УДК 004.4
DOI: 10.26102/2310-6018/2024.47.4.039

Комплексное исследование влияния параметров G1 Garbage Collector на производительность и стабильность JVM

Д.Ю. Золотухина

Финансовая корпорация Открытие, Воронеж, Российская Федерация

Резюме. Актуальность исследования обусловлена необходимостью повышения эффективности управления памятью в высоконагруженных Java-приложениях, где минимизация пауз сборки мусора и поддержание высокой пропускной способности являются критически важными задачами. Данная статья направлена на систематическое изучение параметрического пространства G1 Garbage Collector (G1 GC) и разработку практических рекомендаций по его оптимизации для условий высокой нагрузки. Ведущим методом исследования является эмпирический подход, включающий разработку тестового многопоточного приложения на Java, способного создавать устойчивую нагрузку на память и процессор. Для анализа были использованы контрольная и шесть экспериментальных конфигураций G1 GC, различающихся настройками таких параметров, как размер регионов памяти, порог заполнения кучи, максимальная длительность пауз, доля молодых регионов, количество потоков GC и включение Periodic GC. Результаты измерений ключевых метрик, включая длительность пауз, частоту сборок, пропускную способность и объем освобожденной памяти, были визуализированы и систематизированы с использованием инструмента GCViewer. В статье представлены рекомендации по оптимизации G1 GC, выявлены преимущества перераспределения памяти в пользу молодых регионов и включения Periodic GC, а также показаны ограничения параметра MaxGCPauseMillis при агрессивной настройке. Полученные результаты имеют практическую ценность для разработчиков высоконагруженных приложений, требующих низких задержек и высокой стабильности работы системы. Выводы исследования способствуют углублению понимания работы G1 GC и могут служить основой для дальнейших исследований в области управления памятью JVM.

Ключевые слова: g1 garbage collector, управление памятью, оптимизация jvm, высоконагруженные приложения, тюнинг параметров gc, throughput, длительность пауз, сборка мусора, молодые регионы памяти, производительность java-приложений.

Для цитирования: Золотухина Д.Ю. Комплексное исследование влияния параметров G1 Garbage Collector на производительность и стабильность JVM. *Моделирование, оптимизация и информационные технологии*. 2024;12(4). URL: https://moitvivt.ru/ru/journal/pdf?id=1774 DOI: 10.26102/2310-6018/2024.47.4.039

A comprehensive analysis of the impact of G1 Garbage Collector parameters on JVM performance and stability

D.Yu. Zolotukhina

Financial Corporation Otkritie, Voronezh, the Russian Federation

Abstract. The relevance of the study is determined by the necessity of improving memory management efficiency in high-load Java applications, where minimizing garbage collection pauses and maintaining high throughput are critically important tasks. This article aims to systematically investigate the parameter space of the G1 Garbage Collector (G1 GC) and develop practical recommendations for its optimization under high-load conditions. The primary research method is an empirical approach, which involves the development of a multithreaded Java application capable of generating sustained memory and CPU loads. The study utilized a control and six experimental G1 GC configurations, differing in

parameters such as heap region size, heap occupancy threshold, maximum pause duration, young region size, the number of GC threads, and the activation of Periodic GC. Key metrics, including pause durations, GC frequency, throughput, and freed memory volume, were measured, visualized, and systematically analyzed using the GCViewer tool. The article presents recommendations for G1 GC optimization, highlights the advantages of reallocating memory toward young regions and enabling Periodic GC, and identifies the limitations of the MaxGCPauseMillis parameter in aggressive configurations. The findings have practical value for developers of high-load applications requiring low latency and high system stability. The conclusions contribute to a deeper understanding of G1 GC functionality and can serve as a foundation for further research in JVM memory management.

Keywords: g1 garbage collector, memory management, jvm optimization, high-load applications, gc parameter tuning, throughput, pause duration, garbage collection, young memory regions, java application performance.

For citation: Zolotukhina D.Yu. A comprehensive analysis of the impact of G1 Garbage Collector parameters on JVM performance and stability. *Modeling, Optimization and Information Technology.* 2024;12(4). (In Russ.). URL: https://moitvivt.ru/ru/journal/pdf?id=1774 DOI: 10.26102/2310-6018/2024.47.4.039

Введение

Современные высоконагруженные приложения на языке Java требуют стабильной и предсказуемой работы системы управления памятью, поскольку эффективность этого компонента напрямую влияет на производительность, надежность и общее качество функционирования программного обеспечения. В условиях возрастающего объема данных и растущих ожиданий по времени отклика, оптимизация автоматической сборки мусора (Garbage Collection, GC) в виртуальной машине Java (JVM) приобретает особую значимость. GC не только предотвращает утечки памяти, но и обеспечивает динамическое перераспределение ресурсов, что критически важно для поддержания высокой пропускной способности и минимизации задержек [1].

Среди различных алгоритмов сборки мусора, G1 Garbage Collector (G1 GC) выделяется как один из наиболее подходящих для приложений с большими объемами памяти. Его уникальная архитектура, основанная на разделении кучи на регионы, позволяет эффективно управлять памятью и минимизировать длительные паузы, которые характерны для других сборщиков [2]. Однако универсального решения для настройки параметров G1 GC не существует, и некорректная конфигурация может привести к ухудшению производительности системы, увеличению задержек и снижению стабильности. Проблема заключается в отсутствии комплексных рекомендаций по выбору параметров G1 GC, учитывающих как специфику конкретных приложений, так и общие требования высоконагруженных систем.

Настоящее исследование направлено на устранение этого пробела путем систематического изучения влияния ключевых параметров G1 GC на производительность JVM. Цель работы заключается в разработке практических рекомендаций для настройки сборщика мусора, позволяющих минимизировать паузы, повысить throughput и улучшить использование памяти. Для достижения этой цели были поставлены следующие задачи: создание тестового приложения, моделирующего реалистичную нагрузку на память и процессор; проведение серии экспериментов с изменением таких параметров, как размер регионов кучи, порог заполнения памяти, максимальная длительность пауз, доля молодых регионов, количество потоков GC и использование механизма Periodic GC; анализ и систематизация результатов с использованием инструментов визуализации, таких как GCViewer.

Полученные данные и выводы представляют собой шаг к более глубокому пониманию работы G1 GC и его оптимизации, что актуально как для разработчиков

высоконагруженных приложений, так и для исследователей в области управления памятью JVM [3].

Материалы и методы

В рамках исследования была разработана программа на языке Java, создающая параллельную нагрузку на систему управления памятью в JVM с использованием G1 GC. Программа представляет собой многопоточное приложение, которое генерирует интенсивное использование памяти и процессора на протяжении длительного времени, обеспечивая условия для анализа различных конфигураций сборщика мусора. Основная задача программы заключается в создании временных объектов, распределяемых по регионам памяти, а также в периодическом освобождении ресурсов для вызова процесса сборки мусора.

Программа состоит из основного управляющего потока, запускающего 24 рабочих потока. Каждый поток выполняет задачу создания массивов байтов, которые хранятся в списке. Массивы создаются размером 1 MB и заполняются случайными данными для увеличения нагрузки на процессор. Когда количество элементов в списке достигает 1000, список очищается, что вызывает сборку мусора. Такая структура имитирует сценарии работы реальных приложений, включая создание и освобождение короткоживущих объектов, характерных для высоконагруженных систем. Кроме того, реализация включает искусственные паузы (10 мс) между итерациями, чтобы симулировать поведение приложений с нерегулярной нагрузкой. Программа работает в течение 10 минут, обеспечивая устойчивую нагрузку на память и процессор. Такая реализация позволяет эффективно тестировать параметры G1 GC, включая размеры регионов, максимальное время пауз, частоту сборок и распределение ресурсов между молодыми и старшими регионами.

Для анализа и визуализации результатов работы сборщика мусора в ходе экспериментов использовался инструмент GCViewer. Этот инструмент предоставляет удобный способ интерпретации логов GC, генерируемых JVM, и позволяет получить ключевые метрики, такие как длительность пауз, пропускная способность, частота сборок и использование памяти [4]. GCViewer был выбран за его понятный интерфейс, совместимость с логами G1 GC и возможность получения агрегированных данных. Он обеспечивает наглядное представление временных рядов и сводных характеристик работы GC, что облегчило сравнительный анализ между контрольной конфигурацией и экспериментальными настройками.

Для оценки эффективности работы GC и сравнения результатов были выбраны следующие ключевые параметры, характеризующие производительность и стабильность системы [5]:

1) средняя пауза (avgPause). Позволяет понять общий тренд времени пауз для каждой конфигурации GC. Среднее значение важно для оценки стабильности системы в долгосрочной перспективе;

2) максимальная пауза (maxPause). Демонстрирует худший сценарий в каждом эксперименте. Критически важно для приложений, требующих низкой задержки;

3) количество minor GC (gcPauseCount). Отражает частоту младших сборок, что важно для управления короткоживущими объектами. Снижение количества сборок указывает на более эффективное использование памяти;

4) количество full GC (fullGcPauseCount). Характеризует частоту наиболее ресурсоемких сборок. Увеличение или уменьшение этого значения может напрямую влиять на стабильность и производительность системы;

5) пропускная способность (Throughput). Отражает, сколько времени приложение потратило на полезную работу, а не на паузы GC. Это один из самых прямых индикаторов производительности приложения;

6) объем освобожденной памяти (freedMemory). Позволяет оценить эффективность работы GC в терминах реального освобождения памяти. Сравнение этого параметра между конфигурациями дает представление о том, насколько лучше или хуже G1 GC справляется с освобождением ресурсов.

Тесты проводились на машине со следующими характеристиками:

- процессор: Apple M2 @ 3.49 GHz, 8 ядер, 8 потоков;
- оперативная память: 8 ГБ;
- накопитель: 256GB PCIe® NVMe™ M.2 SSD.

На первом этапе исследования был проведен контрольный замер производительности G1 GC с использованием стандартных параметров его настройки, принятых по умолчанию в JVM. Этот этап служил базовой отправной точкой для последующего анализа изменений, вызванных оптимизацией параметров GC. Контрольный замер позволил собрать исходные данные о работе сборщика мусора, включая такие метрики, как длительность пауз, пропускная способность (throughput), частота сборок (minor и full GC) и объем освобождаемой памяти.

Использование стандартных параметров G1 GC (например, размер регионов, порог запуска сборок и количество потоков GC) позволило объективно оценить начальную конфигурацию, которая часто применяется в реальных приложениях без дополнительной настройки. Это обеспечило основу для сравнения с последующими экспериментами, где изменялись ключевые параметры GC.

Тестовая программа функционировала в неизменных условиях, создавая равномерную нагрузку на память и процессор. Собранные метрики послужили базой для выявления узких мест в стандартной конфигурации, таких как высокие частоты сборок или неэффективное использование памяти. Таким образом, контрольный замер обеспечил количественную оценку исходной производительности и сформировал основу для дальнейшего анализа эффективности оптимизаций.

Параметры командной строки для запуска:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=1M -XX:ParallelGCThreads=8 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=45 -XX:MaxGCPauseMillis=200 -
Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

В рамках эксперимента 1 целью исследования было оценить влияние изменения размера регионов памяти на производительность G1 GC. Для этого был изменен параметр XX:G1HeapRegionSize, определяющий размер отдельных регионов в куче. Его значение было увеличено до 4 МБ вместо стандартного значения 1 МБ, используемого по умолчанию. Это изменение направлено на уменьшение количества регионов, что теоретически должно снизить накладные расходы на их управление и уменьшить частоту сборок [6].

Параметры командной строки были следующими:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=8 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=45 -XX:MaxGCPauseMillis=200 -
Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

При проведении эксперимента 2 целью являлось изучение влияния изменения порога заполненности кучи, инициирующего сборку мусора, на производительность G1 GC. Для этого использовался параметр -XX:InitiatingHeapOccupancyPercent, значение которого было увеличено с базового уровня (по умолчанию 45 %) до 65 %. Это

изменение направлено на уменьшение частоты вызовов сборок мусора за счет задержки момента их инициирования, что потенциально позволяет увеличить объем памяти, обрабатываемой за одну сборку, и снизить нагрузку на процессор [7].

Параметры для запуска программы:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=8 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=60 -XX:MaxGCPauseMillis=200 -
Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

Эксперимент 3 был посвящен анализу влияния уменьшения максимального времени паузы, устанавливаемого параметром -XX:MaxGCPauseMillis, на производительность G1 GC. Для достижения этой цели значение параметра было снижено до 100 миллисекунд, что соответствует потребностям приложений с высокими требованиями к задержкам. Цель эксперимента заключалась в том, чтобы сократить максимальную длительность пауз GC, сохраняя при этом приемлемый уровень пропускной способности системы и стабильности работы [8].

Программа была запущена со следующими параметрами:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=8 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=60 -XX:MaxGCPauseMillis=100 -
Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

В эксперименте 4 исследовалось влияние изменения доли памяти, выделяемой для молодых объектов, на производительность G1 Garbage Collector. Для этого были настроены параметры -XX:G1NewSizePercent и -XX:G1MaxNewSizePercent, задающие минимальную и максимальную долю памяти кучи, резервируемой для молодых регионов. Значения этих параметров были увеличены до 30 % и 50 % соответственно, что позволяет выделять больше памяти для объектов, создаваемых в молодых регионах, и, таким образом, снижать нагрузку на старшие регионы [9].

Параметры для командной строки:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=8 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=60 -XX:MaxGCPauseMillis=100 -XX:G1NewSizePercent=30 -
XX:G1MaxNewSizePercent=50 -Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

Эксперимент 5 был направлен на исследование влияния изменения числа потоков сборки мусора на производительность G1 Garbage Collector. Для этого был настроен параметр XX:ParallelGCThreads, определяющий количество потоков, задействованных при выполнении параллельных этапов сборки. Значение параметра было снижено до 4, что эквивалентно уменьшению ресурсов процессора, выделяемых для GC. Цель эксперимента заключалась в оценке влияния такого изменения на длительность пауз, пропускную способность и общую производительность системы в условиях высокой нагрузки [10].

Параметры запуска:

```
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=4 -
XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -
XX:InitiatingHeapOccupancyPercent=60 -XX:MaxGCPauseMillis=100 -XX:G1NewSizePercent=30 -
XX:G1MaxNewSizePercent=50 -Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar
```

В эксперименте 6 изучалось влияние включения механизма периодических сборок мусора (Periodic GC) на производительность G1 Garbage Collector. Для активации этого механизма был использован параметр -XX:+ExplicitGCInvokesConcurrent, который позволяет запускать сборку мусора с определенными интервалами времени. Цель эксперимента заключалась в минимизации фрагментации памяти, улучшении управления старыми регионами и повышении общей стабильности системы.

Параметры командной строки:
java -Xmx2G -Xms128M -XX:+UseG1GC -XX:G1HeapRegionSize=4M -XX:ParallelGCThreads=4 -XX:ConcGCThreads=2 -XX:+UnlockExperimentalVMOptions -XX:+UseCompressedOops -XX:InitiatingHeapOccupancyPercent=60 -XX:MaxGCPauseMillis=100 -XX:G1NewSizePercent=30 -XX:G1MaxNewSizePercent=50 -XX:+ExplicitGCInvokesConcurrent -Xlog:gc*:file=gc.log:tags,uptime,time,level -jar MyApp.jar

Результаты

Результаты исследования представлены в Таблице 1.

Таблица 1 – Производительность G1 GC для различных конфигураций
Table 1 – Performance of G1 GC for various configurations

Параметр	Контрольная конфигурация	1	2	3	4	5	6
Средняя пауза (avgPause)	0,01988 с	0,0470 с	0,0544 с	0,2859 с	0,0217 с	0,0222 с	0,0179 с
Максимальная пауза (maxPause)	0,98177 с	3,6888 с	1,5082 с	18,6063 с	0,3473 с	0,6525 s	0,8518 с
Количество minor GC (gcPauseCount)	1110	470	426	331	204	150	125
Количество full GC (fullGcPauseCount)	102	210	203	170	200	172	184
Throughput (%)	98,82%	94,66%	94,29%	76,08%	98,53%	98,8%	99,03%
Объем освобожденной памяти (freedMemory)	28,219 MB	24,049 MB	24,857 MB	15,749 MB	35,856 MB	19,820 MB	18,799 MB

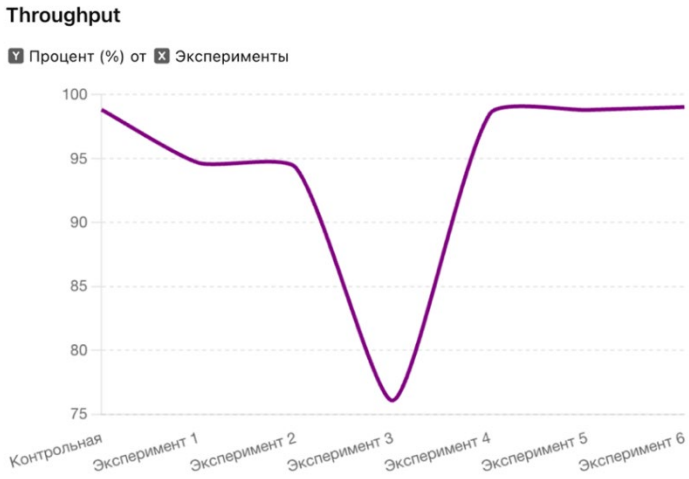

Рисунок 1 – Изменение параметра throughput в ходе исследования
Figure 1 – Change in the throughput parameter during the study

На графике Рисунка 1 представлены значения throughput, полученные в результате каждого эксперимента. В контрольной сборке его значение составило 98,82 %, демонстрируя изначально высокий уровень полезной работы программы. Однако изменение параметров в ходе экспериментов привело к различным результатам. Снижение максимальной паузы GC до 100 мс (Эксперимент 3) существенно понизило throughput до 76,08 %. Напротив, увеличение доли молодых регионов (Эксперимент 4) и

включение Periodic GC (Эксперимент 6) обеспечили восстановление throughput до 98,53 % и 99,03 % соответственно.

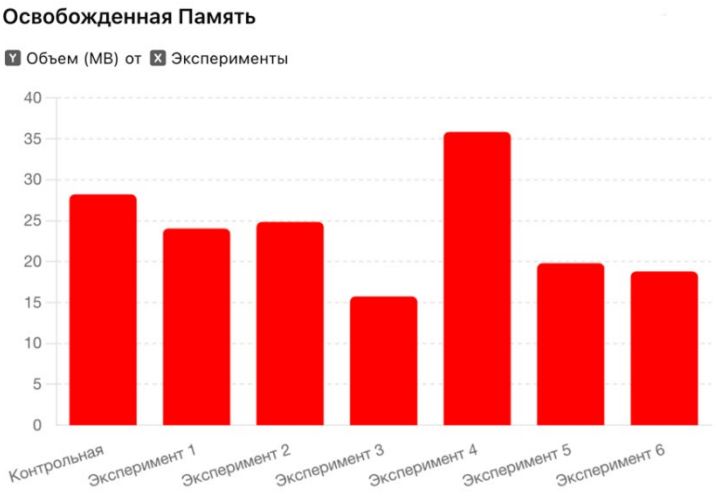

Рисунок 2 – Изменение параметра освобожденной памяти в ходе исследования
Figure 2 – Change in the freed memory parameter during the study

График на Рисунке 2 демонстрирует динамику объема освобожденной памяти (freedMemory). Контрольная сборка освобождала 28,219 MB, что выступает базовым ориентиром для последующих экспериментов. Наибольший объем освобожденной памяти был достигнут в эксперименте 4 (35,856 MB) благодаря перераспределению памяти в пользу молодых регионов. Включение Periodic GC (Эксперимент 6) показало снижение до 18,799 MB.

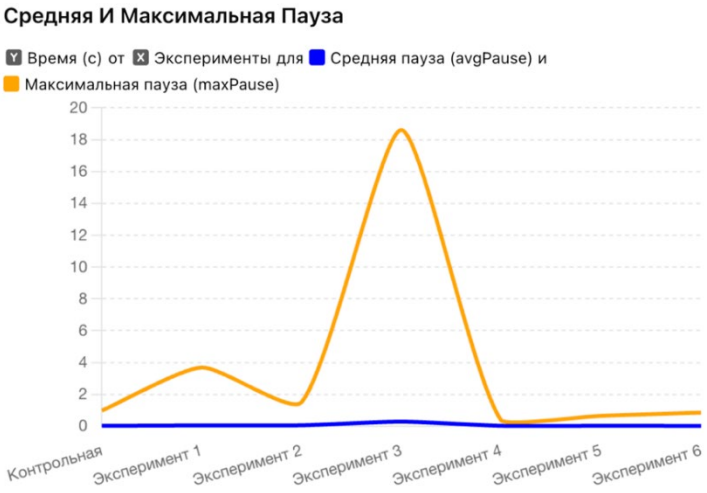

Рисунок 3 – Изменение средней и максимальной паузы в ходе исследования
Figure 3 – Change in average and maximum pause during the study

На графике Рисунка 3 видно, что средняя (avgPause) и максимальная (maxPause) паузы значительно варьировались между экспериментами. Контрольная сборка показала среднюю паузу 0,01988 секунд и максимальную паузу 0,98177 секунд, что является сравнительно сбалансированным результатом. Уменьшение параметра XX:MaxGCPauseMillis (Эксперимент 3) привело к резкому увеличению максимальной паузы до 18,60666 секунд, что делает эту конфигурацию неприемлемой для приложений

с высокими требованиями к задержке. Однако увеличение доли молодых регионов (Эксперимент 4) и включение Periodic GC (Эксперимент 6) обеспечили значительное снижение пауз. В эксперименте 6 максимальная пауза составила 0,85184 секунд, а средняя снизилась до 0,01791 секунд, демонстрируя улучшение по сравнению с контрольной сборкой.

Рисунок 4 – Изменение суммарного количества сборок мусора в ходе исследования
Figure 4 – Change in the total number of garbage collections during the study

График на Рисунке 4 отражает суммарное количество сборок мусора (Minor + Full GC). В контрольной конфигурации было выполнено 1212 сборок, что стало отправной точкой для анализа. Эксперименты показали, что перераспределение памяти в пользу молодых регионов (Эксперимент 4) и включение Periodic GC (Эксперимент 6) позволили значительно уменьшить общее количество сборок до 309. Это свидетельствует о том, что оптимизация распределения памяти и использование Periodic GC могут существенно снизить нагрузку на сборщика мусора.

Обсуждение

Результаты проведенных экспериментов позволяют сделать вывод о значительном влиянии параметров G1 GC на производительность высоконагруженных Java-приложений. Контрольная сборка продемонстрировала базовые показатели, отражающие сбалансированную работу сборщика мусора в условиях стандартной конфигурации JVM. Однако детальный анализ экспериментальных данных выявил как возможности для повышения эффективности G1 GC, так и ограниченные области применения некоторых параметров, требующих дополнительных корректировок.

Изменение размера регионов памяти в первом эксперименте подтвердило гипотезу о снижении частоты младших сборок за счет уменьшения накладных расходов на управление регионами. Однако рост времени обработки более крупных регионов и увеличение пауз GC свидетельствуют о необходимости аккуратного подхода к выбору этого параметра.

Эксперимент с изменением порога заполнения кучи (второй эксперимент) показал, что увеличение этого значения позволяет уменьшить частоту сборок, сохраняя приемлемую пропускную способность. Однако рост средней и максимальной пауз подчеркивает сложность поиска оптимального значения. Эти результаты подтверждают важность адаптивного управления параметром InitiatingHeapOccupancyPercent.

Снижение значения MaxGCPauseMillis (третий эксперимент) продемонстрировало его двойственную природу. С одной стороны, сокращение времени, доступного для выполнения сборки, привело к значительному увеличению числа полных сборок и ухудшению throughput, с другой – подтвердило необходимость совместной настройки с другими параметрами, такими как размер регионов и доля молодых регионов.

Наибольшее улучшение производительности наблюдалось при увеличении доли молодых регионов (четвертый эксперимент) и включении механизма Periodic GC (шестой эксперимент). Эти подходы обеспечили высокую пропускную способность, минимальные паузы и значительное уменьшение частоты сборок. Перераспределение памяти в пользу молодых регионов подтвердило эффективность работы G1 GC в условиях высокой интенсивности создания временных объектов. Включение Periodic GC продемонстрировало преимущества регулярных сборок для минимизации фрагментации памяти и улучшения ее управления, что подтверждается работами, посвященными анализу периодических сборок в системах с большой памятью.

Снижение числа потоков GC (пятый эксперимент) показало неоднозначные результаты. Несмотря на увеличение пропускной способности, рост максимальной паузы указывает на недостаточную эффективность работы в условиях многопоточной нагрузки. Полученные данные подчеркивают необходимость комплексного подхода к настройке G1 GC, включающего балансировку параметров в зависимости от специфики приложений.

Заключение

Результаты проведенного исследования подтверждают критическую значимость параметрической настройки G1 Garbage Collector для обеспечения стабильной и эффективной работы высоконагруженных Java-приложений. В ходе экспериментов было выявлено, что ключевые параметры G1 GC, такие как размер регионов памяти, порог заполнения кучи, максимальная длительность пауз, доля молодых регионов, количество потоков GC и включение Periodic GC, оказывают значительное влияние на производительность и стабильность JVM. Контрольная сборка предоставила базовые показатели, которые послужили отправной точкой для сравнительного анализа. Экспериментальные конфигурации показали, что оптимизация отдельных параметров позволяет достичь значительного снижения длительности пауз, увеличения throughput и улучшения эффективности использования памяти.

Основные выводы исследования включают выявление преимуществ перераспределения памяти в пользу молодых регионов и внедрения механизма Periodic GC, которые обеспечивают высокую пропускную способность и низкие задержки. В то же время было установлено, что агрессивное снижение значения MaxGCPauseMillis и уменьшение количества потоков GC могут приводить к росту максимальных пауз и снижению стабильности системы. Эти результаты подчеркивают необходимость комплексной настройки G1 GC, ориентированной на специфику целевого приложения и его требований к производительности.

Разработанные рекомендации по оптимизации G1 GC включают увеличение доли молодых регионов для приложений с высокой интенсивностью создания временных объектов, осторожный подход к снижению максимального времени пауз, а также использование Periodic GC для минимизации фрагментации памяти. Эти рекомендации имеют практическую значимость для разработчиков высоконагруженных приложений, обеспечивая инструменты для достижения баланса между производительностью и стабильностью.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Sihan W. Java Garbage Collectors. *International Journal of Open Information Technologies.* 2022;10(6):57–61.
2. Zhao W., Blackburn S.M. Deconstructing the garbage-first collector. In: *VEE '20: Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 17 March 2020, Lausanne, Switzerland.* New York: Association for Computing Machinery; 2020. pp. 15–29. https://doi.org/10.1145/3381052.3381320
3. Аникин Д.А. Преимущества автоматизированного управления памятью на примере Java Hotspot. *Столыпинский вестник.* 2022;4(9). URL: https://stolypin-vestnik.ru/wp-content/uploads/2022/11/10-3.pdf
   Anikin D.A. Advantages of automated memory management on the example of Java Hotspot. *Stolypin Annals*. 2022;4(9). (In Russ.). URL: https://stolypin-vestnik.ru/wp-content/uploads/2022/11/10-3.pdf
4. Nisbet A., Nobre N.M., Riley G., Luján M. Profiling and Tracing Support for Java Applications. In: *ICPE '19: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, 07–11 April 2019, Mumbai, India.* New York: Association for Computing Machinery; 2019. pp. 119–126. https://doi.org/10.1145/3297663.3309677
5. Bruno R., Ferreira P. A Study on Garbage Collection Algorithms for Big Data Environments. *ACM Computing Surveys*. 2019;51(1). https://doi.org/10.1145/3156818
6. Pufek P., Grgić H., Mihaljević B. Analysis of Garbage Collection Algorithms and Memory Management in Java. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 20–24 May 2019, Opatija, Croatia.* IEEE; 2019. pp. 1677–1682. https://doi.org/10.23919/MIPRO.2019.8756844
7. Наливайко А.С. Рекомендации по оптимизации потребления памяти в Java. *Молодой ученый*. 2020;(24):59–63.
8. 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.
   Prozorova A.P., Vershinin E.V., Potapova A.E. Impact on the performance of an application with a small amount of memory and the use of Garbage Collector (GC). *Elektronnyi zhurnal: nauka, tekhnika i obrazovanie.* 2019;(1):55–61. (In Russ.).
9. Петров И.А. Верификация кучи памяти объектов виртуальной машины. *Современные информационные технологии и ИТ-образование.* 2021;17(3):603–612. https://doi.org/10.25559/SITITO.17.202103.603-612
   Petrov I.A. Object Memory Verification in Virtual Machines. *Modern Information Technologies and IT-Education.* 2021;17(3):603–612. (In Russ.). https://doi.org/10.25559/SITITO.17.202103.603-612
10. Филатов А.Ю., Михеев В.В. Анализ эффективности потоково-локальной сборки мусора в распределённых системах хранения и обработки данных. *Вестник СибГУТИ*. 2022;(1):77–88. https://doi.org/10.55648/1998-6920-2022-16-1-77-88
    Filatov A.Yu., Mikheev V.V. Application of Thread-Local Garbage Collection to Distributed Systems for Large-Scale Data Processing. *Vestnik SibGUTI.* 2022;(1):77–88. (In Russ.). https://doi.org/10.55648/1998-6920-2022-16-1-77-88

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Золотухина Дарья Юрьевна, ведущий разработчик, Финансовая корпорация Открытие, Воронеж, Российская Федерация.
e-mail: dar.zolott@gmail.com

Daria Yu. Zolotukhina, lead software developer, Financial Corporation Otkritie, Voronezh, the Russian Federation.

Статья поступила в редакцию 16.12.2024; одобрена после рецензирования 23.12.2024; принята к публикации 26.12.2024.

The article was submitted 16.12.2024; approved after reviewing 23.12.2024; accepted for publication 26.12.2024.