Научная статья

УДК 004.4
DOI 10.25205/1818-7900-2025-23-3-79-86

Исследование влияния параметров Golang Garbage Collector на производительность приложений и использование ресурсов системы

Денис Сергеевич Фадеев

Акционерное общество «Сбербанк-Технологии»
Москва, Россия

denisfadeev82@gmail.com

Аннотация

Современные высоконагруженные приложения предъявляют высокие требования к системам сборки мусора (garbage collection), которые используются в языках программирования, в частности в Golang. Данная статья направлена на систематическое изучение параметра GOGC и его влияние на ключевые показатели работы тестового приложения. Результаты измерения выбранных ключевых метрик (количество запусков, продолжительность GC, объем выделенной памяти) были визуализированы с помощью таблицы и графиков. В статье проведен анализ поведения механизма GC и влияние на него параметра GOGC. Результаты исследования позволяют разработчикам высоконагруженных приложений применить их на практике для управления механизмом сборки мусора в зависимости от задач, которые необходимо решить.

Ключевые слова

Garbage Collector, Golang, GOGC, сборка мусора, тюнинг параметров gc

Для цитирования

Фадеев Д. С. Исследование влияния параметров Golang Garbage Collector на производительность приложений и использование ресурсов системы // Вестник НГУ. Серия: Информационные технологии. 2025. Т. 23, № 3. С. 79–86. DOI 10.25205/1818-7900-2025-23-3-79-86

Research the Impact of Golang Garbage Collector Parameters on Application Performance and System Resource Usage

Denis S. Fadeev

Joint Stock Company “Sberbank-Technologies”,
Moscow, Russian Federation

denisfadeev82@gmail.com

Abstract

Modern high-load applications place high demands on garbage collector systems, which are used in programming languages, in particular in Golang. This article is aimed at a systematic study of the GOGC parameter and its impact on the key performance indicators of the test application. The measurement results of selected key metrics, such as the number of GC starts and duration, and the amount of allocated memory, were visualized using tables and graphs. The article analyzes the behavior of the GC mechanism and the effect of the GOGC parameter on it. The research results allow

developers of high-load applications to put them into practice to manage the garbage collection mechanism, depending on the tasks that need to be solved.

Keywords

garbage collector, golang, GOGC, garbage collection, gc parameter tuning

For citation

Fadeev D. S. Research the impact of Golang Garbage Collector parameters on application performance and system resource usage. *Vestnik NSU. Series: Information Technologies*, 2025, vol. 23, no. 2, pp. 79–86 (in Russ.) DOI 10.25205/1818-7900-2025-23-3-79-86

Введение

Современные высоконагруженные приложения требуют устойчивого и прогнозируемого потребления оперативной памяти системы, так как размер ее является конечным значением. Попытка превышения размера имеющейся памяти ведет к сбоям и отказам в работе приложения. Различные языки программирования используют разные подходы к работе с выделением и освобождением памяти, используемой приложением в процессе его работы. Одним из популярных подходов является использование механизма автоматической сборки мусора (garbage collector, GC) [1]. Этот механизм используется в языке программирования Golang [2]. Такой подход значительно упрощает разработку приложения и минимизирует риски утечек памяти. С другой стороны, плата за автоматическую сборку мусора – это дополнительное использование процессорного времени на выполнение операций GC, а также необходимость останавливать выполнение приложения на периоды освобождения неиспользуемой памяти (фаза «stop-the-world», STW), что ведет к увеличению времени выполнения приложения.

Таким образом, механизм GC оказывает влияние на работу приложения в части управления используемой памяти, процессорного времени и фаз STW. Поэтому крайне важно иметь возможность управлять работой механизма сборки мусора. В языке Golang для этого есть два параметра – GOGC и GOMEMLIMIT. Параметр GOMEMLIMIT является ограничителем максимального объема памяти системы, которое может использовать приложение. Параметр GOGC указывает процент выделения новой памяти от размера использованной в момент начала очередной сборки мусора. Подробное описание о работе механизма сборки мусора есть в официальной документации языка программирования Golang [1]. Цель данного исследования – провести серию экспериментов и сформировать рекомендации по использованию параметров механизма GC. Эти рекомендации позволят минимизировать влияние GC на производительность приложения, улучшить использование памяти. Проведенные эксперименты позволят наглядно продемонстрировать возможность изменения баланса между использованием памяти и частотой работы GC. Для достижения этих целей были выполнены следующие задачи: создано тестовое приложение, имитирующее реалистичное потребление памяти; проведено измерение потребления памяти; выявлено влияние работы GC на общую производительность при различных значения параметра GOGC; проведен анализ и систематизация результатов с помощью инструментов Golang для трассировки выполнения приложения; сделана визуализация с помощью таблиц и графиков.

Полученные результаты актуальны для практического применения разработчиками высоконагруженных систем и исследователей в области управления памятью и механизмов GC.

Материалы и методы

Для исследования была разработана тестовая программа на языке Golang, которая представляет собой многопоточное приложение. Это приложение в нескольких потоках генерирует

[1] См.: https://tip.golang.org/doc/gc-guide

интенсивное использование памяти, в результате чего позволяет провести анализ поведения механизма GC при различных значениях параметра конфигурации GOGC.

Программа из основного потока исполнения запускает четыре рабочих потока. Каждый поток создает массив целых чисел типа int размером 10 000, и заполняет его случайными значениями. Создание массива повторяется 500 раз. Так как создание массива оформлено в виде отдельной функции с использованием локальных переменных, то по ее завершению созданная локальная переменная с массивом чисел становится не нужной, что приводит к необходимости высвобождения занятой ею памяти при помощи механизма GC. Таким образом имитируется работа реального приложения, в котором создаются и освобождаются объекты с высокой интенсивностью, характерной для высоконагруженных систем [3].

Для анализа и визуализации результатов работы программы и механизма GC используется стандартная утилита Golang «go tool trace». Для этого в начале работы программы средствами языка Golang запускается механизм трассировки, который позволяет собрать необходимую информацию. Данные трассировки выводятся в файл для последующего изучения при помощи указанной утилиты. Утилита предоставляет визуальное и числовое представление данных временных рядов, характеристик работы GC и использование памяти (heap).

Для оценки эффективности работы GC и тестового приложения в целом были выбраны следующие характеристики, показывающие производительность и потребление ресурсов:

1) максимальное потребление heap. Позволяет оценить рост потребления памяти в зависимости от периодичности запуска GC [4];

2) количество и продолжительность запусков GC. Отображает зависимость выбранного значение параметров GOGC с частотой запуска GC и временем его работы [5];

3) общее время работы программы. Характеризует общее влияние работы механизма GC на производительность программы.

Тесты проводились на компьютере следующей конфигурации:

– процессор: Apple M3 @ 4.05 GHz, 12 ядер, 12 потоков;
– оперативная память: 32 Гб;
– накопитель: 512GB PCIe® NVMe™ M.2 SSD.

Тестовая программа запускалась и работала в неизменных условиях, создавая нагрузку на память и процессор в течение времени исполнения. Использовалась версия языка программирования Golang версии 1.23.0.

Для получения значений выбранных характеристик были проведенны замеры с различными значениями GOGC: в диапазоне от 10 до 100 процентов с шагом 10; в диапазоне от 100 до 1000 процентов с шагом в 100. Данные диапазоны были выбраны исходя из значения по умолчанию GOGC = 100 процентов. Собранные метрики позволили проанализировать поведение механизма GC и выявить закономерности частоты запуска сборки мусора, его продолжительности и объем потребления памяти системы.

Параметры командной строки для запуска приложения с выбранным значение GOGC:

GOGC = <значение> ./app

Серия экспериментов со значением GOGC в диапазоне от 100 до 10 процентов с шагом 10 преследовала цель изучения поведения механизма GC в условиях увеличения требования к экономии памяти. Это требование приводит к увеличению частоты сборки мусора и влияния этого процесса на работу тестового приложения. Цель проведения экспериментов в диапазоне от 100 до 1000 процентов с шагом 100: оценить влияние уменьшения частоты сборки мусора на потребление памяти тестовым приложением, снижение нагрузки на процессор, влияние фаз GC на приложение (см. таблицу).

Время работы приложения и метрики механизма GC
в зависимости от значения параметра GOGC

Application operation time and metrics of the GC mechanism,
depending on the value of the GOGC parameter

GOGC, %	Общее время работы приложения, ms	Количество запусков GC	Продолжительность запусков GC, ms	Объем памяти (Heap), Mb
1000	1382	1	0,5	38,24
900	1384	1	0,6	34,44
800	1387	1	0,5	30,65
700	1381	1	0,5	26,84
600	1375	1	0,5	23,08
500	1385	2	1,1	19,32
400	1380	3	1,3	15,56
300	1379	4	1,7	11,75
200	1382	7	2,8	8,94
100	1375	14	6,0	5,73
90	1377	16	6,3	5,47
80	1376	18	7,2	5,14
70	1377	21	7,9	4,99
60	1378	26	10,0	4,65
50	1371	29	10,5	4,29
40	1372	36	14,7	4,24
30	1372	38	14,9	4,07
20	1367	38	15,1	4,04
10	1374	38	15,1	4,07

Результаты и обсуждение

На рис. 1 представлены значения количества запуска механизма GC в зависимости от установленного процента GOGC в каждом эксперименте. При значениях от 1000 до 600 процентов количество запусков сборки мусора одинаковое – 1 раз. При дальнейшем уменьшении значений параметра GOGC происходит существенное увеличение частоты запуска GC. Возрастание количества сборки мусора происходит вследствие необходимости очистки памяти системы для соблюдения требования установленного для эксперимента параметра GOGC.

График на рис. 2 демонстрирует общую продолжительность работы механизма сборки мусора в зависимости от значения GOGC. Время выполнения одной сборки мусора практически остается неизменным, но количество GC растет, что приводит к большему суммарному времени работы. Следует отметить, что время одного GC очень мало по сравнению с временем работы всего приложения. Суммарное время при максимальной измеренной частоте 38 раз составляет всего 15,1 миллисекунды, тогда как время работы приложения составляет в среднем 1 377 миллисекунд.

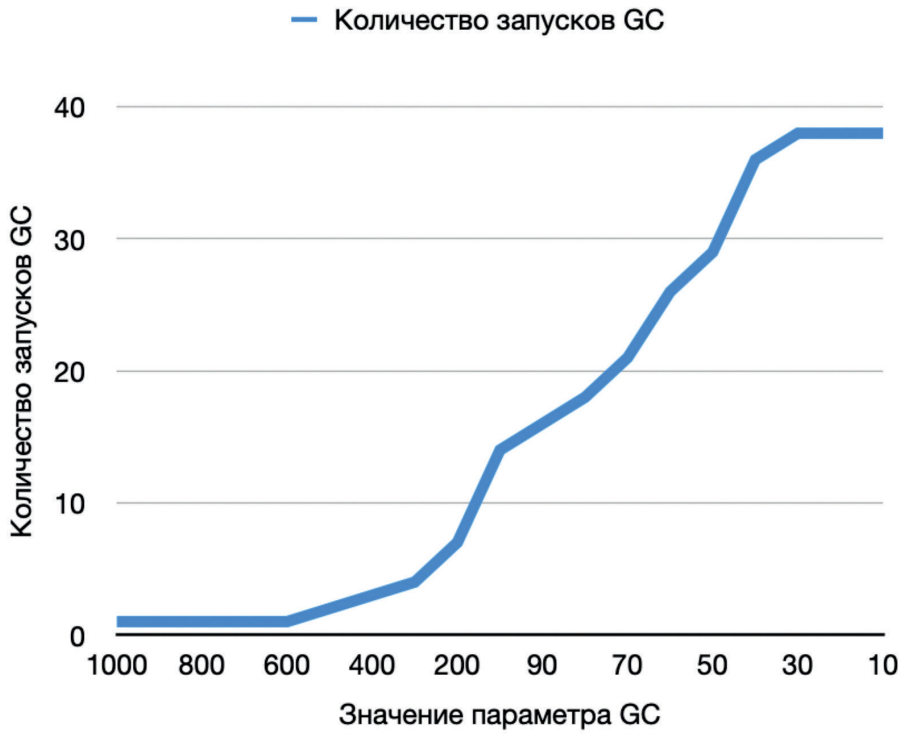

Рис. 1. Зависимость количества запуска GC от значения GOGC
Fig. 1. Dependence of the number of GC starts on the GOGC value

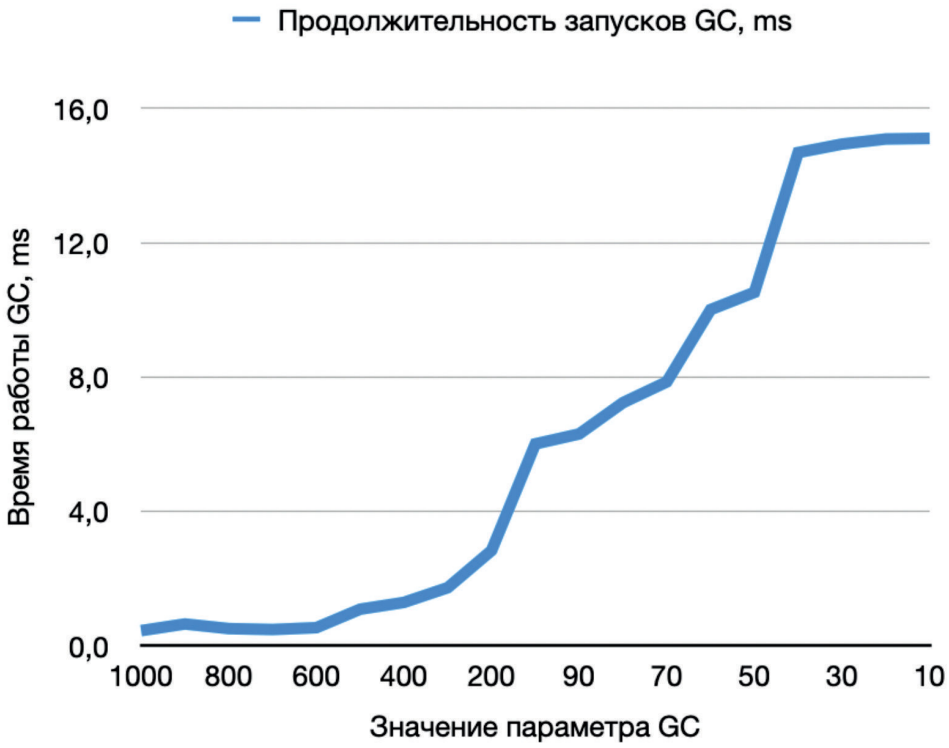

Рис. 2. Зависимость продолжительности работы GC от значения GOGC
Fig. 2. Dependence of the duration of GC operation on the GOGC value

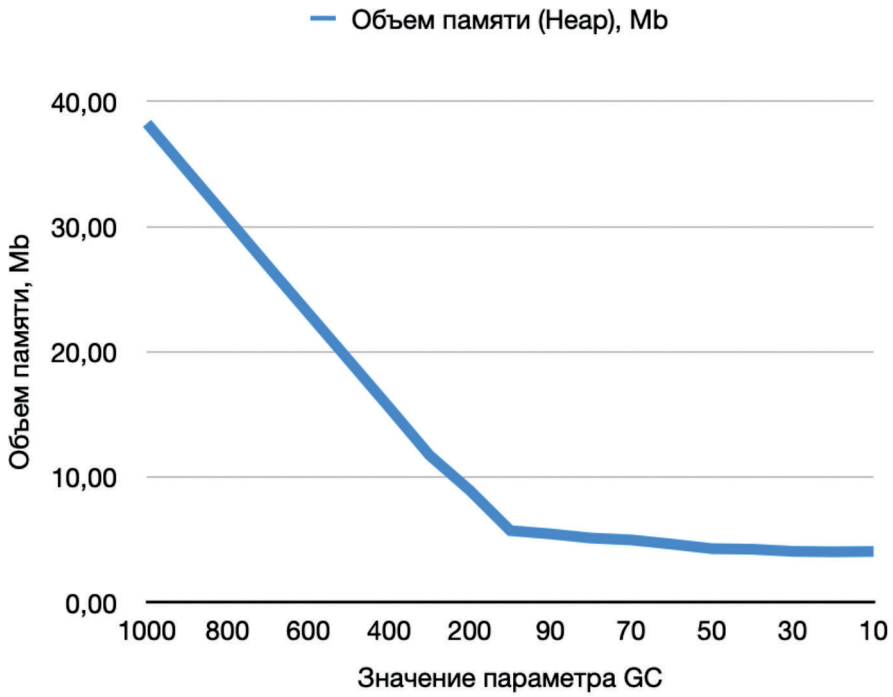

Рис. 3. Максимальный объем используемой памяти в зависимости от от значения GOGC
Fig. 3. The maximum amount of memory used, depending on the GOGC value

График на рис. 3 отражает максимальный объем используемой памяти системы при различных значениях GOGC. В ходе экспериментов было установлено, что частота работы механизма сборки мусора напрямую влияет на количество выделяемой памяти приложения. Таким образом, можно уменьшить количество вызовов GC за счет использования большего количества памяти системы или уменьшить количество выделяемой памяти за счет более частой сборки мусора. На графике видна область, в левую сторону от которой наблюдается резкий рост потребления памяти, а в правую – незначительное изменение используемой памяти при уменьшении значения GOGC. Это говорит о достижении оптимальной эффективности сборки мусора.

Результаты проведенных экспериментов демонстрируют влияние значения параметра GOGC на частоту сборки мусора и использование памяти. С точки зрения выделения памяти значение по умолчанию в 100 процентов является оптимальным, так как его уменьшение не дает существенной экономии памяти. Увеличение же этого значения значительно увеличивает потребление памяти системы. Для тестового приложения увеличение частоты GC практически не влияет на общее время работы приложения. Но для постоянно работающих приложений (сервисов), в которых критически важно время обработки каждого запроса, может быть важнее меньшее количество сборок мусора, при условии, что время обработки запроса сопоставимо с временем этапа GC. В таком случае увеличение значения GOGC позволит уменьшить частоту запуска механизма сборки мусора, что в свою очередь поможет снизить его влияние на обработку запросов. С другой стороны, возрастает вероятность использования всего объема памяти системы, что может привести к сбою приложения и всей системы. Во избежание этого нужно использовать параметр GOMEMLIMIT в сочетании с параметром GOGC.

Эксперименты показали высокую эффективность и скорость работы реализации GC в языке программирования Golang.

Заключение

Исследование было проведено для изучения возможностей управления работы механизма сборки мусора в языке программирования Golang. Были установлены связи между различными значениями параметра GOGC и частотой, общей продолжительностью GC и использованием памяти системы приложением. На примере тестового приложения показана зависимость выбранных метрик от параметра GOGC. Собраны и визуализированы выбранные метрики для демонстрации влияния GC на работу приложения. Установлено, что увеличение частоты GC и использование памяти для значений GOGC менее 100 процентов не приносит практической пользы. Увеличение же более 100 процентов может быть использовано в случае необходимости для снижения влияния сборки мусора за счет использования большего количества памяти. Эти выводы имеют практическую пользу для разработчиков высоконагруженных систем, в которых скорость выполнения операций критически важна.

Список литературы

1. **Аникин Д. А.** Преимущества автоматизированного управления памятью на примере Java Hotspot // Столыпинский вестник. 2022. С. 3. URL: https://stolypin-vestnik.ru/wpcontent/uploads/2022/11/10-3.pdf
2. **Сапожков А. М., Строганов Ю. В., Рудаков И. В.** О реализации метода автоматического управления памятью на языке Golang // Математические методы в технологиях и технике. 2024. № 12-2. С. 12–16.
3. **Шумилов М. И.** Оптимизация высоконагруженных веб-проектов с использованием микросервисной архитектуры // Universum: технические науки: электрон. научн. журн. 2024. 11 (128). URL: https://7universum.com/ru/tech/archive/item/18560
4. **Прозорова А. П., Вершинин Е. В., Потапов А. Е.** Влияние на производительность приложения малого объема памяти и использование Garbage Collector (GC) // Электронный журнал: наука, техника и образование. 2019;(1):55–61.

References

1. **Anikin D. A.** Advantages of automated memory management using the example of Java Hotspot. *Stolypinsky Bulletin*, 2022, p. 3. URL: https://stolypin-vestnik.ru/wpcontent/uploads/2022/11/10-3.pdf (in Russ.)
2. **Sapozhkov A. M., Stroganov Yu. V., Rudakov I. V.** On the implementation of the automatic memory management method in the Golang language. *Mathematical methods in technology and engineering*, 2024, no. 12-2, pp. 12–16. (in Russ.)
3. **Shumilov M. I.** Optimization of highly loaded web projects using microservice architecture. *Universum: technical sciences: electron. scientific journal*, 2024, vol. 11 (128). URL: https://7universum.com/ru/tech/archive/item/18560 (in Russ.)
4. 4. **Prozorova A. P., Vershinin E. V., Potapov A. E.** The impact on the performance of low-memory applications and the use of Garbage Collector (GC). *Electronic journal: science, technology and education*, 2019, vol. (1), pp. 55–61. (in Russ.)

Информация об авторе

Фадеев Денис Сергеевич, ведущий эксперт по технологиям

Information about the Author

Denis S. Fadeev, Leading Technology Expert

Статья поступила в редакцию 04.03.2025;
одобрена после рецензирования 24.06.2025; принята к публикации 24.06.2025

The article was submitted 04.03.2025;
approved after reviewing 24.06.2025; accepted for publication 24.06.2025