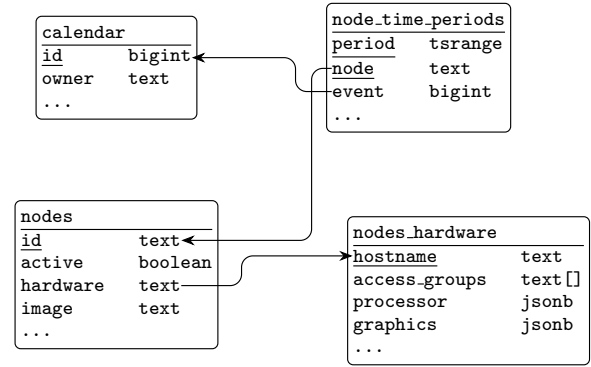


Figure 2: Relevant tables of pos database schema.

formation. The database schema reflects the operational aspects of the testbed. Figure 2 shows an excerpt of the database schema with the relevant tables for our matter. Underlined attributes indicate primary keys and arrows indicate foreign key relationships. The nodes table contains information about all test nodes, such as the currently running image. Additionally, nodes can be deactivated using the active flag. For each test node, more specific details about the hardware of a node is stored in the nodesHardware table. This table also includes the access groups of a node, allowing access to nodes to be restricted to a certain user group. For example, access to the GPU-equipped machines of the chair requires the user to have the corresponding access group. When a calendar booking is created, an event is recorded in the calendar table, while the node_time_periods table links calendar events to test nodes and the reserved time period.

Interacting with a testbed. As seen in Figure 1, users can interact with a testbed using different tools. Through the REST API of the daemon, users can create calendar bookings, allocate nodes, and conduct experiments. The poslib Python library provides a high-level abstraction of this API. The poscli command-line tool is built on top of poslib and offers commands to interact with the testbed from the terminal. In addition, pos provides a web interface for users to manage their calendar bookings.

4. Design

In this section, we present the design of our usage statistics system for pos.

Requirements. To guide the design of the usage statistics system for pos, we derive the following requirements.

Hardware-aware aggregation. The system must support clustering and aggregation of usage statistics based on node hardware characteristics (e.g., GPU-equipped nodes) in order to enable informed resource planning.

Configurable time ranges. The system must provide utilization statistics for configurable time intervals.

Fine-grained filtering. The system must allow filtering based on access groups and node IDs to enable focused analysis of specific subsets of the testbed.

Absolute and relative metrics. The system must report utilization both in absolute terms (e.g., hours used) and relative terms (e.g., share of total usage).

Statistics generation. In order to assess node usage, we can leverage the calendar system of pos, which is

used to reserve nodes for experiments. A calendar booking does not directly imply that the reserved nodes are actively used in this time period. In practice, however, users typically create calendar bookings only when they intend to conduct experiments, making bookings a reasonable proxy for node usage. Therefore, we can use calendar bookings as a proxy for node usage. To generate usage statistics, we use the `node_time_periods` table of the `pos` database (cf. Figure 2), which contains the reserved time periods for each node. In combination with the `nodes` and `nodes_hardware` tables, we can filter the usage statistics based on node IDs and access groups. To support clustering based on different node hardware, the access group is used as a proxy: Some access groups correspond to hardware features, e.g., GPU-equipped nodes belong to a special access group. We cannot rely on the actual hardware columns (e.g., processor or graphics) because they are optional and are not consistently populated with data. The calendar table would only be needed if we wanted to filter based on user IDs, which is currently out of scope and can be addressed in the future.

Software architecture. To ensure consistency with the existing `pos` architecture and maintain a modular system design, the usage statistics functionality is integrated following the established action grouping logic. In the existing `pos` architecture, all available actions are logically grouped. Each group of actions has a dedicated endpoint handler in `posd`, exposing functionality via the REST API. If a group of actions requires persistent data storage, the corresponding database queries are implemented in a dedicated database handler. In `poscli`, each group of actions has a corresponding command module, which implements a new subcommand for the `poscli` command-line tool. Following this architecture, we introduce a new endpoint handler in `posd` for statistics-related actions with the route `/stats/node_utilization`. This handler implements the logic to process incoming requests, validate parameters, and call the database handler to retrieve the required data. To start off with the usage statistics, we want to make integration as easy as possible. Therefore, we do not create new tables for statistics and no new database handler is required. Instead, we implement the query logic in the existing database handler for the `nodes` table. In `poscli`, we implement a new command module `stats` with a subcommand `node_utilization` to allow users to request usage statistics from the command line. The command accepts parameters for time range, clustering, and filtering options.

5. Implementation

In the following, we describe the implementation of our usage statistics system for `pos`, following the design presented in Section 4.

5.1. REST API

To maintain architectural consistency and separation of concerns, we introduce a new endpoint handler in `posd` for statistics-related actions covering the `/stats/` routes. The handler is implemented in the `StatsRouteHandler` class, which extends the `EndpointRouteHandler` class to reuse existing request validation and response handling logic.

The new route `/stats/node_utilization` returns usage statistics via a GET request as the operation is read-only and stateless. The endpoint accepts the following query parameters:

- `start_date` and `end_date` (required): Define the time range for the statistics.
- `filter_nodes`: Only include the defined node ids in the statistics.
- `filter_access_groups`: Only include nodes belonging to the defined access groups in the statistics.
- `group_by_access_groups`: Group and aggregate results by access groups.

The handler processes incoming requests, validates parameters and calls the `NodesDBHandler` database handler to retrieve the required data. The API returns the usage statistics in a structured JSON format, including absolute usage in booked hours and relative usage for each node id. If grouping by access groups is requested, the handler returns statistics for each access group instead of individual nodes. Similar to other testbeds (cf. Section 2), usage statistics are computed on-demand when the endpoint is called.

5.2. Database Query

The query logic is implemented in the existing `NodesDBHandler` database handler, which is responsible for queries related to the `nodes` table. We use parameterized SQL queries and structure the query with Common Table Expressions (CTEs) for better readability and maintainability. The query is structured into the following parts: (i) Join required tables, (ii) Filtering, (iii) Aggregation and (iv) Calculation of relative usage. We describe the subqueries in a simplified manner, leaving out type casts and null checks for clarity.

Listing 1: SQL subquery (i): Join required tables

```

1 WITH base AS (
2   SELECT n.id AS node_id, n.active, hw.access_groups,
3         ntp.period,
4         CASE WHEN :GROUP_BY_AG THEN ag.access_group ELSE n.id
5         END AS id
6   FROM nodes AS n
7   LEFT JOIN nodes_hardware AS hw ON n.hardware = hw.
8       hostname
9
10  LEFT JOIN LATERAL (
11    SELECT unnest(hw.access_groups) AS access_group
12  ) AS ag ON :GROUP_BY_AG = true
13
14  LEFT JOIN node_time_periods AS ntp ON ntp.node = n.id
15 )

```

In Listing 1, we join the required tables based on their foreign key relations to retrieve the required data for the statistics. The resulting `id` column contains the final ID which is either the node ID or the access group, depending on whether grouping by access groups is requested (L3). Since `access_groups` is an array, we use a lateral join with the `unnest()` function to create a row for each access group of a node (L7–L9), if grouping is requested.

Listing 2: SQL subquery (ii): Filtering

```

1 WITH filtered AS (
2   SELECT node_id, id, period FROM base
3   WHERE active = true
4   AND (:FILTER_NODES IS NULL OR node_id = ANY(
5     FILTER_NODES))
6   AND (:FILTER_AG IS NULL OR access_groups && :
7     FILTER_AG)
8   AND (
9     period IS NULL OR period
10    && tsrange(:START_DATE, :END_DATE, '[') )
11 )

```

In Listing 2, we filter the joined data based on the provided parameters. We only include active nodes (L3) and apply filters for node IDs (L4), access groups (L5) and period to be overlapping with the request time range (L6–L9).

Listing 3: SQL subquery (iii): Aggregation

```

1 WITH per_node AS (
2   SELECT id, node_id, COUNT(period) AS booking_count,
3   SUM(EXTRACT(EPOCH FROM (
4     LEAST(upper(period), :END_DATE)
5     - GREATEST(lower(period), :START_DATE)
6   ))) AS booked_seconds
7   FROM filtered
8   GROUP BY id, node_id
9 ),
10
11 per_id AS (
12   SELECT id, SUM(booking_count) AS booking_count, SUM(
13     booked_seconds) AS booked_seconds
14   FROM per_node
15   GROUP BY id
16 )

```

In Listing 3, we first aggregate the filtered data per node ID (L1–L11) to calculate the number of bookings and total booked seconds within the specified time range. We use the LEAST and GREATEST functions to correctly handle bookings that partially overlap with the requested time range (L4–L5). In the second CTE (L11–L15), we further aggregate the data per final ID (either node ID or access group) to get the total bookings and booked seconds.

Listing 4: SQL subquery (iv): Calculation of relative usage

```

1 SELECT id, booking_count,
2   ROUND((booked_seconds / 3600), 4) AS booked_hours,
3   ROUND(booked_seconds / SUM(booked_seconds), 4) AS
4   relative_utilization
5 FROM per_id;

```

In Listing 4, we calculate the final usage statistics by converting booked seconds to booked hours (L2) and calculating the relative utilization as the fraction of booked seconds over the total booked seconds (L3). The final result includes the ID, booking count, booked hours and relative utilization for each node ID or access group.

5.3. Command Line Interface

In poscli, we implement a new subcommand `node_utilization` of the `stats` command module using the Click library [18], which allows users to request usage statistics from the command line. The command accepts the following parameters:

TABLE 1: Top 10 most-used nodes in Blockchain testbed (November 2025)

Node ID	Bookings	Booked hours	Rel. Utilization
smartcash	23	491.4	9.24%
ada	21	483.9	9.09%
chrysos	8	369.0	6.94%
litecoin	13	251.8	4.73%
litecoincash	4	235.7	4.43%
vmexp1	39	208.6	3.92%
bitcoin	63	192.2	3.61%
idex	51	175.8	3.30%
amdexp0	74	165.9	3.12%
algoft	41	164.8	3.10%
stoi	41	149.1	2.80%

- `--start-date DATE` and `--end-date DATE`: Define the time range for the statistics (required).
- `--filter-nodes NODES`: Only include the defined node IDs in the statistics.
- `--filter-access-groups ACCESS_GROUPS`: Only include nodes belonging to the defined access groups in the statistics.
- `--group-by-access-groups`: Group and aggregate results by access groups.
- `--order-by FIELD`: Order the results by the specified field.
- `--desc`: Order the results in descending order.
- `--json`: Print results as json. By default, results are printed as a table.

In poslib, a high-level abstraction of the new daemon API route is implemented in the `node_utilization()` method of the `stats` module. This method accepts the same parameters as the API route and is called by the `poscli` command module to request the statistics from the daemon. Ordering is performed by the `poscli` command module.

6. Case Study

In this section, we present a case study in which we apply the usage statistics to a real-world scenario as a demonstration.

Setup. The usage statistics tool is applied to the *Blockchain* testbed operated by the Chair of Network Architectures and Services at TUM [19]. The testbed includes heterogeneous hardware resources, such as special test nodes equipped with GPUs, P4-switches, and compute nodes, as well as multiple access groups. The statistics are generated on-demand from the historical booking data stored in the `pos` database of the testbed.

Results. Using the implemented usage statistics tool, we generate node utilization statistics for the *Blockchain* testbed for the month of November 2025. Table 1 shows the 10 most-used nodes in terms of booked hours, along with their booking counts and relative utilization. For example, the node `smartcash` was booked for a total of 491.4 hours across 23 bookings, accounting for 9.24% of the total usage in the testbed during this period.

Figure 3 shows the distribution of booking times across the individual access groups, with each segment representing an access group and its booked hours, and its relative utilization. This visualization is based on data



Figure 3: Node usage statistics for the Blockchain testbed, grouped by access groups (November 2025)

generated with the `--group-by-access-groups` option. A node can belong to multiple access groups. In this case, its booked hours contribute to multiple access groups in the aggregated view. The None access group represents nodes that do not belong to any access group. We can see that certain access groups, such as `algorand` and `gpuserver`, have a significant share of the total usage, indicating their popularity among users. The `algorand` nodes are the newest test nodes in the testbed and provide higher performance, making them attractive for computationally demanding experiments. Similarly, the `gpuserver` group consists of nodes equipped with GPUs, which are required for workloads involving machine learning, parallel computing, or hardware acceleration.

7. Conclusion and Future Work

In this paper, we presented the design and implementation of a usage statistics system for the plain orchestrating service (pos). By leveraging the existing calendar system of pos, we are able to approximate node utilization statistics based on historical booking data. The implemented tool allows users to filter and aggregate usage statistics based on various criteria, such as access groups and time ranges. We demonstrated the applicability of our tool through a case study on the *Blockchain* testbed, showcasing its ability to provide insights into resource usage.

For future work, the usage statistics can be extended in several ways. Further filtering options could be added, for example, based on hardware features or users. A dedicated web interface could be developed to visualize node usage or other statistics. For this purpose, it would be beneficial to pre-compute statistics or cache them, reducing the load on the database for repeated requests.

References

- [1] L. Nussbaum, “Testbeds Support for Reproducible Research,” in *Proceedings of Reproducibility’17*. Los Angeles, CA, USA: ACM, Aug. 2017, pp. 24–26.
- [2] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, “The pos framework: A methodology and toolchain for reproducible network experiments,” in *Proceedings of the 17th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT ’21. New York, NY, USA: ACM, 2021, pp. 259–266.
- [3] F. Hermenier and R. Ricci, “How to Build a Better Testbed: Lessons from a Decade of Network Experiments on Emulab,” in *Testbeds and Research Infrastructure. Development of Networks and Communities*, T. Korakis, M. Zink, and M. Ott, Eds. Berlin, Heidelberg: Springer, 2012, vol. 44, pp. 287–304.
- [4] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, “The Design and Operation of CloudLab,” in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC ’19. USA: USENIX Association, Jul. 2019, pp. 1–14.
- [5] “Grid5000,” <https://www.grid5000.fr/>, accessed: 2025-11-30.
- [6] “Chameleon,” <https://chameleoncloud.org/>, accessed: 2025-12-02.
- [7] “FABRIC Testbed,” <https://fabric-testbed.net/>, accessed: 2025-12-02.
- [8] “Planetlab Europe,” <https://www.planet-lab.eu/>, accessed: 2025-12-02.
- [9] “Fed4FIRE+,” <https://www.fed4fire.eu/>, accessed: 2025-12-13.
- [10] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, “A survey on network simulators, emulators, and testbeds used for research and education,” *Computer Networks*, vol. 237, p. 110054, Dec. 2023.
- [11] “Emulab – testbed node usage stats,” https://www.emulab.net/node_usage/, accessed: 2025-11-30.
- [12] “CloudLab,” <https://cloudlab.us/>, accessed: 2025-11-30.
- [13] “Grid5000 – status,” <https://www.grid5000.fr/w/Status>, accessed: 2025-11-30.
- [14] “FABRIC – resources overview,” <https://portal.fabric-testbed.net/resources/overview>, accessed: 2025-12-02.
- [15] “Fed4FIRE+ – D2.7: Federation Operations, Tools and Support,” <https://www.fed4fire.eu/wp-content/uploads/sites/7/d2-07-federation-operations-tools-and-support-update-1.pdf>, 2020, accessed: 2025-12-13.
- [16] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, “Lessons learned from the chameleon testbed,” in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC’20. USA: USENIX Association, 2020.
- [17] “Plain Orchestrating Service (pos) – Documentation,” <https://i8-testbeds.pages.gitlab.lrz.de/pos/cli>, accessed: 2025-11-15.
- [18] “Click – Python Library – Documentation,” <https://click.palletsprojects.com/>, accessed: 2025-12-17.
- [19] “i8 – Blockchain Testbed – Wiki,” <https://gitlab.lrz.de/i8-testbeds/wiki/-/wikis/testbeds/blockchain>, accessed: 2025-12-11.