УДК 681.324
doi:10.15217/issn1684-8853.2015.5.100

ЭВРИСТИЧЕСКИЙ АЛГОРИТМ РАСЧЕТА РАЗМЕРОВ ПАМЯТИ В МНОГОУРОВНЕВОЙ СИСТЕМЕ ХРАНЕНИЯ

Д. А. Маличенко[a, 1], *ведущий программист*
[a]*Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, РФ*

Введение: *рассматривается многоуровневая блочная система хранения данных. Уровни отличаются типом памяти и скоростью работы с данными. Для каждого приложения, работающего с системой, задано требуемое качество обслуживания, выражаемое в виде времени отклика системы на запрос. Распределение адресного пространства приложений между уровнями памяти влияет на время отклика системы. Целью работы является автоматическое распределение областей памяти приложений в зависимости от входного потока заявок таким образом, чтобы требования на качество обслуживания были выполнены.* ***Результаты:*** *сформулирована оптимизационная задача расчета размеров памяти приложений на всех уровнях системы и предложен эвристический алгоритм ее решения. Используются такие характеристики потока запросов, как среднее значение доли запросов, попадающих в кэш, и распределение частот запросов к блокам данных на дисках. Оценка работы алгоритма вычислена с помощью имитационного моделирования системы хранения, на которую подавался поток заявок, снятый ранее с реальных систем хранения. Результаты моделирования демонстрируют, что предложенный алгоритм в некоторых случаях позволяет повысить эффективность работы системы хранения до 10 %.* ***Практическая значимость:*** *предложенный алгоритм может улучшить качество обслуживания в многоуровневой системе хранения данных.*

Ключевые слова *— многоуровневая система хранения, управление многоуровневой системой хранения, Storage Area Network, качество обслуживания, моделирование.*

Введение

Системы хранения данных очень разнообразны по уровню сложности. Для некоторых задач обычные жесткие диски, которые используются в персональных компьютерах, не подходят. Например, у сервисов по бронированию и покупке билетов или у социальных сетей возникают дополнительные требования к системам хранения данных. Требуется хранить больше данных, одновременно обрабатывать заявки от большого числа клиентов, а также быстро работать с данными. Для таких целей используются специализированные сложные системы хранения, которые могут представлять собой целую сеть. В работе рассматривается пример такой системы хранения данных. Для увеличения скорости работы системы используются диски разного типа. Применение только очень быстрых дисков значительно увеличивает стоимость всей системы. Обычно комбинируют диски с разной скоростью работы. В этом случае более востребованные данные хранятся на более быстрых дисках, менее востребованные данные хранятся на медленных дисках. Со временем востребованность данных меняется, и управляющая система должна перекладывать данные между дисками разного типа. С системой хранения работает множество клиентов — приложений. У каждого приложения свой объем востребованных данных, следовательно, пропорции, в которых данные распределены между дисками разного типа, у каждого приложения свои. Одному приложению может требоваться больше быстрой памяти для обеспечения заданной скорости работы, другому меньше, а со временем эта ситуация может поменяться в обратную сторону. Если объем быстрой памяти, выделенный приложению, слишком мал, то все наиболее востребованные данные приложения не поместятся в быструю память, и часть данных будет считываться с медленных дисков, что приведет к уменьшению средней скорости работы с данными для приложения. Кроме дискового пространства у каждого приложения есть своя область в кэш-памяти. Требуемые размеры этих областей также разные для разных приложений и зависят от характеристик потока запрашиваемых данных. В данной работе рассматривается задача выбора для каждого приложения размера выделяемой памяти в кэше и на дисках всех типов.

Многоуровневые системы хранения известны давно. В работе [1] дано функциональное описание такой системы, а в работе [2] предложен алгоритм перемещения блоков данных между дисками разных типов. В работе [3] для идеализированной модели системы хранения представлены аналитические выражения для расчета требуемого размера кэша, а также для размеров дисков и оптимального количества уровней системы хранения. Оптимизация размеров кэш-памяти рас-

[1] Научный руководитель — профессор, проректор по науке, доктор технических наук, директор Института информационных систем и защиты информации, заведующий кафедрой безопасности информационных систем СанктПетербургского государственного университета аэрокосмического приборостроения *Е. А. Крук.*

смотрена в работах [4–6], а распределение места на дисках — в работах [7, 8]. В работе [7] система состоит из двух уровней памяти: SSD-дисков и обычных дисков, причем SSD-память работает в режиме кэша.

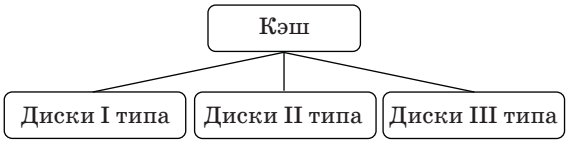
В настоящей работе, в отличие от многих других, рассматривается изоляция приложений друг от друга в кэше и на дисках. Решения алгоритма распространяются не только на диски, но и на кэш, для которого вычисляются требуемые хит-рейты.

Многоуровневая система хранения данных

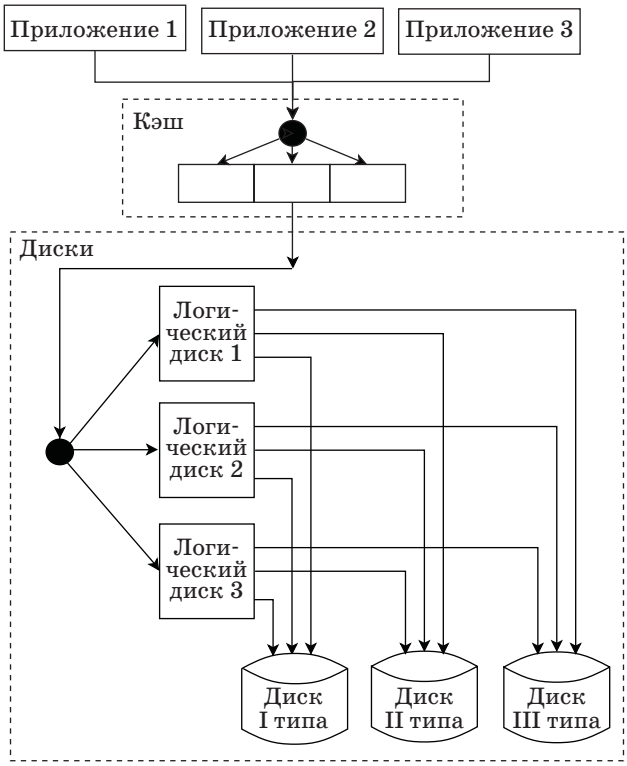
Опишем систему хранения с четырьмя типами памяти. Имеется кэш-память и три типа дисков (рис. 1), которые отличаются объемом и скоростью работы. Запрашиваемый блок данных сначала проверяется на наличие в кэше. Если блок в кэше отсутствует, то он считывается с того диска, на котором записан.

Рассмотрим сценарий, в котором у каждого приложения, работающего с системой, есть своя независимая область в кэше и свой логический диск, при этом физически пространство логического диска распределено между дисками всех типов. Более подробно работа системы проиллюстрирована на рис. 2.

Когда приложение 1 запрашивает блок данных, то система хранения сначала ищет его в выделенной для приложения 1 области кэша. Если блок найден, то время чтения блока равно времени чтения из кэша. Если блок отсутствует в кэше, то он считывается с диска. Приложение работает с логическим диском. На более низком уровне системы хранения лежат физические диски разного типа. Блок данных может лежать либо на быстрых дисках, либо на медленных. Примером быстрых дисков могут служить диски на flash-памяти (SSD-диски), примером медленных дисков могут служить SATA-диски. Расположение блоков данных постоянно меняется системой. Будем считать, что раз в час система перемещает блоки данных в соответствии с вероятностями их запросов. Все блоки сортируются по популярности, и самые популярные записываются на самые быстрые диски, пока место, выделенное для данного приложения, не закончится; далее блоки записываются на более медленные диски. Статистика по вероятностям запросов собирается в течение часа до следующего перемещения блоков.



■ ***Рис. 1.*** Упрощенная схема многоуровневой системы хранения



■ ***Рис. 2.*** Схема работы многоуровневой системы хранения с разделением приложений

Определение основных зависимостей и постановка задачи

Пусть время доступа к блоку данных в кэше равно t_0, а времена доступа к дискам t_1, t_2, t_3 и $t_0 << t_1 << t_2 << t_3$. Пусть объемы памяти в кэше и на дисках соответственно равны S_0, S_1, S_2, S_3. С системой работает n приложений A_1, ..., A_n. Пусть объем памяти, выделенный для i-го приложения на j-м уровне памяти, равен s_j^i. Под уровнем понимается тип памяти. Тогда

$$\sum_{i=1}^{n} s_j^i \le S_j. \qquad (1)$$

Пусть p_j^i — вероятность того, что блок данных, запрошенный i-м приложением, окажется на j-м уровне памяти. Тогда среднее время доступа к блоку данных для i-го приложения

$$\overline{T_i} = p_0^i t_0 + p_1^i t_1 + p_2^i t_2 + p_3^i t_3, \qquad (2)$$

где $p_0^i + p_1^i + p_2^i + p_3^i = 1$.

Пусть заданы ограничения на средние времена доступа к блоку данных для каждого приложения:

$$\overline{T_i} \le R_i. \qquad (3)$$

Следовательно, присутствует ограничение на вероятности p_j^i

$$p_0^i t_0 + p_1^i t_1 + p_2^i t_2 + p_3^i t_3 \le R_i. \quad (4)$$

Вероятности p_j^i связаны с объемами памяти s_j^i. Чем больше памяти приложение использует на j-м уровне относительно других уровней памяти, тем выше вероятность p_j^i. Обозначим эту зависимость так:

$$p_0^i = f_c^i(s_0^i);$$
$$p_j^i = f_d^i(s_j^i, p_0^i),\ 1 \le j \le 3,$$

где $f_c^i()$ и $f_d^i()$ — функции зависимости между размером занимаемой памяти и вероятностью попадания на нее для кэш-памяти и дисков соответственно.

Вероятность p_0^i называют хит-рейтом. Важной характеристикой кэш-памяти является стековое расстояние [9] — это количество уникальных блоков данных, запрошенных между запросом одного и того же блока, плюс один. Например, если блоки запрашивались в последовательности A, B, C, D, E, D, C, B, D, A, то стековое расстояние для второго появления блока A равно пяти. Функция распределения стековых расстояний $F_{sd}^i(x)$ дает вероятность того, что стековое расстояние для следующего запрошенного блока будет меньше либо равно x. Это означает, что если иметь в распоряжении кэш размером x блоков, то хит-рейт кэша будет равен $F_{sd}^i(x)$. Для i-го приложения этот факт можно записать таким образом:

$$p_0^i = F_{sd}^i(s_0^i).$$

Заметим, что $f_c^i(s_0^i) = F_{sd}^i(s_0^i)$.

Рассмотрим теперь $F_{pop}^i(x)$ — функцию распределения вероятностей того, что следующий запрошенный i-м приложением блок данных будет находиться среди первых x блоков с наивысшей вероятностью появления. Иногда $F_{pop}^i(x)$ называют популярностью адресов [9]. Чтобы получить эту функцию для потока запрашиваемых блоков данных, необходимо посчитать частотные вероятности каждого блока. После этого отсортировать пары значений (номер блока, вероятность) по убыванию вероятности. Перенумеруем блоки по порядку после сортировки, начиная от самого вероятного. Функция, задаваемая этими парами, где аргумент — номер блока по порядку, а значением является вероятность, будет функцией плотности вероятности блока, а построенная по ней функция распределения будет искомой $F_{pop}^i(x)$. Если имеется три диска разных типов с размерами S_1, S_2, S_3, и данные упорядочены по степени востребованности так, что самые востребованные хранятся на диске 1, а самые невостребованные — на диске 3, то вероятность того, что данные будут запрошены с диска 1, равна

$$p_1^i = \frac{F_{pop}^i(s_1^i)}{1 - p_0^i} = f_d^1, \quad (5)$$

вероятность запроса данных с диска 2

$$p_2^i = \frac{F_{pop}^i(s_2^i + s_1^i) - F_{pop}^i(s_1^i)}{1 - p_0^i} = f_d^2, \quad (6)$$

а вероятность запроса данных с диска 3

$$p_3^i = \frac{F_{pop}^i(s_3^i + s_2^i + s_1^i) - F_{pop}^i(s_2^i + s_1^i)}{1 - p_0^i} = f_d^3. \quad (7)$$

Таким образом, функция f_d^i для дисков разных уровней будет отличаться. В рассматриваемой системе хранения приложению выделена только часть s_j^i каждого диска, т. е. $s_j^i < S_j$. Перепишем уравнения (5)–(7) другим образом:

$$s_1^i = {F_{pop}^i}^{-1}\left(p_1^i\left(1 - p_0^i\right)\right); \quad (8)$$

$$s_2^i + s_1^i = {F_{pop}^i}^{-1}\left(\left(p_2^i + p_1^i\right)\left(1 - p_0^i\right)\right); \quad (9)$$

$$s_3^i + s_2^i + s_1^i = {F_{pop}^i}^{-1}\left(\left(p_3^i + p_2^i + p_1^i\right)\left(1 - p_0^i\right)\right). \quad (10)$$

При этом на вероятности p_j^i имеется ограничение (4), а на s_j^i — ограничение (1).

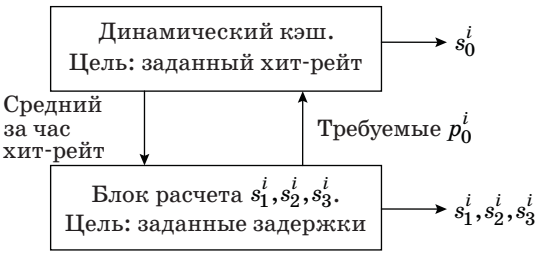
Все необходимые зависимости определены, перейдем к постановке задачи. Пусть известны параметры S_0, S_1, S_2, S_3, t_0, t_1, t_2, t_3, n, $R_{i=1,\dots,n}$, требуется найти такие s_j^i, чтобы выполнялись заданные ограничения $R_{i=1,\dots,n}$.

Кроме заданных параметров, в выражениях (8)–(10) присутствуют функции $F_{sd}^i(x)$ и $F_{pop}^i(x)$, которые меняются в зависимости от потока запросов, и для реальной системы их можно получить только численно, они задаются таблицей. Это приводит к тому, что известные методы решения задач оптимизации в данном случае не применимы. По этой причине нами предлагается эвристический алгоритм решения задачи.

Описание предлагаемого решения

Функционально предлагается заменить исходный кэш со статичным разделением областей между приложениями на описанный в работе [6] динамический кэш, который меняет границы областей приложений в зависимости от заданного для каждого приложения хит-рейта. В настоящей работе вводится дополнительный блок (рис. 3), который раз в час будет пересчитывать для каждого приложения размеры областей, выделяемых системой на дисках разного типа.

Значения s_0^i вычисляются динамическим кэшем [6], чтобы достичь хит-рейтов, заданных для



■ ***Рис. 3.*** Функциональная схема предлагаемого решения

■ Возможные события в результате перераспределения памяти

№	Условие	Действие
1	$\overline{T}_x > R_x, \overline{T}_y < \alpha R_y$	Продолжать увеличивать s_1^x и уменьшать s_1^y
2	$\overline{T}_x < R_x, \overline{T}_y < \alpha R_y$	Остановить перераспределение s_1^i
3	$\overline{T}_y > \alpha R_y$	Сделать шаг назад, т. е. уменьшить s_1^x и увеличить s_1^y

приложений. В предлагаемой схеме хит-рейты задает блок расчета s_1^i, s_2^i, s_3^i. В начале работы системы размеры s_0^i задаются некоторым типовым образом, например, одинаковыми или пропорционально требованиям на отклик системы.

Рассмотрим алгоритм расчета s_1^i, s_2^i, s_3^i. Как упомянуто выше, алгоритм вычисляет границы областей один раз в час. За час собирается статистика, вычисляется $F_{pop}^i(x)$, и вместе с текущими значениями границ по выражениям (5)–(7) могут быть рассчитаны вероятности p_1^i, p_2^i, p_3^i. Используя эти вероятности и p_0^i, которую можно посчитать в кэше, с помощью выражения (2) вычисляем $\overline{T}_i$. Далее все $\overline{T}_i$ разделим на два списка. В первый список попадут $\overline{T}_i \geq R_i$, а во второй список — $\overline{T}_i < \delta R_i$, где δ характеризует необходимый запас по требуемой задержке и представляет собой долю от R_i. В первый список $T1$ попадут задержки тех приложений, для которых требование R_i не выполнилось, а во второй список $T2$ — задержки тех, для которых они выполнились с избытком. Если эти списки не пустые, то далее задача алгоритма состоит в таком изменении s_1^i, s_2^i, s_3^i, при котором для всех приложений требования R_i станут удовлетворяться. Для этого сначала списки $T1$ и $T2$ сортируются по возрастанию. Далее для приложений, чьи $\overline{T}_i$ оказались в начале списков, происходит изменение размеров областей на дисках. Изменения производятся пошагово, по одному слоту, начиная с самой быстрой памяти. Сначала s_1^y приложения с избытком уменьшается на один слот, а s_1^x приложения с недостатком увеличивается на один слот. После этого по формуле (2) снова пересчитываются $\overline{T}_i$ для двух приложений. В таблице представлены различные варианты значений $\overline{T}_i$ и дальнейшие шаги.

Параметр α необходим для того, чтобы приложение с запасом по $\overline{T}_i$ не стало приложением, не удовлетворяющим требованию. Если выполнилось условие 2, то дальше выбирается следующая пара приложений с недостатком и избытком $\overline{T}_i$, причем приложение с избытком может остаться тем же, что было на предыдущем шаге. Если выполнилось условие 3, то резерв быстрой памяти у приложения с избытком закончился. В этом случае после шага назад приложение с недостатком все еще не удовлетворяет требованию. Далее необходимо повторить процедуру перераспределения для следующего уровня памяти s_2^i. Если и в нем будет такой же результат, то выбирается следующее приложение с избытком, и процедура перераспределения памяти повторяется с уровня s_1^i. Далее либо закончатся все приложения с недостатком быстрой памяти, что означает успешное завершение алгоритма, либо закончатся приложения с избытком, а с недостатком останутся. В последнем случае система не сможет удовлетворить текущим требованиям на задержку.

Работа алгоритма основана на предположении, что во время функционирования системы присутствуют достаточно длинные периоды стабильности $F_{pop}^i(x)$, тогда, собрав за определенное время статистику, можно рассчитывать на то, что новые значения s_j^i будут актуальны.

Для того чтобы быстрее реагировать на изменения потока запросов, в алгоритме предусмотрена процедура пересчета требуемого хит-рейта. Она работает каждые 10 мин, в то время, когда пересчет s_1^i, s_2^i, s_3^i не производится. По текущим значениям s_1^i, s_2^i, s_3^i с помощью формул (5)–(7) вычисляются p_1^i, p_2^i, p_3^i. Далее из этих формул можно выразить вероятность p_0^i, которая и является требуемым хит-рейтом. Найденное значение передается в блок управления кэшем, который должен в результате работы выдерживать заданные p_0^i.

Результаты моделирования

Определим коэффициент Q, по которому предложенный алгоритм будет оцениваться и сравниваться с системой без пересчета размеров областей на дисках. Для этого введем вспомогательный коэффициент

$$q_i = \frac{N\{\overline{T}_i \leq R_i\}}{N}, \qquad (11)$$

где $N\{\overline{T}_i \leq R_i\}$ — количество десятиминутных интервалов, в течение которых среднее время считывания блока данных не превышало заданное для приложения i, а N — общее количество

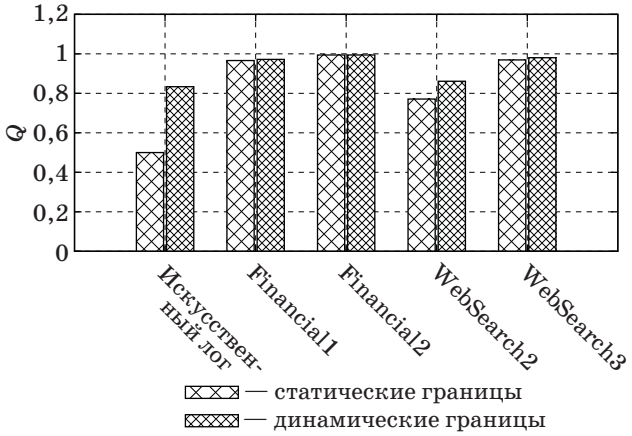
десятиминутных интервалов работы системы. Тогда

$$Q = \frac{1}{n}\sum_{i=1}^{n} q_i, \qquad (12)$$

откуда видно, что Q характеризует работу системы в целом по всем приложениям.

Для вычисления коэффициента Q воспользуемся имитационным моделированием. В качестве модели кэша будем использовать LRU (*Least Recently Used*) стек [9]. В случае с постоянными размерами областей памяти в кэше будем использовать несколько LRU-стеков фиксированного размера. В случае с изменяемыми размерами областей размеры будут пересчитываться [6]. В качестве модели дисков будем использовать таблицу, в которой записано, к диску какого типа относится каждый блок данных. Каждый час блоки будут перемещаться между дисками разного типа согласно $F^i_{pop}(x)$, как описано выше. В качестве потока заявок использовались логи реальных систем хранения [10]. На рис. 4 представлены пары коэффициентов Q для одного синтезированного лога и для нескольких общедоступных логов реальных систем хранения. Первый коэффициент получен для системы с размерами s^i_j, выставленными пропорционально требованиям на задержку (статические границы). Второй коэффициент получен при работе системы с предложенным алгоритмом пересчета s^i_j (динамические границы).

Логи Financial1, Financial2 сняты с систем обработки транзакций в реальном времени в финансовых организациях. Логи WebSearch2, WebSearch3 собраны с известных поисковых систем [10]. Из рисунка видно, что наибольший выигрыш получается на синтезированном логе. Как было отмечено, предлагаемый алгоритм опирается на предположение о наличии достаточно длинных периодов времени, на которых функция $F^i_{pop}(x)$ постоянна. В синтезированном логе это условие строго выполняется, поэтому выигрыш наибольший. В реальности поведение $F^i_{pop}(x)$ непредсказуемо. Кроме предсказуемости функции популярности для успеха алгоритма необходимо, чтобы приложениям требовались разные объемы памяти для выполнения заданных ограничений и чтобы эти объемы достаточно отличались от выделенного изначально объема. На логах



■ ***Рис. 4***. Результаты моделирования предложенного алгоритма

Financial1 и WebSearch3 наблюдается незначительный выигрыш предложенного алгоритма. На логе Financial2 разницы нет. Это связано с тем, что свободных для перераспределения ресурсов не было и выдерживались изначально выставленные границы. На логе WebSearch2 выигрыш предложенного алгоритма составляет 12 %. Можно также отметить, что благодаря введенным защитным параметрам α и δ, на всех рассмотренных логах критерий Q не стал хуже по сравнению с исходной системой (со статическими границами). Также были исследованы логи и других систем. На них выигрыш составил 1–10 %.

Заключение

Работа предложенного в статье алгоритма перераспределения места на дисках основана на предположении о стабильности распределения популярностей адресов. Если это выполняется, то алгоритм дает ощутимое преимущество, что показано с помощью моделирования на искусственном потоке запросов адресов. При использовании потока с реальных систем данное предположение может не выполняться, однако моделирование показало, что довольно часто и на реальных потоках запросов алгоритм дает выигрыш до 10 %. Это позволяет сделать вывод, что алгоритм можно использовать как помощника при конфигурировании системы хранения.

Литература

1. **Morenoff E., McLean J.** Application of Level Changing to a Multilevel Storage Organization // Communications of the ACM. Mar. 1967. Vol. 10. Iss. 3. P. 149–154. doi:10.1145/363162.363183
2. **Ver Hoef E. W.** Design of a Multi-Level File Management System // Proc. of the 1966 21st National Conf. 1966. P. 75–86. doi: 10.1145/800256.810684
3. **Chow C. K.** Determination of Cache's Capacity and its Matching Storage Hierarchy // IEEE Transactions on Computers. Feb. 1976. Vol. C-25. Iss. 2. P. 157–164. doi: 10.1109/TC.1976.5009230
4. **Lai P., Fan R.** Makespan-Optimal Cache Partitioning // IEEE 21st Intern. Symp. on Modeling, Analysis

& Simulation of Computer and Telecommunication Systems (MASCOTS). 14–16 Aug. 2013. P. 202–211. doi: 10.1109/MASCOTS.2013.28
5. **Yuan L.** A Locality-based Performance Model for Load-and-Compute Style Computation // IEEE Intern. Conf. on Cluster Computing (CLUSTER). 24–28 Sept. 2012. P. 566–571. doi: 10.1109/CLUSTER.2012.25
6. **Дужин В. С.** Алгоритм перераспределения кэша между приложениями в системе хранения данных // Научная сессия ГУАП: сб. докл.: в 3 ч. Ч. I. Технические науки. СПб.: ГУАП, 2012. С. 73–75.
7. **Liu X., Salem K.** Hybrid Storage Management for Database Systems // Proc. of the VLDB Endowment. June 2013. Vol. 6. Iss. 8. P. 541–552. doi: 10.14778/2536354.2536355
8. **Canim M.,** at al. An Object Placement Advisor for DB2 Using Solid State Storage // Proc. of the VLDB Endowment. Aug. 2009. Vol. 2. Iss. 2. P. 1318–1329.
9. **Feitelson D. G.** Workload Modeling for Computer Systems Performance Evaluation. — NY.: Cambridge University Press, 2015. — 564 p.
10. **UMass Trace Repository.** http://traces.cs.umass.edu/index.php/Storage/Storage (дата обращения: 10.09.2015).

UDC 681.324
doi:10.15217/issn1684-8853.2015.5.100

Heuristic Algorithm for Calculating the Size of Tiers in a Hierarchical Storage System

Malichenko D. A.[a], Senior Programmer, dml@vu.spb.ru
[a]Saint-Petersburg State University of Aerospace Instrumentations, 67, B. Morskaia St., 190000, Saint-Petersburg, Russian Federation

Introduction: A hierarchical storage system which serves many applications is discussed. Levels in the hierarchy differ in their memory type and response time. For each application, a certain quality of the service is assigned in terms of latency. The allocation of the address space among applications and memory levels affects the response time of the system. The purpose of the work is automatic memory allocation at every level taking into account the request flow characteristics in order to meet the service quality demands. **Results:** A memory partitioning optimization problem is formulated, and a heuristic algorithm is proposed for its solution. The cache hit rate and distribution function of popularity of the requested memory blocks are used as flow characteristics. An assessment of the storage system efficiency is calculated by simulating a storage system fed by a request flow saved from real storage systems. The calculated assessments show up to 10% gain for the proposed algorithm. **Practical relevance:** The proposed algorithm can improve the service quality in a hierarchical storage system.

Keywords — Hierarchical Storage System, Hierarchical Storage Management, Storage Area Network, Quality of Service, Simulation.

References

1. Morenoff E., McLean J. Application of Level Changing to a Multilevel Storage Organization. *Communications of the ACM*, March 1967, vol. 10, iss. 3, pp. 149–154. doi:10.1145.363162.363183
2. Ver Hoef E. W. Design of a Multi-Level File Management System. *Proc. of the 1966 21st National Conf.*, 1966, pp. 75–86. doi: 10.1145/800256.810684
3. Chow C. K. Determination of Cache's Capacity and its Matching Storage Hierarchy. *IEEE Transactions on Computers*, February 1976, vol. C-25, iss. 2, pp. 157–164. doi: 10.1109/TC.1976.5009230
4. Lai P., Fan R. Makespan-Optimal Cache Partitioning. *IEEE 21st Intern. Symp. on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 14–16 August 2013, pp. 202–211. doi: 10.1109/MASCOTS.2013.28
5. Yuan L. A Locality-based Performance Model for Load-and-Compute Style Computation. *IEEE Intern. Conf. on Cluster Computing (CLUSTER)*, 24–28 September 2012, pp. 566–571. doi: 10.1109/CLUSTER.2012.25
6. Duzhin V. S. Cache Partitioning Algorithm in a Storage System. *Nauchnaia sessiia GUAP* [Proceedings of Scientific Session of SUAI]. Saint-Petersburg, GUAP Publ., 2012, vol. III "Technical science", pp. 73–75 (In Russian).
7. Liu X., Salem K. Hybrid Storage Management for Database Systems. *Proc. of the VLDB Endowment*, June 2013, vol. 6, iss. 8, pp. 541–552. doi: 10.14778/2536354.2536355
8. Canim M., Mihaila G. A., Bhattacharjee B., Ross K. A., Lang C. A. An Object Placement Advisor for DB2 Using Solid State Storage. *Proc. of the VLDB Endowment*, August 2009, vol. 2, iss. 2, pp. 1318–1329.
9. Feitelson D. G. *Workload Modeling for Computer Systems Performance Evaluation*. New York, Cambridge University Press, 2015. 564 p.
10. *UMass Trace Repository*. Available at: http://traces.cs.umass.edu/index.php/Storage/Storage (accessed 10 September 2015).