Исследование работы сборщика мусора java

Характеристики оборудования: 8Гб оперативной памяти, процессор intel core i5 8265u.

Опции JVM:

-Xms2048m
-Xmx2048m
-Xlog:gc=debug:file=./logs/gc-%p-%t.log:tags,uptime,time,level:filecount=5,filesize=10m
Опционально:
-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:+UseConcMarkSweepGC
-XX:+UseG1GC

Результаты.

Таблица 1. Получена на основании лога работы программы

	Момент завершения последнего цикла вычислений, сек	размер immortalList	Количество созданных объектов	Момент замера, сек	размер immortalList на момент замера	Количество созданных объектов на момент замера
Using G1	135,761	14480000	159280000	100,035	14320000	157520000
Using Parallel	170,2	14000000	154000000	100,544	12900000	141900000
Using Concurrent Mark Sweep	126,54	11100000	122100000	101,093	11070000	121770000
Using Serial	178,873	13990000	153890000	100,078	13070000	143770000

Таблица 2. Получена на основании *-Xlog:gc*

	Accumulated pauses	number of pauses	Accumulated gc pauses	full gc pauses min	full gc pauses max	Время до OutOfMemoryError
Using G1	100,69	220	98,62	1,06144	1,57733	191.300
Using Parallel	108,16	72	105,93	1,26441	2,77963	177.550
Using Concurrent Mark Sweep	198	128	189,23	2,30832	4,34666	292.602
Using Serial	255	142	251,88	1,68783	2,39532	347.550

График 1. Количество созданных объектов MemoryFillObject в зависимости от времени мс.

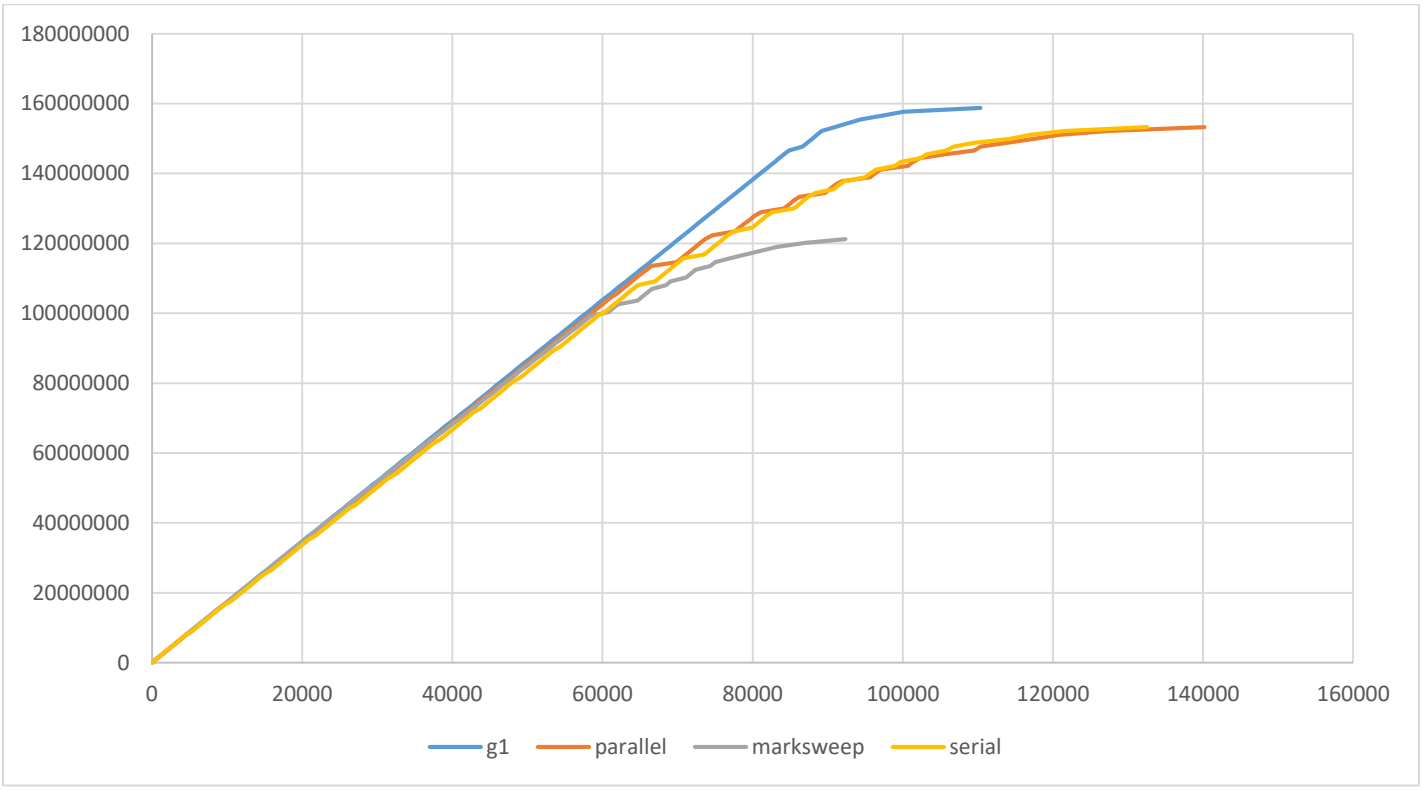
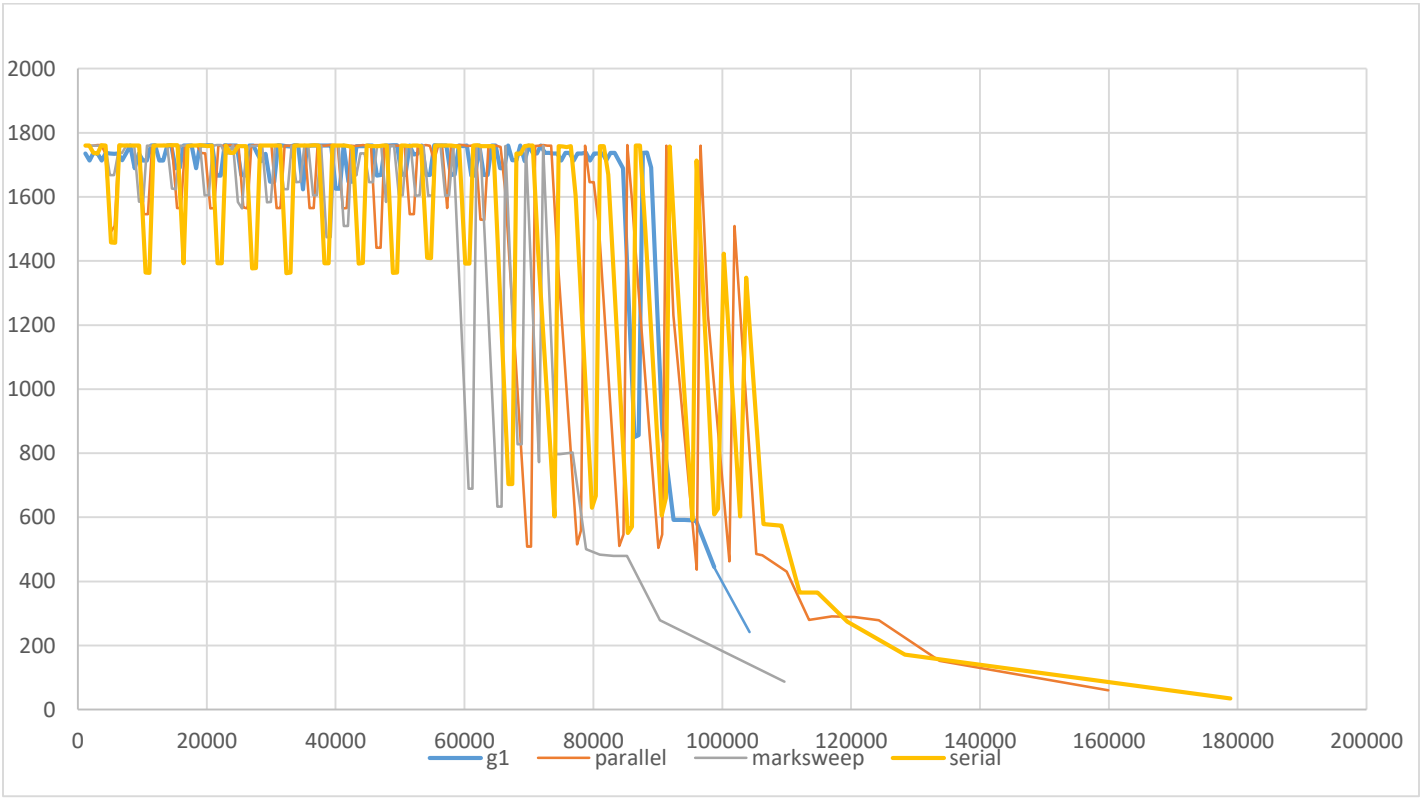


График 2. Усредненная производительность в зависимости от времени (количество созданных объектов в мс).



Выводы

Согласно графикам, можно выделить три этапа работы программы:

1) стабильный этап;
2) этап активной сборки мусора, когда производительность приложения начинает проседать, но иногда восстанавливается. Этот этап хорошо виден на графике 2 между 60000 и 100000 мс.
3) Завершающий этап, когда производительность падает катастрофически.

На первом этапе стабильнее всего G1 collector, а худшим вариантом является использование serial collector.

Serial collector и Parallel Collector немного отстают по производительности от G1, при этом они переходят на этап 2 раньше чем G1, но дольше на нем задерживаются. Parallel Collector гораздо раньше выбрасывает OutOfMemoryError, что делает его предпочтительнее чем Serial collector.

Худший по производительности CMS collector первым достигает этапа 2 на 60 секунде, а этапа 3 на 80-й, при этом прекращение работы программы происходит только на 292 секунде.

В целом Garbage-First (G1) Collector выглядит наиболее предпочтительным вариантом для использования.

При использовании Garbage-First (G1) Collector:

1) достигнута лучшая производительность на момент отсечки (примерно 100 сек), так и за время работы программы;
2) наименьшие максимальные паузы и наименьшее общее время пауз, при большем их количестве;
3) второе время до остановки работы программы с OutOfMemoryError.

На втором месте Parallel Collector , который проигрывает по производительности и паузам, но имеет следующие преимущества:

1) приложение немного позже переходит к завершающему этапу;
2) быстрее завершает работу приложения, 177,5 сек против 191,3;

Таким образом при использовании Parallel Collector минимизируется время когда сборщик мусора занимает все ресурсы и программа не выполняет полезной работы.