УДК 004.043 DOI: 10.17587/it.25.216-222

Д. Р. Потапов, аспирант, potapovd36@gmail.com,
Воронежский государственный университет, Воронеж

Исследование эффективности применения кэша для использования в самоадаптирующихся контейнерах данных

Проведено исследование зависимости эффективности кэша от среднеквадратичного отклонения (сигмы) и соотношения скоростей хранилищ для нормального распределения. Кроме того, выявлены основные закономерности для определения оптимального размера кэша для реализации самоадаптирующихся контейнеров данных. Исследование проведено для различных структур данных языка c# (Dictionary, Sorted Dictionary и Sorted List) и различных алгоритмов вытеснения кэша (Least Recently Used и Most Recently Used), эффективность оценивалась по времени поиска в самоадаптирующемся контейнере данных.

Ключевые слова: *хранение данных, эффективность кэша, оптимальное хранилище данных, нормальное распределение, самоадаптирующийся контейнер данных, оптимальный размер кэша, нагрузка на контейнер данных*

Введение

Самоадаптирующийся контейнер данных [1] — это ассоциативная структура данных, которая меняет логику своей работы в зависимости от нагрузки [2—5]. Нагрузка представляет собой различные последовательности операций вставки, выборки и удаления элементов из хранилища данных. Самоадаптирующиеся контейнеры данных могут быть реализованы путем использования различных структур данных в зависимости от нагрузки и условий работы контейнера [6—8]. Однако наиболее эффективно такие контейнеры могут проявить себя в условиях большого объема данных. В таком случае часть данных хранится в более быстрой памяти, а другая часть — в более медленной памяти или на удаленном источнике. Для приложений, где данные статичны или число запросов на получение данных намного превышает число запросов на добавление элементов в хранилище данных, наиболее оптимальным будет использование кэша [9].

Кэш представляет собой набор записей "ключ—значение", расположенный в памяти с большей скоростью доступа, предназначенный для ускорения обращения к данным, расположенным в хранилище данных с меньшей скоростью доступа. Кэширование применяется как на аппаратном уровне [9] (в процессорах, жестких дисках), так и на программном уровне [9] (в операционных системах, на сетевом уровне (в сетях доставки контента (CDN), системах DNS, сессиях), интернет-приложениях, базах данных, мобильных приложениях, сервисах кэширования в оперативной памяти (Redis [10], Memcached [11, 12])). Помимо выигрыша в производительности при чтении данных кэш может использоваться и для ускорения записи (отложенная запись [13]), однако в данной статье рассматривается использование кэша только для чтения.

Самоадаптирующийся контейнер данных, использующий программный кэш, был реализован с использованием архитектурного шаблона "кэш на стороне" (cache-aside) [14]. При данной архитектуре кэш располагается в оперативной памяти, а основное хранилище — на жестком диске или на удаленном источнике (рис. 1), что позволяет программно изменять размеры кэша в процессе работы, чтобы обеспечить максимальную эффективность. Кэш и основное хранилище являются ассоциативными структурами данных, которые хранят пары "ключ—значение" (<Tkey, Tvalue>).

Важнейшим фактором, который определяет нагрузку на контейнер, является набор запрашиваемых ключей, который в общем случае

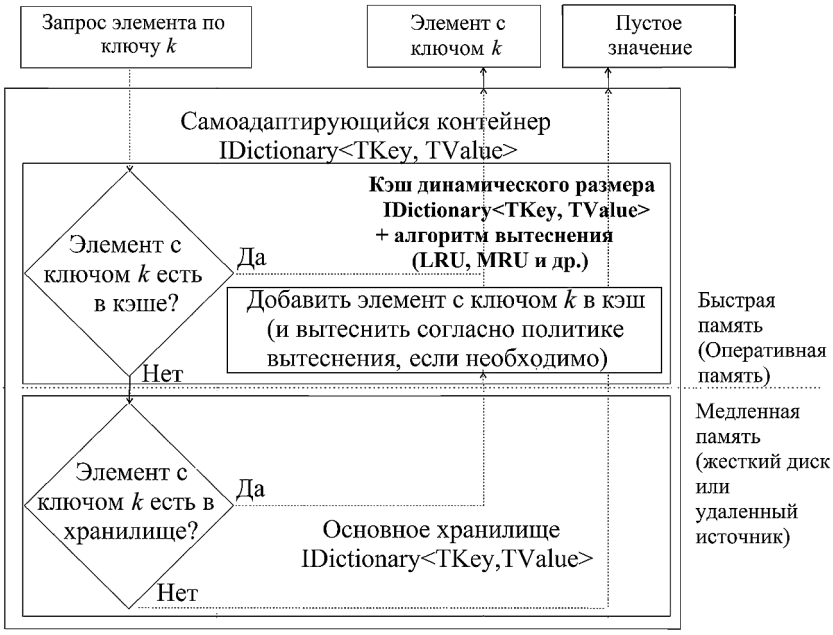

Рис. 1. Общая схема устройства самоадаптирующегося контейнера данных с использованием кэша

может быть описан с помощью нормального распределения. Таким образом, параметр этого распределения — среднеквадратичное отклонение (σ) [15] может влиять на размер кэша. Также необходимо учитывать такой фактор, как соотношение скоростей доступа к кэшу и основному хранилищу.

Постановка задачи

В данной статье проведено исследование зависимости эффективности кэша от среднеквадратичного отклонения и соотношения скоростей хранилищ для нормального распределения. Целью исследования является выявление основных закономерностей, формул и табличных значений для определения оптимального размера кэша в зависимости от перечисленных выше параметров, которые могут быть использованы для автоматического изменения размера кэша в процессе работы самоадаптирующегося контейнера данных.

Описание метода решения задачи

Для решения поставленной задачи было проведено тестирование. Для этого было реализовано консольное приложение языка C#, которое использует:

1) библиотеку Troschuetz.Random для генерации ключей по закону нормального распределения;

2) класс Stopwatch [16] языка C# для подсчета времени выполнения (в тактах процессора) функции поиска в контейнере;

3) программу Excel 2013 [17] для визуализации полученных численных результатов.

Тестирование с помощью данного приложения проводили на следующей программно-аппаратной платформе:

1. Processor: Intel Core i7 6500U @ 2.50 GHz (2 cores).
2. RAM: 16 GB.
3. OS: Windows 10.
4. MS Visual Studio 2015.

В процессе исследования был проведен ряд тестов, параметры которых были заданы следующим образом:

1) элементом кэша и основного хранилища является пара "ключ—значение" (<string, int>), где ключом является строка языка C# (string) длиной в 10 символов, значением выступает случайное число типа int языка C#;

2) основное хранилище содержит 10^5 элементов, которые генерируются случайным образом;

3) запрашиваются 10^7 элементов по ключам, которые генерируются следующим образом:

a) для заданного среднеквадратичного отклонения σ согласно закону нормального распределения $N(0, \sigma^2)$ генерируются целые числа;

b) полученные числа трансформируются в строки длиной в 10 символов путем добавления нулей слева для недостающих символов. Например, для целого числа 1256 строка будет сформирована в виде "0000001256", а для числа 965169 — в виде "0000965169".

Таким образом, параметр σ показывает, насколько большой имеется разброс значений в нагрузке, т. е. чем больше σ, тем меньше вероятность повторного запроса ключа, чем меньше σ, тем больше вероятность повторного запроса;

4) в ходе эксперимента основное хранилище эмулировалось с помощью ассоциативной структуры данных языка C# Dictionary<string, int>, расположенной в оперативной памяти. Таким образом, соотношение скорости кэша и основного хранилища (далее обозначается k) для эксперимента равно единице ($k = 1$);

5) для каждого теста отдельно вычисляется время, затраченное на поиск в кэше ($t_{\text{кэш}}$) и поиск в основном хранилище ($t_{\text{осн. хранилище}}$).

Один тест представляет собой комбинацию из нескольких параметров:

1. Размер кэша. Рассматривались значения 10, 9100, 18 190, 27 280, 36 370, 45 460, 54 550, 63 640, 72 730, 81 820, 90 910, 100 000, 500 000, 1 000 000, 2 000 000 элементов в кэше.

2. Среднеквадратичное отклонение набора запрашиваемых ключей $\sigma = \{10, 50, 100, 500, 1000, 5000, 10\,000, 15\,000, 20\,000\}$ при математическом ожидании $\mu = 0$ для нормального распределения $N(\mu, \sigma^2)$.

3. Используемый алгоритм вытеснения: LRU (Least Recently Used) [18, 19] и MRU (Most Recently Used) [20]. Данное исследование направлено на нахождение общих зависимостей, независимо от конкретных алгоритмов вытеснения. Поэтому для тестирования были выбраны алгоритмы, имеющие принципиально противоположные стратегии вытеснения (LRU вытесняет неиспользованный дольше всех элемент, а MRU — последний использованный), а результаты тестов были усреднены.

4. Ассоциативная структура данных для кэша, реализованная в языке C# (Dictionary <string, int>, SortedDictionary<string, int> и SortedList<string, int>) [21].

Было проведено по 20 тестов для каждой из комбинаций параметров и вычислено среднее арифметическое время поиска в кэше и основном хранилище по всем комбинациям и таким образом получено среднее время $t_{\text{ср. кэш}}$ и $t^{k=1}_{\text{ср. осн. хр}}$ для всех пар (σ, размер кэша) при $k = 1$ (так как во время эксперимента основное хранилище расположено в оперативной памяти, а следовательно, соотношение скоростей равно единице).

Вычислительный эксперимент и анализ

Используя полученные в результате эксперимента данные по среднему времени поиска в кэше и в основном хранилище, расположенном в оперативной памяти, можно вычислить общее время работы самоадаптирующегося контейнера для ситуаций, когда основное хранилище содержит достаточно много элементов и располагается на жестком диске или удаленном источнике. В этом случае соотношение скорости кэша и основного хранилища будет больше единицы ($k > 1$), так как скорость самых современных SSD дисков в разы меньше скорости оперативной памяти, а для HDD дисков и удаленных источников достигает сотен тысяч раз и более (в зависимости от конфигурации компьютера и скорости доступа к удаленному источнику). Таким образом, среднее время поиска в основном хранилище может быть вычислено как $kt^{k=1}_{\text{ср. осн. хр}}$ (так как время поиска на диске или удаленном источнике в k раз больше, чем время поиска в оперативной памяти), а среднее общее время работы контейнера для всех пар (σ, размер кэша) может быть получено по формуле

$$t_{\text{ср. общ}} = t_{\text{ср. кэш}} + kt^{k=1}_{\text{ср. осн. хр}}, \qquad (1)$$

где k — необходимое соотношение скорости кэша и основного хранилища; $t_{\text{ср. кэш}}$ — среднее время поиска (в тактах процессора) в кэше, расположенном в оперативной памяти; $t^{k=1}_{\text{ср. осн. хр}}$ — среднее время поиска (в тактах процессора) в основном хранилище, расположенном в оперативной памяти (при $k = 1$).

С использованием формулы (1) и средств Excel были построены графики зависимости времени работы контейнера от сигмы и размера кэша для различных значений k. В результате анализа этих графиков было выявлено, что при указанных выше условиях и входных данных можно выделить три различные по характеру графиков группы в зависимости от k:

1) $k < 8$ (кэш ненамного быстрее основного хранилища). В этом случае наиболее оптимальным решением будет не использовать кэш вообще, поскольку время выполнения без него меньше, чем при любых размерах кэша. Это связано с тем, что при небольших значениях k время, затраченное на поиск в кэше, больше, чем выгода от его использования. На рис. 2 и 3 (см. вторую сторону обложки) наглядно показано что для любой σ минимальные значения достигаются при нулевом размере кэша. В табл. 1 представлены результаты тестов для $k = 5$;

2) $k > 14$ (кэш намного быстрее основного хранилища). В этом случае оптимальным размером кэша является примерно 8σ (согласно математической статистике в диапазоне $(-4\sigma; 4\sigma)$ располагаются 99,997 % всех значений для нормального распределения) [22], т. е. практически все элементы, участвующие в выборке, помещаются в кэш, и затраты на поиск в таком кэше окупаются. Больше этого значения увеличение кэша не дает ощутимого прироста в производительности. На рис. 4 и 5 (см. вторую сторону обложки) можно заметить, что время поиска достигает минимума для каждой σ при достижении размера кэша примерно 8σ, после чего меняется незначительно. В табл. 2 представлены результаты эксперимента для $k = 20$.

Таблица 1

Время поиска в зависимости от σ и размера кэша для $k = 5$

Размер кэша	σ								
	10	50	100	500	1000	5000	10000	15000	20000
0	11008775	11717288	12987867	21848120	33667977	61566760	68428464	69680182	70807284
10	26940705	33890779	40835969	68498983	81871135	99303146	102834053	103707956	103845549
9100	23845447	31277860	35182731	51799736	65341135	166050254	184057853	191102304	194483186
18190	23840746	31314295	35246913	51764485	65095889	158558086	189196597	203905118	209896699
27280	23923557	31357142	35175279	51649246	65459080	136677576	187540742	207563583	217477651
36370	23852812	31345004	35122799	51525828	65150845	115482041	186774839	208546705	221866535
45460	23830789	31370141	35141213	51487477	65000654	109702657	172906803	208553253	222735013
54550	23879114	31352730	35219454	51850471	65122474	109123617	159923849	207654561	224457537
63640	23846348	31325336	35178629	51432401	65344654	109195183	147747251	197333128	223470878
72730	23856182	31365298	35156191	51602573	64936630	109068561	129887982	183386692	223257982
81820	23931219	31407959	35134752	51612698	64890774	109982322	129943863	173610780	215949778
90910	23824473	31292224	35211458	51477154	64830806	109140683	129291463	162604933	201759919
100000	23843379	31376769	35201755	51553538	64965822	108824473	129628560	147536611	191798905
500000	23872934	31377701	35154747	51678465	65384576	109447917	128939390	141179342	150550711
1000000	23840667	31339943	35078915	51574599	65066264	109459951	129133388	140370802	149799327
2000000	23836529	31327811	35159711	51846836	64963718	108802369	128970369	140220465	149080165

Таблица 2

Время поиска в зависимости от σ и размера кэша для $k = 20$

Размер кэша	σ								
	10	50	100	500	1000	5000	10000	15000	20000
0	44035099	46869151	51951467	87392480	134671909	246267041	273713854	278720729	283229134
10	61378441	88280668	115161873	224040782	276603584	345967157	359823183	363293504	364105260
9100	23853405	31312333	35246751	52067457	65825853	367023041	441143427	468451518	481804017
18190	23848658	31348402	35310551	52030195	65582426	268362245	399521701	455330614	479776381
27280	23931456	31391515	35238668	51913748	65941924	184213669	349734798	428521244	468664433
36370	23860900	31379294	35185284	51791098	65633320	120602442	300075423	394017569	448067855
45460	23838958	31404270	35204942	51752677	65486242	111741472	246978712	359526265	422497948
54550	23887135	31386604	35283781	52119815	65607091	111159454	204013413	324071662	397874795
63640	23854226	31359105	35242288	51700179	65836834	111223692	166663594	284596358	369389281
72730	23864142	31399760	35219158	51868719	65421600	111099971	133618742	246133997	342868562
81820	23939240	31441928	35197994	51879990	65371615	112033657	133690887	215972201	312121471
90910	23832615	31325819	35274286	51743793	65314687	111181060	133029939	186473018	277001355
100000	23851499	31410873	35265944	51819655	65451104	110867690	133388158	156539510	249907663
500000	23880920	31411112	35216670	51945440	65872213	111491062	132668858	146516036	157480155

При увеличении k увеличивается разность между временем поиска без использования кэша и временем поиска с использованием оптимального размера кэша, т. е. эффективность использования увеличивается с ростом k.

Однако не всегда есть возможность создать кэш необходимого размера. В этом случае оптимальным решением может оказаться не максимально доступный размер кэша, а отказ от его использования. Это связано с тем, что затраты на поиск в кэше могут быть больше, чем прирост производительности от его использования. На рис. 6 (см. вторую сторону обложки) изображена зависимость времени поиска от размера кэша для различных σ, где наглядно виден рост времени поиска при использовании недостаточного кэша и плавное уменьшение при увеличении доступного размера кэша. На основании проведенных экспериментов можно выделить два случая оптимального размера кэша.

1. Если доступный размер кэша больше определенного значения (но меньше 8σ) (далее $x_{опт}$), то оптимальный размер кэша равен максимально доступному значению.

2. При доступном размере кэша меньше $x_{опт}$ оптимальный размер кэша равен 0.

При этом $x_{опт}$ аппроксимировано может быть вычислено для определенного k и σ (или близких значений из таблиц) по формуле

$$x_{опт} = \frac{\left(y_0 - y_1 + \frac{(y_2 - y_1)}{(x_2 - x_1)} \cdot x_1\right)}{\frac{(y_2 - y_1)}{(x_2 - x_1)}},$$

где y_0 — время в тактах при нулевом размере кэша; y_1, y_2 ($y_2 > y_0 > y_1$) — время в тактах на концах отрезка $(x_1; x_2)$; x_1 и x_2 — размеры кэша, следующие друг за другом в таблице.

3) $14 > k > 8$. При данном соотношении скоростей оптимальным может оказаться как решение из пункта 1, так и из пункта 2 (в зависимости от k и σ, табл. 3). На рис. 7 (см. вторую сторону обложки) наглядно показано, что оптимальным размером кэша (минимальным значением времени поиска) может быть как нулевой размер, так и максимально доступный размер кэша.

Таблица 3

Время поиска в зависимости от σ и размера кэша для $k = 11$

Размер кэша	σ								
	10	50	100	500	1000	5000	10000	15000	20000
0	24219304	25778033	28573307	48065864	74069550	135446873	150542620	153296401	155776024
10	40715800	55646735	70566331	130715703	159764115	197968750	205629705	207542175	207949433
9100	23848631	31291649	35208339	51906824	65535022	246439369	286892083	302041989	309411519
18190	23843911	31327938	35272368	51870769	65290504	202479750	273326639	304475316	317848572
27280	23926716	31370891	35200634	51755047	65652218	155692013	252418364	295946648	317952364
36370	23856047	31358720	35147793	51631936	65343835	117530201	232095073	282735051	312347063
45460	23834057	31383792	35166705	51593557	65194889	110518183	202535567	268942458	302640187
54550	23882322	31366279	35245185	51958209	65316321	109937952	177559674	254221401	293824440
63640	23849499	31338844	35204092	51539512	65541526	110006587	155313788	232238420	281838239
72730	23859366	31379083	35181378	51709031	65130618	109881125	131380286	208485614	271102214
81820	23934427	31421546	35160049	51719615	65083111	110802856	131442673	190555348	254418455
90910	23827730	31305662	35236590	51583809	65024358	109956834	130786853	172152167	231856493
100000	23846627	31390410	35227431	51659985	65159935	109641760	131132399	151137771	215042408
500000	23876129	31391065	35179516	51785255	65579630	110265175	130431177	143314019	153322489
1000000	23843821	31353546	35104710	51682270	65259442	110281483	130629389	142494953	152557330
2000000	23839726	31341438	35184410	51954870	65157271	109614881	130472266	142340488	151810443

Заключение

В данной работе представлен результат исследования эффективности применения кэша в зависимости от распределения данных (σ) и соотношения скорости основного хранилища и кэша. В результате анализа полученных данных были выявлены три группы закономерностей в зависимости от соотношения скоростей. В первой группе использование кэша не целесообразно, во второй группе оптимальный размер кэша составил 8σ (или может быть вычислен по указанным в статье формулам в случае недостатка памяти), в третьей группе оптимальное решение может быть получено из таблиц. Полученные закономерности могут быть использованы для динамического изменения размера кэша в зависимости от параметров нагрузки при реализации самоадаптирующегося контейнера данных.

Список литературы

1. **Потапов Д. Р., Артемов М. А., Барановский Е. С.** Обзор условий адаптации самоадаптирующихся ассоциативных контейнеров данных // Вестник ВГУ. Серия "Системный анализ и информационные технологии". Воронеж, 2017. № 1. С. 112—119.

2. **Зобов В. В., Селезнев К. Е.** Инструмент для моделирования нагрузки на контейнеры данных // Матер. 14-й науч.-метод. конф. "Информатика: проблемы, методология, технологии", 10—11 февраля 2011 г. Воронеж, 2014. Т. 3. С. 154—161.

3. **Потапов Д. Р., Селезнев К. Е.** Визуальное представление результатов анализа и сравнения контейнеров данных // Информатика: проблемы, методология, технологии. Матер. XVI Междунар. науч.-метод. конф. 2016. С. 123—128.

4. **Потапов Д. Р.** Визуальное сравнение и анализ контейнеров данных // Математическое и компьютерное моделирование, информационные технологии управления (МКМИТУ-2016). Воронеж. 2016. С. 178—181.

5. **Потапов Д. Р.** Разработка и реализация метода для анализа и сравнения контейнеров данных // Вестник Факультета прикладной математики и механики. 2016. № 5. С. 30—36.

6. **Потапов Д. Р.** Анализ применения многомерных структур данных в самоадаптирующихся контейнерах данных // Междунар. науч.-техн. конф. "Актуальные проблемы прикладной математики, информатики и механики". Воронеж, 18—20 декабря 2017 г. С. 436—442.

7. **Потапов Д. Р.** Обзор методов построения многомерных контейнеров данных "ключ—значение" для использования в самоадаптирующихся контейнерах данных // Прикладная информатика. 2018. № 2 (74). С. 69—82.

8. **Потапов Д. Р., Артемов М. А., Барановский Е. С., Селезнев К. Е.** Обзор методов построения контейнеров данных "ключ—значение" для использования в самоадаптирующихся контейнерах данных // Кибернетика и программирование. 2017. № 5.

9. **Jacob B., Ng S., Wang D.** Memory Systems: Cache, DRAM, Disk. San Francisco: Morgan Kaufmann Publishers, 2007. 900 p.

10. **Carlson J. L.** Redis in Action. NY: Manning Publications, 2013. 320 p.

11. **Sanderson D.** Programming Google App Engine with Python. Sebastopol, CA: O'Reilly Media, 2015. 538 p.

12. **Soliman A.** Getting Started with Memcached. Birmingham: Packt Publishing, 2013. 56 p.

13. **Таненбаум Э.** Архитектура компьютера. СПб.: Питер, 2003. 704 с.

14. **Cache-Aside** pattern. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/cache-aside

15. **Королюк В. С., Портенко Н. И., Скороход А. В., Турбин А. Ф.** Справочник по теории вероятностей и математической статистике. М.: Наука, 1985. 640 с.

16. **Deitel P., Deitel H.** C# 6 for Programmers (6th Edition) (Deitel Developer Series). Prentice Hall, 2016. 768 p.

17. **Walkenbach J.** Microsoft Excel 2016 bible: the comprehensive tutorial resource. John Wiley & Sons, 2015. 1152 p.

18. **Megiddo N., Modha D. S.** Outperforming LRU with an Adaptive Replacement Cache Algorithm // Computer, April 2004. Vol. 37, N. 4. P. 58—65.

19. **O'Neil E. J., O'Neil P. E., Weikum G.** The LRU-K Page Replacement Algorithm for Database Disk Buffering // Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. SIGMOD '93. New York, NY, USA: ACM, 1993. P. 297—306.

20. **Chou H.-T., DeWitt D. J.** An evaluation of buffer management strategies for relational database systems // Proceeding VLDB '85 Proceedings of the 11th international conference on Very Large Data Bases.1985. Vol. 11. P. 127—141.

21. **Troelsen A., Japikse P.** C# 6.0 and the. NET 4.6 Framework. New York, NY, USA: Apress, 2015. 1832 p.

22. **Casella G., Berger R. L.** Statistical Inference. Duxbury Resource Center, 2001. 688 p.

D. R. Potapov, Postgraduate, e-mail: potapovd36@gmail.com,
Voronezh State University, Voronezh

Cache Efficiency Research for Using in Adaptive Data Storage

There are a lot of applications with a large amount of the static data or data which is using for reading mostly. At this rate cache applying improves performance greatly. In this case an adaptive data storage can change cache size during execution to achieve maximum efficiency. There are two main factors which affect container performance. The first one is a set of requesting data, which in common case can be described as Gaussian distribution. The second is a difference between speed of a main container and the cache. This paper provides a research results of a cache efficiency depending on the standard deviation of normal distribution and storage speed coefficient. The survey takes into consideration combinations of several parameters. One of

such parameters is a data structure. The research is carried out for different programming language C# data structures (Dictionary, SortedDictionary and SortedList). Another important parameter is a cache replacement policy. Various cache algorithms (Least Recently Used and Most Recently Used) are also examined in this research. Adaptive storage performance is evaluated as sum of a search time in the main container and a search time in the cache. The research results are analyzed in the article and the main patterns for calculating the optimal cache size for using in adaptive data storage are identified.

Keywords: *store the data, cache efficiency, optimal data storage, normal distribution, adaptive data container, optimal cache size, data structures, container load, cache algorithms, dynamic cache*

DOI: 10.17587/it.25.216-222

References

1. **Potapov D. R., Artemov M. A., Baranovskii E. S.** *Obzor uslovii adaptatsii samoadaptiruyushchikhsya assotsiativnykh konteinerov dannykh* (Review adaptation conditions of adaptive associative data storages), *Vestnik Voronezhskogo gosudarstvennogo universiteta. Seriya: Sistemnyi analiz i informatsionnye tekhnologii*, 2017, no. 1, p. 112—119 (in Russian).

2. **Zobov V. V., Seleznev K. E.** *Instrument dlya modelirovaniya nagruzki na konteinery dannykh* (Tool for modeling the load on data containers), *Materialy chetyrnadtsatoi nauchno-metodicheskoi konferentsii "Informatika: problemy, metodologiya, tekhnologii",* 10—11.02.2011 g. Voronezh, VGU, 2014, vol. 3, pp. 154—161 (in Russian).

3. **Potapov D. R., Seleznev K. E.** *Vizual'noe predstavlenie rezul'tatov analiza i sravnenija kontejnerov dannyh* (Visual presentation of results of analysis and comparison of data containers), *Informatika: problemy, metodologija, tehnologii. Materialy XVI Mezhdunarodnoj nauchno-metodicheskoj konferencii,* 2016, pp. 123—128 (in Russian).

4. **Potapov D. R.** *Vizual'noe sravnenie i analiz kontejnerov dannyh* (Visual comparison and analysis of data containers), *Matematicheskoe i komp'juternoe modelirovanie, informacionnye tehnologii upravlenija (MKMITU-2016)*, Voronezh, 2016, pp. 178—181 (in Russian).

5. **Potapov D. R.** *Razrabotka i realizacija metoda dlja analiza i sravnenija kontejnerov dannyh* (Development and implementation of a method for analyzing and comparing data containers), *Vestnik Fakul'teta prikladnoj matematiki i mehaniki*, 2016, no. 5, pp. 30—36 (in Russian).

6. **Potapov D. R.** *Analiz primenenija mnogomernyh struktur dannyh v samoadaptirujushhihsja kontejnerah dannyh* (Multidimensional data structures usage in adaptive data storages review), *Sbornik trudov Mezhdunarodnoj nauchno-tehnicheskoj konferencii "Aktual'nye problemy prikladnoj matematiki, informatiki i mehaniki",* Voronezh, 18—20 December 2017, pp. 436—442 (in Russian).

7. **Potapov D. R.** Existing methods of multidimensional "key-value" storages construction for using in adaptive data storages review, *Journal of applied informatics*, 2018, 2 (74): 69—82.

8. **Potapov D. R., Artemov M. A., Baranovskii E. S., Seleznev K. E.** *Obzor metodov postroenija kontejnerov dannyh "kljuch-znachenie" dlja ispol'zovanija v samoadaptirujushhihsja kontejnerah dannyh* (Existing methods of "key—value" storages construction for using in adaptive data storages review), *Kibernetika i Programmirovanie*, 2017, no. 5 (in Russian).

9. **Jacob B., Ng S., Wang D.** Memory Systems: Cache, DRAM, Disk, San Francisco, Morgan Kaufmann Publishers, 2007, p. 900.

10. **Carlson J. L.** Redis in Action, NY, Manning Publications, 2013, 320 p.

11. **Sanderson D.** Programming Google App Engine with Python, Sebastopol, CA, O'Reilly Media, 2015, 538 p.

12. **Soliman A.** Getting Started with Memcached, Birmingham, Packt Publishing, 2013, 56 p.

13. **Tanenbaum A.** *Arhitektura komp'yutera* (Computer architecture), Saint-Petersburg, Piter, 2003, 704 p. (in Russian).

14. **Cache-Aside** pattern, available at: https://docs.microsoft.com/en-us/azure/architecture/patterns/cache-aside

15. **Koroljuk V. S., Portenko N. I., Skorohod A. V., Turbin A. F.** *Spravochnik po teorii verojatnostej i matematicheskoj statistike* (A Handbook on Probability Theory and Mathematical Statistics), Moscow, Nauka, 1985, 640 p. (in Russian).

16. **Deitel P., Deitel H.** C# 6 for Programmers (6th Edition) (Deitel Developer Series), Prentice Hall, 2016, 768 p.

17. **Walkenbach J.** Microsoft Excel 2016 Bible: the comprehensive tutorial resource, John Wiley & Sons, 2015, 1152 p.

18. **Megiddo N., Modha D. S.** Outperforming LRU with an Adaptive Replacement Cache Algorithm, Computer, April 2004, vol. 37, no. 4, p. 58—65.

19. **O'Neil E. J., O'Neil P. E., Weikum G.** The LRU-K Page Replacement Algorithm for Database Disk Buffering, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. SIGMOD '93*, New York, NY, USA, ACM, 1993, p. 297—306.

20. **Chou H.-T., DeWitt D. J.** An evaluation of buffer management strategies for relational database systems, *Proceeding VLDB '85 Proceedings of the 11th international conference on Very Large Data Bases*, 1985, vol. 11, pp. 127—141.

21. **Troelsen A., Japikse P.** C# 6.0 and the.NET 4.6 Framework, New York, NY, USA, Apress, 2015, 1832 p.

22. **Casella G., Berger R. L.** Statistical Inference, Duxbury Resource Center, 2001, 688 p.