

# JMH Reader

Giordano Tinella

Advanced Verification and Validation (2024)

## Introduction

### The goal

The goal of this project is to create an environment that allows users to upload JSON files generated by JMH and visualize the summarized data for each benchmark added to the system. This will enable users to easily review all the information related to specific commits.

### Java Microbenchmark Harness (JMH)

JMH (Java Microbenchmark Harness) is a framework developed by the OpenJDK community for creating, running, and analyzing microbenchmarks in Java. It is specifically designed to measure the performance of small code snippets or components within the Java Virtual Machine (JVM).

The JSON file generated using JMH can include various types of information. This ranges from environment-specific details, such as the JVM version and computer specifications, to more technical aspects, like the number of forks used for performance measurement and parameters that can vary depending on the context.

## Instructions for working with the tool

### Working with JMH

First, you need to generate the JSON files for a specific benchmark. To do this, you can use the following GitHub repository: <https://github.com/eclipse/eclipse-collections>. I recommend using version 10.0.0. Once you have cloned the repository, set up the environment by following these steps:

#### Install dependencies:

- `mvn install -DskipTests -am -pl eclipse-collections,eclipse-collections-api,eclipse-collections-forkjoin,eclipse-collections-testutils`
- `mvn install -DskipTests -am -f jmh-scala-tests/pom.xml`

#### Build jmh-tests:

- `mvn package -DskipTests -am -f jmh-tests/pom.xml`
- list the available benchmarks: `java -jar jmh-tests/target/microbenchmarks.jar -l`

#### Generate the JSON file:

- `java -jar jmh-tests/target/microbenchmarks.jar org.eclipse.collections.impl.jmh.LongIntMapTest.put -rf json -wi 1 -r 1 -f 1 -i 1 -bm ss`
- The previous command will generate a JSON file named “jmh-result”

\* The JSON generation can be customized to meet your needs (the benchmark used for this example is “LongIntMapTest”, but you can use any benchmark you prefer). This repository includes a PDF named “parameters” that lists all the available options for generating the JSON.

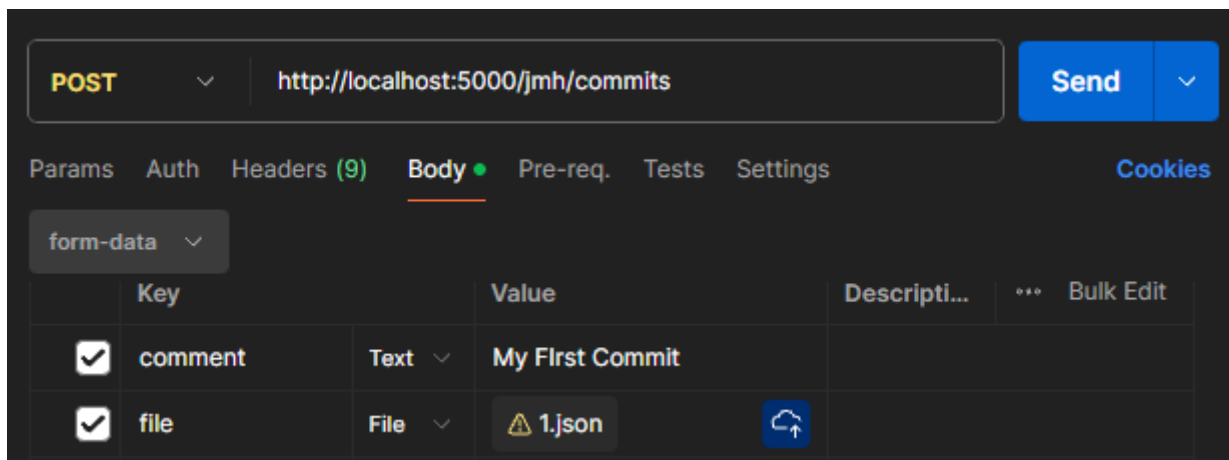
## Working with JMH-Reader

Run the app:

- Clone the repository “<https://github.com/GiordanoT/jmh-reader>”
- Run the command “docker compose up -d” in the root directory
- Wait a few minutes for the app to start up

Add a commit:

- DO an http request POST on “http://localhost:5000/jmh/commits”, the payload should contain:
  - comment (a comment for the commit)
  - file (the json file generated using JMH)



The screenshot shows a REST client interface with a POST request to `http://localhost:5000/jmh/commits`. The request body is set to `form-data` and contains two fields:

	Key		Value	Descripti...	...	Bulk Edit
<input checked="" type="checkbox"/>	comment	Text	My First Commit			
<input checked="" type="checkbox"/>	file	File	1.json			

Visualize the dashboard:

- Go to “<http://localhost:5001>”

## Into the APP

### Dashboard

The homepage is designed to display benchmarks added to the system and allow visualization of each single commit when the eye button is clicked.

Dashboard

Benchmarks

 org.eclipse.collections.impl.jmh.LongIntMapTest.put

 org.eclipse.collections.impl.jmh.MaxTest.serial\_lazy\_jdk

### Benchmark


On the Benchmark page, you can view all the commits added to the system. Each commit includes a delete button, allowing you to remove specific commits if needed. Since JMH can use different parameters, there may be multiple blocks of statistics data for each commit.


Benchmark: **org.eclipse.collections.impl.jmh.MaxTest.serial\_lazy\_jdk** (4 commits)


- #1 Comment: 1 JDK: **1.8.0\_361** Forks: 2 Iterations: 3 Threads: 1 Mode: **ss** Unit: **s/op** 
- #2 Comment: 2 JDK: **1.8.0\_361** Forks: 2 Iterations: 3 Threads: 1 Mode: **ss** Unit: **s/op** 
- #3 Comment: 3 JDK: **1.8.0\_361** Forks: 2 Iterations: 3 Threads: 1 Mode: **ss** Unit: **s/op** 
- #4 Comment: 4 JDK: **1.8.0\_361** Forks: 2 Iterations: 3 Threads: 1 Mode: **ss** Unit: **s/op** 

Parameters: No Parameters

All forks statistics

 General Distribution

 Statistics

 Scatter Plot

Single forks statistics


 FORK 1

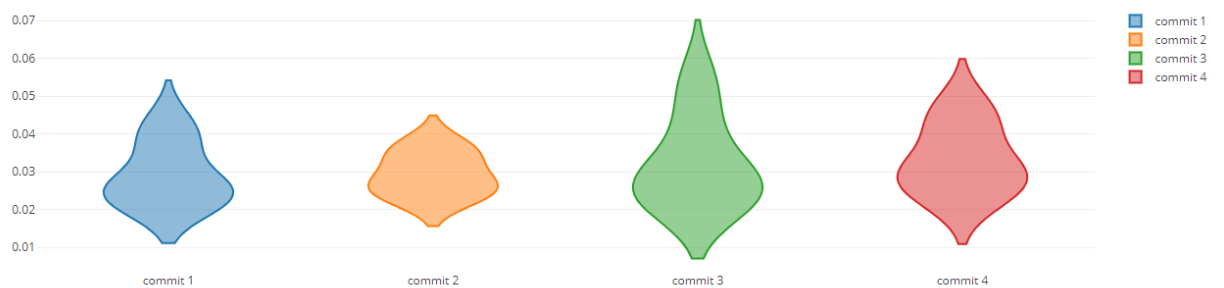
 FORK 2

## Data Section

In the first block of the commit data, you can view general statistics, including the overall metrics for each commit across all forks. In the second block, you can examine detailed behavior of each commit within specific forks. The data is divided as follows:

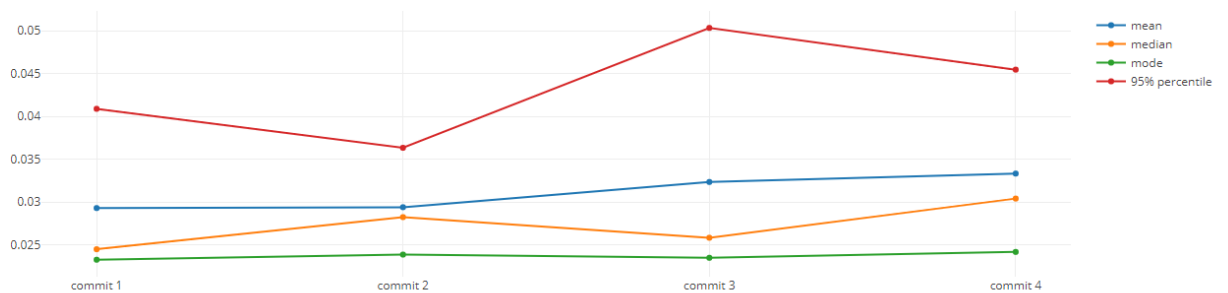
- **General Distribution:** Here you can examine multiple violin charts, each representing the distribution of a commit across all forks. These charts allow us to visualize a range of metrics, such as minimum, maximum, lower fence, and upper fence, providing a comprehensive view of the commit's performance.

 General Distribution



- **Statistics:** Here you can review various metrics for each commit across all forks, including the mean, median, mode, and 95th percentile.

 Statistics



- **Scatter Plot:** Here you can visualize the distribution of each commit across different forks. This enables a better understanding of a commit's behavior in each fork and facilitates comparison between commits across forks.

 Scatter Plot



- **Fork #:** Here you can examine the distribution of all commits for a specific fork, displayed using violin charts.

 FORK 1

