## Алгоритм обработки запроса на запись

* 1. На вход поступает пакет (*LBA*, *data*). *LBA* - внешний адрес записи блока (6 байт), *data* - массив с пользовательскими данными
  2. Массив *data* разбивается на равные блоки (их количество пусть будет *BN*) по размеру блока дедупликации (*DBS*)
  3. Делается запрос в *PackageT* по *LBA*, где *PackageT* - таблица сопоставления внешних и внутренних адресов.   
     *PackageT* возвращает либо массив из BN элементов вида (*hash*, *iLBA*), либо пустой массив. *hash* - это значение хэш-функции для блока дедупликации (6 байт), *iLBA* - внутренний адрес блока данных на бесконечной ленте (которая является абстракцией диска) (5 байт)  
     Если получен не пустой массив, тогда он расширяется ещё одним булевым флагом для каждого элемента
  4. Для каждого подблока (размера *DBS*) исходных данных выполняются следующие шаги. Всего итераций *BN*
     1. Вычисляется хэш этого блока (*hash*)
     2. Делается запрос в *DedupT* по значению *hash*, где *DedupT* - таблица дедупликации  
        *DedupT* возвращает либо информацию о существующем блоке (*iLBA*, *c*) с таким же *hash*, либо ничего не возвращает. Значение *c* является счётчиком, который показывает, в скольких *LBA* этот блок дедупликации используется  
        Пока нет решения по поводу того, что делать со счётчиком и возможностью записать весь диск нулями
     3. Если на шаге 1.4.2 вернулось пустое значение, то в этом случае блок считается новым и производится попытка добавления этого блока  
        Вызывается функция добавления блока у объекта *Reserved Chunk*. Этот объект имеет внутреннее состояние, которое резервирует пространство на диске (ленте). Резервирование выполняется за счёт хранения номера блока (отступа на ленте) (*chi*), при этом гарантируется, что никакой другой контроллер не хранит этот же самый номер  
        **Очередь.** В первую очередь *Reserved Chunk* проверяет очередь сборщика мусора, которая хранит список адресов *iLBA*, доступных для перезаписи. Если очередь не пуста, то берётся и удаляется первый элемент (*qiLBA*) из этой очереди (*q* означает, что это *iLBA* из очереди). Отправляется запрос на запись на диск данных (*qiLBA, data [i]*). В качестве результата возвращается *qiLBA*.  
        **Чанк.** Если очередь оказалась пустой, то объект *Reserved Chunk* проверяет наличие места в чанке. Чанк имеет фиксированный размер (*RCHS*), можно хранить количество блоков (*cbs*), которые были записаны от начала, тогда проверка сводится к тому, чтобы проверить, что количество блоков в чанке меньше возможного количества блоков в чанке. Если место закончилось, то выполняется резервирование нового чанка (при этом *cbs* становится 0). Если место есть, то блок записывается в конец, значение *cbs* увеличивается на 1. В качестве результата возвращается *chi \* RCHS + cbs \* DBS.* Информация об увеличении счётчика распространяется с помощью широкого вещания на другие контроллеры.  
        **Завершение.** После записи блока на диск вызывается функция добавления информации в *DedupT* с аргументами (*hash*, *iLBA*, *1*).
     4. Если на шаге 1.4.2 вернулось не пустое значение, то в этом случае блок считается дедуплицированным (уже есть на диске), и далее возможны два варианта. Блок мог измениться и остаться прежним, это можно определить, сравнив *iLBA* из *DedupT* и *iLBA* из *PackageT*.  
        **Блок не изменился.** Блок содержит те же самые данные, что были записаны ранее (случай, когда *iLBA* совпали). Тогда мы просто поднимаем флаг в массиве (по индексу текущего блока в цикле), который мы расширили в пункте 1.3  
        **Блок обновился.** Блок содержит новые данные, но уже дедуплицированные (случай, когда iLBA не совпали). Это по определению является новым использования блока дедупликации по конкретному LBA, поэтому мы должны будем увеличить значение счётчика (*c*) на 1 для блока с таким *hash-*значением. Вызывается функция увеличения счётчика по *hash*-значению у *DedupT*.
     5. Информация о блоке дедупликации (*hash*, *iLBA*) добавляется в массив по индексу текущего блока в цикле. Этот массив (после цикла) будет записываться в *PackageT*.
  5. Полученный массив с информацией про (*hash*, *iLBA*) для каждого блока дедупликации записывается в *PackageT*. Размер массива также *BN* элементов по 6 + 5 байт.
  6. В случае, если в пункте 1.3 *PackageT* вернула не пустой массив, запускается цикл по всем элементам этого массива (расширенной версии с флагом).
     1. Если флаг элемента массива не выставлен, то это означает, что этот элемент больше не используется, поэтому необходимо уменьшить значение счётчика в *DedupT*
  7. На этом обработка запроса записи завершена

## Алгоритм обработки запроса на чтение

* 1. На вход поступает пакет (*LBA*).
  2. Делается запрос в *PackageT* по *LBA*. *PackageT* возвращает либо массив из BN элементов вида (*hash*, *iLBA*), либо пустой массив
  3. Если в пункте 2.2 вернулся пустой массив, то это означает, что блок данных по переданному *LBA* никогда не был записан
  4. Если в пункте 2.2 вернулся не пустой массив, необходимо для каждого элемента этого массива (*hash*, *iLBA*) сделать запрос на чтение с диска по *iLBA*. После этого, прочитанные блоки (равные по размеру DBS), необходимо в том же порядке конкатенировать и отдать в том же порядке, в котором они быле в результате от *PackageT*
  5. На этом обработка запроса чтения завершена

## Алгоритм сбора мусора

* 1. Алгоритм запускается через некоторые промежутки времени в отдельном потоке, который является демоном
  2. Алгоритм пробегает по содержимому *DedupT* и смотрит на значение счётчика *c* у элементов (количество использований блока), пока не выполнится критерий останова
  3. Если значение счётчика с опускается до 0, то это означает, что блок больше нигде не используется и его можно перезаписать
  4. Значение *iLBA* элемента со значением счётчика 0 помещается в очередь, описанную в пункте 1.4.3. При этом физически с записью на диске ничего не происходит
  5. Если размер очереди из *iLBA* достиг предельного значения (например, 1000), то алгоритм поиска “мусора” уходит в спячку. Также он уходит в спячку, если прошёл целый круг по записям из *DedupT* (случай, когда нет “мусора”)
  6. В данном случае можно запускать демонов, которые будут искать iLBA с “мусором” так, что *iLBA % N == n*, то есть, чтобы найденные iLBA, доступные для перезаписи, могли быть использованы только соответствующим контроллером, номер которого (*n*) равен остатку при деления *iLBA* на общее число контроллеров (*N*). Таким образом будет гарантироваться то, что не будет конкуренции за один и тот же *iLBA*

## Алгоритм резервирования чанков

* 1. Алгоритм запускается, когда текущий чанк у контроллера заполнен. Алгоритм запускается в том же потоке и является блокирующей операцией
  2. Контроллер рассылает оповещения на все остальные контроллеры по UDP, датаграмма имеет следующий формат “code, controller id” (2 байта). В данном случае code всегда равно 1
  3. После этого контроллер переходит в режим ожидания с таймаутом. Во время этого режима контроллер слушает ответы от других контроллеров и записывает их в массив. Ответы имеют формат “code, chunk id, other controller id”. Значение code всегда равно 2. Значение chunk id – номер текущего чанка, зарезервированный у контроллера other controller id. При этом значение other controller id может быть положительным и отрицательным, но абсолютное значение одинаково. Отрицательное значение other controller id означает, что у этого контроллера тоже закончился чанк и он находится в процессе резервирования.
  4. Если значение от контроллера не было получено, то оно считается 0
  5. После того как все значения получены или вышло время, если все ответы были положительны или их абсолютные значения больше номера текущего контроллера, то контроллер находит максимум из ответов и добавляет 1. Если были отрицательные other controller id, которые по абсолютному значению меньше номера текущего контроллера, то текущий контроллер делает паузу на случайное время до 100 мс и повторяет попытку с пункта 4.1.

