The Memory & Storage Design of Neo4J

Fabian Klopfer

November 27, 2020

Abstract

In this document I describe the internals of the storage layer of the popular native graph database Neo4J. A detailed description of how records are stored into which files and how these are accessed is to be elaborated on in the following article. This is a translation, update and extension of previous work done by Michael Brendle.

1 Introduction

Relational databases store tables of data. The links considered in this category of DBMS are mostly used to stitch together the fields of an entry into one row again, after it has been split to satisfy a certain normal form. Of course one may also store tables where one table stores nodes and the other table's fields are node IDs to represent relationships.

However, in order to traverse the graph, one has either to do a lot of rather expensive look ups or store auxiliary structures to speed up the look up process. In particular when using B-trees as index structure, each look up takes $\mathcal{O}(\log(n))$ steps to locate a specific edge. Alternatively one could store an additional table that holds edge lists such that the look up of outgoing or incomming edges is only $\mathcal{O}(\log(n))$ which would speed up breadth first traversals. But still one has to compute joins in order to continue the traversal in terms of depth. Another way to speed things up is to use a hash-based index, but this also has a certain overhead aside from the joins.

In contrast to relational data base management systems, native graph databases use structures specialised for this kind of queries. In the remainder of the document I discuss based upon Michael Brendle's work what structures and mechanisms the graph database Neo4J uses in order to achieve this superior performance in the domain of graphs.

First of all, let us consider the high level architecture of a database management systems as shown in figure 1 — with a focus on the storage and loading elements.

Here The disk space manager, sometimes also called storage manager, handles de-/allocations, reads & writes and provides the concept of a page: A disk block brought into memory. For that it needs to keep track of free blocks in the allocated file. Optimally both a disk block and a page are of the same size. One crucial task of a disk space manager is to store sequences of pages into continuous memory blocks in order to optimize data locality. Data locality has the upside, that one needs only one I/O operation to load multiple pages. To summarize the two most important objectives of a storage manager are to provide a

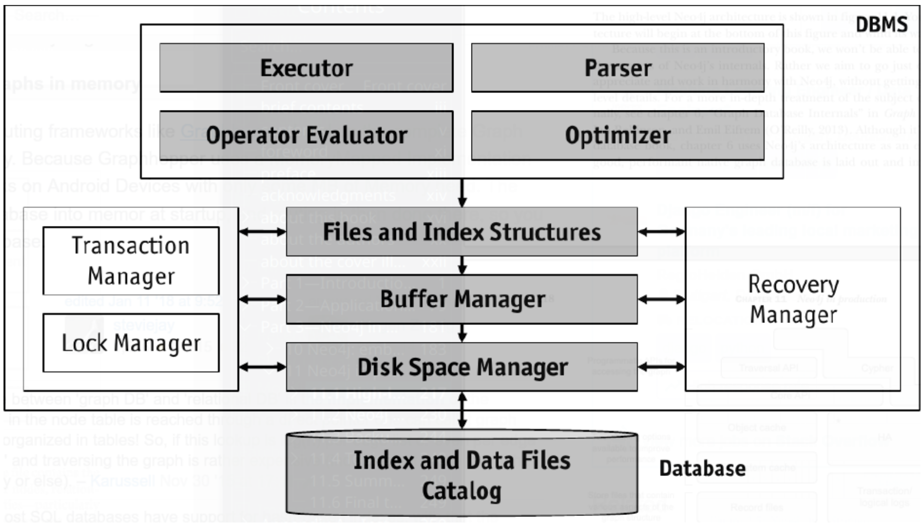

Figure 1: The typical structure of a relational database management system

locality-preserving mapping from pages to blocks based upon the information in the DBMS and to abstract physical storage to pages, taking care of allocation and access. A buffer manager is used to mediate between external storage and main memory. It maintains a designated pre-allocated area of main memory — called the buffer pool — to load, cache and evict pages into or from main memory. It's objective is to minimize the number of disk reads to be executed by caching, pre-fetching and the usage of suitable replacement policies. It also needs to take care of allocating a certain fraction of pages to each transaction.

The final memory and storage model relevant component of the of a database management system is the file layout and possible index structures. In order to store data a DBMS may either store one single or multiple files to maintain records.

A file may consist of a set of pages containing a set of slots. A slot stores one record with each record containing a set of fields. Further the database needs to keep track of free space in the file: A linked list or a directory must record free pages and some structure needs to keep track of the free slots either globally or per page.

Records may be of fixed or of variable size, depending on the types of their fields. Records can be layout in row or column major order. That is one can store sequences of tuples or sequences of fields. The former is beneficial if a lot of update, insert or delete operations are committed to the database, while the latter optimizes the performance when scans and aggregations are the most typical queries to the system.

Another option is to store the structure of the records along with pointers to the values of their fields in one files and the actual values in one or multiple separate files. Also distinct types of tables can be stored in different files. For example entities and relations can be stored in different files with references to each other, thus enabling the layout of these two to be specialized to their structure and usage in queries.

Files may either organize their records in random order (heap file), sorted or using a hash function on one or more fields. All of these approaches have upsides and downsides when it comes to scans, searches, insertions, deletions and updates.

To mitigate the effect that result from selecting one file organization or another, the concept of indexes have been introduced. Indexes are auxiliary structures to speed up certain operations that depend on one field. Indexes may be clustered or unclustered. An index over field F is called clustered if the underlying data file is sorted according to the values of F. An unclustered index over field G is one where the file is not sorted according to G. In a similar way indexes can be sparse or dense. A sparse index has less index entries than records, mostly one index entry per file. This can of course only be done for clustered indexes as the sorting of the data file keeps the elements between index keys in order. An index is dense if there is a one to one correspondence between records and index entries. There are different variants of storing index entries which have again certain implications on the compactness of the index and the underlying design decisions.

In another view of database management systems architectures, this boils down to the design decisions one has to make when implementing the storage layer and the access layer as shown in figure 1.

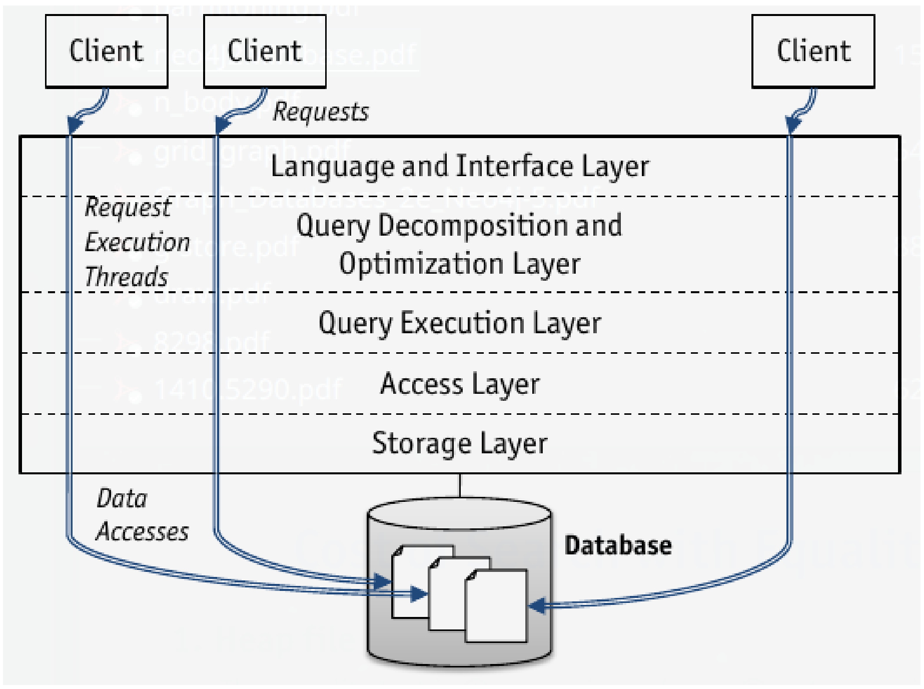
Client
Client
Client
Requests
Language and Interface Layer
Request Execution Threads
Query Decomposition and Optimization Layer
Query Execution Layer
Access Layer
Storage Layer
Data Accesses
Database

Figure 2: The architecture of a database management system from another point of view.

All these considerations make choosing different file splits, layouts, orderings, addressing schemes, management structures, de-/allocation schemes and indexes a complex set of dependent choices. These depend mainly on the structure of the data to be stored and the queries to be run.

When restricting to graph structures where nodes and relationships are allowed to have properties and labels and types respectively, this allows one to narrow down some of the design decisions. In particular the example of a popular graph native database is what I discuss in the next sections.

To get an overview of the architecture let us consider figure 1.

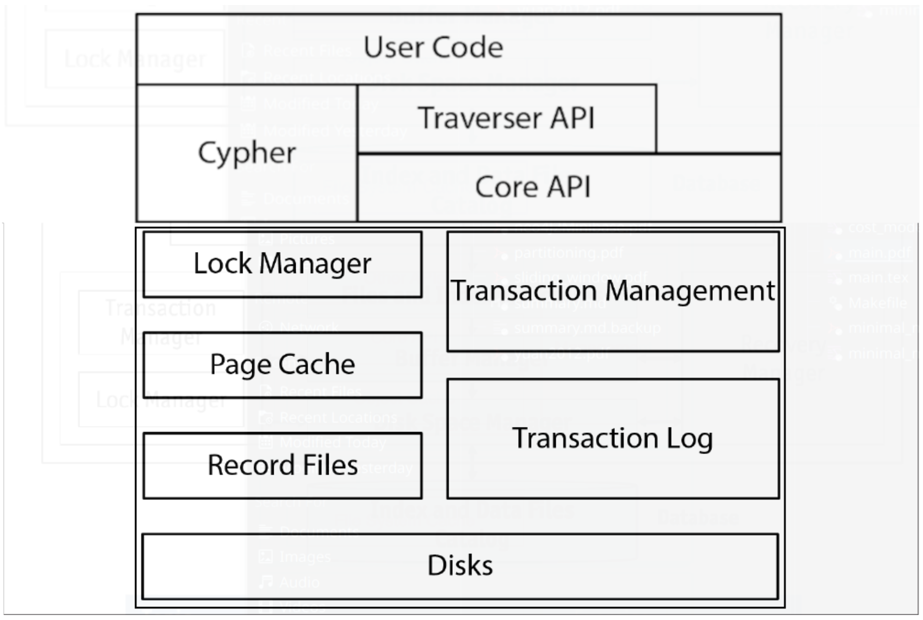

Figure 3: The typical structure of a relational database management system

2 Disk Space Management or Storage Management

`neo4j/community/io/src/main/java/org/neo4j/io/fs/`

`neo4j/community/storage-engine-api/src/main/java/org/neo4j/storageengine/api/`

`neo4j/community/native/src/main/java/org/neo4j/internal/nativeimpl/LinuxNativeAccess.ja`

https://neo4j.com/developer/kb/how-deletes-workin-neo4j/

http://g-store.sourceforge.net/th/index.htm

https://www.youtube.com/watch?v=NlT21Ceg3y0

https://neo4j.com/docs/operations-manual/current/performance/space-reuse/#space-reuse

3 Buffer Management

`neo4j/community/io/src/main/java/org/neo4j/io/pagecache`
https://www.slideshare.net/thobe/an-overview-of-neo4j-internals

4 File Layout, Record Structure & Management

`neo4j/community/record-storage-engine/.../kernel/impl/store/record/`
`neo4j/community/record-storage-engine/src/main/java/org/neo4j/kernel/impl/store/`

https://neo4j.com/developer/kb/understanding-data-on-disk/

Ancient: https://www.slideshare.net/thobe/an-overview-of-neo4j-internals
https://skillsmatter.com/skillscasts/2968-neo4j-internals

http://key-value-stories.blogspot.com/2015/02/neo4j-architecture.html

https://groups.google.com/g/neo4j/c/cxClivwF94k

5 Example

6 Conclusion