Development and implementation of accelerated methods of data access

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. In this paper we address the problem of organizing a permission-conscious access to files and their metadata. A large amount of data is required to be stored and analyzed in medicine, bioinformatics, astronomy, and banking. For effective data storage, it is necessary to apply different technologies depending on the data. We have developed a system that consists of the following components: a relational DBMS to store user permissions and file metadata; an off-the-shelf distributed search engine to store activity logs; a distributed cache with permissions; a file system; a special application that interacts with all the respective components. We describe the overall architecture of our approach and discuss each component in detail. To offload database server and to speed up the permission checking we have devised a simple data caching scheme that is performed on an application server. The idea was to manually rewrite queries that are used by our application to check permissions. As a result, queries that use several tables request only non-cached tables from the database. We conclude our paper with the experimental evaluation of our system and the proposed caching approach.

1. Introduction
A large amount of data poses new challenges for storage and processing systems. The amount of data in the world is constantly increasing. A large amount of data can provide information that will help you make smarter decisions. For example, pharmaceutical companies around the world can increase patient safety and the effectiveness of clinical trials by collecting large amounts of information. However, the additional features that a large amount of data provides require the development of new data storage systems.

The data storage system should store a large amount of information, store it under any circumstances, provide constant access to it and not lose data. Important system parameters are security [1], fault tolerance [2] and performance.

We use caching to speed up the information retrieval from the database. The search engine is used to efficiently store immutable data. We store file metadata in a database for quick search through it.

Many applications have a layered architecture, usually consisting of a client, an application server, and a database [3]. Bottlenecks at the application server level can be resolved easily and cheaply by increasing the number of servers. All application state is maintained by a database that typically runs on a single high-performance computer. A user query can call tens or even hundreds of database queries. The total response time observed by the user is often characterized by the execution time of queries to the database, especially when it is very busy. SQL databases do not scale well [4]. To improve performance and scalability, you need to reduce the load on the database. One way to solve
this problem is database caching, which is usually located at the application server level (figure 1) [5]. The purpose of database caching at the application server level is to transfer part of the load from the database server to intermediate servers.

![Multi-tier application architecture with database caching](image)

**Figure 1.** Multi-tier application architecture with database caching.

The intermediate server has a local database that stores a copy of some data from the database. This allows you to evaluate some queries locally.

There are several caching options: caching records of database tables (for example, TimesTen [6]) and caching query results [7]. The cache can be used as a key-value store or a materialized view. For example, in Facebook, the memcached key-value store is used as a cache [8]. Materialized views are used for complex queries with a large number of lines [9].

There are different caching schemes depending on the data and queries. Caching can be optimized if the queries are related to each other [10] or the data has a graph structure [11]. Caching is used for ORM (Object-Relational Mapping) technology [12]. In some cases, multi-level caching is used [13]. There is a semantic caching scheme [14]. You can analyze the logical formula of the query and determine whether it can be performed using the cache.

Queries often select data based on several database tables. In this case, the considered schemes for executing the query either do not use the cache, or use the group cache [6]. When using a group cache, data is automatically loaded into the cache if it is referenced from another table. In this article, we consider the scheme in which we cache not all tables involved in the query. This scheme can be used if some database tables cannot be cached, because, for example, this data changes frequently or requires a lot of memory.

Data can be stored not only in the database, but also in the search engine. There is often no transaction support in the search engine, and the data is stored in a denormalized form. Denormalization improves extraction performance (since data from different tables are not required to be combined), uses more space (since some data must be stored several times), but makes it difficult to maintain consistency and relevance (since any change must apply to all instances). However, the search engine is well suited for data that is written once and then read a lot. Examples of search engines are Elasticsearch and Solr, which are based on Lucene.

Data can also be stored in files in the file system. File systems are well suited for storing large amounts of unstructured data, but are poorly suited for storing structured data.

Databases and search engines use indexes to speed up data search. Indexing increases the time of addition or change, since it is necessary to update the indexes, but significantly reduces the search time. Databases use a B-tree based index, the search engines mainly use an inverted index.

There are data storage systems that use several technologies. For example, data can be stored in the HDFS file system and the Solr search engine [15]. This system does not support data modification, and structured data is stored both in the search engine and in the file system. There are also data storage systems consisting of a database and file system.

This article will cover both the use of caching methods when working with data, and the use of file systems and search engines. It is expected that the data access time is mainly reduced due to caching the results of database queries.

The contribution of this paper is: 1) a developed method for caching database queries that requires query decomposition. 2) the design of the data storage system with multi-user access, which uses different technologies for different types of data.

2. The query caching method

Figure 2 shows the proposed caching scheme. The query to the database should be divided into two queries. The first query does not access cached tables, and the second is executed only if the result is
not in the cache. Processing of the received data at the application server level should also be added. Consider an example (figure 3). A similar query can be executed in the developed system.

Suppose that some entities are stored in the database (the Entity table) and each entity has access rights for each user (the Permission table). Suppose that the Entity table cannot be cached. Entity permissions can be cached.

Suppose we want to select all entities for which a given user has rights. Let the granted field be 1 if there are rights. The query will look like this:

```
SELECT DISTINCT e.* FROM ENTITY e JOIN PERMISSION p
  ON e.id = p.ENTITY_ID WHERE p.USER_ID = :userId AND GRANTED = 1;
```

We will store all access rights in the cache by the composite key of the user_id and entity_id fields, the value will be the granted field. To fill the cache, we will use the query:

```
SELECT * FROM PERMISSION
  WHERE USER_ID = :userId AND ENTITY_ID = :entityId.
```

We can get all the entities from the database using the query `SELECT * FROM ENTITY`. From the received entities, it is necessary to leave only those for which this user has rights. We can do this using the cache.

Consider another example (figure 4).

```
SELECT DISTINCT e.* FROM ENTITY e
JOIN PERMISSION p ON e.id = p.ENTITY_ID
JOIN ENTITY_INFO ei ON e.id = ei.ENTITY_ID
JOIN GROUP_USER gu ON p.GROUP_ID = gu.GROUP_ID
WHERE (p.USER_ID = :userId OR gu.USER_ID = :userId) AND GRANTED = 1;
```

Figure 2. Proposed caching scheme.  
Figure 3. Example of an ER diagram of the database №1.

Figure 4. Example of an ER diagram of the database №2.
It is necessary to split this query into two. The first query will request a database each time:

```sql
SELECT * FROM ENTITY e JOIN ENTITY_INFO ei ON e.id = ei.ENTITY_ID
```

The results of the second query will be cached:

```sql
SELECT * FROM PERMISSION p JOIN GROUP_USER gu ON p.GROUP_ID = gu.GROUP_ID
WHERE (p.USER_ID = :userId OR gu.USER_ID = :userId)
AND p.ENTITY_ID = :entityId
```

Then we filter the resulting entities from the second query using the cache.

If changes occur in cached tables, then it is necessary to clear or update the cache.

3. Developed data storage system

The developed storage system consists of a file system, a database, an access rights cache, and a search engine. The database stores file metadata (for example, size, who created and modified, and when, name, path, and other user metadata). It is possible to use arbitrary keys to store metadata. The key name is stored in the Metadata_key table, and the value is stored in the Metadata table (figure 5). One key can have multiple values. The system allows arbitrary search for files by their metadata.

![Figure 5. An ER diagram of metadata.](image)

For each file, each user or group of users may have read, write, and delete permissions. Permissions are frequently requested and rarely changed, so a cache can be used. If the client sends a request to update the cached data, the application server updates it both in the cache and in the database. All events in the system (creation, modification, deletion of files or metadata) are logged into the Elasticsearch search engine.

The client requests some information (metadata, files or occurred events) from the application server, and the application server requests the necessary subsystems to produce the desired result. The overall structure of the system is shown in figure 6.

![Figure 6. The overall structure of the system.](image)

The application was written in Java using the Spring framework. Apache Tomcat was used as an application server. The ehCache cache was chosen because it is convenient to use it with Spring. We used Hibernate and Spring Data to access the database.

Oracle database was used. Clients request an application using the HTTP protocol (REST API). Several clients can simultaneously send requests to the system. We use transactions to execute queries in the database in parallel. Cache settings are described in xml format. You can specify the lifetime of the cached data or that the data is not deleted by time, the maximum amount of data in memory, whether data is stored on disk, cache replacement policies (can be LRU – least recently used, LFU – least frequently used, FIFO – first in first out, the oldest element by creation time).
4. Testing the efficiency of caching
The storage system was deployed for testing on a local computer with an HDD. Table 1 shows the average execution time for several queries without caching and with caching. Queries perform complex metadata searches. All queries are different, but have a common part – filtering based on permissions. The Permission table in the database contains about 5 million entries for 130 thousand entities and 100 users. Database size is about 5 GB. Queries return from 100 to 1000 entities. Caching significantly reduces query execution time. The following is SQL query number 6 from table 1 without caching:
\[
\begin{align*}
&\text{select distinct e3.*} \\
&\text{from ENTITY e1 left join METADATA m1 on e1.ID=m1.OWNER_ID} \\
&\text{left join METADATA KEY mk1 on m1.METADATA_KEY_ID=mk1.ID} \\
&\text{cross join ENTITY e2 left join METADATA m2 on e2.ID=m2.OWNER_ID} \\
&\text{left join METADATA_KEY mk2 on m2.METADATA_KEY_ID=mk2.ID} \\
&\text{cross join ENTITY e3 left join METADATA m3 on e3.ID=m3.OWNER_ID} \\
&\text{left join METADATA_KEY mk3 on m3.METADATA_KEY_ID=mk3.ID} \\
&\text{left join PERMISSION p on p.ENTITY_ID = e3.ID} \\
&\text{left join GROUP_USER gu on p.GROUP_ID = gu.GROUP_ID} \\
&\text{where p.granted = 1 and (p.user_id = 'user' or gu.user_id = 'user')} \\
&\text{and mk3.KEY_NAME='k3' and lower(m3.VALUE) like 'v31'} \\
&\text{and mk2.KEY_NAME='k2'} \\
&\text{or lower(m2.VALUE) like 'v21' or lower(m2.VALUE) like 'v22'} \\
&\text{or lower(m2.VALUE) like 'v23' or lower(m2.VALUE) like 'v24'} \\
&\text{or lower(m2.VALUE) like 'v25' or lower(m2.VALUE) like 'v26'}} \\
&\text{and (e1.ID not in (select e4.ID from ENTITY e4 inner join METADATA m4 on e4.ID=m4.OWNER_ID inner join METADATA_KEY mk4 on m4.METADATA_KEY_ID=mk4.ID where mk4.KEY_NAME='k1') \\
&\text{or mk1.KEY_NAME='k1' and lower(m1.VALUE) like 'v1') \\
&\text{and e1.ID=e2.ID and e2.ID=e3.ID;}}
\end{align*}
\]

With caching in this query there are no two join statements with the Permission and Group_user tables and a condition in the where statement that checks access rights. Access rights are in the cache and, if necessary, loaded from the database. This query searches for 6 values of v21, ..., v26 for the k2 key. In queries #1-5, the search is performed by 1-5 values respectively.

### Table 1. Time of query execution.

| Query number | Average execution time, seconds |
|--------------|--------------------------------|
|              | Without cache | With cache |
| 1            | 24             | 2          |
| 2            | 49             | 3          |
| 3            | 66             | 4          |
| 4            | 93             | 5          |
| 5            | 125            | 6          |
| 6            | 148            | 7          |

5. Conclusion
A data storage system has been developed that can efficiently store data depending on its characteristics. It is necessary to use different technologies for different data. Unstructured data is
stored in the file system, structured data is stored in the database, immutable data is stored in the
search engine, and frequently requested data is cached.

A caching scheme was proposed that caches some of the tables involved in the query. The query to
the database is simplified; there is no need to do time-consuming join operations with cached tables.
Unfortunately, in this case, unnecessary data is obtained from the database, which will be filtered later
using the cache. Thus, the load on the application server increases and the amount of data transferred
between the database and the application server increases. The load on the database is generally
reduced. This is usually required because the load on the relational database is not well parallelized.
The efficiency of the proposed caching scheme depends on the data, cache update rate, server and
network characteristics. In some cases, performance may degrade.

References
[1] Zegzhda D P, Poltavtseva M A and Lavrova D S 2017 Systematization and security assessment
of cyber-physical systems *Automatic control and computer sciences* 51 835-841
[2] Bolsunovskaiya M V, Shirokova S V and Loginova A V 2018 Developing a procedure for
conducting a security audit of a software package for predicting storage system failures
*MATEC Web of Conferences* 245 10007
[3] Schuldt H 2009 Multi-tier architecture *Encyclopedia of database systems*
[4] Medvedev T A and Struchkov I V 2013 Middleware Architecture for Relational Database
Access in a Scalable Hybrid Cloud System *St. Petersburg State Polytechnical University
Journal. Computer Science. Telecommunication and Control Systems* №1(164) 9-16
[5] Ghandeharizadeh S, Yap J and Nguyen H 2014 Strong consistency in cache augmented SQL
systems *Proceedings of the 15th International Middleware Conference* 181-192
[6] Lahiri T, Neimat M A and Folkman S 2013 Oracle TimesTen: An In-Memory Database for
Enterprise Applications *IEEE Data Eng. Bull.* 36 6-13.
[7] Sinaga A M and Sibarani P 2016 The Implementation of Caching Database to Reduce Query's
Response Time *MATEC Web of Conferences* 58 03014
[8] Nishtala R, Fugal H, Grimm S, Kwiatkowski M, Lee H, Li H C, McElroy R, Paleczny M, Peek
D, Saab P, Stafford D, Tung T and Venkataramani V 2013 Scaling Memcache at Facebook
*10th USENIX Symposium on Networked Systems Design and Implementation* 385-398
[9] Ghandeharizadeh S and Yap J 2012 Materialized Views and Key-Value Pairs in a Cache
Augmented SQL System: Similarities and Differences *Technical report, University of
Southern California, Database Laboratory Technical Report 2012-05*
[10] Glasbergen B, Abebe M, Daudjee K, Foggo S and Pacaci A 2018 Apollo: Learning Query
Correlations for Predictive Caching in Geo-Distributed Systems *Proceedings of the 21st
International Conference on Extending Database Technology (EDBT)* 253-264
[11] Ma H, Shao B, Xiao Y, Chen L J and Wang H 2016 G-SQL: fast query processing via graph
exploration *Proceedings of the VLDB Endowment* 9 900-911
[12] Gupta P, Zeldovich N and Madden S 2011 A trigger-based middleware cache for ORMs
*ACM/IFIP/USENIX 12th International Middleware conference* 329-349
[13] Lin Y T, Hsiao Y H, Lin F P and Wang C M 2016 A hybrid cache architecture of shared
memory and meta-table used in big multimedia query *15th International Conference on
Computer and Information Science (ICIS)* 1-6
[14] Kumar P M and Vaideeswaran J 2012 Semantic based efficient cache mechanism for database
query optimization *International Journal of Computer Applications* 43 14-18.
[15] Voinov N, Drobinsev P, Kotlyaros V and Nikiforov I 2017 Distributed OAIS-Based digital
preservation system with HDFS technology *IEEE 20th Conference of Open Innovations
Association (FRUCT)* 491 - 497