Demonstration

Chronos: The Swiss Army Knife for Database Evaluations

Marco Vogt Alexander Stiemer Sein Coray Heiko Schuldt
{firstname.lastname}@unibas.ch
Databases and Information Systems Research Group
Department of Mathematics and Computer Science
University of Basel, Switzerland

ABSTRACT

Systems evaluations are an important part of empirical research in computer science. Such evaluations encompass the systematic assessment of the run-time characteristics of systems based on one or several parameters. Considering all possible parameter settings is often a very tedious and time-consuming task with many manual activities, or at least the manual creation of evaluation scripts. Ideally, the thorough evaluation of a complete evaluation space can be fully automated. This includes the set-up of the evaluation, its execution, and the subsequent analysis of the results. In this paper, we introduce Chronos, a system for the automation of the entire evaluation workflow. While Chronos has originally been built for database systems evaluations, its generic approach also allows its usage in other domains. We show how Chronos can be deployed for a concrete database evaluation, the comparative performance analysis of different storage engines in MongoDB.

1 INTRODUCTION

Scientific practice considers the development of novel theories and their empirical evaluation. In computer science, empirical evidence is to a large extent obtained by the systematic evaluation of systems. Essentially, such a systematic evaluation includes the thorough assessment of the quantitative behavior of a system based on one or several parameters. This means that systems need to be run over and over again with a modified set of parameters. This is a tedious and highly repetitive task, but essential for obtaining insights into the run-time characteristics of the system under evaluation (SuE).

In the database community, systems evaluations are mostly based on benchmarks that combine test data and certain access characteristics (queries). Systems evaluations need to be tailored to the characteristics of an SuE and to the parameters of the latter that determine their behavior. Nevertheless, the overall evaluation workflows, seen from a high-level perspective, show quite some commonalities even across systems. First, the SuE needs to be set up with the exact parameters of a particular evaluation run. This also contains the configuration of the SuE — in the database world, this includes the generation of benchmark data and their ingestion into the system. Second, the SuE needs to undergo a warm-up phase, for instance filling internal buffers, to make sure that the behavior of the SuE reflects a realistic use. Third, the actual evaluation is run. In the case of a database benchmark, this is the execution of the predefined queries in a given query mix. The evaluation finally generates data which at the end needs to be analyzed.

In most cases, these steps are implemented to a large extent by means of manual activities and are highly repetitive. Even in cases of (semi-)automated evaluations, they need to be re-started over and over again with varying parameters.

Ideally, a complete set of evaluation runs that systematically and thoroughly assess a given parameter space of an SuE can be fully automated. The only requirement on an SuE is that its evaluation workflow does not require any human interaction, allowing such SuEs to be evaluated using a generic evaluation toolkit. In order to provide systems *Evaluations-as-a-Service (EaaS)*, such a toolkit, once linked to the SuE, has to fulfill the following requirements: (i) It has to feature an easy to use UI for defining new experiments, for scheduling their execution, for monitoring their progress, and for analyzing their results. (ii) The toolkit has to support different SuEs at the same time and in particular parallel executions of benchmarks of these systems. (iii) To allow SuEs to be considered in long-running evaluations, the toolkit has to exhibit a high level of reliability. This includes an automated failure handling and the recovery of failed evaluation runs. (iv) The toolkit has to provide mechanisms for archiving the results of the evaluations as well as of all parameter settings which have led to these results. (v) Already existing evaluation clients should be easily integrable into the toolkit which also has to support developers in building new clients. (vi) The toolkit has to offer a large set of basic analysis functions (e.g., different types of diagrams), support the extension by custom ones, and provide standard metrics for measurements (e.g., execution time). Finally, (vii) it should be easy to apply the toolkit to new SUEs.

In this paper, we introduce Chronos, a generic evaluation toolkit for systematic systems evaluations. The contribution of this paper is twofold: First, we describe the functionality of Chronos which automates the entire evaluation workflow and assists users in the analysis and visualization of the evaluation results. Second, we demonstrate how Chronos can be configured for the evaluation of different SuEs and for the systematic analysis of a complete evaluation space by using two different storage engines in MongoDB[1] as running example.

Since its first version, Chronos has been used for the evaluation of several research prototypes, including ADAM$_{pro}$ [6], Beowulf [9], Icarus [10], and Polypheny-DB [11] as well as in various student evaluation activities. It has been released on GitHub[2] under the MIT open source license.

The remainder of this paper is structured as follows: In Section 2 we introduce Chronos and Section 3 shows Chronos at work. Section 4 discusses related work and Section 5 concludes.

2 CHRONOS

Chronos is a toolkit for defining, monitoring, and analyzing the results of evaluations. It consists out of two main building blocks (see Figure 1): First, *Chronos Control* (in green), the heart of the evaluation toolkit that provides a web UI and a RESTful API for managing evaluations; second, the *Chronos Agents* (in red)

[1] https://www.mongodb.com
[2] https://github.com/Chronos-EaaS

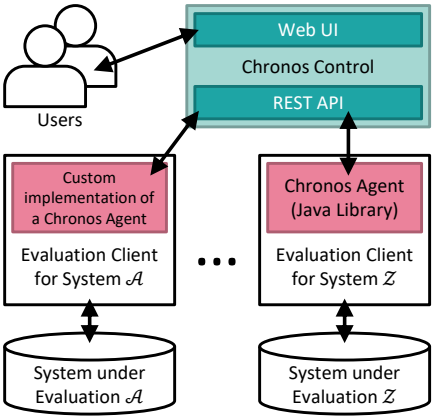


Figure 1: Chronos Toolkit Architecture Overview

which interact with the (existing) evaluation clients and which leverage the REST API to get the required information to perform the evaluations. While it is possible to write a custom Chronos Agent, there is also a generic Chronos Agent available for Java. This allows to easily integrate Chronos into existing projects. In this section, we present the implementation of Chronos and how to integrate it into existing projects.

2.1 Data Model

The data model of Chronos contains projects, experiments, evaluations, jobs, systems, and deployments.

Project. A project is an organizational unit which groups experiments and allows multiple users to collaborate on a specific evaluation. Access permissions are handled at the level of projects so that every member of a project has access to all experiments, evaluations, and their results. Users can archive entire projects, i.e., make their evaluation settings and the results persistent.

Experiment. An experiment is the definition of an evaluation with all its parameters; when executed, it results in the creation of an evaluation. Like projects, experiments can be archived.

Evaluation. An evaluation is the run of an experiment and consists of one or multiple jobs. If the objective of an evaluation is, for example, to compare the performance of two storage engines of a database system for different numbers of threads, every job would execute the benchmark for a specific number of threads for each engine. Depending on the evaluation, the execution of jobs can be parallelized if there are multiple identical deployments of the SuE.

Job. A job is a subset of an evaluation, e.g., the run of a benchmark for a specific set of parameters and a given DB storage engine. The result of every job is stored together with its log output. A job can be in one of the following states: *scheduled*, *running*, *finished*, *aborted*, or *failed*. Jobs which are in the status *scheduled* or *running* can be aborted and those which are *failed* can be re-scheduled.

Result. A result belongs to a job and consists of a JSON and a zip file. Every data item which is required for the analysis within Chronos Control is stored in the JSON file. Additional results can be stored in the zip file (e.g., for further analysis outside of Chronos).

System. A system is the internal representation of an SuE. For every SuE, it is defined which parameters the SuE expects, how the results are structured, and how they should be visualized.

Deployment. A deployment is an instance of an SuE in a specific environment. There can be multiple deployments of an SuE at the same time. Deployments serve two purposes: First, they allow to simultaneously execute evaluations in different (hardware) environments or different versions of the SuE; second, they allow to parallelize the evaluation in case of multiple identical deployments.

2.2 Implementation

In the following section, we give details on the implementation of Chronos' architecture as depicted in Figure 1.

Chronos Control. It is designed as a web application allowing the management and analysis of evaluations using common web browsers. It offers a RESTful web service for clients benchmarking the SuEs. Chronos Control has only a few run-time requirements: Apache HTTP Server[3], PHP[4], MySQL[5] or MariaDB[6], and git[7]. For the SuE extensions, Mercurial[8] is also supported. The provided installation script handles the complete setup of Chronos including the creation of the necessary database schema and user account.

User Interface. Chronos' web UI is based on Bootstrap[9] and comes with an advanced session and role-based user management to support the deployment in a multi-user environment. A key feature of the Web UI is its modular architecture which enables the easy integration of different SuEs. For every SuE, only the available parameters for an experiment and information on how the results are to be visualized have to be specified. Both can be done completely UI-based. Chronos Control therefore provides several parameter and diagram types. Parameter types include Boolean, check box, and value types as well intervals and ratios. For the result visualization, Chronos provides bar, line, and pie diagrams. If more parameter types and diagrams are required, the built-in set of types can be extended by providing an external repository containing PHP scripts with additional implementations.

REST Interface. Chronos' REST API is used for both the clients requesting information to perform their benchmarks (e.g., requesting job descriptions or submitting results) and for the integration of the Chronos toolkit into existing evaluation workflows. For this, the API offers methods to, for example, schedule an evaluation which is caused by a successful build of the SuEs build bot. To support the smooth evolution of Chronos, the API is versioned. This allows new clients to simultaneously use the newly developed features while other clients still use older versions of the REST API.

Chronos Agent. Chronos Agents are clients or client libraries connecting to Chronos' REST API that perform or trigger the actual evaluation workload. Agents are essential to link existing or newly developed evaluation clients of the SuEs with Chronos. An agent can be implemented in any language supporting the

[3] https://httpd.apache.org/
[4] https://secure.php.net/
[5] https://mysql.com/
[6] https://mariadb.org/
[7] https://git-scm.com/
[8] https://mercurial-scm.org/
[9] https://getbootstrap.com/

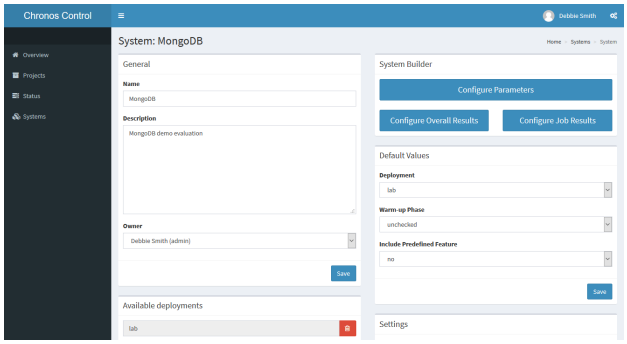


Figure 2: Configuration of a System in Chronos' Web UI

access to the RESTful web service. Along with Chronos Control, we provide a reference implementation of a generic agent library written in Java (also available on GitHub[2]). This reference implementation handles all the communication with Chronos Control including the upload of the results via HTTP or FTP. The latter allows to use a different server or a NAS for storing the results which also reduces the load and storage requirements on the Chronos Control server. During its run, the agent periodically sends the output of the logger to Chronos Control and the agent library allows to easily update the progress of the evaluation. Furthermore, the agent library already measures basic metrics which are returned to Chronos Control along with the results.

Integrating the Chronos Agent library into an existing evaluation client is the only part which requires programming. All other steps can be done completely UI-based. However, the required amount of programming is rather negligible, since the agent library already provides an interface with all necessary methods to be implemented. Depending on the existing evaluation client, this usually narrows down to calling already existing methods of the evaluation client.

3 CHRONOS AT WORK

In this section, we present the two main workflows supported by Chronos: First, the registration of an SuE in the Chronos toolkit, and second, the necessary steps to perform an actual evaluation.

The first workflow demonstrates how easy it is to integrate the Chronos Agent into an existing evaluation client using the Java library. After building a Chronos-enabled evaluation client and setting-up an instance of the SuE, it needs to be registered in Chronos Control. This can be done by either specifying the parameters required for the evaluation client and the SuE in Chronos Control or by providing a path to a git or mercurial repository. Figure 2 shows the overview page of a system in the Chronos Web UI. Once the system has been created and its parameters have been configured, the first workflow is finished and Chronos is ready for executing evaluations against this SuE.

The second workflow starts with the creation of a project. Afterwards, one or multiple experiments are defined (Figure 3a). To schedule work for a Chronos Agent, an evaluation needs to be created which consists of one or multiple jobs as depicted in Figure 3b. Figure 3c shows the overview page of a job providing the current status of the job including its progress and the log output. Furthermore, it allows to abort a scheduled or running job or to reschedule a failed one. The timeline shows all events associated with this job. When the Chronos Agent has finished its work, the evaluation results are visualized (Figure 3d).

The separation of experiments and evaluations comes in handy if certain evaluations need to be repeated multiple times. This is the case during the development of an SuE, for example, in the bug-fixing phase, or for the quality assurance monitoring the performance of an SuE over subsequent change sets.

A demonstration that has been prepared to show Chronos' capabilities considers two workflows using the comparative evaluation of two storage engines of MongoDB (wiredTiger and mmapv1) as an example. This demonstration allows to create short running evaluations for the two MongoDB deployments and to directly analyze the results in the Chronos Web UI.

The MongoDB Chronos agent is available on GitHub[2] and the demo that will be presented at the conference is summarized in a short video[10] that shows how Chronos can be used for the comparative evaluation of the two storage engines of MongoDB.

4 RELATED WORK

Like the Chronos toolkit, the *PEEL framework* introduced in [1] automates the evaluation process and helps improving the reproducibility of evaluations. PEEL, however, is designed with a focus on Machine Learning applications running on frameworks like Hadoop MapReduce, Spark, Flink, and others. In PEEL, the SuE and the experiments are described using XML or Scala classes. These documents are compiled into a bundle containing the PEEL framework, the required configuration, and the evaluation data sets. The bundle is then deployed on the target machine running the SuE.

PROVA! [7] is a distributed workflow and system management tool for benchmarks on HPC systems. It allows to easily benchmark applications on different systems and architectures. Similar to Chronos it visualizes the results. While Chronos is a general-purpose tool for all kinds of evaluations and benchmarks with a focus on database evaluations, PROVA! is specifically tailored to the HPC domain.

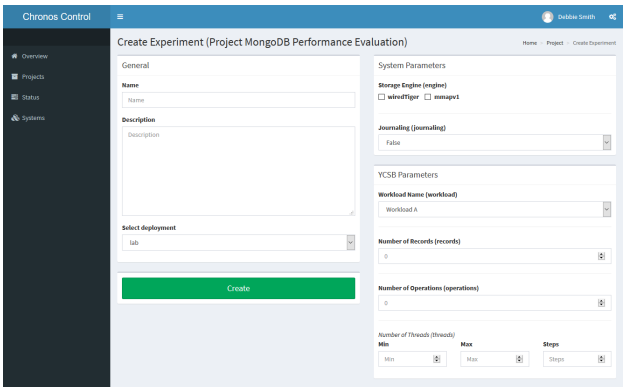
OLTP-Bench [5] is a benchmarking framework which provides implementations for in total 15 different transactional, web-oriented, and feature testing benchmarks including YCSB [4], TPC-C[11], and CH-benCHmark [3]. In contrast to OLTP-Bench, Chronos addresses the complete evaluation workflow and thus also includes the definition of the experiments and the analysis of the results. However, a combination of Chronos (automation of the evaluation workflow) and OLTP-Bench (definition of various benchmarks) would even further facilitate the definition, set-up, execution, and analysis of evaluations. In our future work, we thus plan to develop a Chronos Agent that wraps the OLTP-Bench so as to combine both systems.

The *TREET* testbed [8] allows developers of trust and reputation systems (e.g., online marketplaces) to evaluate their systems using standardized test cases. Further, TREET can be flexibly extended with custom agents and test cases allowing the testing of the developer's application. Like Chronos, but for a different domain, TREET supports the execution of experiments and the comparison between different trust and reputation systems.

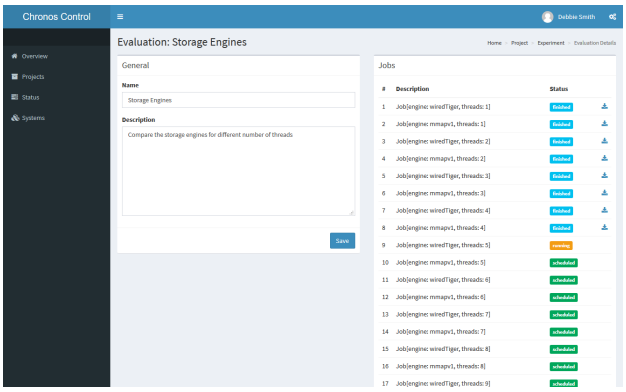
5 CONCLUSION

In this paper, we have presented the evaluation toolkit Chronos, a first step towards the concept of Evaluation-as-a-Service. Chronos automates the entire evaluation workflow and assists users in the results analysis. The Chronos toolkit is available on

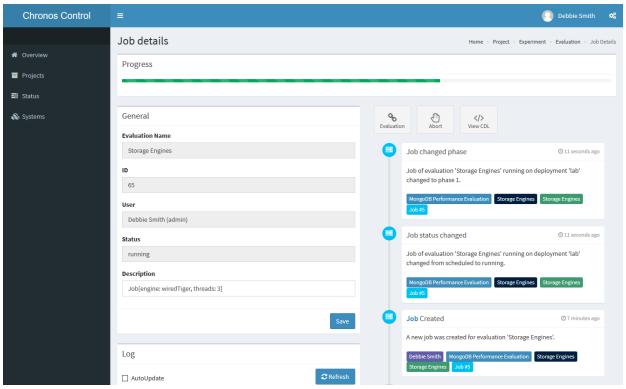
[10] https://youtu.be/fNmsZH4HOl0
[11] http://www.tpc.org/tpcc/



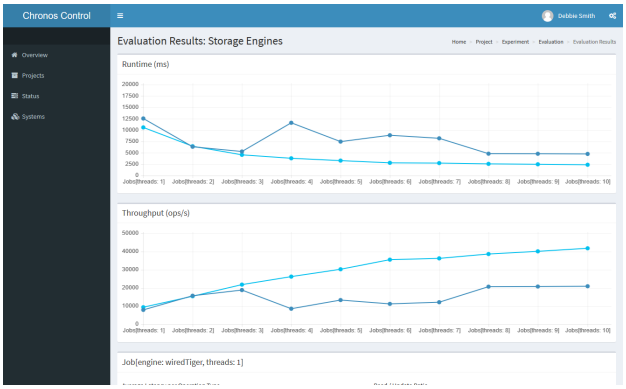
(a) Creation of an Experiment



(b) Details of a Running Evaluation



(c) Details of a Running Job



(d) Basic Result Analysis done by Chronos Control

Figure 3: Basic Evaluation Workflow

GitHub[2]. Future releases of Chronos will be extended with the functionality for setting up the infrastructure of an SuE automatically, for example, in an on-premise cluster or in the Cloud. Also, we plan to release additional reference implementations of agent libraries, for instance for Python.

ACKNOWLEDGMENTS

The work was partly funded by the Swiss National Science Foundation (SNSF) in the context of the project Polypheny-DB (contract no. 200021_172763). The authors would like to thank Filip-Martin Brinkmann and Ivan Giangreco. Their feedback on the evaluation of ADAM$_{pro}$ [6] and PolarDBMS [2] on the basis of Chronos has significantly improved the system. Moreover, the authors like to thank Dina Sayed for her helpful comments on an earlier version of this paper.

REFERENCES

[1] Christoph Boden, Alexander Alexandrov, Andreas Kunft, Tilmann Rabl, and Volker Markl. 2017. PEEL: A Framework for Benchmarking Distributed Systems and Algorithms. In *Performance Evaluation and Benchmarking for the Analytics Era (TPCTC)*. Springer, Munich, Germany, 9–24. DOI: http://dx.doi.org/10.1007/978-3-319-72401-0_2

[2] Filip-Martin Brinkmann and Heiko Schuldt. 2015. Towards Archiving-as-a-Service: A Distributed Index for the Cost-effective Access to Replicated Multi-Version Data. In *Proceedings of the 19th International Database Engineering & Applications Symposium (IDEAS'15)*. ACM, Yokohama, Japan, 81–89. DOI: http://dx.doi.org/10.1145/2790755.2790770

[3] Richard Cole, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, Florian Waas, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, and Thomas Neumann. 2011. The Mixed Workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*. ACM, Athens, Greece, 1–6. DOI: http://dx.doi.org/10.1145/1988842.1988850

[4] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proc. of the 1st ACM Symposium on Cloud Computing*. ACM, Indianapolis, Indiana, USA, 143–154. DOI: http://dx.doi.org/10.1145/1807128.1807152

[5] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the VLDB Endowment* 7, 4 (2013), 277–288. DOI: http://dx.doi.org/10.14778/2732240.2732246

[6] Ivan Giangreco and Heiko Schuldt. 2016. ADAM$_{pro}$: Database Support for Big Multimedia Retrieval. *Datenbank-Spektrum* 16, 1 (2016), 17–26. DOI: http://dx.doi.org/10.1007/s13222-015-0209-y

[7] Danilo Guerrera, Antonio Maffia, and Helmar Burkhart. 2019. Reproducible stencil compiler benchmarks using prova! *Future Generation Computer Systems* 92 (2019), 933–946. DOI: http://dx.doi.org/10.1016/j.future.2018.05.023

[8] Reid Kerr and Robin Cohen. 2010. TREET: the Trust and Reputation Experimentation and Evaluation Testbed. *Electronic Commerce Research* 10, 3 (2010), 271–290. DOI: http://dx.doi.org/10.1007/s10660-010-9056-y

[9] Alexander Stiemer, Ilir Fetai, and Heiko Schuldt. 2016. Analyzing the Performance of Data Replication and Data Partitioning in the Cloud: The BEOWULF Approach. In *IEEE International Conference on Big Data*. IEEE, Washington DC, USA, 2837–2846. DOI: http://dx.doi.org/10.1109/BigData.2016.7840932

[10] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2017. Icarus: Towards a Multistore Database System. In *IEEE International Conference on Big Data*. IEEE, Boston, MA, USA, 2490–2499. DOI: http://dx.doi.org/10.1109/BigData.2017.8258207

[11] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2018. Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore. In *IEEE International Conference on Big Data*. IEEE, Seattle, WA, USA, 3364–3373. DOI: http://dx.doi.org/10.1109/BigData.2018.8622353