Jurnal Mantik, 5 (3) (2021) 1591-1597

Published by:Institute of Computer Science (IOCS)

Jurnal Mantik

Journal homepage: www.iocscience.org/ejournal/index.php/mantik/index

Performance Analysis of BLoC and Provider State Management Library on Flutter

Regawa Rama Prayoga[1], Ghifari Munawar[2], Rahil Jumiyani[3], Alifia Syalsabila[4]

[1234] Informatics and Computer Engineering Department,
[1234] Politeknik Negeri Bandung, Bandung 40012, Indonesia

E-mail: [1]regawa.rama.tif417@polban.ac.id, [2]ghifari.munawar@polban.ac.id, [3]rahil@jtk.polban.ac.id, [4]alifia.syalsabila.tif417@polban.ac.id.

A R T I C L E I N F O

Article history:

Received: August 02, 2021
Revised: September 15, 2021
Accepted: October 15, 2021

Keywords:

Performance Analysis,
Flutter,
State Management Library,
BLoC,
Provider
Android

A B S T R A C T

A proper choice of state management library can increase the performance of the Flutter for Android application development. BLoC and Provider are the most popular state management libraries nowadays. Unlike setState, which rebuilds all widgets, the state management library allows apps to rebuild only those widgets that have state changes. However, it is not known yet how much it increased the performance by using BLoC or Provider as a state management library. This study focused on the effect of using state management libraries on the widget build process in Flutter. It was conducted by developing setState, BloC, and Provider. The performance was measured by CPU utilization, memory usage, and execution time. Experimental results show that applications that use BLoC result in (1) CPU utilization of 0.45% (2.14% more efficient), (2) memory usage of 23.27 MB (8.19% more efficient), and (3) execution time of 3.54 seconds (16.36% more efficient). The application that uses Provider results in (1) CPU utilization 0.54% (2.57% more efficient), (2) memory usage of 32.01 MB (11.27% more efficient), and (3) execution time of 4.10 seconds (19.44% more efficient). However, it should be noted that these results occur when BLoC and Provider are implemented in the leaf widget.

1. Introduction

The development of tools to facilitate application development is increasingly diverse. One of them is the development of the Software Developer Kit (SDK). Flutter is a Google-owned User Interface (UI) SDK used to develop cross-platform applications [1]. Applications developed with Flutter built from blocks of UI components called widgets. Flutter provides widgets that are under existing application design standards, making it easier to create UI. The appearance of the widget can change according to the configuration and state of the widget [2]. "State is information that can be read synchronously when the widget is built and might change during the lifetime" [3]. State of the widget: a representation of dynamic data that affects the display on the screen. By default, when the widget state changes, the widget needs to be rebuilt by calling the setState method. That way, Flutter will rebuild all the widgets in the UI view. Unlike setState, which rebuilds all widgets, the state management library allows apps to rebuild only those widgets that have state changes.

BLoC and Provider are the most favorite state management libraries Flutter for developer [4]. BLoC (*Business Logic Component*) is stream/observable based library state management [5]. BloC divides the application into three layers, namely the presentation layer, business logic, and data layer. In BLoC, events from the UI affect the state of a widget. It processed the event at the business logic layer and will return to a new state. While Provider works by wrapping inherited widgets and passing state change information through its components. The provider will notify the observer (Consumer) when there is a data change in the state manager class (ChangeNotifier). Consumer and ChangeNotifier can communicate via

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

ChangeNotifierProvider. Using BLoC and Provider state management libraries can limit the build scope, so that only widgets that are rebuilt are those that have state changes [1]. The fewer widgets that are built, the more efficient resources are used.

Performance efficiency of a software product can be seen based on several characteristics including, time behavior, resource utilization, and capacity [6]. Time behavior is the extent to which response and processing times, and the throughput level of the application or system, meet the existing requirements [6]. Other metric is resource utilization, resources on mobile devices are CPU, memory, battery and network [7]. previous studies [8][9] believe that CPU usage, execution time, and memory usage are the main aspects of measuring the performance.

This study examines the efficiency of using the BLoC and Provider state management libraries in the build process compared to the default mechanism of the Flutter application using setState. In this study, we built three applications that implemented three different state management libraries, namely setState, BLoC, and Provider. We measure the performance of the application based on CPU utilization, memory usage, and execution time using the Snapdragon Profiler tool.

2. Method

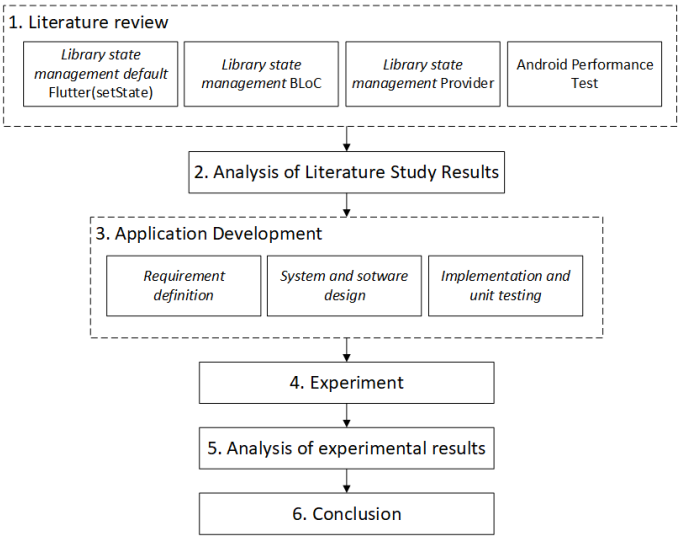

Fig 1. Methodology

We divide this study into six parts (1) Literature review (2) Analysis of literature and study result (3) Application development (4) Experiment (5) Analysis of experimental result (6) Conclusion. The literature review phase is carried out to study the sources and materials needed to conduct research. This analysis of literature and study result phase is done to analyzing the results of the study to get the factors that affect the performance of CPU utilization, memory usage and execution time in the application build process. The application development phase is carried out to create a tool whose performance will be measured (object of research). The application built is a movie catalog application with the name MovDB. The MovDB film catalog application provides information about films, such as film titles, film genres, film posters to film synopsis. The following shows the use case of the application.

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

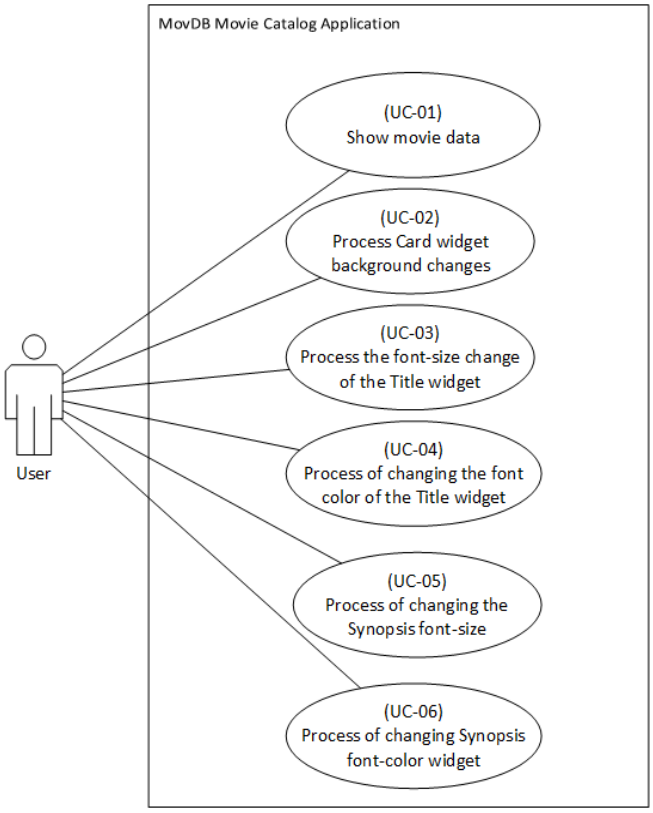

Fig 2. MovDB usecase diagram

The following Fig 3 is the implementation of the developed application.

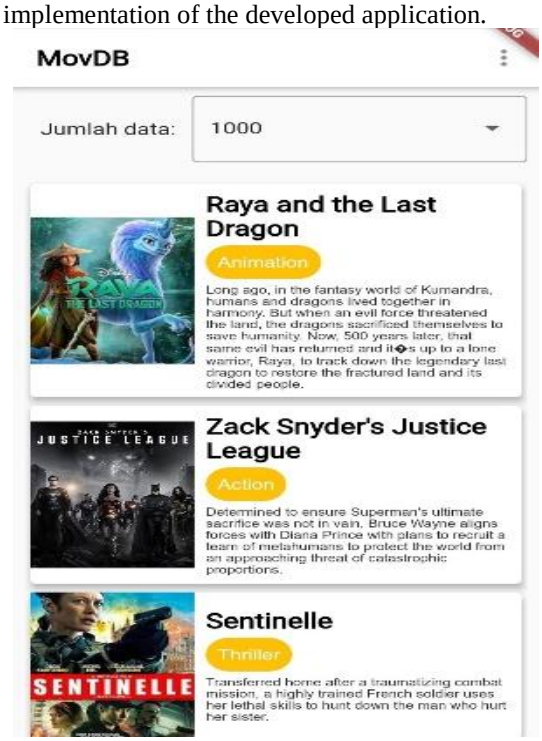

Fig 3. Implementation

After developing all three Flutter applications (setState, BLoC and Provider), the experimental phase is carried out to get measurement data for CPU utilization, memory usage and execution time from applications

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

that have been developed. The experimental scenario is designed based on the factors that have been identified in the Analysis of Literature Study Results, namely the type of library state management, the amount of data, and the position of the widget. The following scenario is shown in **Error! Reference source not found.**.

Table 1
Experiment Scenario

Scenario ID	Position	Description
SK-01	Parent	Change background state on Card widget
SK-02	Leaf	Change the font-size state of the Text:title widget
SK-03		Change state font-color on widget Text:title
SK-04		Change the font-size state of the Text:synopsis widget
SK-05		Change the font-color state of the Text:synopsis widget

This experiment will be carried out on 1,000 and 10,000 data for more significant testing. Each scenario is run three iterations in order to get a more valid value. The iteration results are then averaged. These scenarios will measure CPU usage, memory usage and execution time of Flutter(setState), Provider and BLoC default applications. The combination of components in the scenario will affect the number of widgets that need to be created. This will affect resource usage (CPU usage, memory usage) and execution time so that performing the three state management libraries can be seen as measured. The experiment was carried out on an android device with Android 10 specifications, 4GB RAM and a Qualcomm Snapdragon 460 Processor with the Snapdragon Profiler performance monitoring tool.

The analysis phase conducted to analyze the experimental results get conclusions about CPU utilization, memory usage, and execution time performance on the default Flutter application, applications built using the Provider state management library and applications built using the BLoC state management library. The analysis is carried out by calculating the performance efficiency value with the following formula.

$$\text{Efficiency (metric)} = \frac{\text{Average difference (metric)}}{\text{Average } (metric) \text{of setState application}} * 100 \quad (1)$$

3. Result and Analysis

This section analyzes the efficiency of the state management library based on the experiments that have been carried out. Before that, we will first show the difference of each measured variable between the default Flutter(setState) application and the BLoC state management library application and the difference between the default Flutter(setState) application and the Provider state management library application. The comparison is shown based on the scenarios that have been run. Fig 4 show the CPU utilization comparison between setState and BLoC also between setState and Provider. It is show that the difference between setState and BLoC, setState and Provider in SK-01 in the two data amounts is negative. This shows that BLoC and Providerhas a worse CPU utilization value than setState in that scenario.

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

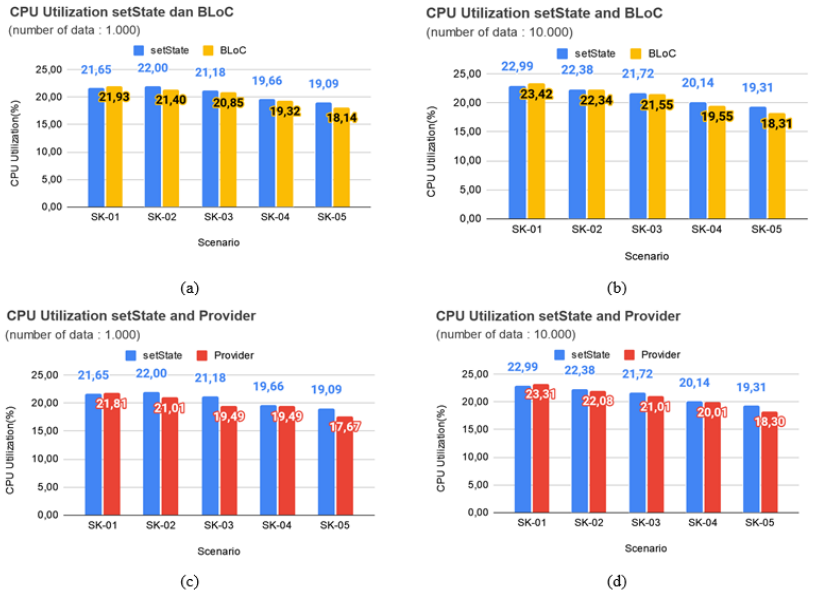

Fig 4. CPU utilization comparison of (a)(b)setState and BLoC, (c)(d) setState and Provider

Fig 5 shows the measurement results of memory usage on the application. It can be seen that in the difference between setState and BloC also setState and Provider in SK-01 data of 1,000 is negative while data of 10,000 is positive. This shows that BLoC and Provider has a worse CPU utilization value than setState at 1000 data but at 10,000 BLoC data is better than setState.

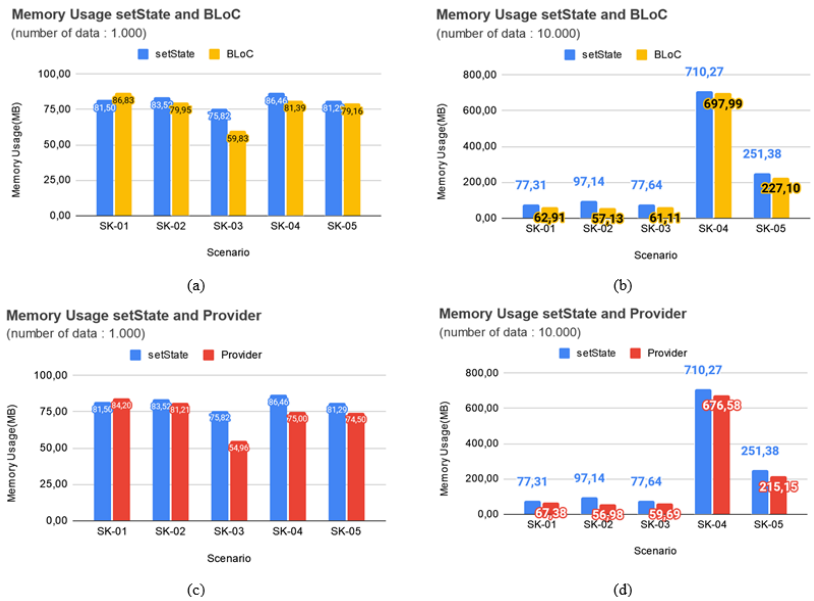

Fig 5. Memory usage comparison of (a)(b)setState and BLoC, (c)(d) setState and Provider

The above anomaly can occur due to the following possibilities.

a. Because the application limits the allocation of memory usage.

b. The large amount of data combined with the process of changing the background color of the Card widget can trigger excessive garbage collection. Dart as Flutter's programming language automatically takes care of memory allocation problems with garbage collection [10]. Garbage collection will reuse memory that is not accessible by the process. In general, the occurrence of garbage collection cannot be controlled. Garbage collection that occurs when an intensive process takes place can increase processing time [11].

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

Fig 6 show the execution time comparison between setState and BLoC also setState and Provider. It is show that the difference between setState and BLoC, setState and Provider in SK-01 is negative. This shows that BLoC and Provider has a worse CPU utilization value than setState in that scenario.

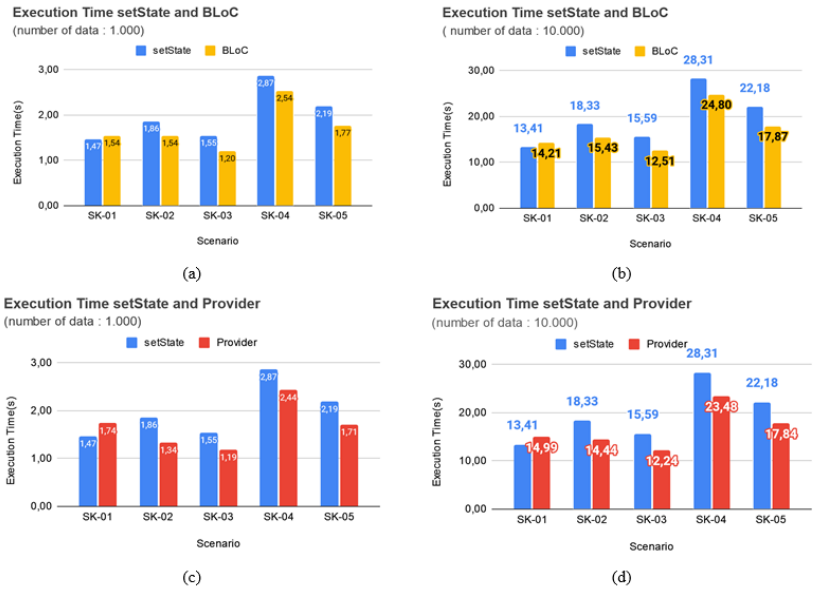

Fig 6. Execution time comparison of (a)(b)setState and BLoC, (c)(d) setState and Provider

From the comparison of experimental results, it is known that the state management library has a better performance when used on the leaf than on the parent widget. Therefore, the value calculated to measure efficiency is only the value in the scenario in the leaf. **Error! Reference source not found.** shows the efficiency of three metrics measured for the BLoC and Provider state management libraries compared to the default Flutter(setState).

Table 2
Efficiency of Block and Privider State Management Libraries

Metric (%)	Data	Efficiency	
		BLoC	Provider
CPU utilization	1.000	2,70	5,19
	10.000	2,14	2,14
Memory usage	1.000	8,18	12,66
	10.000	8,19	11,27
Execution time	1.000	16,79	21,21
	10.000	16,36	19,44

4. Conclusion

Based on the experimental results, we can conclude that the application with the BLoC state management library has smaller CPU utilization, memory usage and execution time values compared to the default Flutter (setState) application at the widget position in the leaf. Likewise, applications with library state management Providers have lower CPU utilization, memory usage and execution time values than the default Flutter(setState) application. However, the default Flutter(setState) application has a smaller value than the application with the BLoC state management library and the Provider state management library in the widget position in the parent. Thus, a research question is answered regarding how much influence the

Accredited "Rank 4"(Sinta 4), DIKTI, No. 36/E/KPT/2019, December 13th 2019.

use of BLoC and Provider state management libraries compared to the default Flutter mechanism (setState) has on the efficiency of Android application performance, with details as follows.

3.1 CPU utilization

The average CPU utilization value of the BLoC library state management version of the application with 1000 and 10,000 data, respectively, is 0.55% (2.70% more efficient than setState) and 0.45% (2.14% more efficient than setState. The average CPU utilization value of the application version of the library state management Provider with 1000 and 10,000 data, respectively, is 1.06% (5.19% more efficient than setState) and 0.54% (2.57% more efficient) versus setState. Both BLoC and Provider both have more efficient CPU utilization values than setState. This is because there are fewer widget objects that have to be rendered during the build process on the BLoC and Provider versions of the application, so the process is lighter.

3.2 Memory usage

The average memory usage value of the application version of the BLoC library state management with the amount of data 1000 and 10,000 respectively is 6.69 MB (8.18% more efficient than setState) and 23.27 MB (8.19% more efficient than setState). The average memory usage value of the application version of the library state management Provider with 1000 and 10,000 data, respectively, is 10.36 MB (12.66% more efficient than setState) and 32.01 MB (11, 27% more efficient than setState). Both BLoC and Provider, both have more efficient memory usage values than setState. That is because the setState version of the application instantiates more objects than BLoC or Provider. That way the memory allocation will increase according to the number of objects that must be stored.

3.3 Execution time

The average execution time of the BLoC state management library version of the application with 1000 and 10,000 data, respectively, is 1.06 seconds (16.79% more efficient than setState) and 3.54 seconds (16.36% more efficient than setState). setStates). The average execution time of the application version of the library state management Provider with 1000 and 10,000 data, respectively, is 0.45 seconds (21.21% more efficient than setState) and 4.10 seconds (19.44% more efficient). versus setState). The number of widgets that need to be rebuilt influences this value. The fewer the number of widgets built, the shorter the build and render time. The ideal execution time of a mobile application from a user perspective is 1 to 2 seconds [12]. We can conclude that, although the execution time of the state management library is generally more efficient, for the amount of data 10,000 it still cannot reach the ideal execution time.

5. References

[1] V. Katz, Mike; Moore, Kevin; Ngo, *Flutter Apprentice (First Edition)_ Learn to Build Cross-Platform Apps.pdf*. .

[2] "Introduction to widgets - Flutter." https://flutter.dev/docs/development/ui/widgets-intro (accessed May 16, 2021).

[3] "State class - widgets library - Dart API." https://api.flutter.dev/flutter/widgets/State-class.html (accessed Sep. 14, 2021).

[4] "Search results for state management." https://pub.dev/packages?q=state+management (accessed May 26, 2021).

[5] "List of state management approaches - Flutter." https://flutter.dev/docs/development/data-and-backend/state-mgmt/options (accessed May 16, 2021).

[6] "ISO 25010." https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (accessed May 16, 2021).

[7] I. G. I. Publishing, "Mobile Application Benchmarking Based on the Resource Usage Monitoring," pp. 64–75, 2009, doi: 10.4018/jmcmc.2009072805.

[8] B. Wisnuadhi, G. Munawar, and U. Wahyu, "Performance Comparison of Native Android Application on MVP and MVVM," no. January, 2020, doi: 10.2991/aer.k.201221.047.

[9] N. S. Sibarani, G. Munawar, and B. Wisnuadhi, "Analisis Performa Aplikasi Android Pada Bahasa Pemrograman Java dan Analisis Performa Aplikasi Android Pada Bahasa Pemrograman Java dan Kotlin," *9th Ind. Res. Work. Nationmal Semin.*, no. Juli, pp. 319–324, 2018.

[10] "Flutter | Using the Memory view." https://flutter.dev/docs/development/tools/devtools/memory (accessed Sep. 14, 2021).

[11] "Overview of memory management | Android Developers." https://developer.android.com/topic/performance/memory-overview (accessed Aug. 16, 2021).

[12] R. Zhou, S. Shao, W. Li, and L. Zhou, "How to define the user's tolerance of response time in using mobile applications," *IEEE Int. Conf. Ind. Eng. Eng. Manag.*, vol. 2016-Decem, pp. 281–285, 2016, doi: 10.1109/IEEM.2016.7797881.