Crowd-tuning with GPTune History Database: Design Overview, Data-sharing Policy, and Plan to Maintain Security

Younghyun Cho [*] James W. Demmel [*] Xiaoye S. Li [†] Yang Liu [†] Hengrui Luo [†]

September 2021

The success of autotuning depends on collecting a sufficient number of performance data samples, but collecting performance data is an expensive task especially when tuning HPC codes. To address this challenge, we propose the idea of *crowd-tuning* using the *GPTune history database* that allows users to share performance data with other users at different sites. In this note, we present an overview of the history database and its API, and then discuss our data-sharing policy and plan to maintain security.

1 Design

We provide an open-repository at `https://gptune.lbl.gov` to collect and share performance data in a *reusable*, *reproducible*, and *reliable* way. Collected data is managed using MongoDB [1] and can be converted into JSON [4]. The JSON data format is compatible with our autotuner GPTune [5], and it will not be difficult to use the data for other autotuners and software frameworks. Collected performance data can be used for a variety of scenarios, e.g. achieving the best available parameter configuration for a widely used code, using historical data for transfer learning, building and using a performance model, or solving a challenging tuning problem with multiple users. Figure 1 depicts the idea of crowd-tuning that we envision.

Tuning an HPC code typically means choosing optimal tuning parameters to optimize the performance of the code. All performance data therefore includes the tuning parameter configuration and its evaluated result. That information, however, may not be sufficient for other users who need to run the code on different HPC systems and/or with different compile/link-time dependencies. Therefore, our database also records the machine and software configuration on which the code is evaluated. The challenge of recording such an environmental configuration is how to parse the information (automatically) in a way so that other users can easily query the performance data. In case the HPC code is installed with an automatic tool such as CK [2] or Spack [3], the user can inform the database about which software is installed with CK or Spack, so that our database can automatically parse the software configuration and record it in the database. Otherwise, the user needs to manually describe the software dependency and version information with our JSON format. The repository contains separate databases that already have well-defined information for existing HPC systems (mostly DOE supercomputers) and software packages (e.g. compilers, tools, libraries). The repository can therefore parse the user-provided information internally to match the tag names with the well-defined machine/software information in the database. Users can also add their own machine configuration in the database. In Figure 1, we illustrate how the database handles users' queries according to these principles of downloading and uploading performance data.

For reliability, the repository allows only registered users to upload and record the user information (username and email), but we also provide some options for users who do not wish to disclose their information to other users. Each performance data in the repository can have a different accessibility level, therefore, interested users should register at `https://gptune.lbl.gov/account/signup/`.

[*]University of California at Berkeley (younghyun@berkeley.edu, demmel@cs.berkeley.edu)
[†]Lawrence Berkeley National Laboratory (xsli@lbl.gov, liuyangzhuan@lbl.gov, hrluo@lbl.gov)

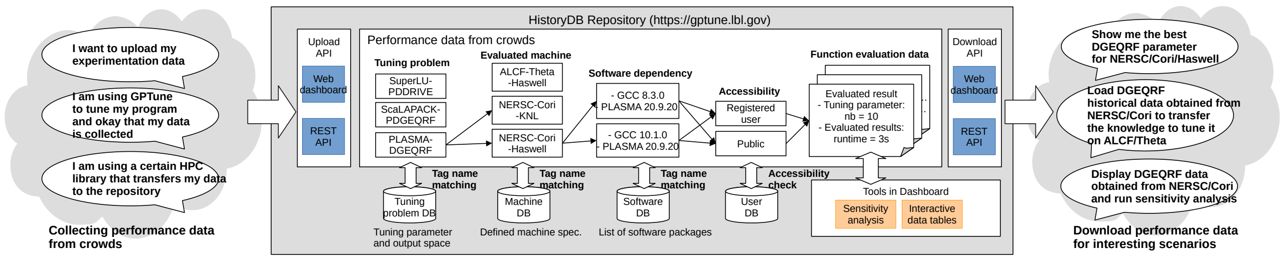
I want to upload my experimentation data
I am using GPTune to tune my program and okay that my data is collected
I am using a certain HPC library that transfers my data to the repository
Collecting performance data from crowds
HistoryDB Repository (https://gptune.lbl.gov)
Upload API
Web dashboard
REST API
Performance data from crowds
Tuning problem
SuperLU-PDDRIVE
ScaLAPACK-PDGEQRF
PLASMA-DGEQRF
Evaluated machine
ALCF-Theta -Haswell
NERSC-Cori -KNL
NERSC-Cori -Haswell
Software dependency
- GCC 8.3.0 - PLASMA 20.9.20
- GCC 10.1.0 - PLASMA 20.9.20
Accessibility
Registered user
Public
Function evaluation data
Evaluated result - Tuning parameter: nb = 10 - Evaluated results: runtime = 3s
Tag name matching
Tag name matching
Tag name matching
Accessibility check
Tuning problem DB
Machine DB
Software DB
User DB
Tuning parameter and output space
Defined machine spec.
List of software packages
Tools in Dashboard
Sensitivity analysis
Interactive data tables
Download API
Web dashboard
REST API
Show me the best DGEQRF parameter for NERSC/Cori/Haswell
Load DGEQRF historical data obtained from NERSC/Cori to transfer the knowledge to tune it on ALCF/Theta
Display DGEQRF data obtained from NERSC/Cori and run sensitivity analysis
Download performance data for interesting scenarios

Figure 1: Proposed crowd-tuning approach.

2 Interface

We offer two interfaces to access the repository: (1) interactive web-dashboard or (2) an HTTPs RESTful API (called crowd-tune API). In what follows, we briefly introduce these interfaces. More detailed examples are given at `https://gptune.lbl.gov/docs/src/historydb_repository.html`.

2.1 Interactive Dashboard

We provide an interactive web dashboard at `https://gptune.lbl.gov/repo/dashboard`, where users can query the available performance data for a specific machine and software configuration(s). The dashboard also provides several useful tools. For example, users can directly query a performance model and use it for sensitivity analysis and performance prediction. To upload data, users can use another dashboard at `https://gptune.lbl.gov/repo/upload`, where they can select the tuning problem and upload performance data in our JSON format (file/text). A more detailed user manual is provided at `https://gptune.lbl.gov/docs`.

2.2 Programmable API

The programmable API is an API to access the repository via HTTPs based on REpresentational State Transfer (REST) principles, which means that the database resources can be identified via a uniform resource identifier (URL) and accessed in various programming languages like C++/Python.

API Format. To download performance data, the user can use a general URL form in the following, where `tuning_problem_name` is the name of tuning problem, `machine/software/user_configurations` contain lists of machine/software/user configurations to download.

```
GET https://gptune.lbl.gov/direct-download/?tuning_problem_name=ScaLAPACK-PDGEQRF
  &machine_configurations=[...]&software_configurations=[...]&user_configurations=[...]
```

Similarly, the general URL form for uploading a function evaluation result is as follows, where `tuning_problem_name` is the name of tuning problem to be queried, and `function_evaluation` is the performance data to be uploaded.

```
POST https://gptune.lbl.gov/direct-update/?tuning_problem_name=ScaLAPACK-PDGEQRF&
  &function_evaluation={...}
```

The function evaluation data contains the tuning parameter configuration and its evaluated outputs, and the machine and software configuration on which the parameter configuration is evaluated. For the JSON format describing a function evaluation data, please refer to our user guide [5] and an online manual at `https://gptune.lbl.gov/docs`.

API Key. Unlike the web-dashboard where the user can login from a web browser, all API requests need to contain an API key with a `x-api-key` header (e.g. {"X-Api-Key": "your_API_key"}). Each user can generate one or more API keys at `https://gptune.lbl.gov/account/access-tokens`. When generating an API key, the user can select a display option whether the user agrees to display their user info or prefers to be anonymous. Note that, users have to manage their API keys securely, because API keys are used instead of passwords. Users have responsibility for any violations conducted using user API keys.

Examples. Our online manual contains some more detailed examples of using the programmable API with API keys (`https://gptune.lbl.gov/docs/src/historydb_repository.html`).

3 Data-Sharing Policy

We will assume that our users agree to our data-sharing policy that includes:

- For a user account, we will collect username (ID), the user's first and last names, and the user email and affiliation in our database. The database records are kept confidential. When uploading data, users can choose whether to disclose their username, affiliation, and email address in the data.
- To assure provenance and avoid uploading bad data, the repository requires access credentials to upload performance data and download data that requires access permission.
- Each performance data can have a different accessibility level: publicly available, private, or sharing with specific users/groups. Users can create specific user groups for data-sharing.
- GPTune will have an optional automatic synchronization mode, where the collected function evaluation data can be automatically uploaded to the repository.
- Users have to manage their API keys securely, and have responsibility for any violations conducted using user API keys.
- There is a possibility that we may delete/alter/modify users' performance data for management purposes (generally we will not need to modify the performance results, but there is a possibility of changing the database structure).

4 Maintaining Security

This section discusses our efforts and plans to maintain security in the database.

- To maintain security, all requests to the repository, either using the programmable API or the interactive dashboard, have to be done over HTTPs.
- Our database, where all user and performance data is stored, uses NERSC's storage and can only be accessed within a workload container at NERSC. In other words, any access to the storage is shielded by authenticating with a certain workload manager Rancher2 [1].
- User password and authentication: A password is required to sign-in. We use Python Django's user authentication module, so we do not record the user's raw password in the database, but only a hash [2].
- User API keys: Since the API keys can be used instead of user passwords, we provide an option to use public and private key pairs for API keys. In this case, the user keeps the private key and we record only the public key in the database.
- We regularly run a web vulnerability scanner (e.g. Arachni [3]) to detect possible vulnerabilities.

[1] `https://rancher.com/`

[2] `https://docs.djangoproject.com/en/3.2/topics/auth/default/`

[3] `https://www.arachni-scanner.com/`

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. We used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

References

[1] K. Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly Media, Inc., 2013.

[2] G. Fursin. Collective Knowledge: organizing research projects as a database of reusable components and portable workflows with common APIs. *arXiv preprint arXiv:2011.01149*, 2020.

[3] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. The Spack Package Manager: Bringing Order to HPC Software Chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[4] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč. Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[5] W. M. Sid-Lakhdar, Y. Cho, J. W. Demmel, H. Luo, X. S. Li, Y. Liu, and O. Marques. GPTune User Guide. `https://gptune.lbl.gov/documentation/gptune-user-guide`, 2021.