АДАПТИВНОЕ ОБЪЕДИНЕНИЕ ЗАПРОСОВ ДЛЯ RAID-МАССИВА НА ОСНОВЕ ТВЕРДОТЕЛЬНЫХ НАКОПИТЕЛЕЙ

Васенина А. И., студент кафедры системного программирования СПбГУ, младший разработчик исследовательской лаборатории RAIDIX, Vasenina.A@raidix.com

Аннотация

Одним из основных недостатков RAID-массивов на основе контрольных сумм являются дорогостоящие RMW-операции. Для последовательной записи количество этих операций можно существенно сократить используя объединение запросов.

В данной работе рассматривается алгоритм, управляющий объединением входящих запросов записи на основании анализа входящей нагрузки.

Введение

Устройства хранения данных постоянно развиваются. Помимо объема, существенной характеристикой накопителя является скорость доступа к данным. Современные твердотельные накопители способны обрабатывать во много раз большее количество операций в секунду, чем их предшественники. Рынок высокопроизводительных дисков быстро растет и производителям систем хранения данных приходится находить новые решения, чтобы удовлетворять требованиям потребителей.

Одной из наиболее распространенных технологий в области систем хранения данных являются RAID-массивы [1]. Избыточность хранения данных, а, следовательно, и отказоустойчивость может достигаться в RAID-массивах двумя возможными путями:

1. Зеркалированием
2. Хранением контрольных сумм

В первом случае участвуют два диска. при этом один из них становится точной копией другого, таким образом объем памяти RAID 1 сокращается в половину от используемых накопителей.

Во втором случае RAID делится на полосы данных, в каждой из которых есть сегменты данных и кодовые сегменты. При этом число кодовых сегментов фиксировано, то есть для RAID 6 с двумя кодовыми сегментами мы получаем уменьшение итогового объема RAID-массива на два объема используемых накопителей. Такая экономическая выгодность

RAID-массивов на основе контрольных сумм особенно актуальна для дорогостоящих высокопроизводительных дисков.

Основным недостатком RAID-массивов на основе контрольных сумм является наличие тяжеловесных операций чтения/записи (в дальнейшем RMW-операций [2]), которые возникают при записи в полосу данных блока размером меньше длины полосы данных. При этом для RAID-массива с хранением двух контрольных сумм одна операция записи превращается в 3 операции чтения и 3 операции записи.

Объединение запросов

Существенно сократить количество RMW-операций для последовательной записи позволяет объединение запросов. Допустим, в полосу данных RAID-массива на 4-х дисках с 1 контрольной суммой последовательно поступает три запроса длиной в сегмент полосы данных. В таком случае, обрабатывая эти операции по отдельности мы получим по 4 операции на каждый запрос. В итоге 12 запросов.

Если же при поступлении первого запроса мы бы не стали сразу начинать обработку, а дождались двух других запросов, то RMW-операция не была бы необходима: мы могли бы вычислить контрольную сумму сразу же по записываемым сегментам. Таким образом выполнив только 4 операции записи (см. Рис. 1).

Рисунок 1: Сокращение числа выполняемых операций за счет объединения запросов

В операционных системах Linux задача объединения запросов возлагается на планировщики ввода-вывода [3]. В области планирования

постоянно ведутся исследования и появляются новые разработки. Увеличение скорости накопителей привело к переходу от единой очереди (blk-sq), используемой такими планировщиками как noop, deadline и cfq, к технологии blk-mq [4], заменяющей одну очередь на количество очередей, соответствующих количеству ядер центрального процессора.

Для исследования вопроса применимости планировщиков в качестве решения, было проведено тестирование предела производительности планировщиков ввода-вывода. Тестирование проводилось на блочном устройстве с отсутствующей логикой обработки запроса: на каждый поступающий запрос это блочное устройство сразу возвращало команду успешного завершения запроса.

Такой тест позволяет узнать скорость поступления запросов в блочное устройство, а следовательно и скорость планировщика запросов. Тестирование производилось при помощи FIO в количество потоков 32, каждый из которых имел глубину очереди 32. Результаты тестирования приведены на рис. 2.

Тип нагрузки	none	blk-sq			blk-mq		
		noop	deadline	cfq	none-mq	mq-deadline	kyber
Случ. чтение	7273	348	323	3240	3130	352	2971
Случ. запись	5411	350	322	3296	2983	352	2924
Посл. чтение	7021	345	321	190	3079	355	2920
Посл. запись	5111	346	320	192	2952	352	2875

Рисунок 2: Тестирование предела производительности стандартных планировщиков ввода- вывода. Результаты представлены в тысячах операций в секунду

Как можно увидеть по результатам тестирования, использование планировщиков ввода-вывода, даже таких как noop, не выполняющих перестановку или объединение запросов, существенно снижает производительность системы, по сравнению с использованием блочного устройства без планировщика (none). При этом данная производительность стала бы лимитом для нашей системы, поэтому данное решение непригодно для высокопроизводительной системы. В связи с этим было принято решение о разработке собственного алгоритма объединения последовательных запросов записи на основе красно-черных деревьев и управляющего этим процессом алгоритма на основе анализа входящей нагрузки.

Эффективность объединения последовательных запросов записи

Для случайной записи объединение запросов не имеет смысла, так как ожидание не приводит к появлению новых запросов рядом с исходным и уходит впустую, только задерживая первый запрос.

Для определения эффективности объединения запросов была проведена серия тестов для различных конфигураций последовательной записи. На рис. 3 вы можете видеть результаты для записи блоком 4 килобайта и для записи блоком 8 килобайт. При этом по горизонтали меняется количество потоков, а по вертикали глубина очереди. Более темным цветом отмечены меньшие результаты по производительности при сравнении результатов без объединения запросов и результатами с объединением запросов. Тесты проводились на RAID 6 с 6-ю дисками. То есть длина сегментов данных в полосе данных была равна 64.

Паттерн 1: размер блока 4k

Глубина очереди \ Количество потоков	1	8	32
1	29.9	214	569.3
4	62.8	408.6	990.2
8	111.6	543.7	1241
16	134.1	664.6	1431.5
32	165.9	1016.8	2023.4

Без объединения запросов

Паттерн 2: размер блока 8k

Глубина очереди \ Количество потоков	1	8	32
1	38.9	280.6	643.1
4	77.8	491.5	1068
8	134.1	690.2	1285.1
16	200.7	849.9	1598.5
32	258.1	1257.5	2193

Без объединения запросов

Глубина очереди \ Количество потоков	1	8	32
1	20.3	154.6	446.5
4	31.3	216.1	699.4
8	50.1	317.4	966.6
16	333.8	3818.5	6347.8
32	416.8	5065.7	6369.3

С объединением запросов

Глубина очереди \ Количество потоков	1	8	32
1	27.9	206.8	632.8
4	46.7	314.4	936.9
8	547.1	4595.7	6346.7
16	576.5	5732.3	6392.8
32	735.2	6328.3	6429.7

С объединением запросов

Рисунок 3: Тестирование эффективности объединения запросов для последовательной записи. Результаты в Mb/s

Из серии этих тестов мы можем вывести следующее правило эффективности объединения запросов: глубина очереди (QD), умноженная на размер блока (BS) должна быть больше или равна длине сегментов данных в полосе данных (stripe data len).

$$QD*BS >= \text{stripe data len}$$

Алгоритм

Зонирование RAID-массива

В реальных условиях часто встречаются случаи, когда разные части RAID-массива действуют разные типы нагрузки. Ввиду этого, определять необходимость объединения запросов выгоднее не для всего RAID-массива, а для некоторой его области. Для этого разделим все полосы данных в RAID-массиве на равные группе. Каждой группе будет соответствовать свой бит в битовой карте, ответственной за осуществление объединения запросов. Значение каждого бита будет определяться нашим анализатором входящей нагрузки. При этом каждому выставленному биту мы можем доверять только определенный промежуток времени, поэтому каждому биту в битовой карте соответствует элемент из массива времен обновления битовой карты (см. Рис. 4).

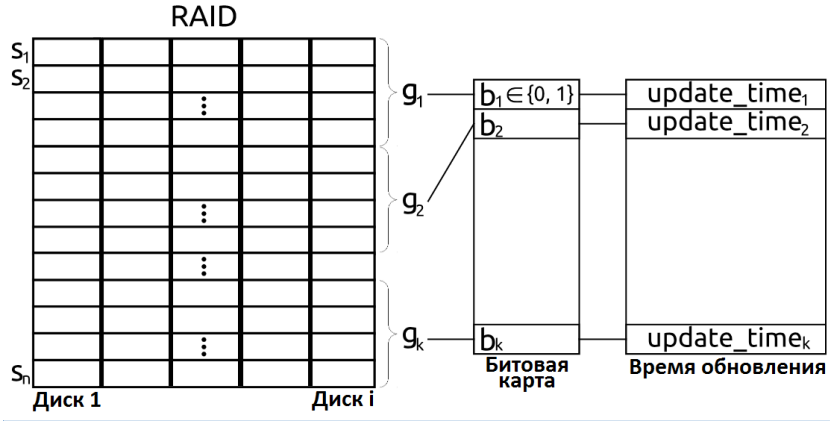
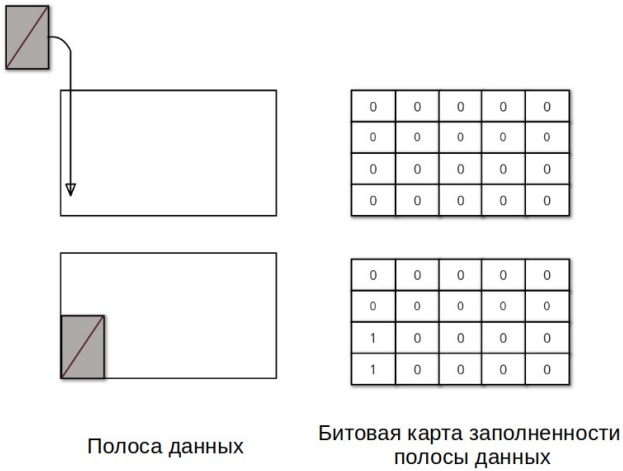


Рисунок 4: Разделение RAID-массива на группы

Детектор последовательной нагрузки

При поступлении запроса записи для этого запроса формируется виртуальная структура полосы данных. При этом мы создаем в этой структуре битовую карту, в которой каждому биту соответствует 4 килобайта (минимальная длина запроса к твердотельному накопителю). По мере поступления запросов мы отмечатем, в какие участки полосы данных были получены запросы записи (см. рис. 5).



Риунок 5: Процесс заполнения битовой карты заполненности полосы данных

Так как структура полосы данных существует только пока существуют необработанные запросы, направленные в эту полосу данных, мы можем с высокой уверенностью утверждать, что если битовая карта полностью заполнилась единицами, то в зону, которой принадлежит эта полоса данных, осуществляется последовательная запись.

Детектор скорости поступления запросов

Наше время ожидания для запроса всегда варьируется между нулем и максимумом, задаваемым требованиями к системе. При этом ноль отвечает за отсутствие ожидания запросов. Так что для описания следующего детектора будем считать, что t всегда больше нуля.

Мы считаем, что скорость поступления запросов в полосу данных в идеальной ситуации равномерна. Черная полоса на графике на рис. 6 обозначает минимальную скорость, с которой запросы должны поступать в полосу данных, чтобы мы успели накопить полную полосу данных до истечения максимального времени. Теперь оценим скорость поступления запросов для двух примеров: для точки (t1,a) текущая скорость выше нашей теоретически необходимой, а для точки (t2,b) ниже. Таким образом, для точки (t1,a) мы делаем вывод, что успеем накопить полную полосу данных и должны продолжить ожидание. Для точки (t2, b), напротив, накопить полную полосу данных мы не сможем, поэтому ждать дальше не имеет смысла. Будем называть этот случай неудачным ожиданием.

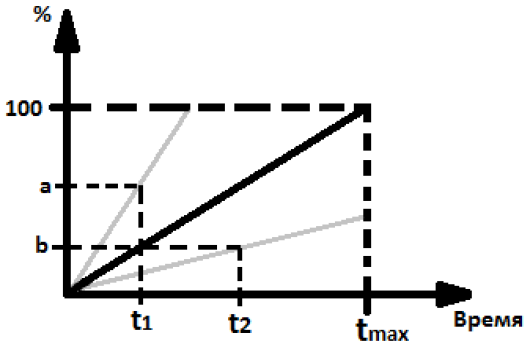


Рисунок 6: Теоретическая оценки скорости заполнения полосы данных

Если бы скорость заполнения полосы данных всегда была постоянна, то мы бы могли на основании одного неудачного ожидания определять отсутствие необходимости объединения запросов. На практике же скорость поступления запросов не постоянна и функция заполненности полосы данных приобретает ступенчатый вид, что не дает нам однозначно определять эффективность объединения запросов на основании одного измерения. Поэтому решение об отсутствии необходимости применения объединения запроса принимается при наличии некоторой доли неудачных ожиданий в группе.

Заключение

Комбинация представленных алгоритмов позволяет применять объединение запросов только в случае, где это обеспечит выигрыш в производительности, то есть для быстрой последовательной записи.

Литература

1. EMC Education Services. От хранения данных к управлению информацией. — 2016.
2. Abhishek Singhal Rob Van der Wijngaart Peter Barry. Atomic Read Modify Write Primitives for I/O Devices. — Intel White Paper. — 2008.
3. Robert Love. Linux Kernel Development, Third Edition. — 2010.
4. Matias Bjørling Jens Axboe† David Nellans† Philippe Bonnet. Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems. — 2013.