A caching model for a quick file access system

N. V. Ermakov and S. A. Molodyakov

Peter the Great St. Petersburg Polytechnic University, Polytechnicheskaya 29, St.Petersburg 195251, Russia

E-mail: kolyaermakov@yandex.ru

Abstract. The amount of stored and transmitted information is constantly increasing. The search and retrieval of the required data become more difficult as it grows. In this paper, we consider the developed storage system. The system stores information about pharmaceutical research. This system effectively stores data depending on its features. The main attention is paid to the issue of data caching at the application server level. To speed up data retrieval, we suggest using a two-section cache. The first section contains the results of queries to the database, and the second contains data from the file system. Analytical and simulation caching models have been developed for research. In the model, you can specify the distribution of requests, the size of the cache sections, the number of requests, and other parameters. The model uses the LRU cache replacement policy. The graphs of model research depending on the cache size and other parameters are provided. The analytical model considers the ideal case, so it shows better results compared to the simulation model.

1. Introduction

Currently, companies operating in various industries use information systems to store, search, and process information. Examples of information systems are banking systems, online stores, ticket reservation systems. Over many years of work, various institutions and enterprises have accumulated large amounts of information, which continue to grow [1], [2]. A large amount of data provides additional capabilities, but requires the development of new data storage systems. The bottleneck of such data warehouses is the search and retrieval of information.

File systems usually provide storage of poorly structured and unstructured information, leaving further structuring to applications. The file naming rules, the method of accessing data stored in a file, and the structure of this data depend on the specific file management system and, possibly, on the file type. From the point of view of an application program, a file is a named area of external memory that can be written to and from which data can be read. The file management system takes care of allocating external memory, mapping file names to the corresponding addresses in external memory, and providing access to data. The file management system does not know the file structure, it is known only to the program that works with it.

Versatility is both an advantage and a disadvantage of file systems. There are several needs of information systems that are not covered by the capabilities of file management systems: maintaining a logically consistent set of files; providing a data manipulation language; information recovery after various kinds of failures; simultaneous data modification in a single file by several users. The structure of information is often very complex, and although data structures are different in different
information systems, there are often many similarities between them. Database management systems solve many problems that are difficult or impossible to solve when using file systems [3] - [5].

Databases allow you to obtain data on difficult conditions when it is not enough to obtain data by file name. The need for a database occurs when there is a need for something beyond the minimum that the file system provides. If there is no need for indexing and high-speed data processing, then you can use the file system. There are applications for which files are enough; applications for which it is necessary to decide what level of work with data in external memory they require, and applications for which you certainly need a database.

Despite all the advantages of databases, a file system is better suited for storing a large amount of unstructured data than a database [6]. Therefore, if you need to store both structured and unstructured data, the file system is used together with an SQL, NoSQL database, or search engine [7]. An SQL database stores data in a normalized form, but is poorly distributed across several nodes, unlike NoSQL databases and search engines. The Elasticsearch and Solr search engines, which are based on Lucene, can be used as document-oriented NoSQL databases. Lucene uses an inverted index, and databases use the B-tree index by default. Search engines have shorter search time, but longer insertion time compared to other databases, for example, MongoDB [8].

Usually the search for stored information in databases and file systems is already optimized and performance improvement is possible only with the increased use of hardware resources. Application developers have very limited opportunities to accelerate performance (actually only an index). Therefore, the main attention should be paid to methods and algorithms for working with data at the application level.

One way to improve storage performance is caching. The essence of caching is that part of the data is duplicated in the storage with a higher access speed – in the cache. If the necessary information is in the cache, then you get a significant gain in time.

The cache has an important role in the work of almost any web application at the level of working with databases, web servers, and on the client. The HTTP Protocol has several different headers for managing caching processes at the client and network level. Requests can be cached not only by the end-user but also by various intermediate proxies and CDN networks. Cached data can be stored on the client for some time, after this time the data is again requested from the server and updated. This caching method is used if it is acceptable to have outdated data on the client for some time. To always have up-to-date data, you need to transfer the modification date of cached data in a request to the server. The server will either return newer data or respond that the data has not changed.

The cache row replacement strategy is an algorithm that is applied to a filled cache and determines the order in which rows are removed from the cache. New data blocks are inserted in their place. The optimal algorithm should choose data for replacement, which will not be accessed in the future for the longest time. In practice, it is unrealizable, because it is impossible to predict which data is most likely to be unclaimed. Therefore, many other algorithms are used that are inferior to the optimal one.

Analytical and simulation modeling of the caching system is used to evaluate effectiveness depending on various parameters. For analytical modeling, typically, the processes of system functioning are written in the form of some functional relationships. In simulation modeling, the system is replaced by a model that describes the processes as they would take place. In analytical modeling, models based on Markov chains can be used [9] - [11]. For a simulation of various cache coherence protocols (MSI, MESI, and others), models based on state diagrams are used [12], [13].

The data storage system was developed for EPAM. It is designed to store information about pharmaceutical research. The system consists of a file system, a database, a search engine, and an application program hosted on a server. All necessary information is classified and distributed. Unstructured data is stored in the file system, structured data is stored in the database, and unchanged data is stored in the search engine. The database stores file metadata (for example, size, name, path, and other user-defined metadata). All events that occur in the system (authorized creation, modification, deletion of files or metadata) are stored in the Elasticsearch search engine. The system
allows you to perform arbitrary file searches for any given metadata. The results of frequently used database queries are cached.

This paper aims to investigate the possibilities of improving the performance (increasing search speed and decreasing file access time) of the data storage system by using caching more effectively. Analytical and simulation models of a two-section cache were developed for research. This cache contains the results of database queries and files from the file system.

2. Analytical caching model

Figure 1 shows the caching scheme in the data storage system. Clients request files from the application based on a specified condition. The application server requests a list of files that meet this condition from the database. The query results are placed in the S1 cache segment. The application server then requests the required files from the file system. The requested files are placed in the S2 cache segment. The requested data can also be placed in the L1 client cache. If data may be required by several clients, then they can be cached in proxy servers L2. When requesting data, the client or proxy server sends the modification time of cached data. The server either sends the updated data or responds that the data in the cache is up to date.

\[ T_{S1} \] and \[ T_{S2} \] are the average read times from cache segments S1 and S2. \( T_1 \) and \( T_2 \) are the average read times from the database and file system. \( len_1 \) and \( len_2 \) are the sizes of the cache segments S1 and S2. \( n_1 \) is the number of requests to the database and \( n_2 \) is the number of requested files. In real storage systems, some data is always requested more often than others. Suppose that the “numbers” of database queries and the “numbers” of requested files from users are distributed according to the normal law. \( \sigma_1 \) and \( \sigma_2 \) are standard deviations of the “numbers” of queries to the database and files. “Numbers” that are next to the mathematical expectation are requested more often.

A normal distribution with a mathematical expectation \( \mu \) and a standard deviation \( \sigma \) has a probability density function (1) and a distribution function (2).

\[
 f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)
 \]

\[
 F(x) = \frac{1}{2} (1 + \text{erf} \left( \frac{x-\mu}{\sqrt{2}\sigma} \right) ) \quad (2)
 \]

\text{erf} is the error function (3).

\[
 \text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt \quad (3)
 \]

Suppose that the cache always stores the most frequently requested data with “numbers” from \( \mu - len/2 \) to \( \mu + len/2 \), where \( len \) is the cache size. Then equation (4) is the probability of a cache hit.

\[
 P = \frac{len}{\sum_{i=1}^{len} f(x_i)} \quad (4)
 \]
\[ p_{\text{hit}} = F\left(\mu + \frac{\text{len}}{2}\right) - F\left(\mu - \frac{\text{len}}{2}\right) = \text{erf}\left(\frac{\text{len}}{2\sqrt{2}\sigma}\right) \]  

Equation (5) is the total operating time of the system.

\[ T_{\text{total}} = n_1(p_{\text{hit}}T_S + (1 - p_{\text{hit}})T_1) + n_2(p_{\text{hit}}T_S + (1 - p_{\text{hit}})T_2) \]

Equation (6) is the total system running time without caching.

\[ T_{\text{without cache}} = n_1T_1 + n_2T_2 \]

Equation (7) is the gain of using caching. The gain depends on the ratio of requests to the database and the files \(n_1/n_2\), the ratio of the cache length and the standard deviation \(\text{len}/\sigma\) for each segment, the read time from the database \(T_j\), the file system \(T_2\) and the cache segments \(T_{S1}\) и \(T_{S2}\).

\[ gain = \frac{T_{\text{without cache}}}{T_{\text{total}}} = \frac{n_1T_1 + n_2T_2}{n_1\text{erf}\left(\frac{\text{len}_1}{2\sqrt{2}\sigma}\right)T_{S1} + \text{erf}\left(\frac{\text{len}_1}{2\sqrt{2}\sigma}\right)(1-T_1) + n_2\text{erf}\left(\frac{\text{len}_2}{2\sqrt{2}\sigma}\right)T_{S2} + \text{erf}\left(\frac{\text{len}_2}{2\sqrt{2}\sigma}\right)(1-T_2)} \]

3. Simulation caching model

Figure 2 shows the functional diagram of the developed caching model created in MATLAB / Simulink. The model consists of two normal distribution number generators, two cache models, and a part that calculates the benefits of caching. Number generators produce fractional numbers, which are then rounded. In the model, you can specify the standard deviation of the generators, the size of the cache segments, the average read time when it hits the cache and misses. We believe that the average read time for cache hits and misses is 1. To calculate the gain, the average time without caching \(T_1 + n_2/n_1 \times T_2\) is multiplied by the current time and divided by the total time with caching.

![Fig. 2. Caching model in MATLAB / Simulink.](image)
longest time is popped out of the queue and the cache, if there was a cache miss (result = 0) and the cache size is the maximum. Two counters count the number of cache hits and misses.

![LRU cache model](image)

**Fig. 3.** Block “LRU cache model”.

### 4. Results

Let’s assume the average file size is 1 GB, the average response size from the database is 100 bytes, the cache segment size for the database is 100 KB, and for files, it is 10 GB. On average, the cache contains \( \text{len}_1 = 1000 \) database queries and \( \text{len}_2 = 10 \) files. The number of requests to the database is \( n_1 = 10000 \) and to the file system is \( n_2 = 100 \). The standard deviation of requests to the database is \( \sigma_1 = 100\sqrt{10} \) and to the file system is \( \sigma_2 = \sqrt{10} \). About 1000-2000 various database queries and 10-20 files are called because 99.7% of requests fall in the range from \(-3\sigma\) to \(+3\sigma\). The average read time from the database cache is \( T_{S1} = 1 \), from the database is \( T_1 = 10 \), from the file cache is \( T_{S2} = 10 \), from the file system is \( T_2 = 10000 \). We investigate the dependence of the gain from using the cache depending on one parameter with the other fixed parameters and compare it with the gain according to (7).

Figure 4 shows the dependence of gain on the size of the second cache segment \( \text{len}_2 \). When \( \text{len}_2 \geq 14 \), the gain stops increasing and is equal to 6.3. This is because the cache is initially empty. Therefore, there will always be cache misses at first, even the cache size is larger than all the different requested data. Also, the LRU replacement policy is worse than the ideal replacement policy in the formula. We expect that the gain calculated by the formula will always be greater than the gain obtained from the simulation. In this case, the gain is limited to \( 100/14 = 7.14 \). This result can only be achieved if reading data from the database and the caches takes almost no time compared to reading from the file system.

![Gain vs Cache Size](image)

**Fig. 4.** The dependence of the gain on the S2 cache size.

Figure 5 shows the dependence of the gain on the read time from the database \( T_2 \) (when a cache miss occurs in the second segment). The longer the time, the greater the gain.
Figure 5 shows how the gain depends on the number of requested files \( n_2 \). The gain decreases with the growth of \( n_2 \) to about \( n_2 = 10 \) and then begins to grow. This happens because when \( n_2 \) is small, the cache is empty and cache misses occur. With a large \( n_2 \), the gain is 4-4.5, i.e. about 75-78% of cache hits occur, which is worse than the estimate of 88.6% in the ideal case (8).

\[
\begin{align*}
p_{\text{hit}} &= \text{erf}\left(\frac{\text{len}}{2\sqrt{2\sigma}}\right) = \text{erf}\left(\frac{10}{2\sqrt{20}}\right) = 0.886
\end{align*}
\]  

(8)

5. Conclusions
This article discusses the analytical and simulation models of a two-section cache. This cache can be used for data storage systems that consist of several components, such as a database and a file system. An ideal case is considered in the analytical model, and therefore it shows the best results. The developed models can be used to determine the optimal cache size in the data storage system depending on the distribution of requests. Caching can significantly speed up the retrieval of data that is often requested and rarely changed.

References
[1]  Laboshin L U, Lukashin A A and Zaborovsky V S 2017 The Big Data approach to collecting and analyzing traffic data in large scale networks Procedia Computer Science 103 536-542
[2]  Voinov N, Garzon K R, Nikiforov I and Drobintsev P 2019 Big Data processing system for analysis of GitHub events 2019 XXII Int. Conf. on Soft Computing and Measurements (SCM) 187-190
[3]  Coronel C and Morris S 2019 Database systems: design, implementation, and management. (Boston, USA: Cengage Learning)
[4]  Ramakrishnan R and Gehrke J 2003 Database management systems (McGraw Hill)
[5]  Connolly T and Begg C 2015 Database systems: practical approach to design, implementation, and management (Harlow: Pearson Education Limited)
[6]  Sears R, Van Ingen C and Gray J 2006 To blob or not to blob: Large object storage in a database or a filesystem? Technical Report MSR-TR-2006-45
[7]  Ermakov N V and Molodyakov S A 2019 Development and implementation of accelerated
methods of data access J. Phys.: Conf. Ser. 1326
[8] Abubakar Y, Adeyi T S and Auta I G 2014 Performance evaluation of NoSQL systems using YCSB in a resource austere environment Int. J. of Applied Information Systems 7 23-27
[9] Ben-Ammar H, Hadjadj-Aoul Y, Rubino G and Ait-Chellouche S 2019 On the performance analysis of distributed caching systems using a customizable Markov chain model J. of Network and Computer Applications 130 39-51
[10] Chen C and Beltrame G 2017 An adaptive Markov model for the timing analysis of probabilistic caches ACM Transactions on Design Automation of Electronic Systems (TODAES) 23 1-24
[11] Ammar H B, Chellouche S A and Aoul Y H 2017 A Markov chain-based Approximation of CCN caching Systems 2017 IEEE Symp. on Computers and Communications (ISCC) 327-332
[12] Kehagias D and Raptis I 2017 An Android-based MESI cache coherence simulator The 5th Int. Virtual Conf. on Advanced Scientific Results 194-199
[13] Sensfelder N, Brunel J and Pagetti C 2019 Modeling cache coherence to expose interference 31st Euromicro Conf. on Real-Time Systems (ECRTS 2019)