Tool to monitor Thread Pool of Spring Web Application

Han Yongjun[1], Kim Kyeonghyeon[2], Yoon Jaehwan[3], and Kin Junyoung[4]

[1]Sungkyunkwan University, Department of Civil Engineering, Computer Science Engineering
[2]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering
[3]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering
[4]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering

December 12, 2024

Abstract

Monitoring and visualizing the performance of web applications is critical to understanding resource utilization and system behavior. Existing tools provide comprehensive solutions for capturing and analyzing system metrics, but often fail to provide detailed insights into individual thread performance, especially in terms of memory usage. This report presents the development of a new visualization tool designed to improve existing monitoring approaches. This graphical approach allows users to intuitively monitor real-time memory usage and thread status.

1 Introduction

Efficient management and monitoring of application performance have become paramount. With the increasing complexity of software architectures and the need for real-time data analysis, developers are seeking solutions that provide actionable insights into applications. In Korea, Java holds a significant market share of 35%, with the Spring framework playing a dominant role in the backend development environment. Approximately 72% of Java developers use Spring as the basic framework for key applications running on Tomcat servers based on thread pool skills. This proposal outlines state-of-the-art tools designed to improve monitoring capabilities of Spring applications through real-time thread

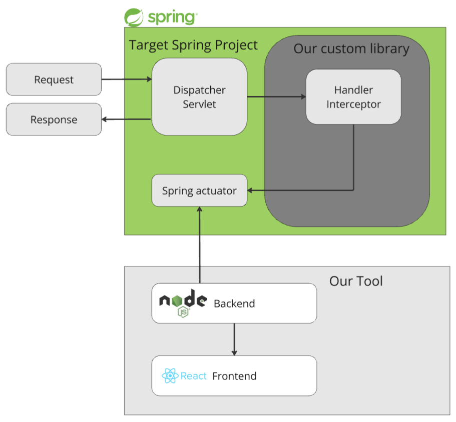


Figure 1: Overall architecture

pool visualization. The tool combines Spring Actuator, a powerful module that exposes operational information about applications, and React, which quickly updates the DOM component in the face of many state changes in threads, allowing developers to monitor critical metrics such as memory usage and thread activity in real time. This not only provides a deeper understanding of application behavior, but also helps identify performance bottlenecks and optimize resource utilization.

2 Design

2.1 Architecture Design

The overall architecture depicted in figure 1, composed of 3 components: Library, front-end, and back-end. Each component has their role: collecting data, managing the data, and interacting with user. The library is declared by the user in the user's spring web application. This library is for collecting data of the Spring web application's data, which is specific data(memory usage, uri, error, executingTime) of each thread running. Through this library, the data collected is loaded on endpoint created by Spring Actuator. Our tool fetch this data and show to user. As our tool is bifurcated into two components: backend and frontend, backend can focus on managing the data and frontend can focus on interacting with high cohesion and loose coupling.

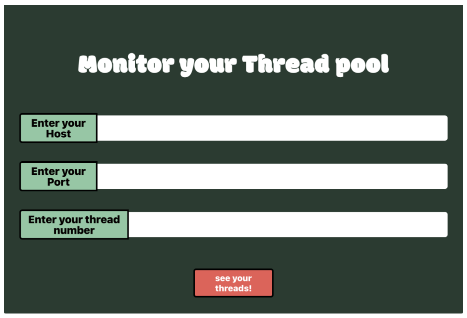
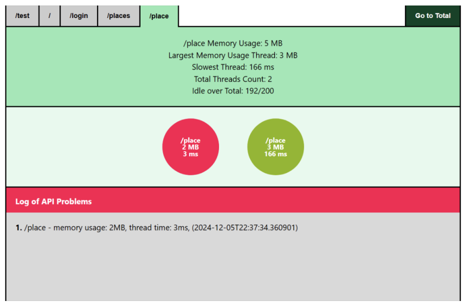


Figure 2: Initial Page



/place Memory Usage: 5 MB
Largest Memory Usage Thread: 3 MB
Slowest Thread: 166 ms
Total Threads Count: 2
Idle over Total: 192/200

```
1. /place - memory usage: 2MB, thread time: 3ms, (2024-12-05T22:37:34.360901)
```

Figure 3: Main Page

3 Implementation

3.1 UI/UX Design

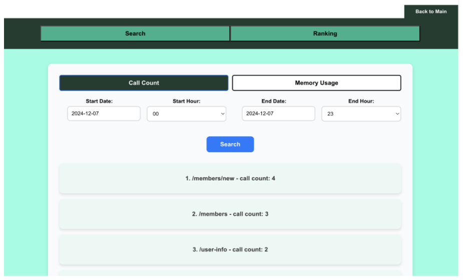
Followings are explanation of the representative User Interface and the corresponding User Experience applied to our service.

3.1.1 Initial Page

The Start Page(Figure 2) allows user to input key parameters such as host, port, and thread numbers of user's Spring application.

3.1.2 Main Page

The Main Page(Figure 3) focuses on visualizing thread pool activity in real-time. Users can monitor various metrics such as thread execution, memory usage, and call counts. This visualization helps users quickly identify which API's memory usage is large, which is likely to cause large resource usage of the server. On the top, we can switch between categories cut between the leading slash and the upcoming slash. On click, total memory usage, largest memory usage thread, slowest thread, and total active threads of the category are given.



Start Date: 2024-12-07
Start Hour: 00
End Date: 2024-12-07
End Hour: 23

```
1. /members/new - call count: 4
2. /members - call count: 3
3. /user-info - call count: 2
```

Figure 4: Summary Page

Idle thread count is also displayed beneath all information. In the middle, we have threads in bubbles. Red indicates erroneous bubble and green indicates a well running thread. At the bottom, list of erroneous logs are shown with detailed information. We can click the button on the top right corner to direct to the total page.

3.1.3 Summary Page

The Summary Page enhances user experience by offering comprehensive search and ranking functionalities. Users can specify a date and time range to analyze specific metrics like call counts and memory usage. Additional features, such as viewing average or maximum memory usage, provide flexibility and detailed insights for more effective analysis.

3.2 Library

As the library should be declared on the Spring application, the library is implemented with Java. We overrided Handler Interceptor, an interface that allows developers to intercept requests and responses during the processing lifecycle. It provides hooks for executing additional tasks at various points in the request handling process.

Key methods of Handler Interceptor.

preHandle(). This method is called before the request is processed. It can be used to validate requests or check authentication informtion. If this method returns false, the request handling is aborted.

postHandle(). This method is invoked after the handler is executed but before the view is rendered. It can be used to midofy data or perform additional processing.

afterCompletion(). This method is called after the request handling and view rendering are complete. It is useful for cleaning up resources or logging.

Library's goal is to get data(uri, memory usage, execution time, error) of each request. For uri, we use preHandle() method by reading request object. For memory usage, we use afterCompletion() method, using ThreadMXBean, which is The management interface for the thread system of the Java virtual machine. For error, we use postHandle() method. As this method is called only when task is done successfully, we add new attribute on request object(isError = true) and set this value as false on postHandle() so that we can distribute if there is error. Lastly, for executionTime, we calculate it by (time when preHandle() is called) - (time when afterCompletion is called). Additionaly, if there are too much data on the Actuator endpoint, time for backend to fetch data might take too long. Therefore, data on the endpoint is reset every 5 seconds.

3.3 Backend

node.js node.js is a runtime environment that allows developers to build highly scalable applications using JavaScript. node.js has low overhead because of its single threaded nature. Its drawback is that there are lots of overhead for managing lots of requests. However our backend requires only one client : user. Therefore drawbacks of node.js doens't count on our backend. Therefore, our backend is implemented by node.js .

Ranking service For ranking service, choosing a sorting algorithm is important factor for time complexity and also space complextiy. The built-in function of Javascript, sort(), was used to filter the API. The sort function uses the 'timsort' method with O(nlogn) time complexity. This algorithm is a combination of merge sort and insertion sort. Therefore this algorithm takes advantages : stable and efficiently aligned according to data size, fast when there is little. The advantages of mergesort are stable and efficiently aligned according to data size, and the advantages of insertionsort are very fast when there is little or almost no data. The criteria for small and large amounts of data are divided into 32.

File management As the data on Spring Actuator endpoint of user's Spring application is being reset every 5 seconds, we need to save the data which are eliminated. Therefore, our backend manages those data as file, and data are saved as json.

RESTful API A REST API is an application programming interface (API) that follows the design principles of the REST architectural style. REST is short for representational state transfer, and is a set of rules and guidelines about how you should build a web API. The communication with frontend has been implemented as a RESTful API, ensuring clear separation of roles and operation through a consistent interface.

json The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects. The backend manages, communicate with frontend the data as json format.

3.4 Frontend

React React is a JavaScript library for building user interfaces, focusing on creating reusable components and managing state efficiently. It enables developers to build dynamic, fast, and scalable web applications. The requirement for the frontend of our tool is component-based and easily update-able. React with component-based architecture and virtual DOM is the best fit for frontend implementation. The frontend is consisted of the following three pages.

Start Page 1. Functionality: This page asks the client to input the host, port and total number of threads for the targeted Spring Boot project.

2. Data Fetching: Fetch from the backend(¡backend¿/api) regarding custom endpoint of the actuator to retrieve data of the host at port.

3. Navigation: Retrieve data and input (total number of threads) are passed down to Main Page.

Main Page 1. Functionality: This page displays the status of each API categorized by the first split of the URI between the slashes.

2. Data Fetching: Fetch from the backend(¡backend¿/api/error) regarding cumulative error logs of APIs.

3. Layout: Information is presented in the vertical order of categories (horizontally scrollable), category summary (plain text), active threads in the category (vertically scrollable), error logs (vertically scrollable).

4. Visualization: active threads in the category are colored red for erroneous ones and green for well-running ones.

5. Navigation: Data passed down from the Start Page is passed down to the Summary Page.

Summary Page 1. Data Fetching and Pagination: Fetch from the backend regarding the cumulative data within specified range. To prevent overly heavy communication between front and back while fetching due to huge load of data,

data are pre-sorted in backend and sent to the front 10-by-10 by asking the corresponding page number.

2. Layout: Information is presented with the tab to switch between search engine and ranking.

2.1 Search layout: Information is presented in the vertical order of search tab, search specification (date and time, horizontal elements), search button, search results (10 vertical elements), and page navigation.

2.2 Ranking layout: Information is present in the vertical order of ranking type (call count, memory usage), seach specification (date and time, horizontal elements), search button, search results (10 vertical elements), and page navigation.

3. Functionality: This page displays the cumulative data in ascending or descending order within input, specified range.

4. Navigation: Data passed down from the Main Page is passed back to the Main Page.

4 Evaluation

We evaluated this tool by testing the performance degredation and overhead due to the file size which is managed by backend part.

4.1 Library

To evaluate the performance of our library, we tested for single request, and load test, for 1000 requests in 1 sec. For several APIs of sample Spring applications, we got the result under (Table 1) , and we repeated 10 times per each applications. The goal of this test is to find out the difference between performance when the library is used and not used. The test with both situation was progressed in same environment and to minimize the influence of the network, we set Spring application running locally. The reason for the result : sometimes slow, sometimes faster is because its performance was overshadowed by network traffic, which means the library's affect for the Spring applications is meanless.

Test	Performance Degradation
Single Request	too small to measure (sometimes slow, sometimes faster)
Load Test (1000 requests in 1 sec)	same as above

Table 1: Evaluation for library

4.2 Reading File

As we saved data of Spring application, which is managed as file by the backend of our tool, size of the file is critical factor for our tool's performance. For example, if there is too much data in file, when frontend send request to backend for searching service, the time for backend to read file and filter might be very slow. As a result, we conducted a test to findout performance degradation related to the size of file and results are on Table 2.

Test	Response time
1000 data in file	5 ms
10,000 data in file	40 ms
100,000 data in file	200 ms

Table 2: Evaluation for reading file

5 Limitation

We found several limitations of this tool and they are under.

5.1 Monitoring purpose only

Our tool is specifically designed for monitoring thread pool performance, which restricts its ability to modify the target project in real time. While the system provides detailed visualizations and insights into thread pool behavior, it does not offer direct intervention or configuration capabilities. This non-intrusive approach ensures that the tool does not interfere with the stability of the target system during execution, but it also means that developers must manually implement any necessary optimizations based on the insights provided.

5.2 Performance Degradation

When processing large volumes of data, the tool encounters performance challenges. Visualizing extensive datasets can lead to delays and increased response times, affecting the user experience. As the amount of monitored data increases, the system's efficiency decreases, raising concerns about scalability. To improve performance, future iterations of the tool will need to incorporate more efficient data processing and rendering techniques.

5.3 Limited Insight into Active Threads

The tool provides a total count of active threads, but it does not differentiate between custom application threads and basic Spring framework threads. This blending of thread types makes it difficult to identify specific performance issues related to individual threads. Additionally, the lack of detailed insights into each thread limits the depth of analysis available to developers. Addressing this limitation would require deeper integration with Spring's thread management system, which may conflict with the tool's monitoring-only design philosophy.

6 Related Work

We analyzed 3 popular tools to monitor backend server and comparison with our solutions are below. We described drawbacks of previous tools on the lefthand and how our tool covered that problem on the righthand.

6.1 Prometheus

Prometheus	Our tool
No specific data for each thread	We provide specific data for each thread.
Must be associated with other visualization programs.	There is no need for external programs.

Table 3: Comparison with Prometheus

6.2 Data Dog

Data Dog	Our tool
Heavily priced plans for large scale	Free

Table 4: Comparison with Data Dog

6.3 Graphite

Graphite	Our tool
Constraints in real-time monitoring and alerting.	Serve data as Serve data as a real-time
Steep Learning Curve for Queries	Doesn't need any learnings.

Table 5: Comparison with Graphite

7 Conclusion

This tool well visualizes the per-thread status and ranking within specified range along with custom dependency on Spring project. Although target projects with heavy data may not be suitable due to performance degradation, the tool aims to support the first step for new and light users to get use to the metrics. Rather than tight lines and long texts, handy design with short texts and bubbles with colors ease the data analysis. Next step for the tool is to grow up with the new and light users. This tool is expected to expand its functionality on handling heavy data by optimizing the transfer between the Spring actuator, backend, and frontend, then show even more detailed information such as running, waiting, or terminated state of each active thread. Ultimately, the tool aims to become the leading program that focuses on handy shape and color-based visualization for Spring projects.

8 References

1. Spring. "Spring Framework." [Online]. Available: https://spring.io/. [Accessed: Dec. 12, 2024].
2. GitHub. "The State of Open Source and AI." [Online]. Available: https://github.blog/news-insights/research/the-state-of-open-source-and-ai/. [Accessed: Dec. 12, 2024].
3. JetBrains. "Java Developer Ecosystem 2023." [Online]. Available: https://www.jetbrains.com/lp/devecosyst
2023/java/. [Accessed: Dec. 12, 2024].
4. MDN Web Docs. "Array.prototype.sort() - JavaScript — MDN." [Online].
Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global$_{O}bjects/Array/sort.$[
$Dec.$12, 2024].