Performance Dashboard for Continuous Benchmarking of HPC Libraries

Chingun Ariunbat, Jamil Bagga, Walter Alexander Böttcher, Darius Schefer, Maximilian Schik

2021-05-11

Contents

1 Introduction

Performance Dashboard for Continuous Benchmarking of HPC Libraries

PSE SS21

2 Goals

2.1 Required

heading NICHT IMPLEMENTIERT **M1**

yet more placeholder

Schnelle Weiterleitung Kurz- zu Lang-URL **M2**
Implementiert durch: FR1

Authentifizieren mit E-Mail oder Facebook **M3**
Implementiert durch: FR2

Rechtlichte Vorgaben werden eingehalten **M4**
Implementiert durch: FR3 FR4

template

2.2 Optional

Authentifizieren mit Github **K1**
Implementiert durch: FR2

Seite mit Betreiberinfo keine entsprechende Anforderung **K2**

template

2.3 Limitation

Keine Wahl Kurz-URL **A1**

template

3 Usage

template

4 Product Environment

template

5 Functional Requirements

Schnelle Weiterleitung NICHT GETESTET **FR1**
Implementiert: M2

template

template NICHT GETESTET **FR2**
Implementiert: M3 K1

template

Auf jeder Seite ist ein Link “Impressum” NICHT GETESTET **FR3**
Implementiert: M4

template

Auf jeder Seite ist ein Link “Datenschutz” NICHT GETESTET **FR4**
Implementiert: M4

template

Daten werden persistent gespeichert NICHT GETESTET **FR5**
Implementiert:

template

6 Nonfunctional Requirements

Modernes Design **NF1**

template

Persistenz **NF2**

template

Erweiterbarkeit **NF3**

template

7 Product Data

Benchmark Results (Name in progress) **PD1**

Format: JSON/CSV
Description:

- saved on server
- algorithm result data (time, storage, accuracy, convergence(?))

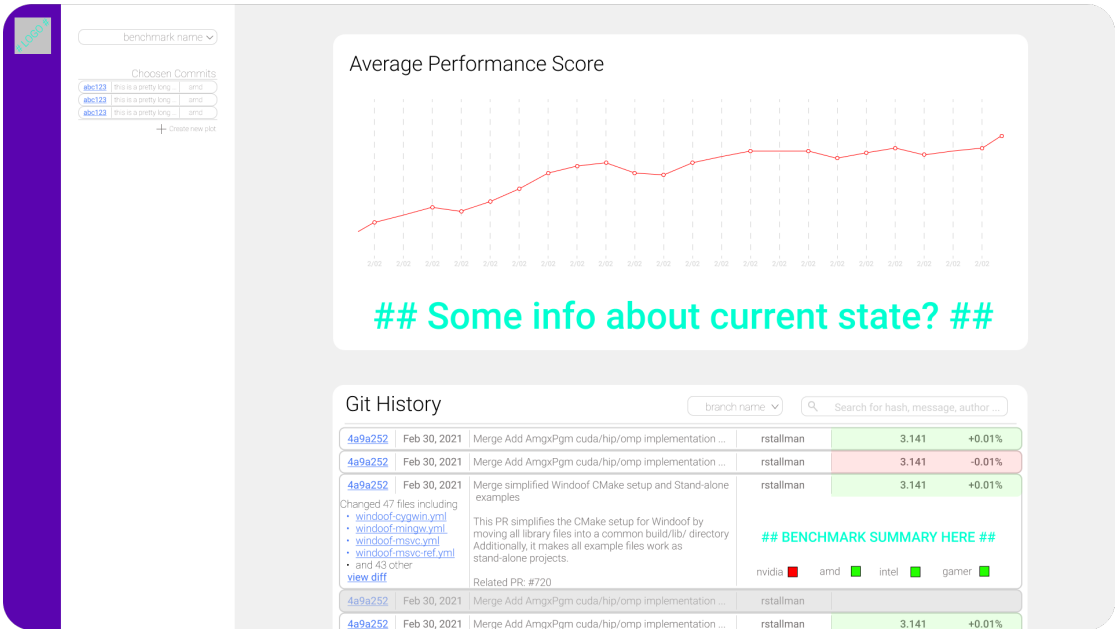
Git Histories **PD2**

Format: ??? (WIP)

Templates **PD3**

Format: JSON (?)

8 User Interface



just a placeholder

9 Tests

10 Scenarios

Scenario name: pushAndInspect
Participating actor: Ted: Developer

- Ted pushes his work to a git repository and fires off a benchmark test
- Ted opens the web app and selects his last pushed change
- Ted chooses a type of visualization
- The app creates the given type of visualization with the benchmark results from the selected change

Scenario name: visualizeFromTemplate
Participating actor: Greta: User

- Greta opens the web app
- Greta chooses a template for a visualization
- Greta chooses which commit she wants to visualize
- The app creates the given type of visualization with the commit

Scenario name: saveTemplate
Participating actor: Greta: User

- Greta opens the web app
- Greta configures a visualization
- Greta saves her visualization as a template for future use

Scenario name: inspect
Participating actor: Greta: User

- Greta wants to see the latest performance benchmarks for the project
- Greta opens the web app and selects the latest change
- Greta chooses a benchmark to compare

- Greta chooses a type of visualization by selecting which value to plot on the x axis and which value on the y axis
- The app creates the given type of visualization with the benchmark results from the selected change

Scenario name: compareImplementations
Participating actor: Greta: User

- Greta wants to know which implementation is the fastest
- Greta opens the web app and selects a benchmark
- Greta selects commits from different branches containing different implementations
- Greta chooses a type of visualization by selecting which value to plot on the x axis and which value on the y axis.
- The app creates the given type of visualization with the benchmark results from the selected change

Scenario name: pushAndCompare
Participating actor: Ted: developer

- Ted pushes his work to a git repository, and fires off a benchmark test.
- Benchmark results are fed into the database.
- Ted opens the web app and selects his last pushed change>
- Ted selects a previous change that he wants to compare to.
- Ted chooses a type of visualization.
- The app creates the given type of visualization with the benchmark results from the selected changes.

Scenario name: badPerformance
Participating actor: Ted: developer

- Ted pushes his work to a git repository, and fires off a benchmark test.
- Benchmark results are fed into the database.

- Our dashboard-backend realizes that the benchmark data for this change is far worse than usual.
- Ted gets notified that his last pushed change significantly worsened the performance and the related details about that.

Scenario name: impossiblePerformance
Participating actor: Ted: developer

- Ted pushes his work to a git repository, and fires off a benchmark test.
- Benchmark results are fed into the database.
- Our dashboard-backend realizes that the benchmark data for this change is theoretically impossible.
- Ted gets notified that his last pushed change has improved the performance above the theoretical maximum and the related details about that.

Scenario name: shareVisualization
Participating actor: Greta: User

- Greta found an interesting visualization for something.
- Greta clicks a *share* button next to the visualization.
- Greta gets a link she can share with others that redirects them to the exact same visualization.

Scenario name: visualizeCommitWithoutBenchmark
Participating actor: Greta: User

- Greta opens the web app and wants to visualize benchmark data for a specific commit. This commit has no benchmark data attached to it, only the commit before and the commit after.
- Greta can't click on the commit because it is greyed out.

Scenario name: takeVisualizationFromHistory
Participating actor: Greta: User

- Greta opens the web app and visualizes something. She then visualizes something else. Her previous visualizations are stored in a list somewhere.

- Greta decides to take another look at a previous visualization.
- Greta picks her previous visualization and gets the previous visualization.

Scenario name: postBenchmarkResults
Participating actor: bencharkCI: CI

- The benchmarkCI processes a benchmark and gets some results.
- The benchmarkCI posts the results to the backend of the system using the API supplied by the system.
- The benchmark results are stored in the backend database system.

11 Use Cases

Use case name: Visualize
Participating actors: initiated by User
Entry conditions: configuration is available
Flow of events:

1. Web app fetches the data specified in the configuration from the backend
2. The web app uses the fetched data to generate a plot according to the specification
3. The generated plot is visualized in the web app

Exit conditions: The plot specified by the configuration gets shown to the User.
Quality requirements: Shouldn't take more than 30 seconds?

Use case name: Compare commits
Participating actors: Server
Entry conditions: benchmark results for all selected commits are available
Flow of events:

1. The backend compares the performance of all selected commits
2. ???

Exit conditions: The server caches the results for later use
Quality requirements: wip

Use case name: InspectChange
Participating actors: initated by User
Entry conditions: benchmark data for commit is available
Flow of events:

1. User selects a single change from the history
2. Server generates comparison `-> CompareCommits`

Exit conditions: Results are displayed in the web app
Quality requirements: If data is present: no more than 30 seconds?

Use case name: SaveTemplate
Participating actors: User, Server
Entry conditions: Web app is currently displaying data
Flow of events:

1. User selects the “save template” option
2. User is prompted for name for the template
3. The Server generates a JSON file representing the template

Exit conditions: Template is stored on the server for later use
Quality requirements: Requires less than 1kB of storage in the database

Use case name: UseTemplate
Participating actors: User, Server
Entry conditions: A template is stored in the database
Flow of events:

1. The User selects the “open template” option
2. User is shown a list of all saved templates
3. User selects a template from the list

Exit conditions: The User’s dashboard is set to the selected template
Quality requirements: ???

Use case name: PostBenchmarkResults
Participating actors: Developer, Server
Entry conditions: The application was benchmarked
Flow of events:

1. The developer makes a POST request to the server containing the new benchmark data in JSON format
2. The Server converts the received data into the correct format
3. The Server stores the received data in the database

Exit conditions: The received performance data is stored in the database
Quality requirements: ???

Use case name: ShareVisualization
Participating actors: User
Entry conditions: A visualization exists already
Flow of events:

1. The User selects the “share visualization” option on the web app dashboard with the visualization on it
2.

Exit conditions:
Quality requirements:

Use case name: ExportVisualization
Participating actors: User
Entry conditions: A visualization exists already
Flow of events:

1. The User selects the “export visualization” option on the web app dashboard with the visualization on the dashboard
2. The web app serializes the data as a JSON file
3. The web app prompts a download of the serialized JSON file

Exit conditions: The User is offered a download of an export of the visualization
Quality requirements: This may not take longer than 30 seconds. (Probably.)

Use case name: ImportVisualization (probably)
Participating actors: User
Entry conditions:
Flow of events:

1. The user selects the “import visualization” option on the web app
2. The web app promps the user to upload a JSON File. (or to paste it?)
3. The web app parses the data within the JSON File, and displays it on the web app.

Exit conditions:
Quality requirements:

Use case name: Notify
Participating actors: Server, User
Entry conditions: A comparison has finished (see compare commits)
Flow of events:

1. The server has finished the comparison of the latest commit with benchmark data with its predecessing commit with benchmark data.

2. The server fires a POST Request to all of its webhook subscribers, containing a notification that the latest change was negatively impacting the performance, a bried description of how it did so, and a link to the web app for immediate review.

3. The user sees this notification on its webhook service (e.g. Discord).

Exit conditions: The server fires a POST Request to all webhook subscribers
Quality requirements:

Use case name: UseCaseTemplate
Participating actors: Actors
Entry conditions: Entry cond.
Flow of events:

1. Flow 1

2. Flow 2

Exit conditions: Exit cond.
Quality requirements: Quality Requirements(?)

Glossary

CI Continuous Integration.

developer Person working on the project that is to be benchmarked.

JSON JavaScript Object Notation.

template Configuaration of a visualization.

visualization (WIP) uhhh... graphs and stuff on one website?.