

"Method Invocation Monitoring in PDFBox"

Prepared for

Prof. Benoit Baudry

Prepared by

Ali Maher

7th of Oct. 2024

Abstract

This report presents a case study of **monitoring method invocations** and system properties in the "**PDFBox**" project, based on a task from the RICK paper. The goal was to track runtime method calls from the "**PDFBox**" project and its third-party dependencies, alongside real-time system properties such as CPU usage and memory consumption. Using **AspectJ** for monitoring method invocations, and JMX (Java Management Extensions) for system metrics, the "**PDFBox**" commands *export:text*, *encrypt*, and were executed 100 times to analyze their impact on system resources. The setup involved integrating a real-time CPU usage chart using JFreeChart, providing live feedback on system performance during command execution. This report details the method used, the challenges faced, and the results obtained, offering insights into the runtime behavior and performance of "**PDFBox**" under repeated workloads.

Contents

Introduction	3
Methodology	4
• Phase 1 (Environmental Setup)	4
• Phase 2 (Repository Setup and Cloning)	4
• Phase 3 (Method Invocation Monitoring with AspectJ)	5
• Phase 4 (System Properties Monitoring with JMX)	6
• Phase 5 (Repeating Commands for Performance Analysis)	7
Build and Running Process	8
Results	9
Personal Insights	11
Other	11
Conclusion	11
References	12

Introduction

In the field of document management, working with PDF files programmatically is a common requirement. Apache PDFBox is an open-source Java library that allows for the creation, manipulation, and extraction of content from PDF documents. It is widely used for tasks such as text extraction, encryption, image export, and more, making it a powerful tool in enterprise-level PDF processing.

This project is based on a task outlined in the RICK paper, which focuses on analyzing method invocations and system performance during the execution of Java projects, specifically those that depend on third-party libraries. PDFBox was chosen as the case study, primarily because it heavily relies on external libraries like Log4j2 for logging and BouncyCastle for cryptography, making it a suitable candidate for runtime method invocation tracking.

The main objective of this project was twofold:

1. **Method Invocation Monitoring:** Using AspectJ, the goal was to monitor the methods invoked by PDFBox and its dependencies during the execution of various commands, such as `export:text`, `encrypt`, and `export:images`. This would provide insights into the internal workings of the PDFBox library and its interactions with third-party libraries.
2. **System Properties Monitoring:** Alongside method invocation tracking, it was crucial to monitor system properties, including CPU usage and memory consumption, during the execution of these commands. This was achieved through Java Management Extensions (JMX) and integrated with JFreeChart to visualize real-time CPU usage in a graphical format.

To ensure thorough testing, commands were executed 100 times in a controlled environment. This repetition allowed for the gathering of sufficient data to analyze the impact of PDFBox's execution on system resources under repeated workloads.

The remainder of this report covers the approach used to monitor both method invocations and system properties, the challenges faced during implementation, and the results obtained from the experiments. By combining AspectJ and JMX monitoring techniques, this study provides valuable insights into both the functional and performance aspects of PDFBox.

Methodology

• Phase 1 (Environmental Setup)

To ensure the experiment ran efficiently and without compatibility issues, all development tools and dependencies were updated to their latest versions. This was critical for ensuring smooth integration of libraries like **AspectJ**, **JMX**, and **JFreeChart**, as well as for handling the repeated execution of resource-intensive **PDFBox** commands. Below are the key components of the development environment:

- **Integrated Development Environment (IDE):**
The project was developed using **IntelliJ IDEA**, which was updated to the latest version at the time of the experiment.
- **Java Development Kit (JDK):**
The **Java Development Kit (JDK)** was updated to the latest stable release, **JDK 23**, ensuring compatibility with the project's dependencies and libraries.
- **Apache Maven:**
Maven was used to manage the project's dependencies and build process. The Maven version was updated to **3.9.9**, allowing seamless integration of external libraries like **AspectJ** and **JFreeChart** through the project's `pom.xml` file. Maven also facilitated the handling of build automation for the repeated execution of commands.
- **Libraries:**
 - **AspectJ:** The **AspectJ** version used was compatible with **JDK 23** and integrated through Maven.
 - **JFreeChart:** The **JFreeChart** library was used for real-time charting of system properties and was added as a dependency in the Maven configuration.
 - **JMX:** Built-in support from the latest JDK allowed us to leverage **JMX** for system property monitoring without any additional configuration.

• Phase 2 (Repository Setup and Cloning)

To begin the experiment, the **Apache PDFBox** repository was cloned from the official Apache GitHub repository. This allowed for local modifications, the addition of monitoring tools, and the execution of commands for method invocation tracking and system property monitoring.

The steps involved in setting up the repository were as follows:

1. **Cloning the PDFBox Repository:**
The project was cloned from the official **Apache PDFBox** GitHub repository using the following command:

```
git clone https://github.com/apache/pdfbox.git
```
2. **Project Configuration:**
Once the repository was cloned, the project was imported into **IntelliJ IDEA** as a **Maven**

project. The `pom.xml` file was configured to include the necessary dependencies, such as **AspectJ** for method monitoring and **JFreeChart** for system property charting.

3. Building the Project:

After integrating the necessary dependencies, the project was built using Maven. The following command was used to clean and build the project:

```
mvn clean install
```

This command ensured that all dependencies were correctly resolved and that the PDFBox tools were compiled, making them ready for testing and experimentation.

4. Execution of Commands:

With the project successfully built, various **PDFBox commands** (e.g., `export:text`, `encrypt`, and `export:images`) were executed. These commands were run both manually and in automated loops to gather sufficient data for system property monitoring and method invocation tracking.

• Phase 3 (Method Invocation Monitoring with AspectJ)

For monitoring method invocations, we integrated **AspectJ**, a popular aspect-oriented programming (AOP) framework, into the PDFBox project. AspectJ enables us to intercept method calls at runtime, providing a way to track both project-specific and third-party methods.

The following steps were taken:

- **Setting Up AspectJ:** The **AspectJ Maven plugin** was added to the project's `pom.xml` file. This allowed AspectJ to weave aspects into the compiled PDFBox classes, enabling method-level monitoring.

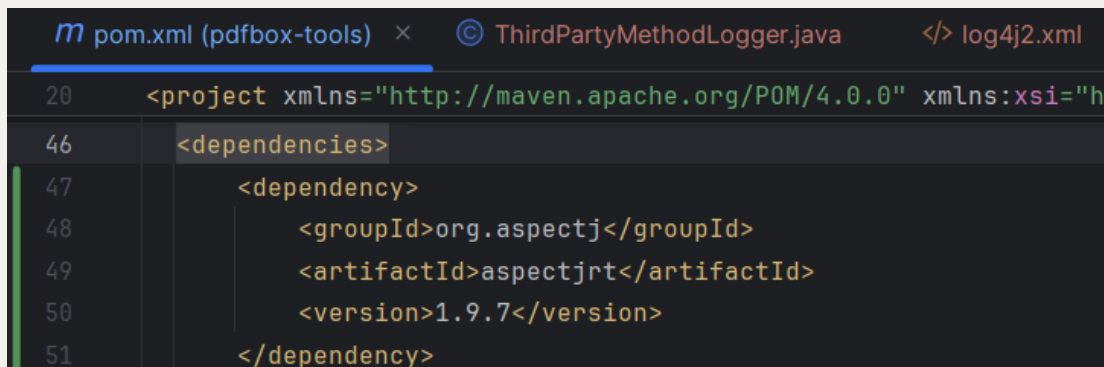
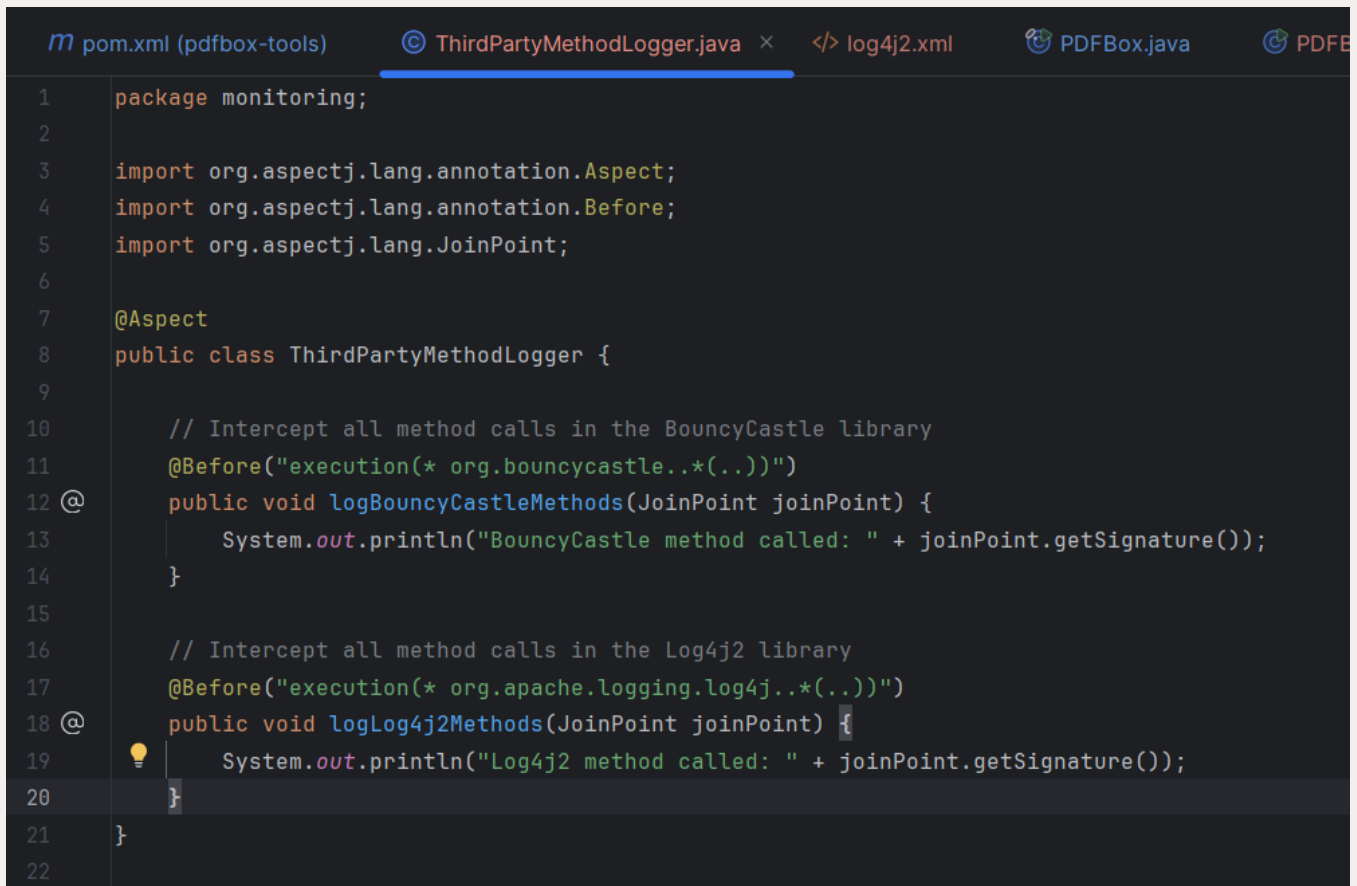


Figure 1: Adding AspectJ dependency to `pom.xml`

- **Aspect Implementation:** A custom aspect (`ThirdPartyMethodLogger`) was developed to log method invocations. It was configured to capture calls to both internal methods and methods from dependencies like **Log4j2** and **BouncyCastle**. The aspect was triggered on method entry points during command execution.



```
1 package monitoring;
2
3 import org.aspectj.lang.annotation.Aspect;
4 import org.aspectj.lang.annotation.Before;
5 import org.aspectj.lang.JoinPoint;
6
7 @Aspect
8 public class ThirdPartyMethodLogger {
9
10     // Intercept all method calls in the BouncyCastle library
11     @Before("execution(* org.bouncycastle.*(..))")
12     @ public void logBouncyCastleMethods(JoinPoint joinPoint) {
13         System.out.println("BouncyCastle method called: " + joinPoint.getSignature());
14     }
15
16     // Intercept all method calls in the Log4j2 library
17     @Before("execution(* org.apache.logging.log4j.*(..))")
18     @ public void logLog4j2Methods(JoinPoint joinPoint) {
19         System.out.println("Log4j2 method called: " + joinPoint.getSignature());
20     }
21 }
22
```

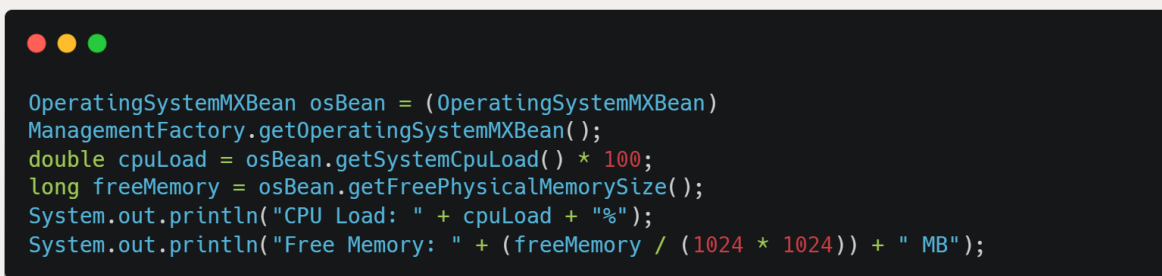
Figure 2: Intercepting method calls in the Log4j2 and BouncyCastle

• Phase 4 (System Properties Monitoring with JMX)

To monitor system properties such as CPU usage and memory consumption, the `OperatingSystemMXBean` class from the Java Management Extensions (JMX) was used. This allowed real-time tracking of system metrics during PDFBox command execution.

The system monitoring was integrated into the project as follows:

1. **System Monitor Class:** A custom `SystemMonitor` class was created to track and log CPU load and memory usage. The monitoring was performed at the start and end of each command execution, and real-time data was captured and displayed using `JFreeChart`.



```
OperatingSystemMXBean osBean = (OperatingSystemMXBean)
ManagementFactory.getOperatingSystemMXBean();
double cpuLoad = osBean.getSystemCpuLoad() * 100;
long freeMemory = osBean.getFreePhysicalMemorySize();
System.out.println("CPU Load: " + cpuLoad + "%");
System.out.println("Free Memory: " + (freeMemory / (1024 * 1024)) + " MB");
```

Figure 3: System Monitoring Code

2. **Real-Time Charting:** To visualize the CPU usage, the JFreeChart library was used to plot CPU load during the execution of PDFBox commands. A time-series chart was updated every second to reflect the CPU usage over time, allowing for live feedback on system performance.

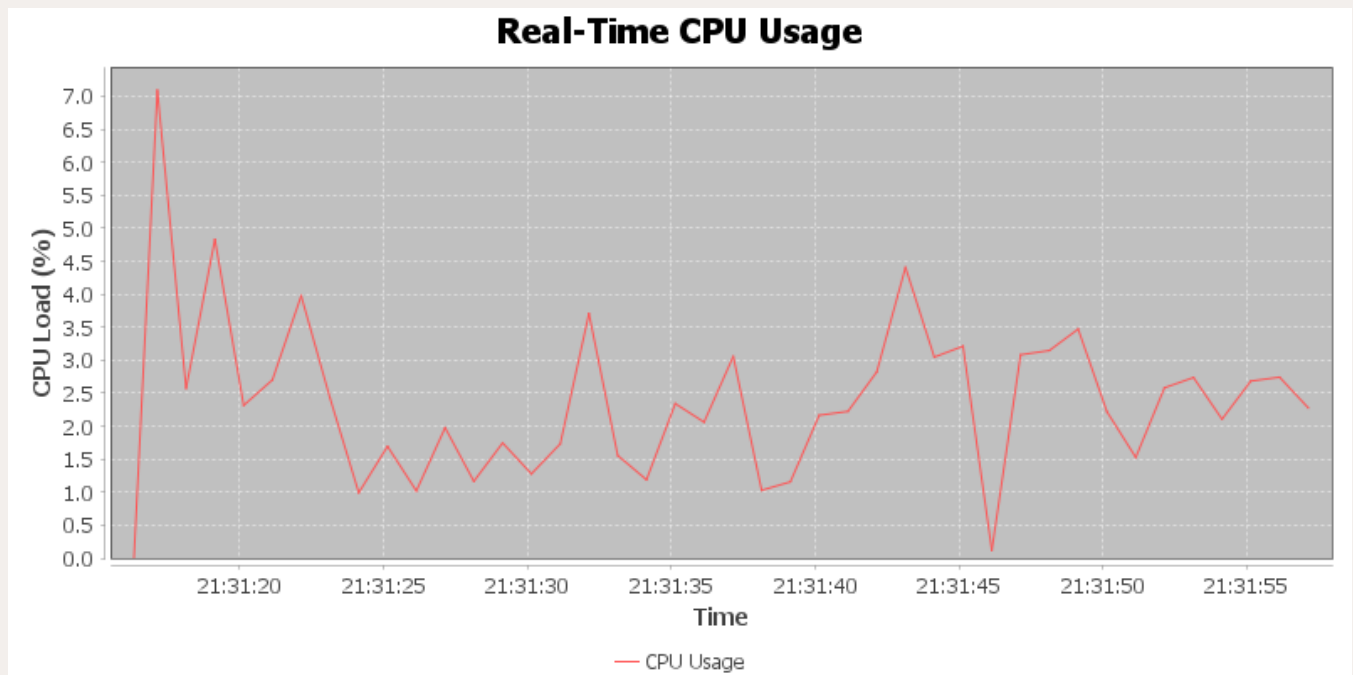


Figure 4: CPU usage chart while running `export:images` on the RICK paper

- **Phase 5 (Repeating Commands for Performance Analysis)**

Initially, each **PDFBox** command (`export:text`, `encrypt`, and `export:images`) was executed manually, but the time taken for each command was too short for the real-time monitoring chart to capture significant data. The rapid execution didn't allow for proper visualization of **CPU usage** and other system properties over time.

To address this, a custom runnable script was written to automatically run each command **100 times**. This provided enough data points to measure how system properties such as **CPU usage** and **memory consumption** changed under repeated workloads. By extending the total runtime through multiple executions, we were able to gather valuable insights into the performance and resource consumption of PDFBox.

Build and Running Process

This section provides an overview of how the project was set up, built, and executed to achieve the monitoring of method invocations and system properties during the execution of various PDFBox commands.

- **Cloning Project:** The first step was to clone the repository containing the modified PDFBox tools and the monitoring framework. This was done using the following command:

```
git clone https://github.com/aliimaher/Pdfbox-Methods-Invocation-Monitoring.git
cd Pdfbox-Methods-Invocation-Monitoring/pdfbox/tools(modified)
```

- **Building the Project:** After cloning the repository, the project was built using Maven to resolve all dependencies and compile the source code. The following command was used to build the project:

```
mvn clean install
```

This command ensures that all necessary dependencies, including AspectJ and JFreeChart, are downloaded and integrated into the project, and the pdfbox-tools module is compiled and packaged.

- **Running PDFBox Commands**

Manual Execution of Commands

Export Text from PDF:

```
java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar export:text --input
sample.pdf --output output.txt
```

Encrypt a PDF File:

```
java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar encrypt --input sample.pdf
--output encrypted-sample.pdf --password 123
```

Extract Images from PDF:

```
java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar export:images --input
test.pdf
```

Automated Execution of Commands (100 Iterations)

To capture comprehensive system performance data, each PDFBox command was executed 100 times using an automation script. This helped to monitor system properties over an extended period, ensuring that sufficient data was collected for performance analysis. The script was executed using the following command:

```
java -cp target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar automation.PDFBoxRunner
```

Results

In this section, you summarize the findings from both the method invocation monitoring and system property tracking. This includes sample logs, graphs, and observations.

1. Method Invocation Results

The following methods were logged during the execution of the export:text command:

```
PS C:\Users\LOTUS\Desktop\pdfbox\pdfbox\tools> java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar export:text --input sample.pdf --output output3.txt
PDFBox method called: void org.apache.pdfbox.tools.PDFBox.main(String[])
CPU Load: 0.0%
2024-10-07 21:53:51 [main] INFO  PDFBox - Export text command added
PDFBox method called: PrintPDF.Duplex[] org.apache.pdfbox.tools.PrintPDF.Duplex.values()
PDFBox method called: TextToPDF.PageSizes[] org.apache.pdfbox.tools.TextToPDF.PageSizes.values()
PDFBox method called: Integer org.apache.pdfbox.tools.ExtractText.call()
PDFBox method called: Writer org.apache.pdfbox.tools.ExtractText.createOutputWriter()
PDFBox method called: long org.apache.pdfbox.tools.ExtractText.startProcessing(String)
PDFBox method called: void org.apache.pdfbox.tools.ExtractText.stopProcessing(String, long)
PDFBox method called: long org.apache.pdfbox.tools.ExtractText.startProcessing(String)
PDFBox method called: void org.apache.pdfbox.tools.ExtractText.extractPages(int, int, PDFTextStripper, PDDocument, Writer, boolean, boolean)
PDFBox method called: void org.apache.pdfbox.tools.ExtractText.stopProcessing(String, long)
CPU Monitoring stopped.
2024-10-07 21:53:51 [main] INFO  PDFBox - PDFBox application finished.
```

Figure 5: Monitoring Methods Invocation of “extract:text” command on a pdf file containing only one single line

The following methods were logged during the execution of the encrypt command:

```
PS C:\Users\LOTUS\Desktop\pdfbox\pdfbox\tools> java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar encrypt --input sample.pdf --output encrypted-sample.pdf -U=123
PDFBox method called: void org.apache.pdfbox.tools.PDFBox.main(String[])
CPU Load: 0.0%
2024-10-07 22:01:08 [main] INFO  PDFBox - Export text command added
PDFBox method called: PrintPDF.Duplex[] org.apache.pdfbox.tools.PrintPDF.Duplex.values()
PDFBox method called: TextToPDF.PageSizes[] org.apache.pdfbox.tools.TextToPDF.PageSizes.values()
PDFBox method called: Integer org.apache.pdfbox.tools.Encrypt.call()
CPU Monitoring stopped.
2024-10-07 22:01:08 [main] INFO  PDFBox - PDFBox application finished.
```

Figure 6: Monitoring Methods Invocation of “encrypt” command on a pdf file

The following methods were logged during the execution of the export:images command:

```
PS C:\Users\LOTUS\Desktop\pdfbox\pdfbox\tools> java -jar target/pdfbox-tools-4.0.0-SNAPSHOT-shaded.jar export:images --input sampleWithImage.pdf
PDFBox method called: void org.apache.pdfbox.tools.PDFBox.main(String[])
CPU Load: 100.0%
2024-10-07 22:12:05 [main] INFO PDFBox - Export text command added
PDFBox method called: PrintPDF.Duplex[] org.apache.pdfbox.tools.PrintPDF.Duplex.values()
PDFBox method called: TextToPDF.PageSizes[] org.apache.pdfbox.tools.TextToPDF.PageSizes.values()
PDFBox method called: Integer org.apache.pdfbox.tools.ExtractImages.call()
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.run()
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.appendRectangle(Point2D, Point2D, Point2D, Point2D)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.clip(int)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.endPath()
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.showGlyph(Matrix, PDFont, int, Vector)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.processColor(PDColor)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.appendRectangle(Point2D, Point2D, Point2D, Point2D)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.clip(int)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.endPath()
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.drawImage(PDImage)
PDFBox method called: void org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.write2file(PDImage, String, boolean, boolean)
PDFBox method called: boolean org.apache.pdfbox.tools.ExtractImages.ImageGraphicsEngine.hasMasks(PDImage)
Writing image: C:\Users\LOTUS\Desktop\pdfbox\pdfbox\tools\sampleWithImage-1.png
PDFBox method called: boolean org.apache.pdfbox.tools.imageio.ImageIOUtil.writeImage(BufferedImage, String, OutputStream)
PDFBox method called: boolean org.apache.pdfbox.tools.imageio.ImageIOUtil.writeImage(BufferedImage, String, OutputStream, int)
PDFBox method called: boolean org.apache.pdfbox.tools.imageio.ImageIOUtil.writeImage(BufferedImage, String, OutputStream, int, float)
PDFBox method called: boolean org.apache.pdfbox.tools.imageio.ImageIOUtil.writeImage(BufferedImage, String, OutputStream, int, float, String)
PDFBox method called: void org.apache.pdfbox.tools.imageio.ImageIOUtil.setDPI(IIOMetadata, int, String)
PDFBox method called: IIOMetadataNode org.apache.pdfbox.tools.imageio.ImageIOUtil.getOrCreateChildNode(IIOMetadataNode, String)
PDFBox method called: IIOMetadataNode org.apache.pdfbox.tools.imageio.ImageIOUtil.getOrCreateChildNode(IIOMetadataNode, String)
PDFBox method called: IIOMetadataNode org.apache.pdfbox.tools.imageio.ImageIOUtil.getOrCreateChildNode(IIOMetadataNode, String)
PDFBox method called: boolean org.apache.pdfbox.tools.imageio.ImageIOUtil.hasICCProfile(BufferedImage)
CPU Monitoring stopped.
2024-10-07 22:12:05 [main] INFO PDFBox - PDFBox application finished.
```

Figure 7: Monitoring Methods Invocation of "export:images" command on a pdf file

These results provide insight into how PDFBox interacts with its own methods and third-party dependencies during execution. Each method call was logged with its signature, giving a clear view of the execution flow.

2. CPU Usage Monitoring Results

Real-time monitoring of CPU usage showed the following patterns:

Table 1: CPU-Time table of executed commands on RICK Paper pdf file

	export:text	export:images	encrypt
Duration Time	1 sec.	40 sec.	~0 %
Average CPU Usage	1 %	2%	~0 %

The CPU usage chart of the "export:text" command is as follow:

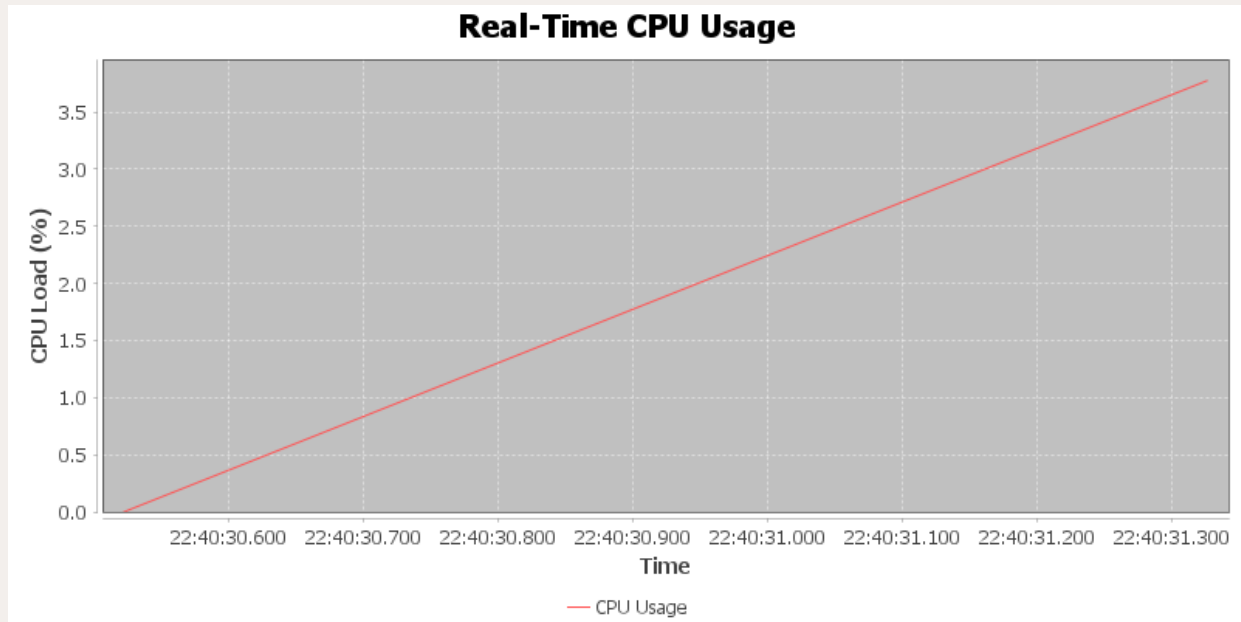


Figure 8: CPU usage chart while running export:text on the RICK paper

Personal Insights

During the process of method invocation monitoring, I was reminded of a challenge—challenge created by my curiosity—I faced with assembly language. I had to find the maximum of two numbers without using any conditions. The core of the solution involved a mathematical formula, but the main difficulty was determining the sign of the numbers, which wasn't directly accessible. To solve this, I leveraged flag monitoring, using shifts and specific instructions to extract the sign of the numbers and complete the formula. This experience made me think about how flag monitoring might have broader applications, perhaps even in tracking method invocations in certain low-level system operations.

Other

- Note: If you need to find the files I created or files I modified, just search "**Ali Maher >>>**" in the project, and you will find them.

Conclusion

In this case study, we successfully monitored both method invocations and system properties during the execution of PDFBox commands, providing a comprehensive view of the library's runtime behavior. By combining AspectJ for method tracking and JMX for system monitoring, we gathered valuable insights into both functional and performance aspects of the PDFBox library.

This approach can be extended to other Java-based projects to analyze how third-party libraries interact with system resources under various workloads, offering potential improvements in resource optimization and performance tuning.

References

1. <https://arxiv.org/pdf/2208.01321>
2. <https://pdfbox.apache.org/3.0/commandline.html>
3. <https://github.com/apache/pdfbox/>

The project files with doc are also available on my [GitHub](#).

Thank you for your time and consideration.