Performance Technical Report

`libraryapi`

Group 4: Isaiah Andrews, Joel Regi Abraham, Salah Salame, Jonathan Taylor

Applied Computer Science & IT, Conestoga College

CSCN 73060: Project VI: Software Efficiency and Performance

Gurpreet Kaur

February 20, 2025

Contents

Abstract:

Throughout this project, we developed a webserver using the CROW framework, and the C/C++ programming languages. The purpose of this project was to evaluate performance under measured workloads and assess whether or not our system is performing adequately. Our results show that initially, the root request had significant load since it fetched for all resources within our database, whereas atomic operations (`POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`) deal with a particular entry at a time. To solve this, we implemented pagination, where only a fixed amount of resources can be gathered with a single `GET` request. This has shown significant improvement in our response times and reduced the overall load on the server.

Introduction

Performance testing evaluates how a system will perform in terms of stability and responsiveness under a particular workload. This project evaluates the role of performance testing within a production environment, where we measure the threshold our system can handle and look for any ways of improvement.

Test Design

The testing for the webserver was conducted in Apache JMeter, covering each "verb" within our API, and made use of graphs to better visualize the data.

- **libraryapi_TestPlan**
  - ↓ **Thread Group**
    - `Root` Request
    - `POST_Add_Book`
    - `PUT_Update_Book`
    - `PATCH_Redescribe_Book`

* DELETE_Delete_Book
* OPTIONS
* Graph: Latencies Over Time
* Graph: Transactions per Second
* Results Tree

API Design

The webserver was implemented using the CROW framework, written in C/C++, where we made use of parametrized URLs to both store and retrieve the right book at any given time. The WEB API was tested against a high volume of users over a fixed interval of time.

HTTP Verb	**Route**	**Description**
GET	/	Displays all the books from the database
POST	/add	Submit a new book record
PUT	/update/{id}	Updates a book record by ID
PATCH	/redescribe/{id}	Updates only the description field for a record by ID
DELETE	/delete/{id}	Deletes a book record by ID
OPTIONS	/	Displays the available verbs/routes of the web server

Table 1: HTTP API Endpoints

Results

For the execution of performance test plan we used JMeter. Our test plan covered each verb that our API was configured to handle. GET, POST, PUT, PATCH, DELETE, OPTIONS and a request for the root URL. Our test plan had two listeners to collect the test data and create graphical representations: Response Latency and Transactions per Second.

Our initial tests showed a significant deviation from our root request when compared to our other methods, where we conclude that since it is attempting to access all the

books stored within the database, the response times will be longer, and are directly tied to the amount of books we have to retrieve.

After adding pagination to the web-server, we were able to make the response times more consistent by not querying all the books on record, and instead query a fixed amount at a time from our database. This removes additional overhead on the webserver and allows for load-balancing on our resources for better management.

First Run

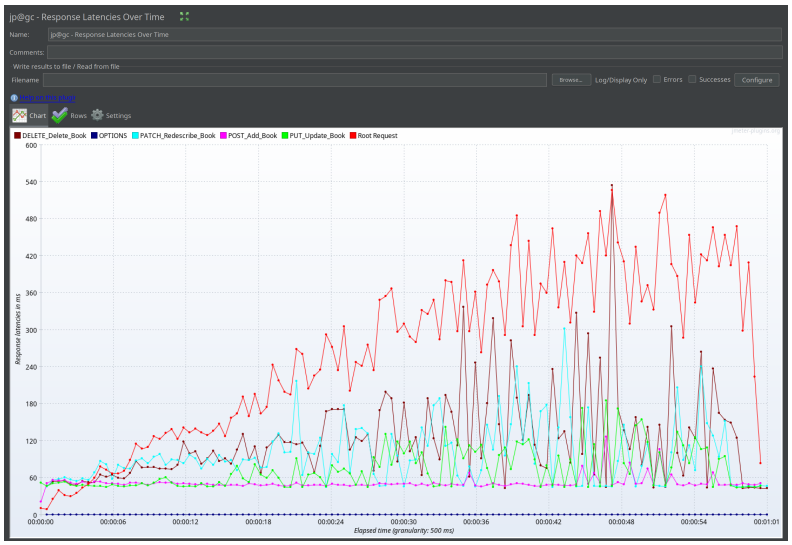

Figure 1: **Response Latency Graph**

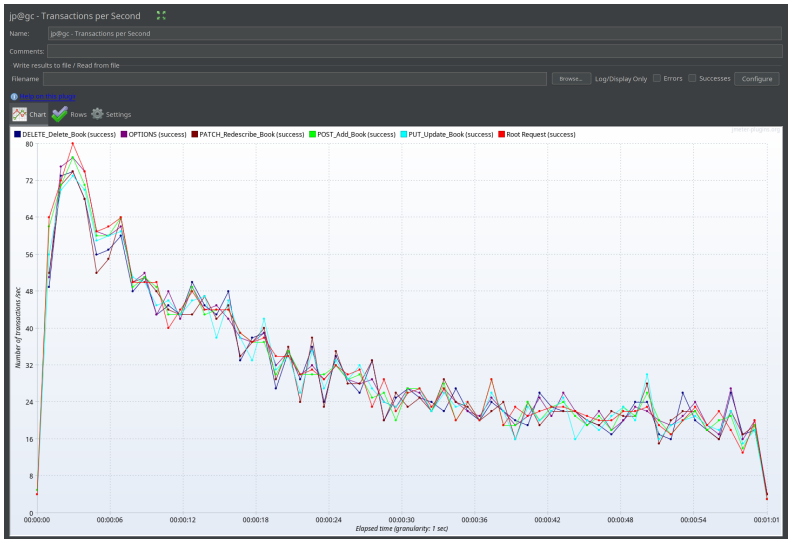

Figure 2: **Transaction per Second Graph**

Second Run

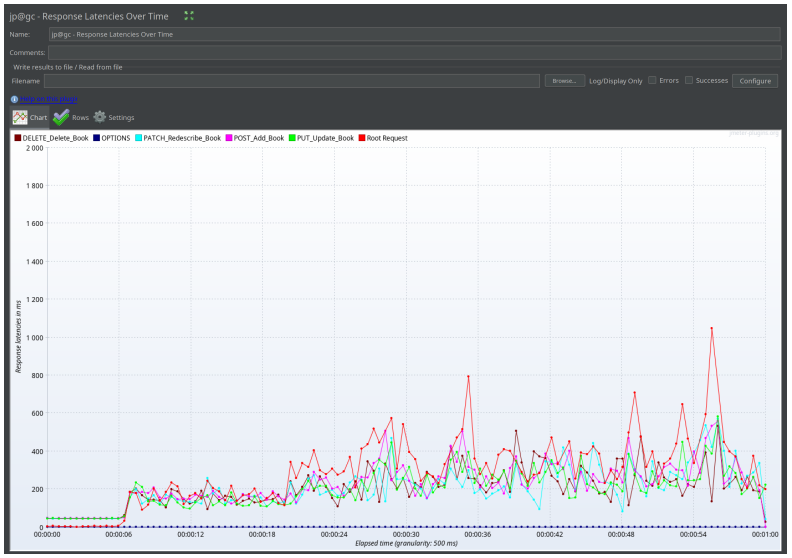

Figure 3: **Response Latency Graph**

Figure 4: **Transaction per Second Graph**

Conclusion & Future Work

Performance testing is a collaborative effort where we measure our compliance with requirements, and business needs, ensuring that information systems are able to perform on day to day business operations. Testing ensures that potential bugs, or performance bottlenecks are accounted for and dealt with at due time, where it can significantly reduce costs, and aid with risk avoidance.