Никита Бегишев, ФКН ПМИ

Сортировка во внешней памяти

Алгоритм

На каждом шаге считываются данные, размером с заданный буфер за раз, формируется Priority Queue, над отдельными run-ами. Run-ом я называю последовательность отсортированных чисел. Получается что на каждом шаге используется M / B run-ов, минус два на буфер для чтения и записи.

На первом шаге считываемые буфером данные сортируются, на всех последующих считываемые данные будут уже отсортированы.

Priority Queue содержит пары объектов с указателем на очередь run-а и первое число в очереди run-а, которое из самой очереди извлекается. Пока Priority Queue не пуста, каждый раз выбирается общий минимум, если очередь run-а закончилась, то производится попытка прочитать следующую часть данных, и заполнить очередь, если границы run-а не достигнуты. Отсортированные данные помещаются в буфер на запись, как только буфер на запись насыщен элементами, производится запись, и буфер очищается.

В программе проверяется ограничение, что заданный буфер хотя бы в 4 раза меньше чем лимит по памяти, это гарантирует, что на каждом шаге будет хотя бы 2 run-а, что в итоге отсортирует весь файл за конечное число шагов.

Для удобства сравнения со встроенной утилитой, в программе есть опция сортировки данных в текстовом виде, что автоматически и выводит результат сортировки также в текстовом виде. Такие данные преобразовываются предварительно в бинарный вид.

Характеристика железа

Воспользуемся утилитой, написанной в первом домашнем задании:

```
Avg read speed: 2103.31 MiB/s, StdDev: 133.95 MiB/s.
Avg write speed: 993.18 MiB/s, StdDev: 48.09 MiB/s.
Avg read latency: 126.69 µs, StdDev: 28.81 µs.
Avg write latency: 113.46 µs, StdDev: 8.31 µs.
Avg read latency: 106.86 µs, StdDev: 1.49 µs.
Avg write latency: 179.41 µs, StdDev: 15.29 µs.
Avg read+write latency: 116.77 µs, StdDev: 17.65 µs.
```

RAM: 16 GB 2133 MHz LPDDR3
SSD: 512GiB

Диск, конечно, быстрый, но как будет видно далее, более важный ресурс для сортировки всё же CPU, что не даст в итоге ультра быструю сортировку, как я ожидал по началу.

Измерение времени работы алгоритма на файле размером 20GiB

```
> time ./sort bin-input-20GiB bin-output-20GiB --ml 1GiB --bs 4MiB
```

Сортировка 20GiB данных в бинарном виде, заняла 12 минут и 16 секунд.

```
> time ./sort bin-input-100GiB bin-output-100GiB --ml 1GiB --bs 8MiB
```

Сортировка 100GiB данных в бинарном виде, заняла 58 минут и 23 секунды.

Подробное исследование для файла меньшего размера (1GiB)

B	K			
	126	**62**	**30**	**14**
16MiB	0m22s	0m28s	0m27s	0m26s
8MiB	0m32s	0m28s	0m27s	0m26s
1MiB	0m28s	0m25s	0m29s	0m29s
256KiB	0m27s	0m43s	0m37s	0m33s

В общем случае выходит, что увеличение размера блока уменьшает время сортировки, а уменьшение K где-то увеличивает, а где-то уменьшает время. Связано это с тем, что с одной стороны увеличение степени ветвления тормозит процесс, однако если степень достаточно мала, то это приводит к появлению дополнительных шагов сортировки. Также, на таких небольших значениях, влиять на уменьшать время может кэш, особенно при первоначальных сортировках.

В каких пропорциях время тратиться между CPU и I/O

Методика вычисления: это конечно, грубая оценка, но всё равно весьма информативная, у утилиты time я использую время user как CPU, а real как всё затраченное время. Операции с диском управляются ядром, поэтому они будут складываться в sys. Остальными мелочами, кажется, можно пренебречь.

(CPU / IO)

B	K			
	126	**62**	**30**	**14**
16MiB	86% / 14%	80% / 20%	79% / 21%	81% / 19%
8MiB	76% / 24%	79% / 21%	79% / 21%	81% / 19%
1MiB	83% / 17%	84% / 16%	81% / 19%	77% / 23%
256KiB	85% / 15%	62% / 38%	67% / 33%	77% / 23%

Здесь нетрудно заметить, что увеличение размера блока увеличивает долю CPU, это вполне очевидно, так как увеличение блока будет увеличивать среднюю скорость чтения с диска, тогда диск использует меньше времени.

Хуже прослеживается зависимость K и процента процессорного времени, но зависимость есть: чем больше ветвление, тем больше времени затрачивает CPU.

Стандартная утилита sort против собственной реализации

Тут сравнивать будем на текстовых данных, собственной реализации придется потратить лишнее время на преобразования данных в бинарный формат и обратно, но алгоритмически она должна всё равно работать эффективнее, посмотрим:

```
> time sort -n text-input-10GiB -S 1G > text-output-10GiB-std
real    9m38.545s
user    6m12.374s
sys     2m10.948s

> time ./sort text-input-10GiB text-output-10GiB --ml 1GiB --bs 8MiB
real    6m55.632s
user    5m32.703s
sys     1m5.884s
```

Выходит, что собственная реализация работает быстрее, 7 минут против 9 минут 40 секунд.