САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Морозов Сергей Валерьевич

Моделирование потока запросов на чтение и запись данных

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор Нестеров В. М.

Санкт-Петербург
2014

Оглавление

Введение

Вращающиеся диски – самый медленный компонент системы хранения данных. Доступ к данным на жестких дисках занимает как правило несколько миллисекунд, тогда как доступ к данным внутри кэш-памяти занимает менее миллисекунды.

Почему нельзя полностью заменить жесткие диски на твердотельные накопители? У организаций возникает потребность в хранении чрезмерно больших объемов данных. Хранение данных на твердотельных накопителях значительно бы ускорило доступ к данным, по сравнению с хранением на жестких дисках, но стоимость оборудования на твердотельных накопителях значительно превышает стоимость хранения на магнитных дисках. В современных системах хранения используются системы, укомплектованные магнитными жесткими дисками и высокоэффективной кэш-памятью, дающей возможность оптимизировать операции ввода-вывода. Такое решение проблемы позволяет значительно уменьшить стоимость за гигабайт памяти ($/GB) и при этом обеспечивает вполне приемлемое время доступа к данным.

Моделирование рабочей нагрузки на систему хранения прежде всего нужно для оценки производительности системы. Смоделированная нагрузка должна быть обобщенной моделью реальной нагрузки. Тестирование оборудования на нагрузку дает заказчику уверенность в том, что система будет работать стабильно в реальных условиях, и упраздняет или уменьшает необходимость испытания оборудования на реальном объекте, что приводит к существенной экономии ресурсов.

Компаниям, где надежность оборудования имеет высокий приоритет, полезно иметь инструмент, который позволяет тестировать оборудование на нагрузку и “на лету” менять интенсивность загрузки системы путем изменения широкого спектра параметров.

Целью настоящей работы является создание тестового пакета, моделирующего нагрузку на кэш-память. Необходимо обеспечить простой способ генерирования нагрузки с возможностью гибкого изменения параметров, влияющих на интенсивность запросов на чтение и запись данных и разброс адресов. Таким образом в рамках работы поставлены следующие задачи:

- Изучить наиболее типичные потоки запросов и алгоритмы генерации нагрузки на кэш-память.
- Реализовать известные алгоритмы генерации нагрузки на кэш-память с возможностью гибкого определения параметров.
- Написать легко расширяемый для включения генерации разных потоков запросов тестовый пакет.

1. Обзор существующих решений

Существует ряд программных продуктов и научных работ, связанных с моделирование поведения различных систем. Ниже приведены некоторые из них:

- *Подсистема кэширования СХД AVRORA [8]:* Бакалаврская работа Золотникова Павла по созданию подсистемы кэширования СХД AVRORA. На вход модели подается готовая трасса запросов, которая генерируется реальной системой, являясь по-сути логом ее работы. Результатом работы модели является временная последовательность команд чтения/записи с указанием адреса и размера данных.

- *Oracle's Sun Unified Storage Simulator [4]:* Виртуальная система хранения данных. Имеется возможность тестирования различных конфигураций.

- *GigaNet Systems VirtualSAN [6]:* Эмулятор оптоволоконной сети хранения данных. Имеется возможность воспроизводить различные состояния сети и различные нарушения в ее работе.

- *SimSANs [5]:* Продукт, использующийся для проектирования и симуляции систем хранения данных.

Тематика работы по моделированию работы кэш-памяти в системе хранения данных AVRORA близка к тематике настоящей работы. Основное отличие в том, что подсистема кэширования СХД AVRORA принимает на вход готовую трассу запросов на чтение и запись данных. В настоящей же работе рассматривается вопрос о создании таких трасс запросов с помощью моделирования.

2. Кэш-память

Кэш-память – это полупроводниковая память, куда временно помещаются данные, для того чтобы снизить время обслуживания запросов на чтение и запись поступающих от какой-либо вычислительной платформы (*хоста*) [1]. Кэш-память позволяет значительно увеличить производительность системы хранения, изолируя хост от механических задержек, связанных с работой магнитных дисков.

2.1. Структура кэш-памяти

Кэш-память состоит из блока данных и теговой памяти. Наименьшей единицей информации является *страница*. Блок данных состоит из набора страниц, а теговая память устанавливает соответствие страницы в блоке данных странице на диске. Структура кэш-памяти представлена на Рис. 1.

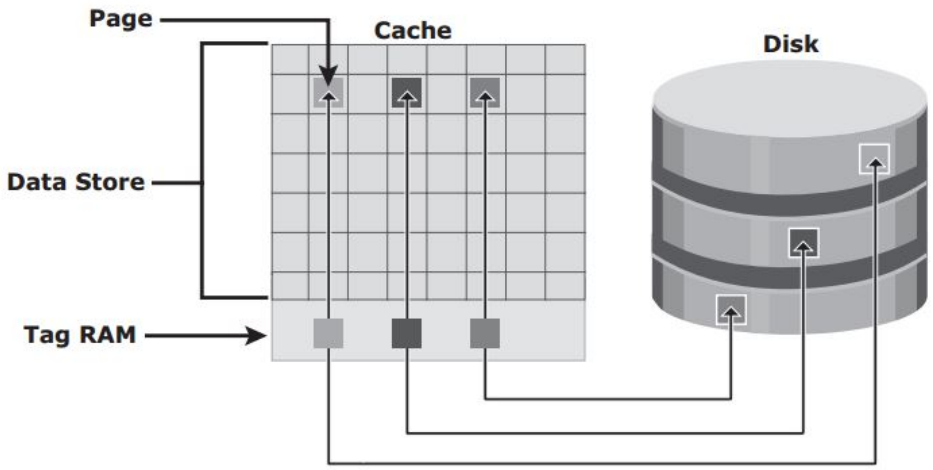


Рис. 1: Структура кэш-памяти

Теговая память также содержит дополнительную информацию. Например флаг *dirty*, который показывает соответствуют ли содержимое страницы в кэш-памяти содержимому страницы на диске, время последнего обращения, и прочее.

2.2. Операции чтения и записи

Чтение

При поступлении запроса на чтение проверятся присутствуют ли запрашиваемые данные в кэш-памяти. Ситуация, когда данные присутствуют в кэш-памяти, называется *read hit*. В этом случае чтение занимает порядка миллисекунды и данные считываются прямо из кэш-памяти и обращения к диску не происходит, что позволяет существенно сэкономить время доступа к данным. Ситуация, когда данные не содержатся в кэш-памяти, называется *read miss*. В этом случае происходится считывание

данных с диска в кэш-память, а затем эти данные посылаются хосту. Время выполнения запроса на чтение значительно увеличивается.

Запись

Запись с использованием кэш-памяти происходит значительно быстрее, чем запись напрямую на диск. Запись обновленных в кэш-памяти данных на диск может быть организована двумя разными способами.

1. *Write-back cache:* Данные записываются в кэш-память и подтверждение об успешной операции записи посылается хосту немедленно. Затем, после нескольких запросов на запись, данные записываются на диск. В этом случае существует риск потери данных, если кэш-память выйдет из строя.

2. *Write-through cache:* Данные записываются в кэш-память и сразу после этого на диск. Только после этого посылается подтверждение хосту об успешно выполненной операции записи. Такой подход увеличивает время операции записи, но риск потери данных сильно снижается.

Для страниц, которые были записаны в кэш-память, но еще не записаны на диск, устанавливается флаг dirty, обозначающий, что страница в кэш-памяти не соответствуют странице на диске. После записи страницы на диск флаг dirty снимается.

2.3. Алгоритмы управления кэш-памятью

Размер кэш-памяти ограничен, поэтому, когда кэш-память заполнена, необходимо записывать новые данные вместо данных, уже находящихся в кэш-памяти. Существуют различные алгоритмы освобождения кэш-памяти, ниже приведены наиболее часто используемые:

1. *Least Recently Used (LRU):* Алгоритм основан на предположении, что данные, к которым не было обращений долгое время, и в дальнейшем использоваться не будут. Когда кэш-память заполнена, и происходит обращение к не содержащейся в стеке страцице, то будет вытеснена страница, к которой не было обращений самое долгое время. Если страница присутствует в кэш-памяти, то она помечается как последняя использованная. LRU алгоритм будет рассмотрен подробнее в следующей секции.

2. *Most Recently Used (MRU):* Алгоритм основан на предположении, что данные, к которым недавно были обращения более не потребуются. Когда кэш-память заполнена, и происходит обращение к не содержащейся в стеке страницы, то будет вытеснена последняя использованная страница. Если страница присутствует в кэш-памяти, то она помечается как последняя использованная.

3. LRU алгоритм управнения кэш-памятью

Предположим, что размер кэш-памяти равен $CacheSize$ страниц. Теговую память кэш-памяти, управляемой LRU алгоритмом, можно представить в виде LRU стека. Максимальная глубина стека равна $CacheSize - 1$, вершина стека имеет глубину 0. Глубина элемента в стеке называется *стековым расстоянием*. Элементы стека это пара *<адрес старницы в блоке данных, адрес страницы на диске>*.

Рассмотрим сначала случай, когда в кэш-памяти еще имеются свободные страницы. При поступлении от хоста запроса на чтение или запись данных просматривается LRU стек. Если запрашиваемый адрес присутствует в стеке на глубине i, то он переносится на вершину стека, а глубина всех элементов стека с номерами меньше i увеличивается на 1. Если запрашиваемый адрес в стеке не присутствует, то текущий размер стека увеличивается на 1, данные по запрашиваемому адресу читаются в кэш-память, и на вершину стека кладется новая пара *<адрес страницы в блоке данных, запрашиваемый адрес на диске>*. Глубина всех остальных элементов стека увеличивается на 1.

Если кэш-память заполнена, и поступает запрос на чтение или запись от хоста, то меняется механизм добавления нового адреса в стек. В страницу в блоке данных, соответствующую элементу стека с номером $CacheSize - 1$ записываются новые данные с диска. Новая пара *<адрес страницы в блоке данных, запрашиваемый адрес на диске>* кладется на вершину стека, а элемент с номером $CacheSize - 1$ выталкивается из стека. Глубина всех остальные элементов стека увеличивается на 1. Работа LRU алгоритма представлена на Рис. 2.

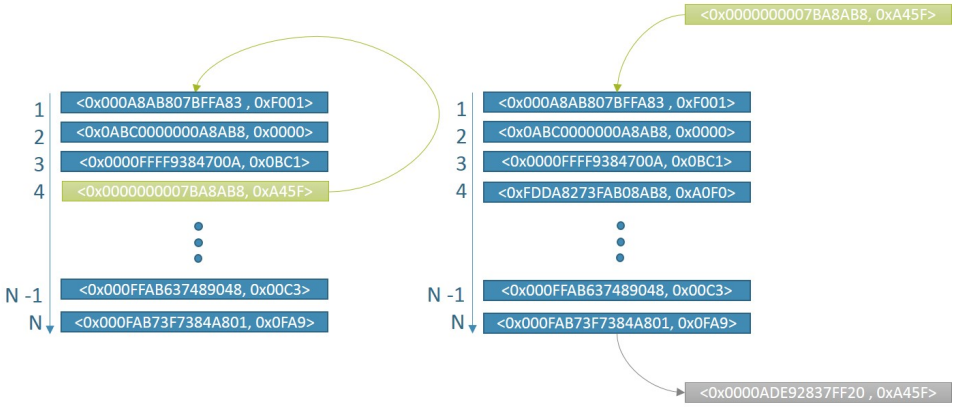


Рис. 2: LRU стек

4. Моделируемые данные

Нагрузку на систему будем представлять в виде последовательности элементарных транзакций (*запросов*). Запрос – это четверка <*Время транзакции, Адрес в кэш-памяти, Размер транзакции, Тип транзакции*> (см. Рис. 3).

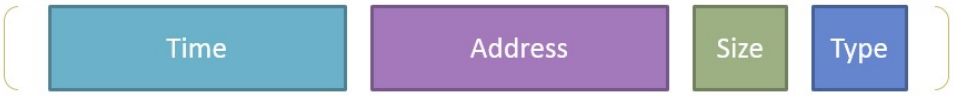


Рис. 3: Запрос (*Request*)

Интенсивности нагрузки изменяется во времени по специальному алгоритму, моделирующему типичное поведение приложений, нагружающих реальные системы. Компоненты запроса подчиняются определенным законам, к примеру, адреса распределены посредством моделирования кэш-памяти с распределением стековых расстояний в соответствие с распределением Парето, а время между последовательными запросами в соответствие с экспоненциальным распределением.

Компоненты запроса:

- Время (*Time*): Время первой транзакции $t_1 = 0$. Время остальных транзакций задается рекурентно $t_i = t_{i-1} + \Delta t_i \ \forall i = 2, ..., n$, где n – количество запросов, которые необходимо сгенерировать. Время между последовательными транзакциями Δt_i распределено в соответствие с экспоненциальным распределением (1). Экспоненциальное распределение имеет единственный параметр λ, называемый параметром скорости (*rate parameter*).

$$F(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & , \quad x \geq 0 \\ 0 & , \quad x < 0 \end{cases} \tag{1}$$

- Адрес (*Address*): Определяется путем моделирования работы кэш-памяти при помощи LRU алгоритма с распределением стековых расстояний в соответствие с распределением Парето (2). Распределение Парето имеет два параметра: k и a. k – параметр положения (*location parameter*), определяет минимально возможное значение $x \geq k$. a – параметр формы (*shape parameter*), определяет "хвост" распределения Парето [2].

$$F(x;k,a) = \begin{cases} 1 - (\frac{k}{x})^a, & x \in [k, +\infty) \\ 0, & x \in (-\infty, k) \end{cases} \tag{2}$$

- Размер транзакции (*Size*): Определенный размер считываемого или записываемого блока данных. Задается вероятностями появления в последовательности запросов (Таблица 1).

Таблица 1: Вероятности появления запросов определенных размеров

512 B	1 KB	2 KB	8 KB	64 KB
0.3	0	0.25	0.35	0.1

- Тип транзакции (*Type*): Запрос на чтение или запись. Задается вероятностями появления в последовательности запросов (Таблица 2).

Таблица 2: Вероятности появления запросов на чтение и запись

Read	Write
0.7	0.3

5. Алгоритм изменения интенсивности нагрузки

Интенсивность нагрузки системы изменяется во времени. Рассмотрим алгоритм, моделирующий типичное поведение приложений, нагружающих систему. Будем считать, что наша система может находиться в нескольких состояниях. Ограничим количество возможных состояний шестью.

В каждый момент времени система находится в определенном состоянии. Для всех состояний определены свои значения параметров распределений и вероятностей для каждого компонента элементарной транзакции. Времена нахождения в состояниях, распределены в соответствие с экспоненциальным распределением (1). Переход из состояния в состояние задается с помощью матрицы состояний (Таблица 3), в ячейках которой располагаются параметры λ_{ij}, задающие параметр скорости, используемого для расчета времени пребывания в состоянии i перед переходом в состояние j. Выбор следующего состояния осуществляется при помощи выбора минимального промежутка времени, необходимого для нахождения в текущем состоянии перед переходом в следующее [2]. Схема работы этого механизма представлена на Рис. 4.

Таблица 3: Матрица состояний

	1	2	3	4
1	0	0.25	0.35	11.4
2	7	0	8.35	13.7
3	6	15.3	0	7.1
4	1	5.5	4	0

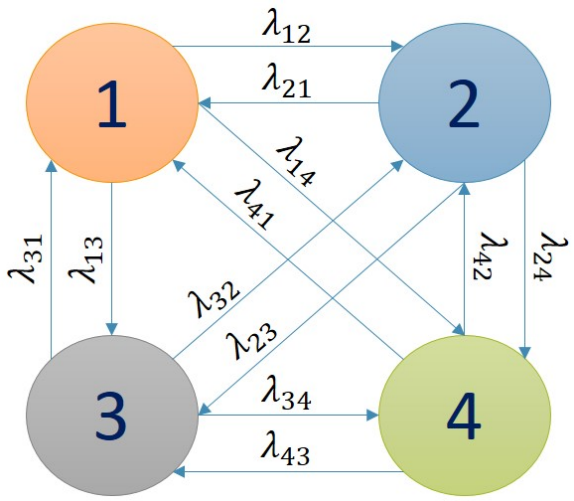


Рис. 4: Схема работы механизма изменяющейся во времени нагрузки

6. Тестовый пакет для моделирования нагрузки на кэш-память

Программа для генерации нагрузки должна быть удобной в использовании и иметь возможность быстрой калибровки параметров под конкретное устройство. Была предпринята попытка создания такой программы. Вышеописанный алгоритм моделирования нагрузки на систему хранения был реализован на языке *C++* в виде динамически подключаемой библиотеки *DLL*. Графический интерфейс написан на языке *C#* с использованием *Windows Forms*.

6.1. Реализация LRU алгоритма

Главную роль при моделировании распределения адресов играют стековые расстояния, распределенные в соответствие с распределением Парето (2). Можно отказаться от хранения в стеке адреса страницы в блоке данных кэш-памяти, в связи с тем, что эта информация никак не используется. Таким образом в стеке хранится только адрес страницы на диске. Когда происходит *cache miss* адрес страницы на диске выбирается равномерно.

При моделировании нагрузки можно задать размер LRU стека, а также выбрать его тип *empty* или *filled*, то есть выбрать будет ли стек заполнен на момент начала генерации запросов или нет.

LRU стек был реализован двумя разными способами. Первый способ – с применением std::deque, а второй с применением std::list. Класс std::deque позволяет вставлять и удалять элементы из начала и конца очереди за $O(1)$, а операция удаления произвольного элемента в худшем случае имеет сложность $O(\frac{CacheSize}{2})$, где $CacheSize$ – размер кэш-памяти (количество страниц). При этом доступ к произвольному элементу осуществляется за $O(1)$. Операция удаления и вставки произвольного элемента std::list имеет сложность $O(1)$, но сложность обращения к произвольному элементу в нашем случае равна $O(\frac{CacheSize}{2})$, в связи с тем, что изначально известен размер LRU стека.

Эксперимент показал, что реализация LRU стека при помощи класса std::deque работает быстрее при генерации трасс с большим количеством запросов, а реализация с применением класса std::list работает эффективнее при генерации трасс с малым количеством запросов. Эмпирическим путем было выяснено, что время работы обоих алгоритмов примерно одинаковое при генерации трасс размером 65000 запросов, поэтому иготовая программа включает в себя две реализации LRU стека, а какая из них будет вызываться при генерации потока запросов зависит от размера трассы. В Таблице 4 представлена зависимость времени исполнения обоих реализаций от количества генерируемых запросов.

Таблица 4: Время выполнения различных реализаций LRU стека

	100 запр.	1000 запр.	10000 запр.	100000 запр.	1000000 запр.
LRU с std::deque	0.008 сек	0.014 сек	0.07 сек	0.558 сек	7.911 сек
LRU с std::list	0.001 сек	0.006 сек	0.062 сек	0.64 сек	14.329 сек

6.2. Вычислительная библиотека

Вычислительная часть тестового пакета для моделирования нагрузки, то есть сам алгоритм, реализован на языке *C++* в виде динамически подключаемой библиотеки *DLL*. Это дает возможность пользоваться этой библиотекой без привязки к графическому интерфейсу.

При реализации был использован объектно-ориентированный подход, поэтому библиотеку легко поддерживать и добавлять новую функциональность, например добавлять новые виды распределений для элементов запроса.

Результатом работы вычислительной библиотеки является бинарный файл с последовательностью запросов. За создание файла отвечает специальный класс IO_Manager, позволяющий записывать и читать такие бинарные файлы.

6.3. Графический интерфейс

Графический интерфейс (Рис. 5) написан на языке *C#* с использованием *Windows Forms*.

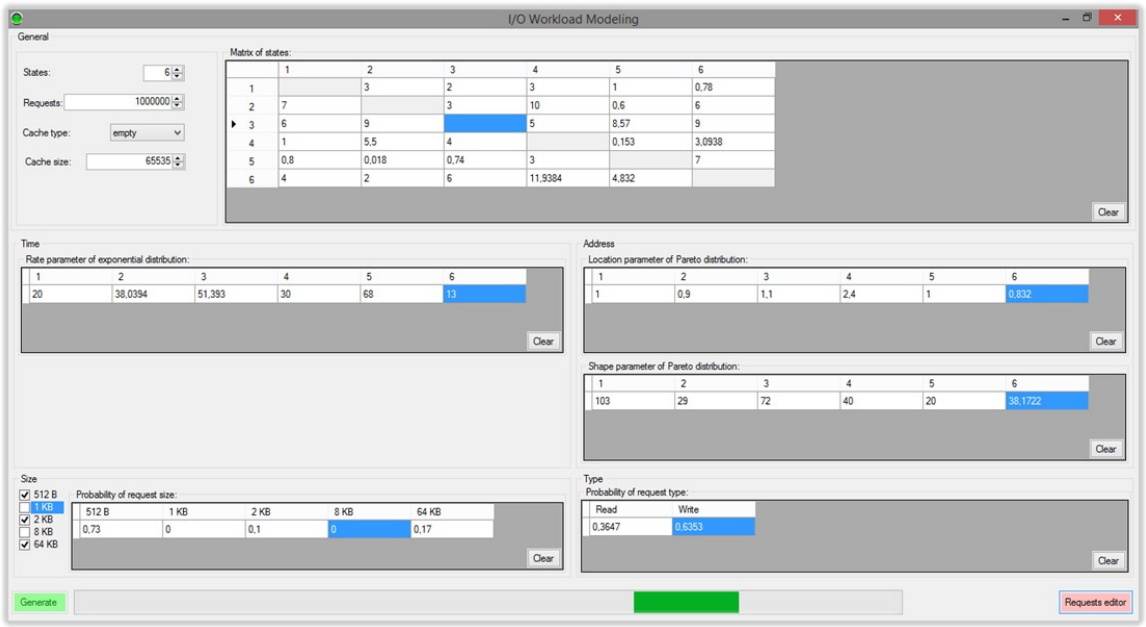
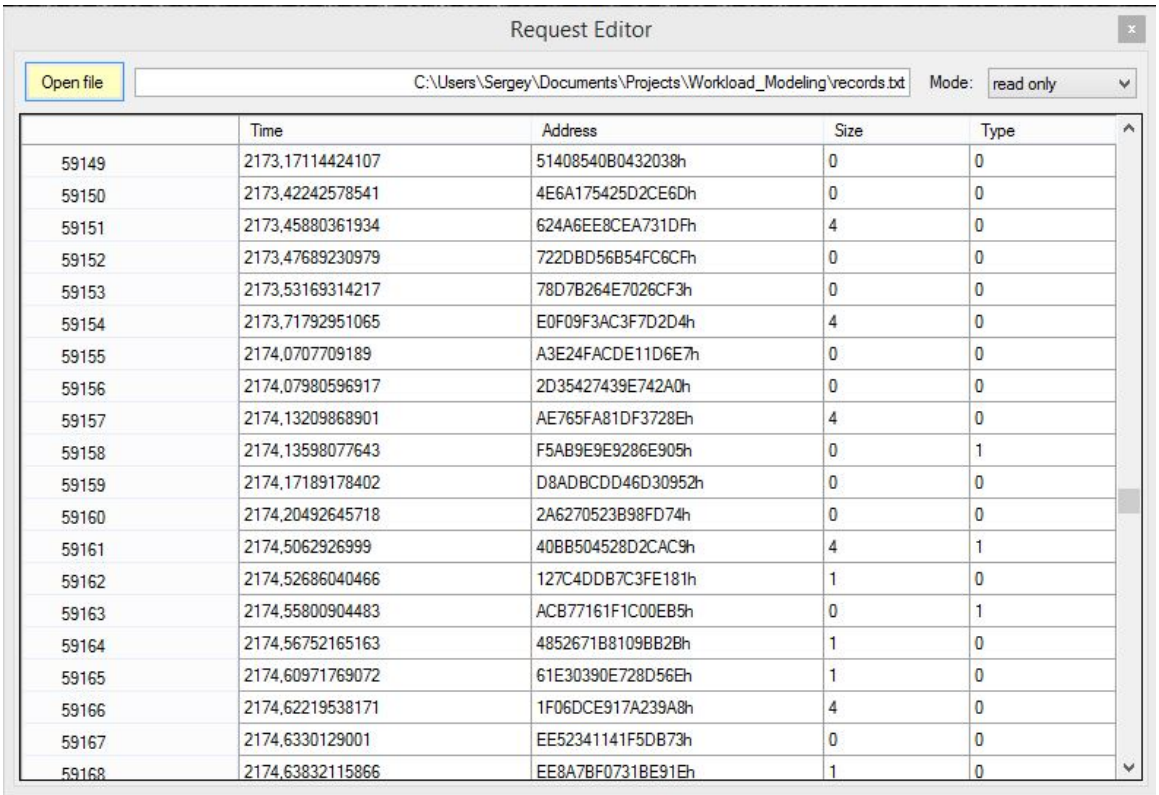


Рис. 5: Графический интерфес тестового пакета

Имеется возможность задать большое количество параметров:

- Количество состояний системы
- Количество запросов
- Размер кэш-памяти (в страницах)
- Тип LRU стека (*empty* или *filled*)
- Матрица состояний
- Параметры распределений или вероятности появления для каждого элемента запроса в любом состоянии системы.

Имеется встроенный графический редактор (Рис.6) запросов работаюший в двух режимах: *только чтение*, *модификация*. В режиме *модификация* можно изменять отдельные элементы каждого запроса. Изменения сразу же будут применены к входному бинарному файлу с трассой запросов.



	Time	Address	Size	Type
59149	2173,17114424107	51408540B0432038h	0	0
59150	2173,42242578541	4E6A175425D2CE6Dh	0	0
59151	2173,45880361934	624A6EE8CEA731DFh	4	0
59152	2173,47689230979	722DBD56B54FC6CFh	0	0
59153	2173,53169314217	78D7B264E7026CF3h	0	0
59154	2173,71792951065	E0F09F3AC3F7D2D4h	4	0
59155	2174,0707709189	A3E24FACDE11D6E7h	0	0
59156	2174,07980596917	2D35427439E742A0h	0	0
59157	2174,13209868901	AE765FA81DF3728Eh	4	0
59158	2174,13598077643	F5AB9E9E9286E905h	0	1
59159	2174,17189178402	D8ADBCDD46D30952h	0	0
59160	2174,20492645718	2A6270523B98FD74h	0	0
59161	2174,5062926999	40BB504528D2CAC9h	4	1
59162	2174,52686040466	127C4DDB7C3FE181h	1	0
59163	2174,55800904483	ACB77161F1C00EB5h	0	1
59164	2174,56752165163	4852671B8109BB2Bh	1	0
59165	2174,60971769072	61E30390E728D56Eh	1	0
59166	2174,62219538171	1F06DCE917A239A8h	4	0
59167	2174,6330129001	EE52341141F5DB73h	0	0
59168	2174,63832115866	EE8A7BF0731BE91Eh	1	0

Рис. 6: Редактор запросов

Заключение

В ходе курсовой работы были изучены наиболее типичные потоки запросов и алгоритмы генерации нагрузки на кэш-память. Реализован тестовый пакет по моделированию нагрузки запросами на чтение и запись данных кэш-памяти. Тестовый пакет включает в себя удобный графический интерфейс с возможностью гибкого изменения входных параметров генератора нагрузки, а так же вычислительную библиотеку, реализующую алгоритмическую часть моделирования потока запросов. Тестовый пакет можно легко расширять, добавляя новые функции. Также можно использовать вычислительную динамически подключаемую библиотеку в своих проектах без привязки к графическому интерфейсу.

Чтобы смоделировать нагрузку для конкретного устроства, необходимо подобрать параметры распределений и вероятности для компонентов запроса для каждого состояния системы. Подбирая параметры, можно определить при какой нагрузке конкретное устройство работает максимально эффективно, выявить отклонения в работе устройства при определенных входных данных. Приведем пример. У компании появилась необходимость замены устройства кэш-памяти в системе хранения данных. Можно смоделировать нормальные условия работы системы хранения данных и работу системы при пиковой нагрузке и просмотреть, как новое устройство кэш-памяти будет реагировать на такие нагрузки. Такое тестирование уменьшает риск того, что новое устройство кэш-памяти не справится с требуемой нагрузкой или будет работать не эффективно.

Продолжением этой работы я вижу испытание генератора нагрузки на реальной системы хранения данных. Это позволит доказать практическую применимость разработанного тестового пакета, а также, определить какие функции и параметры дополнительно включить в тестовый пакет, а какие убрать.

Список литературы

[1] EMC Education Services. Information Storage and Management Second Edition / Ed. by Alok Shrivastava Somasundaram Gnanasundaram.

[2] Feitelson Dror G. Workload Modeling for Computer Systems Performance Evaluation. — The Hebrew University of Jerusalem.

[3] George Alm´asi C˘alin Cascaval David A. Padua. Calculating Stack Distances Efficiently. — University of Illinois at UrbanaChampaign.

[4] Oracle's Sun Unified Storage Simulator. — URL: http://www.oracle.com/webapps/dialogue/ns/dlgwelcome.jsp?p_ext=Y&p_dlg_id=9668083&src=7011671&Act=111 (online; accessed: 26.05.2014).

[5] SimSANs. — URL: http://www.simsans.org/ (online; accessed: 26.05.2014).

[6] VirtualSAN Fibre Channel Emulators. — URL: http://www.giganetsystems.com/docs/VirtualSAN_Product_Brief_v2.1.pdf (online; accessed: 26.05.2014).

[7] А.Н.Бородин. Элементарный курс теории вероятностей и математической статистики.

[8] Модель подсистемы кэширования СХД AVRORA. — URL: http://se.math.spbu.ru/SE/diploma/2013/b/ZolnikovPavel_text.pdf (online; accessed: 26.05.2014).