Санкт-Петербургский государственный политехнический

университет

Институт информационных технологий и управления

Кафедра распределенных вычислений и компьютерных сетей

Работа допущена к защите

Зав. кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ю. Г. Карпов

“\_\_\_“\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2014г.

**ВЫПУСКНАЯ РАБОТА БАКАЛАВРА**

Тема: Исследование и разработка метода выбора размера слота в кэше на основе параметров рабочей нагрузки

Направление: 230100.62 – Информатика и вычислительная техника

Выполнил студент гр. 43507.10 А. А. Суворов

Руководитель, к.т.н., доцент. П. В. Трифонов

Санкт-Петербург

2014

Утверждаю

Зав. кафедрой, д.т.н., проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Карпов Ю.Г. «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2014г.

**З А Д А Н И Е**

**на бакалаврскую работу**

студента гр. 43507.10 А. А. Суворова

1. Тема работы: *Исследование и разработка метода выбора размера слота в кэше на основе параметров рабочей нагрузки*

(утверждена распоряжением по институту от \_\_\_\_\_\_\_№\_\_\_\_\_\_\_\_\_\_)

1. Срок сдачи работы 20 июня 2014.
2. Исходные данные к работе: целью работы является повышение производительности систем хранения данных, использующих кеширование. Необходимо реализовать симулятор кэш-памяти, моделирующий работу LRU-алгоритма на произвольно заданном потоке адресов обращения, а также осуществляющий подсчет его характеристик. Симулятор должен позволять исследовать зависимость эффективности работы кэш-памяти от размера страницы. Необходимо исследовать взаимосвязь между этой зависимостью и параметрами рабочей нагрузки данных потоков адресов. На основе методов машинного обучения необходимо построить алгоритм оценки оптимального размера страницы кэш памяти в зависимости от параметров рабочей нагрузки. На тестовом множестве потоков адресов необходимо оценить процент ситуаций, в которых оценка не оптимальна.
3. Содержание расчетно–пояснительной записки (перечень подлежащих разработке вопросов):
4. Обзор принципов организации кэш-памяти и методов выбора ее параметров.
5. Характеристики рабочей нагрузки на СХД.
6. Описание симулятора кэш-памяти.
7. Алгоритм выбора размера страницы кэш памяти в зависимости от параметров рабочей нагрузки.
8. Оценка качества работы построенного алгоритма.
9. Перечень графического материала с (точным указанием обязательных чертежей): графики зависимости вероятности попадания от размера кэш страницы для различных потоков адресов, промежуточные графики зависимости вероятности попадания от различных характеристик нагрузки.
10. Консультанты по проекту (с указанием относящегося к ним разделов проекта, работы): Фомин А.Д. инженер-консультант. к.т.н.

Дата выдачи задания “\_\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2014 г.

Руководитель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н., доцент. П. В. Трифонов.

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Суворов.

# Реферат

С. 52. Рис. 17. Табл. 5

В данной работе рассматривается проблема выбора параметров кэш-памяти для систем хранения данных (СХД), в зависимости от приложенной нагрузки, на примере размера страницы. Описан классический способ решения данной задачи – моделирование на потоке адресов; приведены его недостатки, а также предложен новый подход, основанный на методах машинного обучения. Построен алгоритм оценки оптимального размера страницы; приведено описание методов его построения, вместе с результатами тестирования и оценкой точности.

Для получения обучающей и тестовой выборок были созданы симулятор кэш-памяти, и ряд программ для автоматизации процесса, описание которых также приведено в работе.

# Abstract

pages 52. pictures 17. tables 5.

The problem of workload based parameters selection for cache memory is considered. This paper describes classical approach for tackling this problem (trace driven modeling) with its limitations and proposes new approach, based on machine learning techniques. Algorithm for estimation optimal page size is provided. Methods, which was used for building this algorithm, and testing results is described too.

For obtaining data set and test set, special software simulator, together with automation scripts were used.

# Оглавление

[Реферат 4](#_Toc391346084)

[Abstract 5](#_Toc391346085)

[Оглавление 6](#_Toc391346086)

[Список обозначений 7](#_Toc391346087)

[Введение 8](#_Toc391346088)

[1. Терминология и постановка задачи 11](#_Toc391346089)

[1.1. Характеристики нагрузки 11](#_Toc391346090)

[1.2. Архитектура систем хранения данных 16](#_Toc391346091)

[1.3. Принципы организации кэш-памяти 18](#_Toc391346092)

[1.4. Моделирование на потоке адресов 21](#_Toc391346093)

[2. Машинное обучение 23](#_Toc391346094)

[2.1. Задача классификации 23](#_Toc391346095)

[2.2. Логистическая регрессия 24](#_Toc391346096)

[2.3. Отбор признаков 26](#_Toc391346097)

[2.4. Решающие деревья 29](#_Toc391346098)

[3. Симулятор кэш-памяти 30](#_Toc391346099)

[3.1. Общая схема обработки потока адресов 30](#_Toc391346100)

[3.2. Архитектура симулятора 33](#_Toc391346101)

[3.3. Интерфейс симулятора 37](#_Toc391346102)

[4. Анализ данных 38](#_Toc391346103)

[4.1. Реализация алгоритмов классификации 38](#_Toc391346104)

[4.2. Оценка точности алгоритмов 41](#_Toc391346105)

[Заключение 46](#_Toc391346106)

[Приложение 1. Пример потока адресов 48](#_Toc391346107)

[Приложение 2. Пример описания нагрузки 50](#_Toc391346108)

# Список обозначений

|  |  |
| --- | --- |
| СХД | Система хранения данных |
| LRU | Least recently used – алгоритм кэширования, при котором в первую очередь вытесняются данные, которые дольше всего не запрашивались |
| ПО | Программное обеспечение |
| RAID | Redundant array of independent disks - избыточный массив независимых дисков. |
| SSD | Solid state drive - Твердотельный накопитель – немеханическое запоминающее устройство на основе микросхем памяти. |
| HDD | Hard (magnetic) disk drive. Жесткий диск. Запоминающее устройство, основанное на принципе магнитной записи. |
| SCSI | Small computer system interface – набор стандартов для физического подключения и передачи данных между компьютерами и периферийными устройствами. |
| LUN | Logical unit number – адрес виртуального логического устройства в СХД (это может быть диск, виртуальный раздел в RAID массиве и др.), используемый для команд SCSI. |
| IOPS | Input/output operations per second. Количество операций ввода-вывода в секунду |
|  |  |
|  |  |
|  |  |
|  |  |

# Введение

В настоящее время наблюдается значительное увеличение объема производимой по всему миру информации. Поставщиками этих данных является большой спектр приложений - от социальных сетей и мобильных приложений, до крупных научных экспериментов (например, LHC - большой адронный коллайдер). По данным компании IDC, занимающейся аналитикой различных рынков, в 2009 году объем созданной, записанной или реплицированной человечеством цифровой информации достиг 0.8 зеттабайт и, согласно прогнозу, к 2020 году увеличится в 44 раза [1]. Хранение и обработка больших объемов данных представляет собой сложную инженерную задачу, которая решается созданием комплексных программно-аппаратных систем хранения данных (СХД).

Перед разработчиками подобных систем встает ряд трудных технических задач. Требуется одновременно обеспечить выполнение ряда требований. Во-первых, современная СХД должна хорошо масштабироваться: должна существовать возможность наращивания числа жестких дисков, объема кэш-памяти, аппаратной модернизации существующей системы, наращивания функционала с помощью дополнительного программного обеспечения (ПО), без значительного переконфигурирования или потерь функциональности. Во-вторых, должна обеспечиваться надежность и отказоустойчивость. Данное требование выполняется полным или частичным резервированием различных компонент системы (блоков питания, путей доступа, процессорных модулей, дисков и т.д.), а также мощной системой мониторинга и оповещения о возможных проблемах. Так, например, для защиты от отказов отдельных дисков используются избыточные массивы независимых дисков (технология RAID), создание полных копий данных внутри дисковой стойки, реплицирование данных на удаленную СХД. В-третьих, должна обеспечиваться высокая производительность, достигающаяся, например, применением более быстрых технологий хранения (например, использованием твердотельных накопителей (SSD), по сравнению с накопителями на жестких магнитных дисках (HDD)). В-четвертых, к промышленным СХД предъявляются жесткие требования по безопасности, что приводит к различным реализациям аппаратного и программного шифрования, разграничению общего дискового пространства и введением различных прав доступа.

Стоит отметить, что вышеперечисленные требования конфликтуют друг с другом. Так, например, внедрение системы шифрования вредит производительности, а репликация на удаленную СХД может существенно понизить уровень безопасности. Кроме того, рынок современных систем хранения далек от монополии – свои решения на нем представляют множество компаний: EMC, HP, IBM, Dell, Hitachi, NetApps, Dothill, Engenio, Adaptec, Raidtec и другие. В условиях интенсивной конкуренции, к вышеперечисленным требованиям добавляется также сопоставимая (с решениями компаний конкурентов) стоимость. Данное требование также вступает в конфликт с другими требованиями. Например, компания монополист могла бы обеспечить беспрецедентную производительность, построив свою СХД целиком на основе дорогостоящих SSD дисков. Но такое решение обладало бы огромной стоимостью на бит хранимой информации и, при наличии конкурирующих компаний производителей, гарантированно было бы быстро вытеснено с рынка.

Одним из широко распространенных в вычислительной технике компромиссов для обеспечения быстрого времени доступа к данным, при малом увеличении стоимости на бит, является технология кэширования. Кэш - промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удаленного источника, однако её объем существенно ограничен по сравнению с хранилищем исходных данных. Кэширование применяется почти повсеместно: жесткими дисками, браузерами, веб-серверами, службами системы доменных имен (DNS - Domain Name System), процессорами и даже системой навигации в современных смартфонах (координаты, полученные со спутника, на некоторое время сохраняются во внутренний кэш, чтобы, если какому-либо приложению понадобится данная информация через малый промежуток времени, можно было не отправлять очередной запрос к спутнику).

Предметом данной работы является повышение производительности систем хранения данных, использующих кэширование. Кэш в современной СХД сам по себе обычно является сложной технической системой, со своей конфигурацией и набором изощренных алгоритмов. Выбор конфигурации для алгоритма кэширования – важная проблема, качественно решив которую, можно существенно повысить производительность всей системы хранения.

Учитывая то, что нагрузка на СХД может сильно отличаться по своим характеристикам, возможным методом выбора наилучшей конфигурации видится адаптация к конкретной нагрузке на конкретной СХД в конкретный период времени. Классическим способом решать вышеописанную задачу является моделирование на потоке адресов. Фактически, данный метод представляет собой моделирование обработки конкретной нагрузки различными алгоритмами кэширования с различными параметрами, по окончании которого выбирается кэш, показавший наивысшую производительность.

Данный метод обладает существенными недостатками. Сбор потока адресов – крайне трудоемкая операция, ведь нужно собрать данные обо всех транзакциях, выполняемых СХД. На время сбора потока адресов время доступа к системе падает, что может быть неприемлемо для промышленной системы, являющейся частью какого-либо бизнес процесса. Также моделирование на таких данных занимает много времени, с учетом того, что обрабатывать данные приходится много раз – каждый раз с разной возможной конфигурацией алгоритма кэширования.

В данной работе изложен новый подход к проблеме выбора конфигурации алгоритма кэширования на СХД, основанный на методах машинного обучения и моделировании на потоке адресов, лишенный вышеперечисленных недостатков. На примере одного параметра конфигурации – размера страницы кэш-памяти, приводится пример сбора обучающей и тестовой выборок, а также построения алгоритма оценки оптимального размера страницы.

Работа организована следующим образом. В главе 1 рассматриваются важные характеристики, которыми описывается нагрузка на СХД, основные принципы организации кэш-памяти; формулируются цели работы; описан классический метод решения данной задачи - моделирование на потоке адресов. В главе 2 рассматриваются теоретические вопросы, затронутые в данной работе: методы машинного обучения для задачи классификации, логистическая регрессия, решающие деревья и методы отбора признаков. В главе 3 рассматривается построенный для решения данной задачи симулятор кэш-памяти, его архитектура и особенности, приводится методика работы с ним, вместе с описанием ПО, автоматизирующим его работу, рассмотрены примеры конкретных файлов с потоками адресов. Глава 4 содержит описание реализаций двух алгоритмов классификации на языке R и реализацию алгоритма отбора признаков, результаты тестирования алгоритмов. В заключении приводится анализ проделанной работы, а также возможные планы развития идеи.

# 1. Терминология и постановка задачи

## 1.1. Характеристики нагрузки

Характеристики нагрузки – один из наиболее важных факторов, влияющих на производительность. Другие факторы включают в себя выбор СХД, файловую систему, пропускную способность сети и др.

Взаимодействие пользователя с системой хранения часто происходит по стандартам SCSI (или аналогичным) – интерфейсу, разработанному для объединения на одной шине различных по своему назначению устройств, таких как жесткие диски, накопители на магнитооптических дисках, стримеры, сканеры и т.д. Система команд SCSI включает общие команды, применимые для устройств всех классов, и специфические для каждого класса. Все команды делятся на три категории:

* обязательные (mandatory);
* дополнительные (optional);
* фирменные (vendor specific).

Может использоваться также взаимодействие по протоколу iSCSI (Internet Small Computer System Interface), который описывает транспортный протокол для SCSI, который работает поверх TCP, а также механизм инкапсуляции SCSI команд в IP сети.

При взаимодействии пользователя с СХД наиболее часто используются следующие SCSI команды (из списков обязательных и дополнительных команд):

* INQUIRY - запрос основных характеристик устройства;
* TEST UNIT READY — проверка готовности устройства;
* READ – чтение;
* WRITE – запись;
* WRITE AND VERIFY – запись и проверка;
* а также некоторые другие.

Для адресации, в рамках взаимодействия по протоколу SCSI, каждое устройство на SCSI-шине имеет как минимум один номер логического устройства (LUN – Logical Unit Number). В некоторых более сложных случаях одно физическое устройство может представляться набором LUN. Для адресации в пределах одного логического устройства используется логический блоковый адрес (LBA - Logical block address). Данные адреса никак не связаны с физическими характеристиками диска (числом цилиндров, головок и секторов): весь диск (логическое устройство) в этой системе адресации считается непрерывным массивом блоков (секторов) одинакового размера, причем их нумерация ведется с нуля. Благодаря этому использовать LBA в программах намного удобнее, чем физические адреса.

Нагрузка, связанная с кэш-памятью создается серией команд для записи и чтения (READ, WRITE). При этом указывается номер логического устройства, логический блоковый адрес и размер данных, которые необходимо записать или прочитать. Далее, под термином нагрузка, в данной работе будет пониматься поток SCSI команд на запись и чтение данных.

Одной из самых важных характеристик нагрузки является количество запросов к СХД за единицу времени (IOPS – input output per second), которые способно обработать система. Может вычисляться либо общее количество запросов, либо количество запросов на чтение и запись по отдельности. Данная характеристика является одним из ключевых параметров при измерении производительности систем хранения. Главным образом, данный показатель помогает определить настройки устройства, при которых оно показывает максимальную производительность. Конкретное значение может варьироваться от системы к системе в зависимости от условий запуска бенчмарка (компьютерная программа, которая служит для сравнения производительности).

IOPS прежде всего характеризует скорость обработки запросов небольшого размера, и наиболее часто применимо для оценки таких задач, как OLTP-база данных и близких к ней. Операции базы данных, как правило, характеризуются обменом относительно небольшими блоками (как правило 4-8KB). Каждая такая операция по считыванию или записи блока является одним IOPS-ом. Таким образом, чем выше показатель производительности в IOPS, тем выше производительность базы данных.

Другим параметром, характеризующим производительность, является скорость передачи данных, так называемый traffic throughput, измеряемый в MB/s, то есть количеством мегабайт, проходящих по интерфейсу ввода-вывода в секунду. Этот параметр характеризует скорость обработки в случае больших блоков. Классическая задача, для которой критична именно скорость передачи данных, это аудио-видео обработка, резервное копирование и др.

Стоит также отметить, что оба данных параметра связаны между собой обратно пропорционально: увеличение IOPS вызывает снижение скорости передачи и наоборот.

Разделяют также запросы к СХД по характеру последовательности адресов. Здесь выделяют последовательный (иначе называемый линейным) и произвольный доступ. Под последовательным доступом понимают операции чтения записи, при которых логические блоковые адреса в цепочке команд следуют последовательно (например, при считывании одного большого файла) в рамках одного логического устройства. При произвольных операциях данные читаются (записываются) случайно из разных областей логического устройства. Разницу между данными типами нагрузки наглядно можно увидеть на рис. 1.1.



Рис. .1.1. Типы нагрузки по характеру последовательности адресов

Другой важной характеристикой нагрузки служит перекос популярности адресов (skew). Перекос популярности показывает на какую долю общего объема памяти приходится определенная доля IOPS. Допустим, известны IOPS по логическим устройствам. Расположив их емкости на оси абсцисс, в порядке убывания приходящей на них нагрузки, можем получить гистограмму, показанную на рис. 1.1.2.

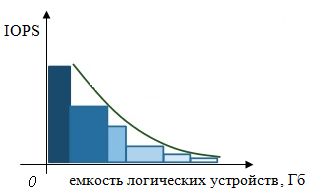


Рис. 1.1.. Гистограмма перекоса популярности адресов

Данная характеристика используется, например, для построения автоматических многоуровневых систем хранения данных (Automated tiered storage). В таких системах, автоматически выделяются области в данных, к которым приходится наибольшее число запросов (“горячие” данные). Такие области автоматически переносятся на более быстрые (но, вместе с тем, более дорогие) устройства хранения внутри одной СХД. Области, запросы к которым происходят реже (“холодные” данные) переносятся на дешевые носители, которые обладают, как правило, большим объемом. В реальных СХД, как правило, большая часть нагрузки работает с малой долей общего объема.

На практике, чаще всего работают с кумулятивной гистограммой (рис. 1.1.3), где по осям откладывают процентные доли нагрузки и емкости. Так, например, на данном графике видно, что на небольшой процент емкости приходится большая доля IOPS. На данном графике можно найти точку, где . Данна точка называется точкой перекоса, например, точка перекоса 80% значит, что 80% всей нагрузки, приходится всего на 20% емкости. При задании перекоса популярности задается либо точка перекоса, либо несколько квантилей, с кумулятивного распределения.

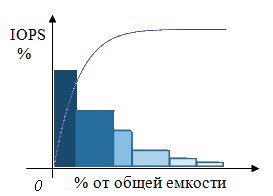


Рис. 1.1.. Кумулятивная гистограмма перекоса популярности адресов

## 1.2. Архитектура систем хранения данных

Как было показано во введении, непрекращающийся рост объемов данных требует от систем хранения эффективного масштабирования для достижения больших емкостей. Данное требование выполняется разработкой грамотной архитектуры, обеспечивающей линейное масштабирование системы.

Рассмотрим, для примера, архитектуру СХД EMC Symmetrix VMAX 20K - массива хранения корпоративного премиум класса (см. рисунок 1.2.1 [13]). Базовой единицей данной архитектуры является плата из двух т.н. директоров (engine).

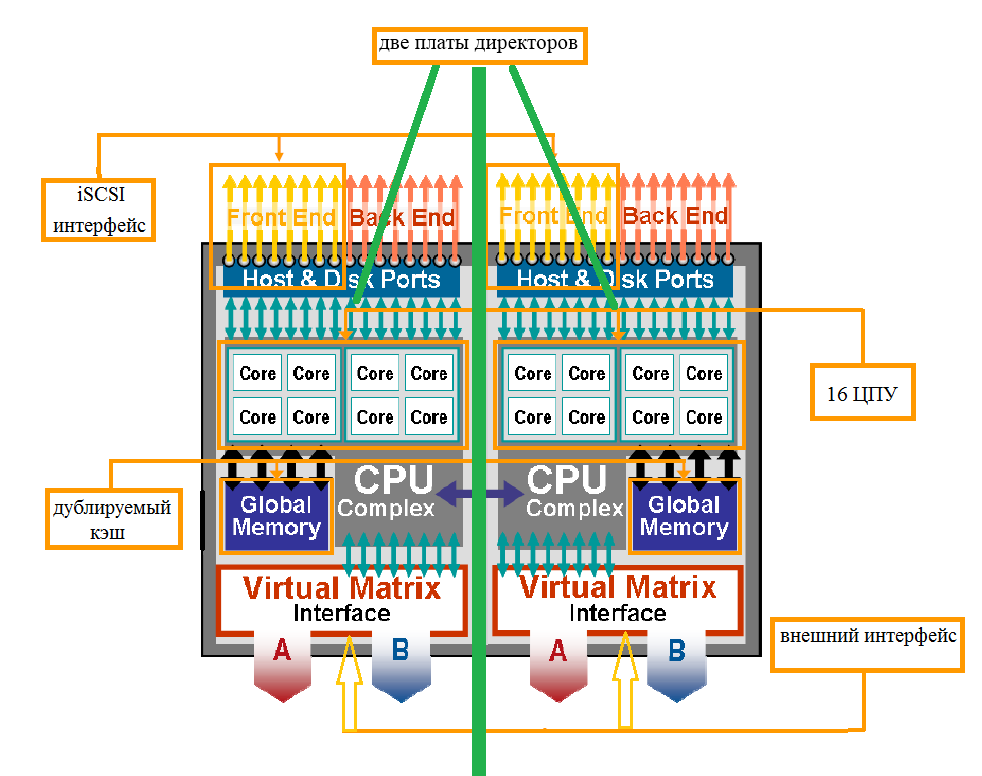


Рис. 1.2.1. Архитектура Symmetrix VMAX 20K

Каждая плата снабжена SCSI-совместимыми интерфейсами, по которым поступают входные запросы (front end) и отправляются запросы к дискам(back end), внешний интерфейс, для объединения плат, общий дублируемый кэш и несколько процессоров, обрабатывающих входные запросы (число зависит от конкретной версии СХД). Кроме того, каждая плата снабжена резервным блоком питания и системой охлаждения. Каждый директор может поставляться с 16, 32 или 64 Гб памяти. Небольшая часть этой памяти отводится под системные нужды, остальная часть - зеркалируется на парном директоре и доступна любому директору внутри системы. Если вся СХД поставлена только с одной платой, память дублируется на парном директоре внутри платы, иначе - может дублироваться между платами.

В рамках одной СХД можно сконфигурировать несколько плат с директорами - см. рис. 1.2.2. Соединения между платами может быть различным. На практике применяется схема соединения на общей шине и схема прямых соединений каждый с каждым (матрица), а также некоторые другие. Детали реализаций данных конструкций выходят за пределы данной работы. Для рассматриваемых далее вопросов, важно понимать из общей архитектуры лишь то, что у системы есть входные-выходные порты, процессоры, обрабатывающие запросы и некоторым образом дублирующийся общий кэш.

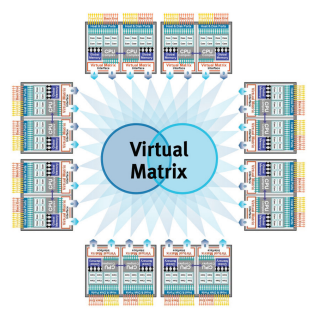


Рис. 1.2.2. Схема объединения плат директоров

## 1.3. Принципы организации кэш-памяти

Любое устройство хранения данных может быть охарактеризовано временем доступа (access time) и стоимостью на бит информации, причем у устройств с меньшим временем отклика стоимость на бит, как правило, больше. Существование различных по скорости и стоимости технологий хранения данных ведет к созданию иерархических систем с кэш-памятью. Кэш-память - сравнительно небольшое по объему, но быстрое хранилище данных, обеспечивающее увеличение производительности за счет принципа локальности ссылок. Принцип локальности ссылок (принцип Деннинга) состоит в том, что большинство реальных программ в течение некоторого отрезка времени обычно работают только с небольшим подмножеством адресов памяти [2. p57] – т.н. рабочим набором (working set). Эффективный кэш должен обладать достаточным объемом, чтобы вместить значительную часть рабочего набора так, чтобы большинство входящих запросов на чтение-запись приходилось на память с меньшим временем доступа. Общее время доступа для системы с кэш-памятью может быть вычислено по формуле:

, (1)

где - число запросов, обработанных кэш-памятью, - число запросов, обработанных основной памятью, и - время доступа к кэш-памяти и основной памяти соответственно, причем выполняется соотношение .

Вопросу проектирования систем кэш-памяти посвящено большое количество работ. Под эффективностью работы кэш-памяти, обычно понимают процент пользовательских запросов, которые обрабатываются кэш-памятью (а значит, быстрее), по сравнению с общим числом запросов:

, (2)

где - общее число запросов к системе. Данный параметр называют процентом попадания (hit ratio).

Известно множество факторов влияющих на эффективность работы кэш-памяти. Их можно разделить на характеристики нагрузки и характеристики непосредственно кэш-памяти. Характеристики нагрузки были рассмотрены в предыдущей главе. Среди характеристик кэш-памяти наиболее важными являются:

* алгоритм вытеснения
* алгоритм предварительной загрузки (prefetch)
* алгоритм обработки последовательных запросов
* общий объем кэш-памяти
* размер страницы (размер слота)

Когда кэш заполнен, задача алгоритма вытеснения выбрать, какую информацию следует удалить, чтобы иметь возможность записать более актуальные данные. Один из наиболее популярных алгоритмов такого типа – алгоритм LRU (Least Recently Used), вытесняет из кэш-памяти данные, которые не использовались дольше всех.

Алгоритм предварительной загрузки должен определить для последовательных запросов, какая группа адресов будет запрошена в следующий момент времени, при условии, что пользователь продолжит отправлять запросы аналогичным образом. Эффективный алгоритм предварительной загрузки, по малому количеству запросов в смежные участки адресного пространства определит последовательный поток запросов, и далее, заранее перенесет в кэш-память данные по некоторому набору следующих адресов. Параметрами данного алгоритма является число последовательных заявок, после которого алгоритм начинает работу, а также объем информации, загружаемый в кэш-память с опережением.

Последовательную нагрузку отличает то, что информация по последовательным адресам используется, как правило, один раз – это, например, процессы резервного копирования в базах данных. Современные СХД используют в том или ином виде отдельные алгоритмы обработки последовательных запросов, чтобы выделить данный тип нагрузки и обрабатывать ее отдельной областью кэш-памяти по отличным правилам вытеснения. Без данной модификации – последовательный поток вытеснит другие данные из кэш-памяти, при том, что по свойству данной нагрузки, повторное обращение к этим данным в короткий промежуток времени маловероятно. Параметрами данного алгоритма является объем памяти, который позволяется занимать в кэш-памяти данным, считанным последовательно.

Ясно, что кэш-память большего размера, как правило, более эффективна. Для того, чтобы количественно оценить насколько, часто проводится подсчет распределения стековых расстояний [3].

Стековым расстоянием нового запроса к некоторому адресу назовем позицию этого адреса в LRU стеке. Или, что то же самое, количество различных адресов в потоке после последнего обращения к этому адресу. В случае отсутствия в стеке, то есть первого обращения, стековое расстояние не определено. Например, рассмотрим поток адресов:

A B C D F C B A

Для последнего адреса А стековое расстояние будет равно 4, так как после предыдущего к нему обращения в потоке было 4 уникальных адреса: B, C, D и F.

В LRU кэше из N страниц, все ссылки со стековым расстоянием будут обработаны кэш-памятью (hit).

Данные в кэш-памяти обычно организованы в виде страниц фиксированного размера, так что все операции между кэш-памятью и основной памятью выполняются над объемом кратным целому числу страниц. Известно, что даже при одинаковом объеме кэш-памяти и при том же алгоритме вытеснения количество попаданий в кэш существенно зависит от выбранного размера страницы [2, p86]. На рисунке 1.3.1, взятом из [4], можно наблюдать данный эффект. По оси Y отмечен процент промахов (miss rate) - доля запросов, обрабатывающихся основной памятью. Уменьшение этого количества в первой половине объясняется принципом локальности ссылок, а увеличение во второй – увеличением «штрафа» за промах. То есть, такой ситуации, когда мы загрузили большую страницу памяти в кэш, но обращений к данному участку памяти больше не последовало.

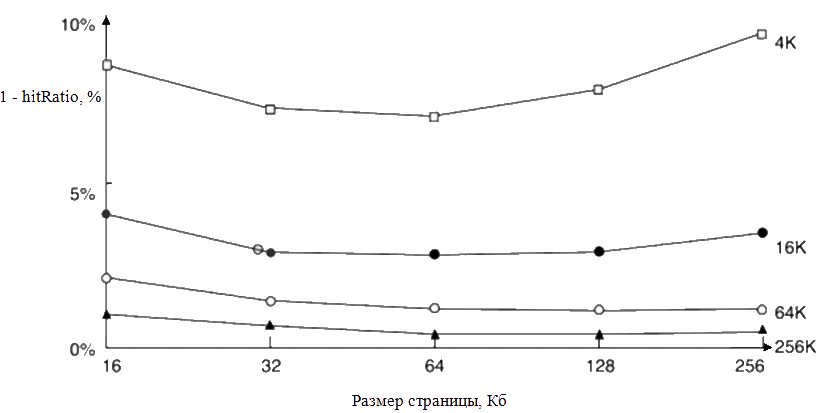


Рис. 1.3.1. Зависимость процента промахов от размера страницы

Для некоторых систем хранения данных размер страницы жестко определен производителем, а для других систем он может быть выбран администратором. В последнем случае наличие автоматического алгоритма выбора оптимального размера страницы кэш-памяти для конкретного типа рабочей нагрузки позволит улучшить производительность системы хранения.

## 1.4. Моделирование на потоке адресов

Как было показано в разделе 1.3, современная реализация кэш-памяти содержит множество параметров, которые можно варьировать, достигая различный процент попадания для конкретной нагрузки. Классическим методом решения задачи выбора системы кэш памяти и ее параметров является моделирование на потоке адресов[5,6]. Собирается полный поток запросов, представляющий собой реальную рабочую нагрузку, приложенную к СХД, в следующем формате:

* временная метка о том, когда пришел запрос
* тип запроса (чтение, запись)
* начальный адрес (имя логического устройства (LUN) + адрес внутри логического устройства)
* размер запрошенных данных

Можно пропустить этот поток адресов через ряд моделей различных систем кэширования с различными параметрами и выбрать тот вариант, который позволяет достичь наибольшего процента попадания.

Данный подход имеет ряд значительных недостатков. Сбор полного потока адресов является очень трудоемкой задачей; при этом время доступа к данным увеличивается, что может быть неприемлемо для работающей коммерческой системы. Объем таких данных, описывающий нагрузку за несколько минут, может достигать нескольких гигабайт. Моделирование на таком объеме данных может занимать большое время, при условии, что необходимо обрабатывать поток адресов несколько раз – меняя размер страницы кэш памяти в модели.

Кроме потока адресов, современные СХД, как правило, позволяют сравнительно легко получить набор агрегированных характеристик нагрузки в терминах, описанных в разделе 1.1. Эта информация дает представление о характере нагрузки, но является недостаточной для проведения моделирования и подсчета процента попадания, так как агрегированные характеристики не несут информацию о порядке прихода запросов.

Целью данной работы является разработка метода оценивания оптимального размера страницы кэша в системе хранения данных на основе агрегированных параметров нагрузки и результатов моделирования поведения модуля кеширования. В главе 4 будет приведено построение нескольких алгоритмов грубой оценки оптимального размера страницы, при этом, все множество возможных значений будет разбиваться на две группы - и больше , где .

Подобного рода алгоритм, может быть в дальнейшем реализован в виде модуля к СХД, который в автоматическом режиме получает набор нужных агрегированных характеристик нагрузки за определенный промежуток времени. Далее, алгоритм пытается классифицировать данную нагрузку к одному из определенных на этапе моделирования типов. Если нагрузка отнесена к типу «больше » - дается указание СХД увеличить размер страницы кэш-памяти. Иным вариантом может быть команда увеличить размер предварительной загрузки в кэш. Алгоритмы, приведенные в главе 4, имеют точность порядка 80% на тестовых данных. Это значит, что в 80% случаях срабатывания алгоритма возможно достигнуть повышения процента попадания.

Нагрузка на СХД может меняться в зависимости от времени суток и множества других факторов. Именно поэтому актуальна задача определения нагрузки, на которой можно достичь более высокий процент попадания, без непосредственного моделирования - чтобы «успевать» за изменениями нагрузки (online - алгоритм). Динамический метод адаптации к нагрузке, таким образом, имеет преимущество перед статической установкой данного параметра (размера страницы).

# 2. Машинное обучение

## 2.1. Задача классификации

Машинное обучение - обширный раздел науки на стыке прикладной статистики, численных методов оптимизации и дискретного анализа, изучающий методы построения моделей, способных обучаться. Алгоритмы машинного обучения получили применение в самых различных отраслях, среди которых: распознавание речи, распознавание текста, задачи технической диагностики, биоинформатика, задачи обнаружения спама, кредитный скоринг, хемоинформатика и др.

Задачу оценки размера страницы по результатам моделирования можно отнести к обширному классу задач, называемых обучение по прецедентам. Сформулируем общую постановку такой задачи. Пусть дано конечное множество прецедентов (объектов), с каждым из которых связана (измерена, собрана) какая-либо информация. Совокупность всех имеющихся описаний прецедентов называется обучающей выборкой. Один из вариантов описания прецедентов - признаковое описание. Введем признаковое описание более формально [7]. Пусть есть наблюдений . Для каждого называют набором признаков (факторов), то есть множеством измеренных характеристик данного наблюдения. Пусть есть также множество соответствующих откликов . Совокупность этих множеств называют обучающей выборкой. Требуется найти функциональное описание , которое бы выражало зависимость откликов от описаний объектов (факторов), т.е. для «новых» объектов, не включенных в обучающую выборку, предсказывало бы отклики. Получение конкретной функции , по обучающей выборке называется обучением.

Если отклики определены на всей вещественной оси, говорят о задаче регрессии. Если множество откликов конечно - говорят о задаче классификации. В нашем случае, множество размеров страниц, которые допускается установить на СХД конечно, поэтому поставленная задача - задача классификации.

## 2.2. Логистическая регрессия

Одним из наиболее популярных методов решения задачи классификации является логистическая регрессия. Эта модель относится к классу линейных классификаторов - алгоритмов классификации, основанных на построении линейной разделяющей поверхности. В случае двух классов разделяющей поверхностью является гиперплоскость, которая делит пространство признаков на два полупространства.

Пусть объекты описываются числовыми признаками:

.

Тогда пространство описания в признаках есть . Пусть **Y** - конечное множество номеров (имен, меток) классов. Пусть задана обучающая выборка пар фактор - отклик:

где - вектор значений признаков, для конкретного наблюдения. Будем использовать бинарную схему для кодирования откликов: , и также, для возможности векторизации вычислений, введем нулевой признак: .

В логистической регрессии моделируется вероятность принадлежности наблюдения конкретному классу. Для этого нужно использовать функцию, множество значений которой лежит в промежутке . В логистической регрессии для этих целей используется функция сигмоид (или logit):

, (3)

где - вектор весов признаков, - порог принятия решения, - скалярное произведение признакового описания объекта на вектор весов. График данной функции всегда имеет S-образную форму, так что, независимо от значения x, значение функции будет лежать в промежутке . Выражение (3) можно преобразовать до следующего вида:

Соотношение называют отношением шансов. Взяв логарифм от обеих частей, получим в левой части логарифм отношения шансов:

При логистической регрессии в качестве моделируемых значений зависимой переменной используется логарифм отношения шансов. Регрессионный коэффициент (вес) - это изменение логарифма отношения шансов данной зависимой переменной на единицу изменения независимой переменной при постоянных значениях всех остальных независимых переменных.

Для того чтобы определить вектор весов признаков, воспользуемся методом максимума правдоподобия. Идея метода состоит в следующем: будем искать такие коэффициенты , что предсказанная вероятность принадлежности к конкретному классу , вычисленная согласно (3), была бы наиболее близка к наблюдаемому значению. Другими словами, будем искать такой вектор весов , что если подставить его в модель для , заданную в (3), получим значения близкие к 0, для тех наблюдений, отклик которых кодируется значением 0, и значения близкие к 1, для тех наблюдений, отклик которых кодируется значением 1. Это можно выразить более формально, введя функцию правдоподобия:

Вектор весов w следует выбирать таким, чтобы максимизировать данную функцию правдоподобия. Следует заметить, что метод максимального правдоподобия используется для подгонки различных моделей, не только логистической регрессии.

После подгонки вектора весов, алгоритм классификации может быть определен следующим образом:

Апостериорные вероятности , помогают оценивать риски, связанные с возможными ошибками классификации.

## 2.3. Отбор признаков

Часто во многих задачах машинного обучения возникает вопрос - какие признаки использовать, а какие нет. Проблема отбора признаков [8] (feature selection) часто возникает из-за того, что на этапах постановки задачи и формирования данных еще не ясно, какие признаки бесполезны или дублируют друг друга. Естественное стремление учесть как можно больше потенциально полезной информации приводит к появлению избыточных (шумовых) признаков. Если признак на самом деле не информативен, то есть не влияет на отклики, его включение в модель может только ухудшить ее качество. Методы обучения должны отличать шумовые признаки и отбрасывать их.

Отбор признаков позволяет установить, какие признаки вносят наибольший вклад в модель, т.е. найти среди множества всех признаков информативные. По мере увеличения числа используемых признаков (сложность модели) средняя ошибка на обучении обычно убывает (более сложная модель может лучше описывать обучающую выборку). При этом ошибка на тестовом наборе сначала уменьшается, проходя через точку минимума, а затем только увеличивается (рис 2.3.1) [7, p 8]. Данное явление называют переобучением. Чрезмерно сложные модели не столько восстанавливают искомую зависимость, сколько аппроксимируют ошибки измерений и погрешности самой модели.

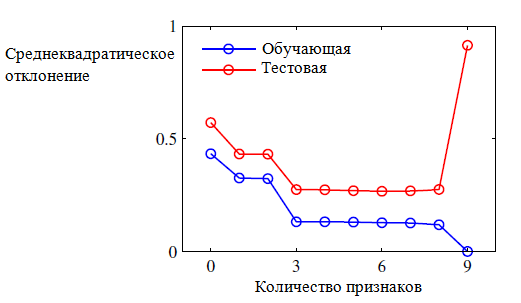


Рис. 2.3.1. Эффект переобучения

Сложность задачи отбора признаков - в ее переборном характере. Если число признаков равно n, то число непустых подмножеств составляет . Прямой перебор всех подмножеств часто оказывается невозможным на практике, поэтому прибегают к различным методам сокращения перебора.

Прежде чем говорить о способах выбора признаков, необходимо сформулировать критерии выбора. Все критерии можно разделить на две большие группы: внутренние и внешние. Внутренний критерий - это функционал, характеризующий качество метода по обучающей выборке. Внутренние критерии нельзя использовать в качестве единственных критериев для выбора модели, так как при этом будет поощряться переобучение. Внешний критерий характеризует качество метода по тем данным, которые не использовались в процессе обучения. Внешний критерий проверяет работу алгоритма в реальных условиях.

Введем функционал средней ошибки алгоритма на выборке :

где функция потерь характеризует величину ошибки алгоритма на объекте . Обычно, для задач классификации:

Тогда примером внешнего критерия может служить функционал средней ошибки на заданной контрольной выборке, называемой ошибкой обобщения (generalization error). Другие возможные критерии: критерий перекрестной выборки (или скользящего контроля), контроль по случайным подвыборкам, бутстрэп и т.п.

При оценке модели по критерию перекрестной выборки, имеющиеся в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Все данные методы страдают от проблемы представительности подвыборок - обобщающие и контрольные подвыборки должны обладать теми же статистическими характеристиками, что и полная выборка. В противном случае выбор модели и настройка ее параметров будут плохо согласованы друг с другом. В задачах классификации рекомендуется сохранять в каждой подвыборке пропорции распределения объектов по классам. Этот прием называется стратификацией.

Для оценки качества предсказаний модели на тестовой выборке при известном качестве на обучающей выборке, при условии, что модель линейна, и мы обучали ее по принципу максимума правдоподобия, применяется информационный критерий Акаике (Akaike's information criterion, AIC). Критерий Акаике является оценкой матожидания средней ошибки на контрольных данных. В общем случае AIC задается как:

где k - число параметров в модели, и - максимизированное значение функции правдоподобия модели. В таком виде можно заметить, что критерий Акаике - сумма внутреннего критерия и штрафного слагаемого, наказывающего чрезмерно сложные модели.

Когда общее число признаков не слишком велико, обучать алгоритм по всем, или почти всем признакам не составляет особой проблемы. В таком случае для отбора признаков применяют стратегию последовательного исключения признаков. Итерации начинаются с полного множества признаков, и далее из набора последовательно исключается по одному признаку так, чтобы значение критерия убывало как можно быстрее.

Также на практике применяется метод последовательного добавления признаков, а также их комбинация - поочередное добавление и удаление признаков.

## 2.4. Решающие деревья

Для решения задачи классификации также часто используют решающие деревья [7, p 678], потому что они дают возможность наглядно интерпретировать результаты. Данные модели широко распространены в медицинской диагностике.

Бинарное решающее дерево для задачи классификации задается бинарным деревом (см. рис. 2.4.1):

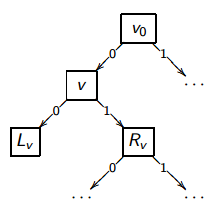


Рис. 2.4.1. Бинарное решающее дерево

Тогда алгоритм классификации запишется в следующем виде:

* Начать с корня (
* Пока текущая вершина не лист:
  + если предикат равен 1 для данного наблюдения :
    - переходим в правое поддерево
  + иначе переходим в левое поддерево
* вернуть класс, соответствующий листу дерева.

Проблема построения оптимального дерева решения в общем случае NP-полная, поэтому существуют различные жадные алгоритмы построения деревьев решения, например см. [9].

# 3. Симулятор кэш-памяти

## 3.1. Общая схема обработки потока адресов

Для обработки потоков адресов был создан симулятор кэш-памяти на основе LRU, который также осуществляет подсчет агрегированных характеристик рабочей нагрузки в терминах раздела 1.1. Решение создавать новый симулятор кэш-памяти, вместо того, чтобы воспользоваться одним из существующих, было принято по нескольким причинам:

* это позволяет реализовать конкретные особенности работы кэш-памяти, реализованные в конкретном семействе СХД
* это позволяет по ходу моделирования вести подсчет характеристик рабочей нагрузки
* это позволяет вести подсчет характеристик рабочей нагрузки в тех терминах, которые используются в конкретном семействе СХД

Описание конкретного формата задания потока адресов приведено ниже. В приложении 1 приведен пример фрагмента такого файла. Первое число - временная метка пришедшего запроса. Тип запроса задается во второй колонке (Read либо Write). Далее идет имя директора (см. раздел 1.2) - данный параметр связан с параллельной обработкой заявок внутри конкретной СХД, с которой был снят данный поток адресов, и не влияет на работу общей кэш-памяти - его мы будем игнорировать. Следующие колонки - имя логического устройства, логический блоковый адрес и размер запроса соответственно. Размеры в данном файле приведены в блоках - величина, равная 512 байтам. Для данного файла, можем заметить, что большинство запросов имеют размер 8 блоков, или, что тоже самое 4 Кб с небольшим числом запросов по 128 блоков. Нагрузка в основном носит произвольный характер (нет большого количества последовательностей из запросов в смежные адреса в рамках одного и того же логического устройства).

Общая схема обработки потоков адресов для сбора обучающей выборки приведена на рис.3.1.1.

Для того чтобы определить объем кэш-памяти для данных экспериментов, была реализована утилита, вычисляющая объем рабочего набора конкретного потока адресов при конкретном размере страницы. Данная программа читает поток адресов, строчку за строчкой. Для каждого запроса вычисляется множество страниц памяти, которые он затрагивает:

* запросы размером меньше, чем размер страницы занимают ровно одну целую страницу кэш-памяти,
* запросы размером больше, чем размер страницы, занимают несколько целых страниц кэш-памяти.

Все множество страниц, которое было посчитано к концу файла и будет рабочим набором. Это эквивалентно моделированию на данном потоке адресов с бесконечным кэшем.



Рис. 3.1.. Общая схема обработки потока адресов

Объем кэш-памяти берется порядка 0.1% - 0.001% от объема рабочего набора. Для описанной далее обучающей выборки, файлы потоков адресов разбивались на части размером от 60 до 120 MB, рабочие наборы были от 10 до 50GB, и кэш размером в 25 MB. Если выбрать кэш слишком маленького размера, по сравнению с рабочим набором, никакой алгоритм вытеснения не обеспечит приемлемого процента попадания, и эффективность не будет зависеть от локальности ссылок. Например, кэш 1 MB при размере страницы в 1024K будет содержать всего 1 страницу. Понятно, что моделирование работы в таких условиях не представляет никакого практического интереса. Если выбрать кэш слишком большого размера, весь рабочий набор может поместиться в нем, и размер страницы тоже никак не будет влиять.

Процессом запуска симулятора занимается скрипт, написанный на языке Python. Он принимает на вход множество файлов с потоками адресов. Для каждого конкретного файла из этого множества, в цикле по всем возможным размерам страницы запускается симулятор с заданными параметрами. Каждый запуск симулятора создает на выходе xml файл, в котором содержится следующая информация (фрагмент данного выходного файла приведен в приложении 2):

* полное распределение размеров заявок

<Size sizeBlocks="8" probability="0.357338" />

* точки 0.05, 0.15 и 0.5 с кумулятивной гистограммы перекоса адресов

<Point capacity="0.05" workPercentage="0.9881" />

* точки 0.05, 0.15 и 0.5 с кумулятивной гистограммы стековых адресов, отдельно, для запросов на чтение и запись

<Point sizeKB="1280" hitPercentage="0.858533" />

* число последовательных запросов на чтение / запись
* число случайных запросов на чтение / запись, которые попали в кэш
* число случайных запросов на чтение / запись, которые не попали в кэш
* общее число запросов на чтение / запись
* оценка размера логических устройств

<Lun name="1d1" sizeGB="90.7031" />

Таким образом мы можем получить, для каждого входного файла с потоком адресов, каталог, содержащий несколько xml файлов с характеристиками нагрузки. Далее запускаются скрипты, которые для множества xml файлов создают csv файл с обучающей выборкой. При этом происходит небольшая модификация данных, для удобства последующего обучения алгоритма на них. Например, агрегируется распределение размеров запросов, чтобы обучающая выборка содержала конечное число признаков. Размеры агрегируются в интервалы (0, 16], (16,32], (32,64], (128, 256], (256, 320], (320, 512] и интервал “больше 512 блоков”.

Далее файл с обучающей выборкой загружается в среду R, где выполняется анализ данных. R - язык программирования для статистической обработки данных и работы с графикой, а также свободная программная среда вычислений с открытым исходным кодом в рамках проекта GNU. R широко используется как статистическое программное обеспечение для анализа данных и фактически стал стандартом для статистических программ.

## 3.2. Архитектура симулятора

При проектировании архитектуры симулятора ставилась цель сделать модель наиболее гибкой. Это позволило бы в будущем использовать данное ПО для моделирования большого спектра систем кэш-памяти с разными параметрами.

При запуске программы происходит обработка параметров командной строки (рис. 3.2.1). Симулятор поддерживает возможность задания размера кэш-памяти, как в виде абсолютного значения, так и в процентах от оценки размера группы хранения. Группа хранения - весь набор LUN, которые встретились в потоке адресов. Класс StorageGroupSizeComputer предоставляет единственный метод preprocess(), реализующий оценку размера группы хранения. За оценку размера группы хранения берется сумма оценок размеров логических устройств. Для каждого логического устройства, его размер оценивается как максимальный адрес, запрошенный в данном потоке адресов у данного логического устройства. Также, данный класс выполняет подсчет числа строк в файле потока адресов, для отображения индикатора процесса. Это позволяет отслеживать степень обработки программой конкретного потока адресов.

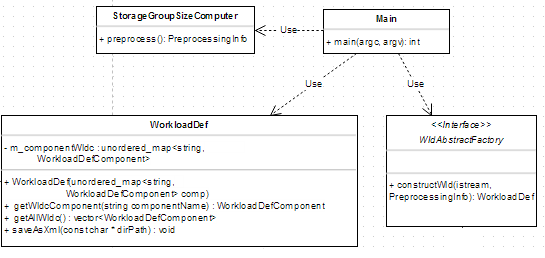


Рис. 3.2.1. Архитектура симулятора. Блок управления

Центральным классом программы является класс с описанием характеристик нагрузки - WorkloadDef. Он может содержать одну и более компоненту нагрузки - по умолчанию вся нагрузка определяется в одну компоненту. В классе предусмотрены методы доступа к определенным компонентам, а также метод для сериализации описания нагрузки в выходной xml-файл.

Созданием конкретного описания нагрузки занимается класс CdbWldFactory, реализующий интерфейс WldAbstractFactory (рис. 3.2.2).

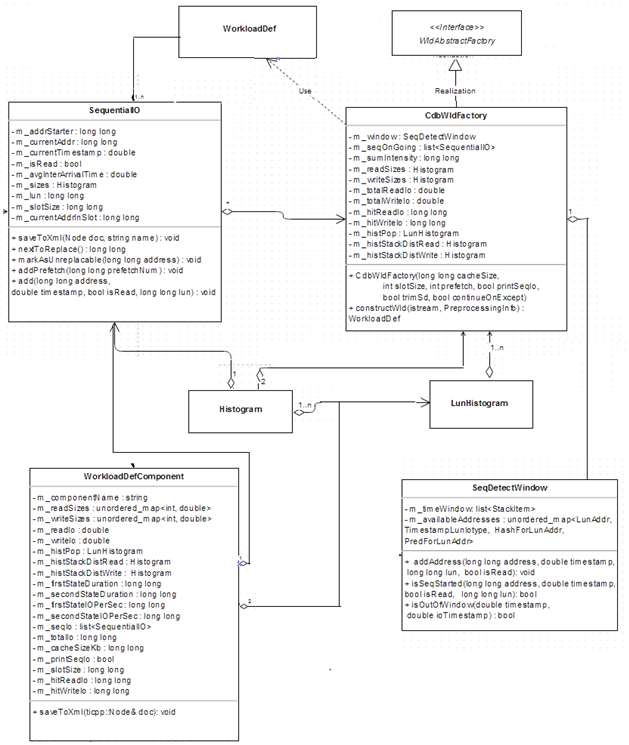


Рис. 3.2.2. Архитектура симулятора. Основной блок

Для оценки того, что будет, если изменить размер кэша, в симуляторе проводится подсчет распределения стековых расстояний по алгоритму, приведенному в [10]. Для возможности отдельного подсчета числа попаданий в кэш для запросов типа чтение и запись строятся две отдельные гистограммы распределения стековых расстояний.

Для обработки потоков последовательных адресов используются классы SequentialIO и SeqDetectWindow. Класс SeqDetectWindow задает конкретное определение последовательного потока адресов, а именно:

* максимальный интервал времени между приходом двух заявок одного потока -
* число смежных адресов в кэше, которые нужно задействовать, чтобы начать поток -
* количество страниц памяти, которое может занимать один последовательный поток -

-ый последовательный запрос вытолкнет из LRU стека страницу, содержащую данные для первого запроса данного потока. Таким образом, минимизируется влияние последовательных потоков на работу кэш-памяти. Первые два запроса из последовательного потока считаются промахами (miss), остальные рассматриваются как попадания в кэш (hit to prefetch). Для этого поддерживается список текущих потоков (CdbWldFactory::m\_seqOnGoing) со следующей информацией (SequentialIO):

* распределение размеров заявок, в рамках данного потока
* адрес логического устройства
* тип запросов в рамках данного потока
* среднее время между приходом запросов
* указатель на страницу, которую нужно будет вытеснить следующей, в рамках данного потока

Если данный запрос не принадлежит ни одному потоку, проверяется, начинает ли данный запрос новый поток. Для этого поддерживает ассоциативный массив, который содержит адреса за последние секунд (SeqDetectWindow::m\_timeWindow). В качестве ключа выступает адрес заявки, LUN и тип операции. Условием начала потока является наличие предыдущих запросов такого же типа к одному логическому устройству в течение последних секунд. В случае удовлетворения условия (SeqDetectWindow::isSeqStarted), создается поток - иначе запрос считается случайным.

Для работы со случайными запросами используется обычный LRU-стек. При приходе нового случайного запроса в стеке отыскивается страница с таким же адресом и перемещается на вершину стека и его стековое расстояние учитывается в распределении. В случае отсутствия адреса в кэше запрос кладется на вершину стека.

Для работы с дискретными распределениями реализован класс гистограмма (Histogram) - рисунок 3.2.3.

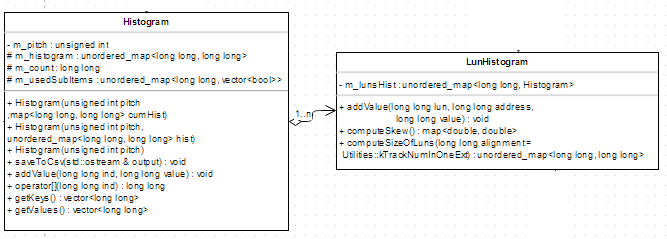


Рис. 3.2.3. Классы для работы с дискретными распределениями

Данный класс внутри себя использует стандартный контейнер - unordered\_map - классической реализацией которого, обычно, является хэш-таблица. В случае распределения размеров запросов ключом является размер запроса, а хранимым значением - вероятность запросов данного размера в данном потоке адресов. Распределение размеров считается отдельно для запросов на чтение и запись. Другой экземпляр данного класса используется для хранения распределения стековых адресов и т.д.

Для подсчета кумулятивной гистограммы перекоса служит класс LunHistogram, содержащий внутри себя гистограмму перекоса для всех LUN.

## 3.3. Интерфейс симулятора

Как уже было показано в разделе 3.2. симулятор представляет собой гибкую модель, с широкими возможностями настройки. Для обработки параметров командной строки используется модуль program\_options из библиотеки boost. Рассмотрим ключи запуска программы:

* “help,h” - выводит инструкцию пользователя,
* "input,I" - задает путь до входного потока адресов,
* "output,O" - задает путь для выходного xml файла,
* "printSd" - путь для csv файла с гистограммами распределения стековых расстояний,
* "slotSize" - размер страницы кэш-памяти в блоках (512 байт),
* "cacheSize" - общий размер кэш-памяти в килобайтах,
* "cacheSizeInPercent" - если задан - значение размера кэш-памяти трактуется как проценты от объема группы хранения, если нет - в килобайтах,
* "printSeqIo" - выводить дополнительную информацию о каждом потоке последовательных адресов
* "maxCacheSizeInMB" - если ключ "cacheSizeInPercent" задан, размер кэш-памяти задается как
* "prefetch" - число страниц памяти, для предварительной загрузки.

# 4. Анализ данных

## 4.1. Реализация алгоритмов классификации

Прежде всего, на данном этапе можно отметить, что график зависимости процента попадания от размера страницы, в большинстве случаев, имеет вид кривой, похожий на результат, описанный в [2,4]. Пример такой зависимости приведен на рисунке 3.2.1. Это экспериментально подтверждает факт влияния размера страницы на эффективность работы кэш-памяти с данными характеристиками.

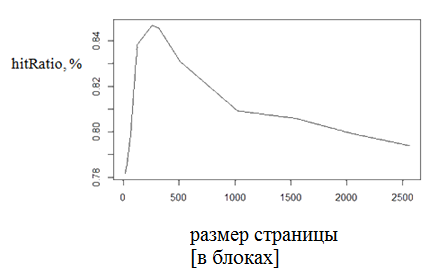


Рис. 4.1.1. Зависимость процента попадания от размера страницы

На данном этапе все множество возможных размеров страниц было разделено на две группы - и больше , где . В качестве входных данных использовалась выборка из m = 239 наблюдений, размерность вектора признаков используемых для предсказаний равнялся n = 11. В качестве признаков, которые не зависят от размера страницы и допустимы для предсказаний, были использованы точки 0.05, 0.15 и 0.5 с кумулятивной гистограммы перекоса популярности адресов и процентное соотношение размеров запросов в интервалах [0, 16], (16,32], (32,64], (128, 256], (256, 320], (320, 512] и заявки большие 512 блоков. Распределения размеров запросов на чтение и запись агрегировались. В представленных данных отсутствуют пропуски.

Для обучения моделей использовалась случайная выборка размера от общего числа наблюдей. Остальные данные были использованы в качестве тестовой выборки. В каждой подвыборке пропорции распределения объектов по классам сохранялись равными.

Для реализации данных моделей была использована встроенная в R функция glm [11]. Основной формат вызова функции таков:

.

Для логистической регрессии будем использовать биномиальное семейство распределений с функцией связи по умолчанию сигмоид.

Функция glm() позволяет также подгонять различные другие распространенные модели, включая пуассоновскую регрессию и модели для анализа выживания. описывает вид модели, которую необходимо подобрать, а data это фрейм с данными обучающей выборки. Функция возвращает объект, содержащий список параметров w а также различные данные о подобранной модели. Для записи формул в R применяется следующий синтаксис:

,

где ~ отделяет зависимую переменную (слева) от независимых (справа). Знак “+” используется для перечисления независимых переменных. Будем использовать запись вида , что означает использовать для обучения все признаки:

После получения моделей был произведен отбор методом последовательного удаления признаков из модели, чтобы оценить информативный набор признаков. Для выбора признаков на основе критерия AIC, использовалась функция step от объекта, который вернула функция glm: . Процесс останавливается, как только AIC начинает расти. Данный метод не гарантирует, в отличие от полного перебора, нахождения глобального минимума критерия.

На той же обучающей выборке, для каждого значения t, построим двоичное решающее дерево. Конкретный вид использованной модели обучения описан в [12]. Для построения использовался пакет rpart языка R. Для построения дерева решений для задачи классификации использовалась функция пакета rpart:

где задает, что нужно построить дерево классификации (а не регрессии), data - дата фрейм с обучающей выборкой.

Для обрезки деревьев использовалась перекрестная проверка (cross validation):

.

Для оценки работы алгоритма на тестовом множестве использовалась функция predict:

где test - дата фрейм с тестовой выборкой, а означает вернуть предсказания в виде вероятностей, а не просто в виде предсказанного класса.

В связи со слабым перекосом (менее 50%) в данных, полученные модели оценивались по значениям точности на тестовом наборе данных (1/3 доля всех данных): вычислялось соотношение количества верно предсказанных наблюдений от их общего числа.

Точность моделей и примеры решающих деревьев анализируются в следующем разделе (см. раздел 4.2).

## 4.2. Оценка точности алгоритмов

Рассмотрим полученные модели для двух классов и больше . Первый класс будем кодировать 1, класс «больше » будем кодировать 0. Данные о точности моделей логистической регрессии приведены в таблице 4.2.1.

Таблица 4.2.1. Точность моделей

|  |  |  |  |
| --- | --- | --- | --- |
| t, Кб | Точность, % (без отбора признаков) | Точность, % (с отбором признаков) | Размер вектора признаков после отбора |
| 512 | 0.7468354 | 0.7721519 | 9 |
| 768 | 0.7341772 | 0.7341772 | 9 |
| 1024 | 0.7721519 | 0.8101266 | 7 |

Для вывода информации о коэффициентах модели логистической регрессии будем пользоваться функцией summary. Рассмотрим модель для , после процедуры отбора признаков (таблица 4.2.2).

Таблица 4.2.2. Коэффициенты модели t=1024K

|  |  |  |
| --- | --- | --- |
| Название признака | Значение | Pr(>|z|) |
| skew 0.15 | 15.448 | 0.00569 |
| skew 0.5 | -510.323 | 0.20579 |
| size16 | -6.981 | 0.00685 |
| size32 | -8.055 | 0.00436 |
| size64 | -5.721 | 0.04964 |
| size128 | -8.537 | 0.01188 |
| size320 | -106.078 | 0.00281 |

Колонка Pr(>|z|) представляет собой значения статистической ошибки первого рода для коэффициентов логистической регрессии. Как видно из таблицы, все признаки, кроме точки 0.5 с кумулятивной гистограммы перекоса популярности, значимые - можем отклонить гипотезу о том, что эти параметры равны 0.

Подобные данные приведены также для модели t=512K и t=768K в таблицах 4.2.3 и 4.2.4 соответственно. У данных моделей, статистически не значимым параметром также оказалась точка 0.5 перекоса популярности. Данный признак означает долю входных запросов, которые относятся к половине всего объема группы хранения. Выборочное среднее данного признака, в данной выборке составляет 0.9991.

Таблица 4.2.3. Коэффициенты модели t=512K

|  |  |  |
| --- | --- | --- |
| Название признака | Значение | Pr(>|z|) |
| skew 0.15 | 9.239 | 0.05945 |
| skew 0.5 | -470.406 | 0.26535 |
| size16 | 158.795 | 0.00310 |
| size32 | 158.276 | 0.00316 |
| size64 | 160.115 | 0.00288 |
| size128 | 158.057 | 0.00308 |
| size256 | 163.893 | 0.00231 |
| size512 | 175.037 | 0.00296 |
| size\_rest | 173.533 | 0.00174 |

Таблица 4.2.3. Коэффициенты модели t=768K

|  |  |  |
| --- | --- | --- |
| Название признака | Значение | Pr(>|z|) |
| skew 0.15 | 15.079 | 0.00514 |
| skew 0.5 | -504.786 | 0.19127 |
| size16 | 144.019 | 0.00273 |
| size32 | 142.459 | 0.00298 |
| size64 | 143.395 | 0.00283 |
| size128 | 142.186 | 0.00287 |
| size256 | 149.895 | 0.00190 |
| size512 | 158.969 | 0.00292 |
| size\_rest | 149.512 | 0.00255 |

Точность решающих деревьев приведена в таблице 4.2.4.

Таблица 4.2.4. Точность решающих деревьев

|  |  |  |
| --- | --- | --- |
| t, Кб | Точность, % | Число вершин |
| 512 | 0.7341772 | 6 |
| 768 | 0.7341772 | 6 |
| 1024 | 0.7974684 | 5 |

Сами решающие деревья приведены на рисунках 4.2.1, 4.2.2 и 4.2.3 соответственно.

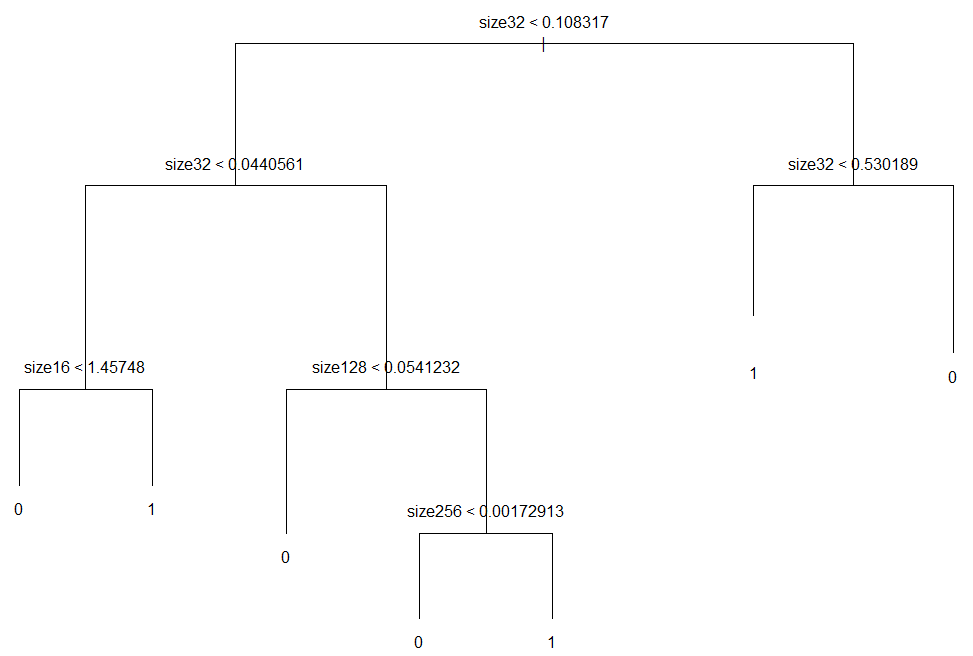


Рис. 4.2.1. Решающее дерево для задачи t=512K

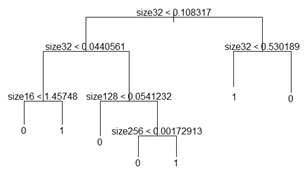


Рис. 4.2.2. Решающее дерево для задачи t=768K



Рис. 4.2.3. Решающее дерево для задачи t=1024K

На данных решающих деревьях, правое поддерево выбирается, если условие в вершине не выполнено. Как можно заметить из сравнения таблиц точность логистической регрессии для данных задач либо такая же, либо выше, но деревья обладают большей возможностью интерпретации. Отбор признаков либо никак не повлиял на точность, либо повысил ее.

На данном этапе также была сделана попытка построить полное дерево решений для предсказания оптимального размера страницы, но результаты пока оставляют желать лучшего - точность составляет не более 60% на отложенных данных. Предполагаемые причины такой низкой точности видятся в следующем:

* малый размер обучающей выборки
* недостаточный набор информативных признаков.

Для многих методов обучения существует рекомендация, что число наблюдений должно быть не меньше квадрата от количество признаков. В данном случае, чтобы соответствовать этой рекомендации, для обучения, распределение размеров запросов на запись было объединено с распределением размеров запросов на чтение. Также в представленном в главе 3 симуляторе, на данный момент, ведется подсчет процента последовательных запросов в терминах страниц кэш-памяти - последовательной нагрузкой считаются запросы, затрагивающие смежные страницы кэш-памяти. При этом не считается процент запросов, в которых логические адреса в действительности идут «стык в стык», хотя данный параметр был бы независим от размера страницы, а значит, пригоден для использования в алгоритме оценки.

На рисунке 4.2.4. показан другой пример графика зависимости процента попадания от размера страницы кэш-памяти. Оптимальным для данного вида нагрузки будет максимально возможный из доступных размеров страниц (для данной обучающей выборки - 2500 блоков). С вероятностью 81% данный тип нагрузки будет определен моделью логистической регрессии (алгоритмом t=1024). Если при этом увеличить размер страницы со стандартных 64 Кб до рекомендованных 1280 Кб, возможно увеличение процента попадания с 40% до 60%.

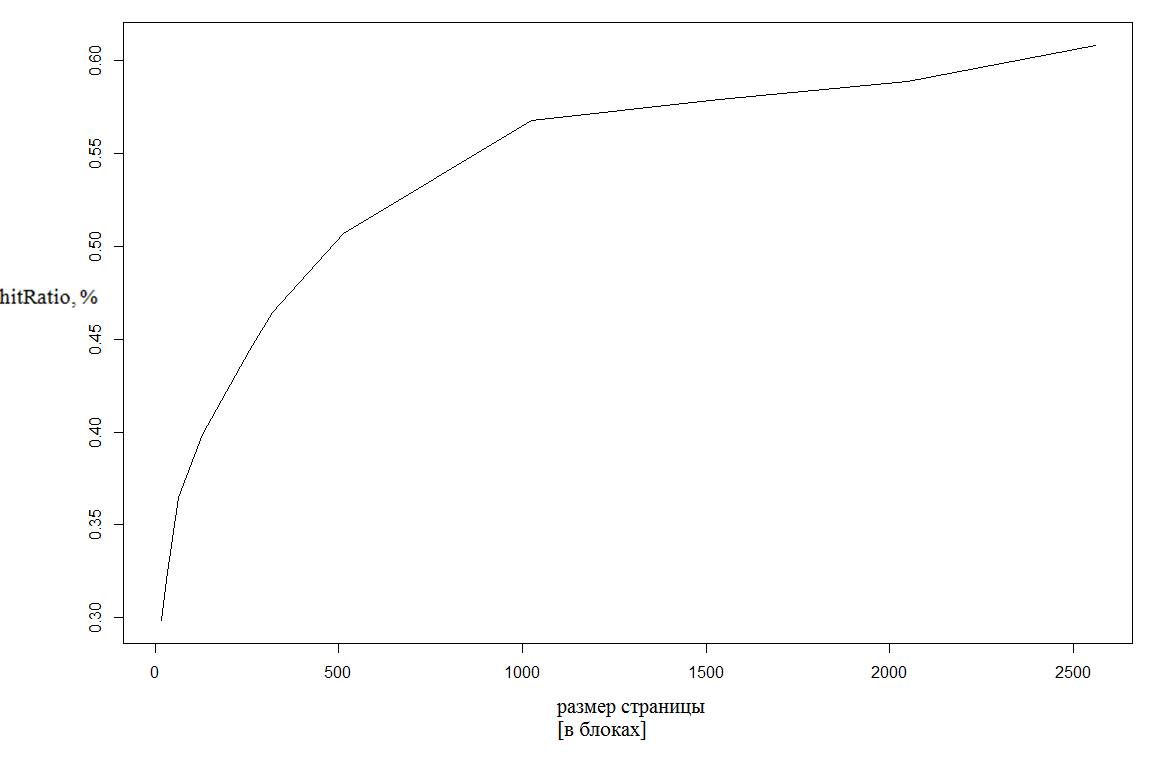


Рис. 4.2.4. Зависимость процента попадания от размера страницы

# Заключение

В рамках данной работы был построен симулятор кэш-памяти, с подсчетом характеристик нагрузки, написаны утилиты для сбора обучающей выборки в автоматическом режиме. Получено решение задачи для двух групп размеров страниц с точностью до 80% на тестовой выборке.

Также, важным результатом работы автор считает получение базовых навыков работы со средой R и освоение базовых методов машинного обучения.

В данный момент продолжается работа по сбору обучающей выборки большего размера, интерпретации результатов модели решающих деревьев и введение в модель оценки доли последовательных заявок, что по получению более обширной обучающей выборки, позволит построить классификатор на большее число классов размеров страниц.

Материалы данной работы были представлены на конференции «Технологии Microsoft в теории и практике программирования. Новые подходы к разработке ПО на примере технологий Microsoft и EMC». Тезисы были опубликованы в сборнике конференции.

**Список использованной литературы**

1. John Gantz, David Reinsel. 2010. The Digital Universe Decade – Are You Ready? URL: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf (дата обращения 10.04.2014).
2. Bruce Jacob, Spencer Ng and David Wang. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann Publishers Inc. 2007.
3. Kristof Beyls, Erik H. D’Hollander. Reuse distance as a metric for cache behavior. In proceedings of the lasted conference on parallel and distributed computing and systems. 2001. 617—662 p.
4. David A. Patterson, John L. Hennessy. Computer Architecture: A Quantitative Approach. Morgan Kaufmann 1990, chapter 5 - Memory Hierarchy Design.
5. Richard Uhlig, Trevor Mudge. Trace-Driven Memory Simulation: A Survey. ACM Computing Surveys. 2004. Volume 29. 128—170 p.
6. Dror Feitelson. Workload Modeling for Performance Evaluation. Springer-Verlag. 2002.
7. Christopher Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc. 2006.
8. К. В. Воронцов. Лекции по методам оценивания и выбора моделей. URL: http://www.ccas.ru/voron/download/Modeling.pdf (дата обращения 10.06.14)
9. Terry M. Therneau, Elizabeth J. Atkinson. An Introduction to Recursive Partitioning Using the RPART Routines. Mayo Foundation 1997.
10. George Almási, Cǎlin Caşcaval, David Padua. Calculating Stack Distances Efficiently. In Proceedings of the 2002 workshop on Memory system performance (MSP '02). ACM. 37—43 p.
11. Роберт И. Кабаков. R в действии. Анализ и визуализация данных в программе R. ДМК Пресс, 2014.
12. Burnham, K. P.; Anderson, D. R. (2002), Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed.), Springer-Verlag.
13. EMC Symmetrix vmax with enginuity. URL: http://www.emc.com/collateral/hardware/product-description/h6544-vmax-w-enginuity-pdg.pdf (дата обращения 23.06.14).

# Приложение 1. Пример потока адресов

Timestamp Type Director LUN Address Size

370.153854 Write 0 0025 28560177 599

370.153856 Read 0 002B 712293600 8

370.154161 Read 0 001F 689146896 8

370.154166 Read 0 0017 863200296 8

370.154536 Read 0 0043 4430568 128

370.155621 Read 0 0045 22659112 8

370.155924 Read 0 0043 4430696 128

370.156300 Read 0 0021 573775336 8

370.156755 Read 0 0015 205145968 8

370.156931 Read 0 001F 398723768 8

370.156942 Read 0 0043 4430824 128

370.157264 Read 0 002D 339859328 8

370.157308 Read 0 002D 238533016 8

370.157312 Read 0 0039 539335792 128

370.158012 Read 0 0039 916204448 128

370.158570 Read 0 0043 4430952 128

370.158821 Read 0 0039 354801432 128

370.159074 Read 0 002B 480352144 8

370.159077 Read 0 0015 710876000 8

370.159079 Read 0 0021 758607112 8

370.159082 Read 0 002D 607852064 8

370.159955 Read 0 0043 4431080 128

370.160458 Read 0 0021 884481504 8

370.160911 Read 0 002B 712293592 8

370.160963 Read 0 001F 438256248 8

370.161163 Write 0 0030 30200407 1

370.161165 Read 0 0039 916204576 128

370.161416 Read 0 0017 875582176 8

370.161419 Read 0 002D 451067344 8

370.162355 Read 0 0039 539336176 128

370.162551 Read 0 002B 1183763968 8

370.162601 Read 0 001F 837292336 8

370.163057 Read 0 0003 485077904 8

370.163234 Read 0 0039 539336432 128

370.163485 Read 0 0017 863200440 8

370.163810 Read 0 002D 339859320 8

370.163862 Read 0 002D 238533008 8

370.164189 Read 0 0039 916204704 128

370.164241 Read 0 0039 539336560 128

370.164244 Read 0 0043 4431208 128

370.165199 Read 0 0039 916204832 128

370.165203 Read 0 0017 875001448 8

370.165250 Read 0 001F 698943040 8

370.165252 Read 0 0039 354801560 128

370.165449 Read 0 002B 712293584 8

370.165504 Read 0 001F 331702672 8

370.165878 Read 0 0017 875582160 8

370.165890 Read 0 0043 4431336 128

370.166962 Write 0 002B 463181984 8

370.166965 Read 0 0014 1206554096 8

370.167020 Read 0 002D 235883960 8

370.167022 Write 0 0030 30200408 3

370.167025 Read 0 0015 710875904 8

370.167028 Read 0 0014 600725392 8

370.167031 Write 0 002D 438088520 8

370.167046 Read 0 0014 247369608 8

370.167215 Write 0 002D 892096960 8

370.167218 Read 0 0014 2112416 8

370.167266 Read 0 0014 2137400 8

370.167269 Read 0 0043 4431464 128

370.167772 Read 0 0003 485077848 8

370.168096 Read 0 0014 185641200 8

370.168099 Read 0 0015 710875912 8

370.168525 Read 0 001F 696317664 8

370.169157 Read 0 0015 1104717472 8

370.169161 Read 0 0021 573786320 8

370.169235 Read 0 0015 710876008 8

370.169669 Read 0 0043 4431592 128

370.169735 Read 0 002B 712293928 8

370.171248 Read 0 0015 712911704 8

370.171256 Read 0 002B 1185782848 8

# Приложение 2. Пример описания нагрузки

<WorkloadDefenition>

<Skews>

<Skew name="unnamed\_skew" granularityGB="0.351563">

<Point capacity="0.05" workPercentage="0.9097" />

<Point capacity="0.15" workPercentage="0.9615" />

<Point capacity="0.5" workPercentage="0.9945" />

</Skew>

</Skews>

<IOsSizes>

<ReadSizeSet name="unnamed\_readSizes">

<Size sizeBlocks="64" probability="0.00585496" />

<Size sizeBlocks="18" probability="4.70332e-005" />

<Size sizeBlocks="2" probability="0.000177382" />

<Size sizeBlocks="96" probability="0.00289725" />

<Size sizeBlocks="4" probability="0.000993073" />

<Size sizeBlocks="72" probability="0.0283691" />

<Size sizeBlocks="144" probability="2.68761e-006" />

<Size sizeBlocks="47" probability="4.03142e-006" />

<Size sizeBlocks="3" probability="0.000215009" />

<Size sizeBlocks="32" probability="0.0263184" />

<Size sizeBlocks="80" probability="0.00329636" />

<Size sizeBlocks="128" probability="0.00743394" />

<Size sizeBlocks="31" probability="1.20943e-005" />

<Size sizeBlocks="136" probability="0.314539" />

<Size sizeBlocks="16" probability="0.556102" />

<Size sizeBlocks="8" probability="0.0252703" />

<Size sizeBlocks="105" probability="5.37522e-006" />

<Size sizeBlocks="40" probability="0.00280587" />

<Size sizeBlocks="24" probability="0.00328561" />

<Size sizeBlocks="121" probability="9.40664e-006" />

<Size sizeBlocks="48" probability="0.00658599" />

<Size sizeBlocks="200" probability="1.34381e-006" />

<Size sizeBlocks="1" probability="0.0018988" />

<Size sizeBlocks="98" probability="9.40664e-006" />

<Size sizeBlocks="10" probability="4.56894e-005" />

.....

</ReadSizeSet>

<WriteSizeSet name="unnamed\_writeSizes">

<Size sizeBlocks="15" probability="0.000698101" />

<Size sizeBlocks="112" probability="0.00196994" />

<Size sizeBlocks="6" probability="0.0039088" />

<Size sizeBlocks="56" probability="0.00168166" />

<Size sizeBlocks="30" probability="0.000110226" />

<Size sizeBlocks="14" probability="0.000480474" />

<Size sizeBlocks="9" probability="0.000850722" />

<Size sizeBlocks="208" probability="8.47896e-006" />

<Size sizeBlocks="94" probability="6.50054e-005" />

.....

</WriteSizeSet>

</IOsSizes>

<CIMPs>

<CIMP name="unnamed\_read\_cimp">

<Point sizeKB="1280" hitPercentage="0.976561" />

<Point sizeKB="3840" hitPercentage="0.983091" />

<Point sizeKB="12800" hitPercentage="0.992135" />

</CIMP>

<CIMP name="unnamed\_write\_cimp">

<Point sizeKB="1280" hitPercentage="0.942895" />

<Point sizeKB="3840" hitPercentage="0.974298" />

<Point sizeKB="12800" hitPercentage="0.988985" />

</CIMP>

</CIMPs>

<Data>

<StorageGroup name="unnamed\_sg">

<Lun name="51e" sizeGB="97.7344" />

<Lun name="546" sizeGB="97.7344" />

<Lun name="54e" sizeGB="93.8672" />

<Lun name="556" sizeGB="97.7344" />

<Lun name="6e4" sizeGB="147.305" />

<Lun name="57e" sizeGB="97.7344" />

<Lun name="4ce" sizeGB="200.742" />

<Lun name="49e" sizeGB="168.047" />

<Lun name="396" sizeGB="201.445" />

<Lun name="2e2" sizeGB="196.172" />

<Lun name="3f6" sizeGB="201.797" />

<Lun name="2b2" sizeGB="142.734" />

<Lun name="5ce" sizeGB="97.7344" />

<Lun name="44a" sizeGB="202.148" />

<Lun name="708" sizeGB="57.6563" />

<Lun name="4c2" sizeGB="198.281" />

<Lun name="31e" sizeGB="194.063" />

<Lun name="ea" sizeGB="43.9453" />

<Lun name="89" sizeGB="73.4766" />

<Lun name="576" sizeGB="95.625" />

.......

</StorageGroup>

</Data>

<TimePeriods />

<Sequentials />

<WorkloadMix name="unnamed\_wldMix">

<SR value="606698" />

<SW value="165890" />

<RH value="3674" />

<RM value="54395" />

<WH value="13171" />

<WM value="95368" />

</WorkloadMix>

<WorkloadDefenitionComponent name="unnamed">

<readIo value="744155" />

<writeIo value="353817" />

<WldMix name="unnamed\_wldMix" />

<StorageGroup name="unnamed\_sg" />

<CIMPRead name="unnamed\_read\_cimp" />

<CIMPWrite name="unnamed\_write\_cimp" />

<Skew name="unnamed\_skew" />

<ReadSizes name="unnamed\_readSizes" />

<WriteSizes name="unnamed\_writeSizes" />

<WldMix name="unnamed\_wldMix" />

</WorkloadDefenitionComponent>

</WorkloadDefenition>