BenchCloud: A Platform for Scalable Performance Benchmarking

Dirk Beyer
LMU Munich
Munich, Germany

Po-Chun Chien
LMU Munich
Munich, Germany

Marek Jankola
LMU Munich
Munich, Germany

Abstract

Performance evaluation is a crucial method for assessing automated-reasoning tools. Evaluating automated tools requires rigorous benchmarking to accurately measure resource consumption, including time and memory, which are essential for understanding the tools' capabilities. BenchExec, a widely used benchmarking framework, reliably measures resource usage for tools executed locally on a single node. This paper describes BenchCloud, a solution for elastic and scalable job distribution across hundreds of nodes, enabling large-scale experiments on distributed and heterogeneous computing environments. BenchCloud seamlessly integrates with BenchExec, allowing BenchExec to delegate the actual execution to BenchCloud. The system has been employed in several prominent international competitions in automated reasoning, including SMT-COMP, SV-COMP, and Test-Comp, underscoring its importance in rigorous tool evaluation across various research domains. It helps to ensure both internal and external validity of the experimental results. This paper presents an overview of BenchCloud's architecture and highlights its primary use cases in facilitating scalable benchmarking.

Demonstration video: https://youtu.be/aBfQytqPm0U
Running system: https://benchcloud.sosy-lab.org/

CCS Concepts

• **General and reference** → **Cross-computing tools and techniques**; • **Computing methodologies** → *Distributed computing methodologies*; • **Computer-systems organization** → *Cloud computing*; *Client-server architectures*; • **Software and its engineering** → *Software verification and validation.*

Keywords

Benchmarking, Remote execution, Job-distribution system, Resource management, Cloud computing, Tool competition, Containers

ACM Reference Format:
Dirk Beyer, Po-Chun Chien, and Marek Jankola. 2024. BenchCloud: A Platform for Scalable Performance Benchmarking. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27 – November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695358

1 Introduction

Large-scale performance evaluation is essential in experimental automated-reasoning research for assessing the effectiveness of various tools and methodologies [1]. Performance-critical automated tools such as verifiers, logic solvers, simulators, and test-case generators require rigorous benchmarking to accurately measure execution time, memory usage, and other performance measures.

There are two major problems to solve: (1) To ensure internal validity, it is essential to use accurate and reliable technology for controlling resources. The benchmarking framework BenchExec [2] can be used to address this problem. (2) To ensure external validity, it is essential to evaluate the performance across a diverse and large benchmark set. The competition SV-COMP 2024 [3] involved over 30 000 verification tasks and evaluated 76 verification tools. We provide BenchCloud as a solution to scale to large experiments.

BenchCloud addresses these scalability challenges by providing an elastic cloud-based infrastructure. This system enhances the capability of BenchExec to handle large-scale experiments and ensures reliable performance measurements. BenchCloud's architecture includes a manager process that distributes benchmark runs across an elastic set of worker nodes. Benchmark results are collected and returned to users, including output files, resource measurements, and execution logs. The system also features run prioritization, allowing for flexible resource allocation based on the importance of different benchmark runs. BenchCloud is elastic in the sense that it is tolerant to disappearing workers and starts using new workers as soon as they appear. By supporting resource management and monitoring across distributed nodes, BenchCloud facilitates rigorous performance evaluations in various research domains [3–5].

Development History. The development of BenchCloud started in 2011 at the University of Passau. The project was initially named VerifierCloud (hence the abbreviation `vcloud` in some references) as it was designed for benchmarking software verifiers on the cloud. BenchCloud made its first public debut in the competition report of SV-COMP 2017 [6]. Now the project is primarily developed and maintained by researchers and students at LMU Munich.

Significance and Impact. BenchCloud has become a pivotal platform for hosting various well-recognized international tool competitions on software verification (SV-COMP [3, 6]), software testing (Test-Comp [4]), and satisfiability modulo theories (SMT-COMP [5]). It ensures reliable performance measurements via BenchExec [2] and scales for parallel benchmark execution across many worker machines, meeting the rigorous requirements of large-scale evaluations for research in software engineering and automated reasoning. While primarily utilized in automated-reasoning communities, BenchCloud can also be used for benchmarking other automated tools. Its unique role in these competitions and software-engineering research underscores its substantial contribution to the development and assessment of cutting-edge tools. Despite being developed and utilized for over a decade, the system has not been adequately documented in the literature. This manuscript addresses this gap by offering an overview of the system.

ASE '24, October 27 – November 1, 2024, Sacramento, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695358

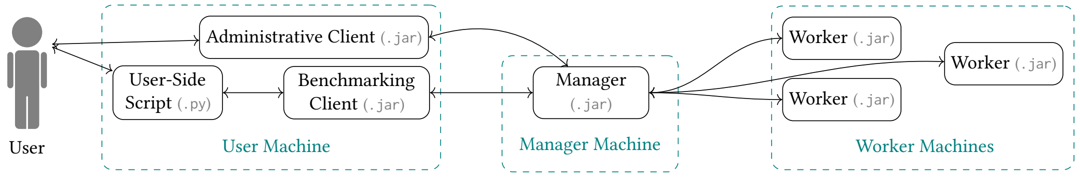

Figure 1: Components of BenchCloud

2 System Description

Figure 1 provides a high-level system overview of BenchCloud, illustrating its four main components: user-side script, clients, manager, and workers. These components work together across multiple machines to facilitate large-scale performance benchmarking. From a user's machine, the user-side script collects the executables of the tool to be benchmarked, the benchmark set (i.e., a set of input files for the tool to execute on), and the experimental settings into a *run collection*, and submits it to the BenchCloud manager through the *benchmarking client*. Upon receiving a run collection, the manager distributes the workload to the worker machines. Each worker machine executes a set of benchmark runs with the resource constraints specified in the experimental settings and reports the results back to the manager. Finally, the manager aggregates the results and sends them back to the user via the client and the user-side script. In this section, we describe each of the main components of BenchCloud in detail.

2.1 User-Side Script

The user-side script `vcloud-benchmark.py`[1] takes as input a *benchmark definition*, which specifies the tool to be benchmarked, the benchmark set, files to be retrieved after a benchmark run, and the experimental setup, including the required CPU model of the worker machines, and limits (memory, time, cores) for each run.

The user-side script is integrated in the BenchExec framework [2] by implementing BenchExec's `Executor` interface, for which the framework provides a default implementation for local execution. Our `Executor` for remote execution tells BenchExec how to process a run collection after aggregating all necessary files, by implementing the method `execute_benchmark()`. Therefore, the user-side script can be used as a drop-in replacement for the program `benchexec`, which uses the default `Executor` provided by the framework.

To benchmark a tool on BenchCloud, it has to be available on the user's machine as executable for the worker machine. If external dependencies, such as libraries, are required for the tool's execution, they should be shipped together with the tool or installed on the worker machines (future versions will support OCI containers [7]). Moreover, the user needs to provide a *tool-info module*[2] to instruct BenchExec on how to assemble the command line to start the tool, which directories to deploy to workers, and how to parse its output.

2.2 Clients

The communication between the user and the BenchCloud manager flows through clients. Here we describe two commonly-used clients: the benchmarking client and the administrative client.

[1]https://github.com/sosy-lab/benchexec/blob/3.24/contrib/vcloud-benchmark.py
[2]Examples: https://github.com/sosy-lab/benchexec/tree/3.24/benchexec/tools

Benchmarking Client. The benchmarking client serves as a bridge between the user-side script and the manager. It is employed by the user-side script for submitting run collections to the manager, and later receiving the benchmarking results from the manager.

Administrative Client. The administrative client is used for administrating a running BenchCloud instance. It features an interactive command-line interface that allows the user to (1) obtain the status and availability of the workers, (2) add and remove workers to/from the system, (3) retrieve the information (e.g., number of finished runs and the submission time) about a run collection, and (4) cancel and adjust the *priority* (explained in Sect. 2.3) of a run collection.

2.3 Manager

The manager is the central component in BenchCloud, responsible for distributing benchmark runs to worker machines and collecting the results. Its three main functionalities are described below.

Job Distribution. The manager maintains an overview of the worker machines, including CPU model, available memory and cores, occupancy status, and current workload. As mentioned in Sect. 2.1, each run collection has a set of resource requirements, with which the worker machines have to comply. The manager assigns runs only to worker machines with a matching CPU model and sufficient available memory and CPU cores, based on these requirements. Each run is assigned dedicated CPU cores and memory, ensuring that no two runs are using the same CPU core and that memory is not overbooked on the worker machines.

In practical scenarios, worker machines are often shared with *external users*, that is, users who use these machines for purposes other than benchmarking and are not part of the BenchCloud system. If the manager detects that an external user is occupying a worker machine, it will stop all currently-executing runs on that machine (and later reassign them to other machines). No further runs will be assigned to that worker machine until the occupancy is cleared. This feature is particularly useful at universities, where computer pools are shared among students and researchers.

Run Scheduling. Multiple users can submit run collections to BenchCloud at the same time. The manager schedules runs based on the specified *priority* of the run collection. BenchCloud offers the following five priority levels:

- `URGENT`: Runs are always scheduled if a worker machine satisfying the resource requirements is available.
- `HIGH` and `LOW`: Runs are scheduled if there are no `URGENT` runs. The ratio of number of runs scheduled for `HIGH` and `LOW` is 2:1.
- `IDLE`: Runs are scheduled only if no runs of higher priority (`URGENT`, `HIGH`, and `LOW`) are in the queue.
- `PAUSED`: No runs are scheduled.

If there are multiple run collections with the same priority from different users, the manager schedules their runs fairly. The priority of a run collection is initially set by the user, but can be later adjusted by the system administrator using the administrative client (see Sect. 2.2). Having different priority levels and the flexibility to adjust them allows BenchCloud users to coordinate the allocation of computing resources according to their needs.

Result Aggregation. After a run collection is completed, the manager collects the computed results and outputs from the worker machines. The result from a run contains the extracted outcome of the tool (using the tool-info module; see Sect. 2.1), the consumed CPU time, and the memory usage (i.e., the peak memory consumption). All results are organized in a structured and standardized XML format used in the BenchExec framework [2]. Additionally, the manager collects the console output (`stdout` and `stderr`) of the tool, as well as the output files specified in the benchmark definition. These aggregated files are then sent back to the user via the benchmarking client. (Users can conveniently post-process and visualize the results using the program `table-generator` in the BenchExec framework.)

2.4 Worker

A worker is a BenchCloud process running on a worker machine. It has the following responsibilities.

Status Monitoring. The worker continuously monitors and reports the status of the machine to the manager. The reports include the current runs being executed, the available computing resources, and whether any external user is occupying the machine. This reporting mechanism ensures that the manager has up-to-date information on the status of all worker machines, which is essential for effective job distribution (see also Sect. 2.3).

Isolated Run Execution and Resource Management. Each BenchCloud worker uses the program `runexec` from the BenchExec framework [2] to execute benchmark runs. `runexec` pins a benchmark run to the CPU cores assigned by the manager and restricts its memory usage according to the given experimental settings using the control groups (cgroups) of the Linux kernel. Besides constraining resource usage, cgroups also support the measurement of resource consumption of the benchmark run. Additionally, `runexec` leverages Linux namespaces for isolated execution, effectively containerizing each benchmark run. This isolation ensures that benchmark runs do not interfere with each other. Linux overlay file systems are used for clean handling of file access and result collection: the benchmarked tool writes to an overlay file system. The combined use of cgroups, namespaces, and overlay file systems ensures the reliability and reproducibility of the benchmark results.

File Caching. In typical benchmarking scenarios, users often benchmarks the same tool across different tasks of a benchmark set or benchmarks different tools on the same tasks. To optimize performance and efficiency, each BenchCloud worker maintains a local file cache for these commonly used files, including both the tool binaries and the benchmark tasks. This caching mechanism effectively reduces network traffic and the time required to transfer files between the manager and the worker machines, and mitigates the risk of saturating the network. By reducing data-transfer overhead, BenchCloud fosters smoother execution of benchmark runs and improves overall system efficiency.

2.5 Implementation and System Requirements

The user-side script and BenchExec are written in Python, whereas the remaining components of BenchCloud are written in Java. Communication between clients, manager, and workers is handled via the `java.net` package. The manager additionally relies on SSH connections to initiate a worker process on a worker machine. For BenchCloud 1.1 [8], Java 11 and Python 3.8 (or newer) are required. In order to ensure process isolation and reliable resource measurement for benchmark runs on worker machines using BenchExec [2], these machines are required to have a Linux-based operating system with modern features like control groups (for managing resource usage), namespaces (for process containerization), and overlay file systems (for file handling).

3 Use Cases

In this section, we present the main use cases and the existing instances of BenchCloud.

3.1 Large-Scale Performance Benchmarking

BenchCloud is a cloud platform designed for large-scale performance benchmarking of automated tools, a common practice in the field of software engineering. At the time of writing, there are four running instances of BenchCloud at LMU Munich, Technical University of Dortmund, Masaryk University of Brno, and ISP RAS. The instance at LMU Munich has the most extensive computing resources, boasting a cluster of over 250 worker machines, with a total of more than 2 400 CPU cores and 10 TB of RAM. These BenchCloud systems are primarily used for research activities in the universities.

Software-Engineering Research. BenchCloud serves as a pivotal platform for performing experimental evaluations with precision and scalability for the software-engineering research community. Once research ideas evolve into tools and mature into experimental setups intended for publication, BenchCloud ensures reliable and consistent performance measurements. Its capability to horizontally scale experiments across multiple machines is invaluable, significantly reducing the time required for evaluations that might otherwise take months on a single machine to just a matter of hours with BenchCloud. The system supports the rigorous demands of scientific research while maintaining an efficient workflow.

International Tool Competitions. BenchCloud plays a crucial role in several prominent international tool competitions, including SV-COMP [3], Test-Comp [4], and SMT-COMP [5], where it serves as a platform for evaluating the performance of numerous automated logic-solving tools. These competitions necessitate rigorous benchmarking of tools across diverse benchmark tasks under controlled environments to assess the tools' effectiveness and efficiency. The BenchCloud instance at LMU Munich supports such evaluation requirements. For example, the 13th SV-COMP in 2024 [3] was executed on 168 worker machines, each equipped with an Intel Xeon E3-1230 v5 CPU. The competition involved 76 verification tools and over 30 000 verification tasks, collectively consuming approximately 4 394 days of CPU time. Notably, while previous editions of

SMT-COMP were conducted on StarExec [9], the 19th SMT-COMP in 2024 utilized the BenchCloud instance at LMU Munich. BenchCloud's scalability and robust resource management make it an ideal choice for hosting large-scale tool competitions.

3.2 Continuous Integration

BenchCloud can also be used for continuous integration (CI) of software projects, especially for detecting functional and performance regressions. For example, the project CPAchecker [10], a configurable software-verification platform developed at LMU Munich, integrates BenchCloud in its Buildbot-based CI pipelines. CPAchecker's Buildbot launches a set of regression tests (as benchmark runs) on BenchCloud at the latest commit of CPAchecker on a daily basis.[3] Every day, the BenchCloud instance at LMU Munich executes over 80 000 benchmark runs for CPAchecker, consuming an average of 85 days of CPU time. Similarly, the development team of JDart [11] at Technical University of Dortmund also incorporates BenchCloud into their CI pipelines. By leveraging BenchCloud for regression testing, the tool developers can quickly identify issues introduced by recent commits. This integration greatly enhances the reliability of the development process, and ensures high code quality and stable performance in the software tools.

4 Related Work

StarExec [9] provides a similar functionality for distributing benchmark runs across computing nodes. It is a web-based benchmarking platform that supports large-scale experimental evaluations across different logic-solving communities, including SMT, SAT, and QBF. StarExec offers a shared infrastructure for storing and managing benchmark tasks and solvers, running solver competitions, and performing comparative evaluations. It used to delegate the resource measurement and control to runsolver [12] and has now adopted `runexec` from the BenchExec framework for this purpose. BenchCloud, on the other hand, is designed to be a cloud-based extension of BenchExec, offering seamless integration with existing components in the framework. This design enables experimental setups that work locally with BenchExec to be easily adapted for remote execution. Additionally, the experimental data gathered by BenchCloud can be directly processed by `table-generator` in BenchExec, streamlining the workflow for researchers.

There are also general-purpose job scheduling and management systems such as Slurm [13] and Kubernetes. Slurm is widely used in high-performance computing environments thanks to its optimized resource allocation and modularity. Kubernetes, designed for container orchestration, provides various features for running containerized applications in parallel. However, these systems are not specifically tailored for benchmarking, and require additional customization to integrate into a scientific evaluation workflow.

Cloud-based load-testing services from providers like AWS and Azure can also be employed to distribute runs at scale. Nonetheless, these services typically provide resource measurement at the node level, whereas BenchCloud collects resource usage for each individual benchmark run. Furthermore, BenchCloud is a free software, while cloud services are generally commercial offerings.

[3] CPAchecker's Buildbot uses a specialized, tool-specific benchmarking client that also automates the tool compilation using worker machines.

5 Conclusion and Future Work

BenchCloud already plays an important role in the automated-reasoning research community, since it is used by three international competitions, and there are at least four independent instances running at European research institutes. Thus, it is time to release the code of this important infrastructure to the public to let more people benefit from it. BenchCloud makes it possible to use the capabilities of BenchExec across many computing nodes, which guarantees reliable and accurate benchmarking on a large scale. The integration with BenchExec facilitates consistency in experimental setups from local to remote execution and convenience in data processing. Furthermore, BenchCloud's components enable system administrators to effortlessly manage a running instance, and the run-prioritization features give users flexibility in resource allocation. Future releases plan to support tool execution in customizable containers [7], reducing the need for users to install the necessary dependencies on the worker machines.

Declarations

Data-Availability Statement. The source code, compiled binaries, and documentation of BenchCloud 1.1 are archived on Zenodo [8].

Funding Statement. This project was funded in part by the Deutsche Forschungsgemeinschaft (DFG) – 378803395 (ConVeY) and 418257054 (Coop), and by the Free State of Bavaria.

Acknowledgement. We thank all developers and users from LMU Munich, Uni Passau, Uni Oldenburg, TU Dortmund, BME Budapest, and Masaryk Uni Brno for valuable feedback and contributions.

References

[1] W. F. Tichy. 1998. Should Computer Scientists Experiment More? *IEEE Computer* 31, 5 (1998), 32–40. https://doi.org/10.1109/2.675631

[2] D. Beyer, S. Löwe, and P. Wendler. 2019. Reliable Benchmarking: Requirements and Solutions. *Int. J. Softw. Tools Technol. Transfer* 21, 1 (2019), 1–29. https://doi.org/10.1007/s10009-017-0469-y

[3] D. Beyer. 2024. State of the Art in Software Verification and Witness Validation: SV-COMP 2024. In *Proc. TACAS (3) (LNCS 14572)*. Springer, 299–329. https://doi.org/10.1007/978-3-031-57256-2_15

[4] D. Beyer. 2023. Software Testing: 5th Comparative Evaluation: Test-Comp 2023. In *Proc. FASE (LNCS 13991)*. Springer, 309–323. https://doi.org/10.1007/978-3-031-30826-0_17

[5] T. Weber, S. Conchon, D. Déharbe, M. Heizmann, A. Niemetz, and G. Reger. 2019. The SMT Competition 2015-2018. *J. Satisf. Boolean Model. Comput.* 11, 1 (2019), 221–259. https://doi.org/10.3233/SAT190123

[6] D. Beyer. 2017. Software Verification with Validation of Results (Report on SV-COMP 2017). In *Proc. TACAS (LNCS 10206)*. Springer, 331–349. https://doi.org/10.1007/978-3-662-54580-5_20

[7] D. Beyer and H. Wachowitz. 2024. FM-Weck: Containerized Execution of Formal-Methods Tools. In *Proc. FM (LNCS 14934)*. Springer. https://doi.org/10.1007/978-3-031-71177-0_3

[8] D. Beyer, P.-C. Chien, and M. Jankola. 2024. BenchCloud Release 1.1. Zenodo. https://doi.org/10.5281/zenodo.13742756

[9] A. Stump, G. Sutcliffe, and C. Tinelli. 2014. StarExec: A Cross-Community Infrastructure for Logic Solving. In *Proc. IJCAR (LNCS 8562)*. Springer, 367–373. https://doi.org/10.1007/978-3-319-08587-6_28

[10] D. Beyer and M. E. Keremoglu. 2011. CPAchecker: A Tool for Configurable Software Verification. In *Proc. CAV (LNCS 6806)*. Springer, 184–190. https://doi.org/10.1007/978-3-642-22110-1_16

[11] K. S. Luckow, M. Dimjasevic, D. Giannakopoulou, F. Howar, M. Isberner, T. Kahsai, Z. Rakamaric, and V. Raman. 2016. JDart: A Dynamic Symbolic Analysis Framework. In *Proc. TACAS (LNCSS 9636)*. Springer, 442–459. https://doi.org/10.1007/978-3-662-49674-9_26

[12] Olivier Roussel. 2011. Controlling a Solver Execution with the runsolver Tool. *JSAT* 7 (2011), 139–144. https://doi.org/10.3233/SAT190083

[13] A. B. Yoo, M. A. Jette, and M. Grondona. 2003. Slurm: Simple Linux Utility for Resource Management. In *Proc. JSSPP (LNCS 2862)*. Springer, 44–60. https://doi.org/10.1007/10968987_3