Using ChatGPT on Improving Program Performance with pprof and Benchmark

Wei-Cheng Lei, Luo-You Jian, Yan-Wen Chen, Li-Der Chou
Department of Computer Science and Information Engineering
National Central University
Taoyuan, Taiwan
{davidleitwm, roy, ywchen } @networklab.csie.ncu.edu.tw, cld@csie.ncu.edu.tw

2023 5th International Conference on Computer Communication and the Internet (ICCCI) | 979-8-3503-2695-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCI59363.2023.10210148

***Abstract*—In the context of limited computing resources, optimizing program architecture is crucial. Therefore, this paper proposes to apply the powerful analytical capabilities of large language models (LLM) to the field of systematic performance optimization. The output of pprof and benchmark is fed into ChatGPT, and the program's performance is improved based on feedback. In the case study, the number of memory allocations for the objective function was successfully reduced from 99 to 1, resulting in a reduction of the test execution time from 8.6 microseconds to 0.36 microseconds. At the same time, the memory allocation was also reduced from 53.5KB to approximately 1KB.**

Keywords— Performance analysis; System optimization; Large Language Models; Prompt Engineering;

I. INTRODUCTION

In modern software development, performance optimization is a critical issue [1][2]. As the complexity of applications and the amount of data increase, performance bottlenecks become more common, which can have negative impacts on user experience and system reliability. To help software developers better understand and solve performance issues, many performance analysis tools have been developed, among which a well-known tool is pprof developed by Google [3].

pprof is a performance analysis tool developed by Google that can be used to analyze the performance of Go programs. The tool supports various data collection modes, including CPU profiling, memory profiling, and so on. pprof is widely used, especially in the Go community. However, since pprof requires embedded collection of code and complex configuration, it can be challenging for developers to use, especially for non-professional developers.

Usually, pprof is used when monitoring large-scale systems to identify services that have particularly high latency or frequently fail due to timeouts. At this point, developers can isolate the bottleneck function and observe the runtime behavior through pprof.

Despite its widespread use and powerful performance analysis capabilities, understanding and using pprof data is still a challenging task for non-professional developers. This calls for more advanced tools to help parse and comprehend the data produced by pprof and provide better optimization suggestions.

Benchmarking is a method of measuring software performance, typically used to compare the efficiency of different implementations. In software performance optimization, benchmarking is crucial because it helps developers quantify the effectiveness of their optimization efforts. Through benchmark analysis of pprof data, developers can gain a better understanding of where performance bottlenecks are located, allowing for better optimization.

In this paper, a new tool named ProfGPT is purposed. The tool utilizes the API of ChatGPT [4] to parse pprof data. ProfGPT can automatically extract valuable performance metrics from pprof files and provide more comprehensive optimization suggestions for the project by leveraging the capabilities of ChatGPT. Therefore, even for non-professional developers, using ProfGPT to parse pprof data has become relatively easy and efficient.

II. BACKGROUND

A. ChatGPT

ChatGPT is a large language model developed by OpenAI based on the GPT-3.5 architecture[5]. It can predict and generate natural language text, and is an optimization and extension of the GPT-3 model. ChatGPT has 175 billion parameters, while GPT-1, GPT-2, and GPT-3 have 117 million, 1.5 billion, and 175 billion parameters, respectively [6]. The GPT series of models are trained on the Transformer architecture and a large amount of text data, and use techniques such as self-attention and residual connections to improve the model's generation and comprehension capabilities. This allows the model to better handle long text sequences and generate high-quality natural language text, such as for question-answering, articles, conversations, and translations. GPT has been applied in various fields, such as P4[7], edge computing[8], self-driving car[9], programmable data plane [10].

The development and advancement of ChatGPT has attracted significant attention and discussion worldwide due to its abilities and potential [11], which have significant value in various application scenarios, such as education and business, making these applications more intelligent and convenient.

Since ChatGPT became a hot topic, the open API of ChatGPT [4] has sparked another wave of intense development. The open API can support various text-based applications, as long as the data is presented in text form, ChatGPT API can interpret it. Many applications, including key summarization, paper review, etc., have amazing effects. Tasks that used to require a lot of manpower and time can now be processed more

efficiently through ChatGPT API, greatly improving the work efficiency of many tedious tasks.

In addition, ChatGPT has also made breakthroughs in another text-based field, optimizing the software development process. For example, CodeGPT tries to generate git commit messages using ChatGPT API, and Naming [12] inputs program text to provide variable naming suggestions, improving readability. These tools optimize the indispensable but tedious work in the development process.

B. *pprof*

pprof is a performance analysis tool used for CPU and memory analysis, commonly used for analyzing performance issues in Go programs. Originally part of the analysis module in Google PerfTools, the goal of pprof is to help developers quickly identify performance bottlenecks and provide effective performance optimization suggestions.

PerfTools was originally developed for internal use at Google. The challenge that Google faced was how to quickly and effectively analyze performance issues in large-scale distributed systems. In this context, PerfTools became very useful, as it could help developers identify performance problems and provide solutions. As Google continued to evolve, PerfTools gradually evolved into an open source project [13], and more people began using it to analyze performance issues in their applications.

C. *Prompt*

Prompt Engineering is a solution designed for natural language processing (NLP) that describes task requirements in the input and standardizes the output structure. Due to the nature of large language models (LLMs), the output cannot guarantee structural stability. This variation is not conducive to program development, as developers cannot use a fixed method to deconstruct and obtain the required information from the output. Prompt Engineering aims to solve the problem of unstable output structure and enable structured output, allowing models to be more widely applied [14].

There are also many people in the community providing various prompts for ChatGPT, and even selling prompts for profit [15]. Because prompts need to be constantly adjusted to optimize them, makes developers willing to spend money to purchase them in order to save time on adjustments.

III. System Architecture

This chapter introduces the system architecture of ProfGPT and explains how the integration of pprof with the ChatGPT API enables the tool to read profiles generated by pprof and provide modification suggestions for performance bottlenecks.

pprof is a performance analysis tool that can be used in two main ways. The first is an interactive command-line tool that allows experienced developers to quickly identify performance issues in their code by entering commands. The second is a web-based visual interface that displays performance results in the form of flame graphs, enabling developers to quickly identify which parts of their code use the most resources. Each approach has its advantages and disadvantages, and developers can choose which to use based on their specific needs.

In the use case, the biggest challenge is how to convert the profile generated by pprof into a text format for parsing by the ChatGPT API. Currently, the ChatGPT API only supports pure text input, so the first challenge for ProfGPT is how to address this issue. After evaluation, it was found that converting the interactive command-line tool into a text format and inputting it into the model is easier, while the web version is relatively more difficult to handle.

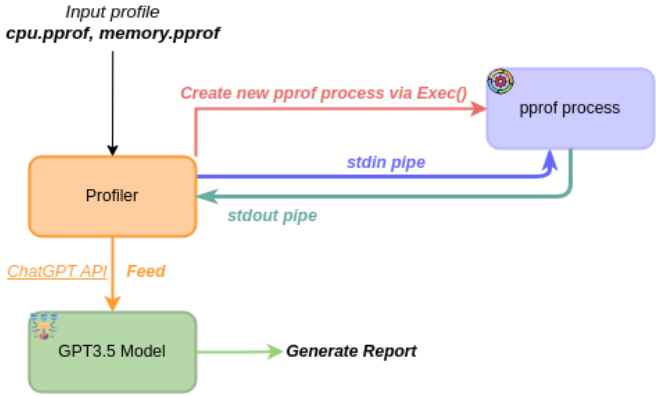

Fig. 1. The architecture of ProfGPT

In the system, first use the pprof tool to generate a binary profile file, then use the Golang exec package to execute the command line pprof tool and pass the profile file generated earlier as an input parameter. Finally, the ChatGPT API analyzes the plain text input, identifies performance bottlenecks, and provides suggestions. This way, by integrating the pprof tool with the ChatGPT API, The automatic identification of performance problems in the program greatly simplifies the process of performance optimization.

IV. Case Study

In the Case Study, a common mistake in Go programming was presented. The mistake is usually caused by a lack of familiarity with the language features or underlying principles by the developers, resulting in poor code performance. The common feature of the mistake is that only a few lines of code need to be modified by developers, without refactoring the entire function's execution logic and sequence, to significantly improve the function's execution speed by tens or even hundreds of times.

The example, "string concat performance issue" is presented in this paper, which is common scenarios encountered when programming in Go language. Through a few simple tests, the performance difference between good and bad writing practices is described to more clearly demonstrate the impact of incorrect writing practices. Next, the poorly written versions are inputted into ProfGPT and analyzed to improve performance. This kind of analysis can allow developers to more deeply understand the reasons for errors and learn how to improve code to enhance performance.

When developing applications in Go, string concatenation is a common scenario, especially when generating SQL query statements, composing log information, or generating HTML documents. Typically, the + operator is used to perform string concatenation or the fmt.Sprintf function. However, this

approach may lead to performance issues when dealing with large amounts of string concatenation.

When processing large amounts of string concatenation, because Go strings are immutable, a new block of memory needs to be allocated each time concatenation occurs, and the content of the old string needs to be copied to the new block of memory. This process requires a lot of time and memory, especially when dealing with a large number of strings of varying lengths, which can easily cause performance problems.

In this issue, several key points are focused on, such as the number of memory allocations during string concatenation and execution efficiency. For some relatively inexperienced developers, the most commonly used method may be to use the + operator supported by Golang's built-in string to complete the task of string concatenation. The effect will be tested with a simple example.

```
const letterBytes = “abcdefghijklmnopqrstuvwxyz”
func randomString(numberOfStr int) string {
   b := make([]byte, numberOfStr)
   for i := range b
      b[i] = letterBytes[rand.Intn(len(letterBytes))]
   return string(b)
}
```

Fig. 2. Generating test data code

Since multiple string concatenation methods need to be tested, a factory function was first written to create various string concat functions. In Figure 3, a factory function is implemented that can take different implementations of concatFunc and perform tests. During the test process, a random string str of length 10 is first generated, then concatFunc is called to concatenate str 100 times, and finally, the performance differences of different implementations of concatFunc are compared.

```
func benchmarkStringContact(b *testing.B, concatFunc func(int,
   string) string) {
   str := random(10)
   for  i := 0; i < b.N; i++
      concatFunc(100, str)
}
```

Fig. 3. Factory function

First, string concatenation using the built-in string + operator in Golang was implemented and a test was written. A benchmark test was generated using the benchmarkStringConcat function as shown in Figure 4.

```
func plusConcat(n int, str string) string {
   s := ""
   for i := 0; i < n; i++
      s += str
   return s
}
func BenchmarkPlusConcat(b *testing.B)
   benchmarkStringConcat(b, plusConcat)
```

Fig. 4. Generating Benchmark Test with the benchmarkStringConcat Function

Afterwards, the go test command is called to run the benchmark test and generate profiles such as cpu.pprof and mem.pprof shown in Figure 5.

```
$ go test -bench=BenchmarkPlusContact \
      -bencheme   -cpuprofile=pluscpu.pprof  \
      -memprofile=plusmem.pprof
```

Fig. 5. "go test" Command and Parameters

The results show that even before using pprof, the number of memory allocations is as high as 99. This is because in Go, using the + operator to concatenate strings causes a new memory space to be allocated every time, since strings are immutable in Go. That is, every time the + operation is executed, a new memory space is allocated to store the new string.

```
$ go test -bench=BenchmarkPlusContact \
      -bencheme   -cpuprofile=pluscpu.pprof  \
      -memprofile=plusmem.pprof
>
goos: linux
goarch: amd64
pkg: github.com/davidleitw/profgpt/ex2
cpu: 12th Gen Intel(R) Core(TM) i5-12500
BenchmarkPlusConcat-12  138002  8745 ns/op
53480 B/op  99 allocs/op
PASS
ok  github.com/davidleitw/profgpt/ex2  1.405s
```

Fig. 6. Benchmark result

When it comes to program optimization, it is often necessary to identify potential performance bottlenecks through benchmarking and profiling. In some scenarios that require extensive string concatenation, the common approach is to use string joining, which involves using the += operator to join multiple strings. However, this method often results in excessive memory allocation and subsequently leads to performance issues and memory wastage when joining a large number of strings. If developers have no specific optimization direction and just want to reduce the number of memory allocations, they can use ChatGPT to interpret the issues reflected in the benchmark and profile and identify the improvement direction. The feedback is summarized as follows:

The plusConcat() function in this code uses string concatenation, which involves using the += operator to join multiple strings. In the benchmarkStringConcat() function, a baseline test is conducted on this function that involves joining a large number of strings.

Although this approach is simple and easy to understand, it has two main performance issues when joining a large number of strings:

- Frequent memory allocation and release: Each time the += operator is used to join strings, a new memory space needs to be allocated for the new string. The more strings that are joined, the larger the memory space that needs to

be allocated. When joining a large number of strings, there will be frequent memory allocation and release, leading to decreased performance.

- Frequent string copying: Each time the += operator is used to join strings, the original string needs to be copied to a new memory space before the new string is joined at the end. When joining a large number of strings, there will be frequent string copying, leading to decreased performance.

Please note that the following prompt is aimed at asking for solutions to optimize code performance. If the developer already has an idea of the performance bottleneck, feel free to modify the prompt and ask ChatGPT to suggest some solutions. Based on ChatGPT's recommendations, the developer can write new tests to verify if the code performance has been improved.

fter being prompted to provide some feasible solutions, ChatGPT proposed several directions. Overall, when it is necessary to concatenate multiple strings together, using "+" may cause unnecessary memory allocation and copying operations, which can affect program performance. Here are several alternative solutions in Golang that can be used instead of "+":

- strings.Builder: This is a structure specifically designed to build strings, and you can add strings to the string being built using its WriteString method. Finally, use the String method to get the concatenated string. Because strings.Builder uses a variable-sized buffer, it can effectively reduce memory allocation and copy operations. However, using strings.Builder requires creating a structure and then gradually adding strings, which may require more code.

- bytes.Buffer: This is a structure similar to strings.Builder, but it is used to process byte slices instead of strings. You can use the WriteString method to write strings and use the Bytes or String method to get the result. This method can also reduce memory allocation and copy operations, but compared to strings.Builder, it is more suitable for handling binary data.

- strings.Join: If you want to concatenate strings from a slice, you can use the strings.Join function to do so. This function takes a slice and a separator, and then concatenates all elements in the slice with the specified separator. This method does not require additional memory allocation, but it requires converting all elements to strings.

With the assistance and testing from a series of ChatGPTs, significant improvements can be seen in both execution efficiency and memory allocation counts from the benchmark results. According to the benchmark results, using the Concat function of the Builder can reduce memory allocation counts from 99 to 8, and using the Grow function can reduce memory allocation counts to 1. In the end, the test execution time was successfully reduced from the original 8.6 microseconds to 0.36 microseconds, while also reducing the memory allocation from 53.5KB to about 1KB, further improving the efficiency and performance of the test.

Why does not strings.Builder need an additional copy when calling String(), even though its underlying storage is also []byte? Looking at the implementation details, strings.Builder uses the unsafe feature as Figure 7.

```
func (b *Builder) String() string {
    return unsafe.String(unsafe.SliceData(b.buf), len(b.buf))
}
```

Figure 7. Implementation of strings.Builder

The implementation as Figure 7 uses unsafe.SliceData() and unsafe.String() functions to directly convert the underlying byte array []byte to a string type result without the need for additional copying operations. This is because in Go, the string type is represented as a read-only byte array, while the []byte type represents a readable and writable byte array. Therefore, the unsafe.SliceData() function can be used to obtain a pointer to the underlying byte array, and the unsafe.String() function can be used to convert the pointer and the length of the byte array to a string type result, thereby avoiding the need for additional memory copying and allocation. Although this approach carries some risk, it can improve the performance and efficiency of string operations.

V. Conclusion

The analysis capability of LLM was successfully applied to the field of system performance optimization in this paper, by utilizing the ChatGPT API to parse pprof data and extract valuable performance metrics, which provided more comprehensive suggestions for project optimization. ProfGPT makes it easier and more efficient for non-professional developers to use pprof data to analyze and provide optimization recommendations. The cooperation between system monitoring and ChatGPT can better understand and solve performance problems. In this study, the memory allocation count of the target function was successfully reduced from 99 to 1, and the execution time was reduced from 8.6 microseconds to 0.36 microseconds, as well as the memory allocation amount from 53.5 KB to about 1 KB. Therefore, the method proposed in this paper is highly valuable in practical applications, helping developers to better optimize system performance, improve system efficiency, and stability.

Acknowledgment

This work was supported in part by the National Science and Technology Council, Taiwan, under Grant no. 111-2221-E-008-062-MY2, 111-2221-E-008-063-MY3, and 111-2218-E-415-001-MBK.

References

[1] Huang, S., Qin, Y., Zhang, X. et al. "Survey on performance optimization for database systems. Sci." China Inf. Sci. 66, 121102 , 2023.

[2] Kougka, G., Gounaris, A., & Simitsis, A. (2017). Kougka, Georgia, Anastasios Gounaris and Alkis Simitsis. "The many faces of data-centric workflow optimization: a survey." International Journal of Data Science and Analytics 6 (2017): 81-107, 2017.

[3] pprof https://github.com/google/pprof

[4] Create chat completion API https://platform.openai.com/docs/api-reference/chat/create?utm_source=soft4fun&utm_medium=post

[5] W. Fedus, I. Goodfellow, and A. M. Dai, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” in Proceedings of the 9th International Conference on Learning Representations (ICLR), Vienna, Austria, 2021.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al., “Language models are few-shot learners,” in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, pp. 6109-6120, 2020.

[7] D. C. Li, P.-H, Chen, L.-D. Chou, "GAP4NSH: A Genetic Service Function Chaining with Network Service Header for P4-based Software-Defined Networks," Journal of Supercomputing, Mar. 2023.

[8] D. C. Li, C.-T. Huang, C.-W. Tseng, L.-D. Chou, "Fuzzy-Based Microservice Resource Management Platform for Edge Computing in the Internet of Things," Sensors, vol. 21, iss. 11, May 2021.

[9] D. C. Li, M Y.-C. Lin, L.-D. Chou, "Macroscopic Big Data Analysis and Prediction of Driving Behavior With an Adaptive Fuzzy Recurrent Neural Network on the Internet of Vehicles," IEEE Access, vol. 10, pp. 47881 - 47895, Apr. 2022.

[10] D. C. Li, M. R. Maulana, L.-D. Chou, "NNSplit-SOREN: Supporting the Model Implementation of Large Neural Networks in a Programmable Data Plane," Computer Networks, vol. 222, Feb. 2023.

[11] L. Zhou, X. Dong, J. Wang, H. Wu, and F. Wei, “Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models,” in IEEE International Conference on Natural Language Processing and Information Retrieval, pp. 1-6, 2022.

[12] naming https://github.com/davidleitw/naming

[13] gperftools https://github.com/gperftools/gperftools

[14] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt,. “A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT.“ ArXiv, abs/2302.11382, 2023.

[15] PromptBase https://promptbase.com/.