Graph Record Layout Research Environment

A graph database with some layout tools & methods

Fabian Klopfer

Databases and Information Systems Group
Department of Computer Science
University of Konstanz

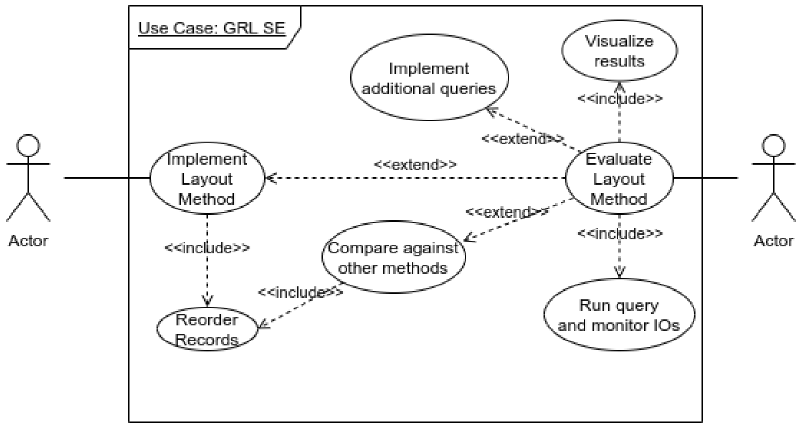
Abstract:
The here described software is focussed on finding new methods to layout the records of a graph database such that the least possible accesses are necessary to satisfy a certain benchmark. A benchmark is here usualy a set of queries, for example traversal-based queries. The current state-of-the-art methods use the graph structure to statically layout the graph on disk using clustering, partitioning or community detection methods to form blocks along with some ordering algorithm. The methods that are to be explored are dynamic — either in terms of the query or in terms of a changing database. This document provides a high-level user guide to the structure of the database and the methods and tools that are implemented. Furthermore the specification as well as a more extensive design document are captured, along with some benchmark and visualization techniques.

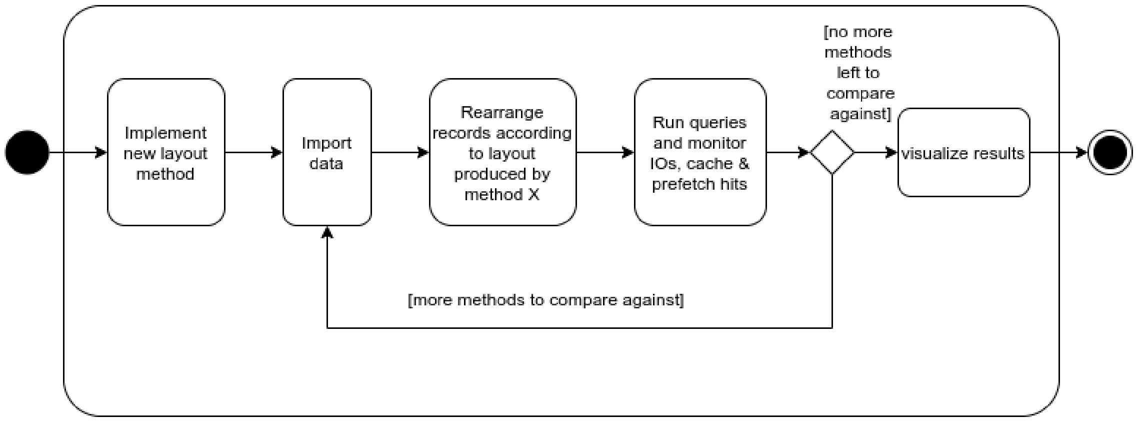
Contents

Chapter 1

User Guide

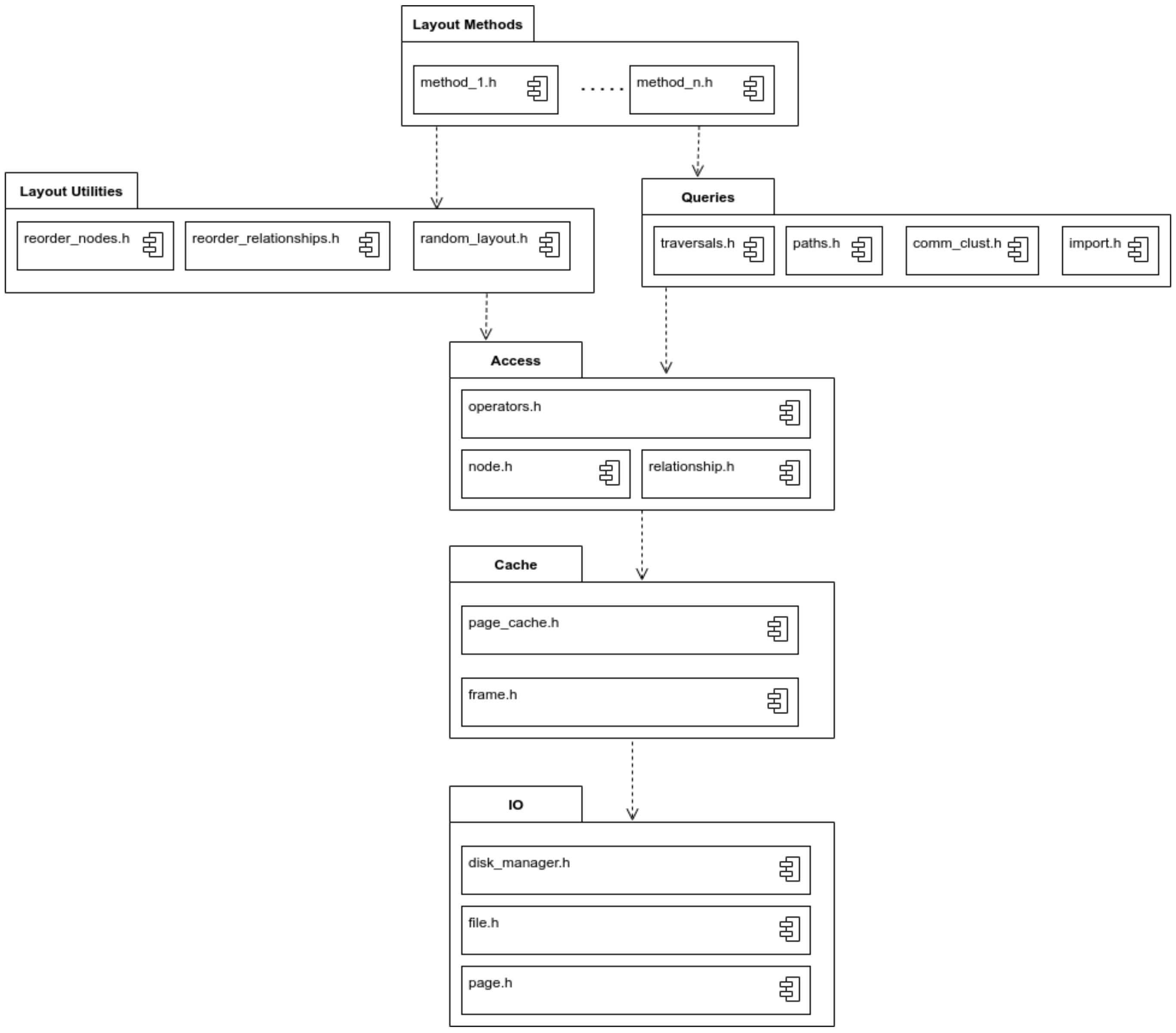
1.1 Introduction




Implement new layout method
Import data
Rearrange records according to layout produced by method X
Run queries and monitor IOs, cache & prefetch hits
[no more methods left to compare against]
visualize results
[more methods to compare against]

1.2 Graph Database

1.2.1 Architectural Overview



1.2.2 Building

1.2.3 Tests & Coverage

1.3 Record Layout Methods

1.3.1 Theoretical Aspects

1.3.2 Static Layout

1.3.3 Dynamic Layout

1.3.4 Benchmarks

Traversal-based Queries

The currently implemented benchmark provides a stepwise less branching traversal of the graph. While BFS and DFS are the most branching traversal possible and Dijkstra resembles this, it depends on the heuristic of A* how branching it is. With the zero heuristic it resembles Dijkstra's, with the perfect heuristic (actual distance to target vertex from source vertex) it does not branch at all (only the neccessary nodes are visited). ALT provides a good heuristic but not a perfect one.
To conclude: The benchmark tests how well the particular record layout deals with the branching factor od the traversal.
A neighbourhood-based layout should perform well for BFS-based traversals but could possibly degrade for a non-branching or DFS traversal.
An access history-based layout should work well for repeated queries of the same path for example.

1.3.5 Visualizations

Total Accesses per Query per Method A bar diagram that shows the number of disk accesses for a set of queries and possibly many methods. Reasonable if you want to compare the performance of different layouts on the same set of queries.

Access sequence Visualizes the addresses that are accessed as a line chart to inversigate if the access is sequential or "jumps" a lot. Samples 50 consecutive accesses at random.

1.4 Conclusion

1.4.1 Summary

1.4.2 Future Work

Chapter 2

Software Requirement Specification and Design Document

2.1 Introduction

Purpose

The purpose of this combined Software Requirements Specification and Design Documents is to describe the features, constraints, demands and the intended design of a research environment for the optimal layout of graph records on disk in detail. This document is intended for both the stakeholders and the developers of the system and will be proposed to Dr. Theodoros Chondrogiannis, the supervising postdoctoral researcher.

Scope

This Software system shall implement a graph record layout research environment, that consists of a graph database along with tools, that rearranges the recrods of the database based on a predefined format, measure the number of disk accesses needed to service a certain query and that provide other layouts to compare against. The time frame is three months, distributed over four months with 75% work load. When the system is implemented, a specific method that is to be dervied shall be too implemented, evaluated and published within the same time frame.

Definitions, Acronyms, Abbreviations

word	shortform	meaning
database	db	a software system to store and alter data in an organized manner.
Operating System	os	An operating system is system software that manages computer hardware, software resources, and provides common services for computer programs.
Portable Operating System Interface	POSIX	A specification for a set of OSes that covers for example Linux, macOS and BSD-style operating systems.
C Programming Language	C	a programming language.
Input/Output	IO	the notion of loading and storing information to media other than RAM and CPU registers. In this document hard drive and solid state disk are meant primarily.
Create, read, update, delete	CRUD	The basic database operations, that allow to create, read, update and delete a record.
Databases and Information Systems	DBIS	The name of the group at the university of Konstanz, at which the software system is build.
Stanford Network Analytics Project	SNAP	A porject of the university of stanford that hosts many large scale graph data sets.
Least Recently Used	LRU	A strategy when evicting pages from a pool of memory.
Breadth-first Search	BFS	A graph traversal scheme, where all neighbours of the current node are visited before continuing with the next node.
Depth-first Search	DFS	A graph traversal scheme, where the next node is considered before visiting all neighbours of the current node.
Single Source Shortest Path	SSSP	The problem of finding the shortest path to all nodes in a graph from one source node.

Overview

In the second section several conditions, assumptions and circumstances will be mentioned, that help charachterizing the software's special use case. In the thrid chapter the concrete requirements are listed. The fith section outlines an overview of the proposed software architecture. The sixth section, finally, elaborates on the subsystems of the archtiecture.

2.1.1 Overall description

Product Perspective

The product shall rely on functions of unixoid OSes. That is it uses the interfaces specified by the OpenGroup in the POSIX.1-2017 specification (also called IEEE Std 1003.1-2017) [11]. There are no relations to other software systems than the operating system during the runtime of the environment.

Product Functions

The product shall support different tasks in graph record layout research:

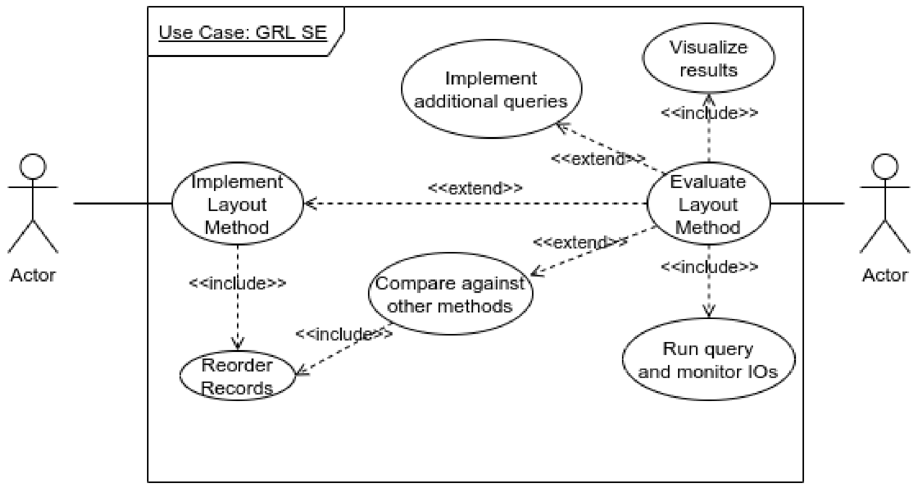
1. Import data into a graph database.
2. Query the database with a certain fixed function.
3. Rearange the record layout on disk with a layout given in a specific format.
4. Monitor the number of disk IOs.
5. Monitor the caching behaviour.
6. Provide interfaces for the implementation of new layout methods.
7. Provide a interfaces to ease the implementation of new queries.

This shall be done using a graph database, that shall save related information to disk and load them into a cache on access, as well as support the common CRUD functionality.

User Characteristics

The potential users are the staff and student assistants of the DBIS group. Therefore standard user will have technical knowldege, and have visited at least a basic course on databases. Futher the users are able to program in C. There are three different types of users of this research environment:

- Researchers: Implements new layout method, benchmarks the result of the method against other layouts, visualizes the result of the benchmark. Should at least know the theory of how records are stored.
- Supervisors: Works on the administration and coordination of the project. Eventually hires other researchers and developers. Also a researcher.
- (Future) Developers: Uses the existing framework to extend the functionality, for example to implement additional queries of a different type, add new features like storing properties and similar things. Needs to know details of database architecture and implementation, more advanced C programming and some software engineering.



Constraints

- The database is expected to be run on a standard notebook and desktop machine, that is it needs to run without momory-related errors on a machine with 8 GiB RAM, no matter the data set size.

Assumptions, Dependencies & References

The environment supports only POSIX-compliant OSes.
A documentation of the low-level design of Neo4J based on Michael Brendle's work and the source code of Neo4J at revision 4.1 [3] is referenced and can be found in the folder `doc/neo4jstorage`.

2.2 Requirement Specification

2.2.1 External interfaces

The POSIX interfaces.
The data format of some of the SNAP data sets.

2.2.2 Functional requirements

1 Data Storage and IO
The system shall

11 store a graph G consisting of nodes and edges (V, E).

12 store the records in a file on disk.

13 be able to grow and shrink the size of the file that is used.

14 be able to read from disk.

15 be able to write information to disk.

2 Data Caching and Memory
The system shall

21 not exceed a certain memory limit for both pages from disk and memory requirements from queries.

22 split an amount of memory into frames.

23 load data from disk into a frame on request.

24 maintain information on unused frames.

25 maintain a mapping from loaded data to frames.

26 evict data from memory when the memory limit is hit

27 be able to prefetch by reading more data than required in a neighbourhood

28 be able to monitor the numer of overall disk accesses.

29 be able to monitor the number of disk accesses that are necessary, if only a certain limited amount of memory is available.

210 be able to monitor the cache hit rate.

211 be able to monitor the prefetch hit rate.

3 Data Access
The system shall

31 be able to calculate the location in the file of nodes and relationships efficiently

32 keep track of free record slots on disk.

33 be able to create, read, update and write nodes.

34 be able to create, read, update and write relationships.

35 provide an interface to easily retrieve and traverse data.

36 provide functions to retrieve common informations about nodes and relationships, like the node degree.

4 Queries
The system shall

41 implement the most basic traversal schemes — BFS and DFS.

42 implement Dijksta's algorithm for finding all shortest paths from a single source (SSSP).

43 implement the A* algorithm for the shortest path problem.

44 implement the ALT algorithm for the shortest path problem.

45 provide data structures for the results of the above algorithms.

46 implement the Louvain method for community detection.

47 implement a random walk.

48 be able to import a set of standard data sets from the SNAP data set collection.

5 Layout Tools
The system shall

51 provide a function to generate a randomized record layout.

52 provide functions to reorganize the record layout on disk, given updated record IDs.

53 provide a function to reorganize the relationships given a layout of the vertices.

54 provide a function to sort the incidence list structure after reorganizing the record structure.

6 Layout Methods
The system shall

61 implement one static and one dynamic history-based layout method for a static database.

62 provide an interface for implementing new layout methods.

2.2.3 Performance Requirements

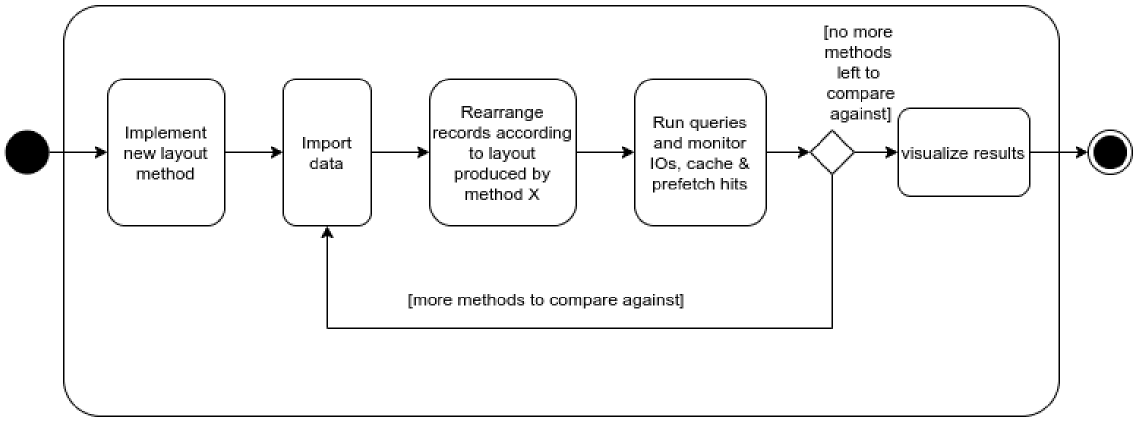
1 The system shall work with limited RAM resources even for very large datasets.

2.2.4 Software System Attributes

1 Maintainability

2 Extensibility

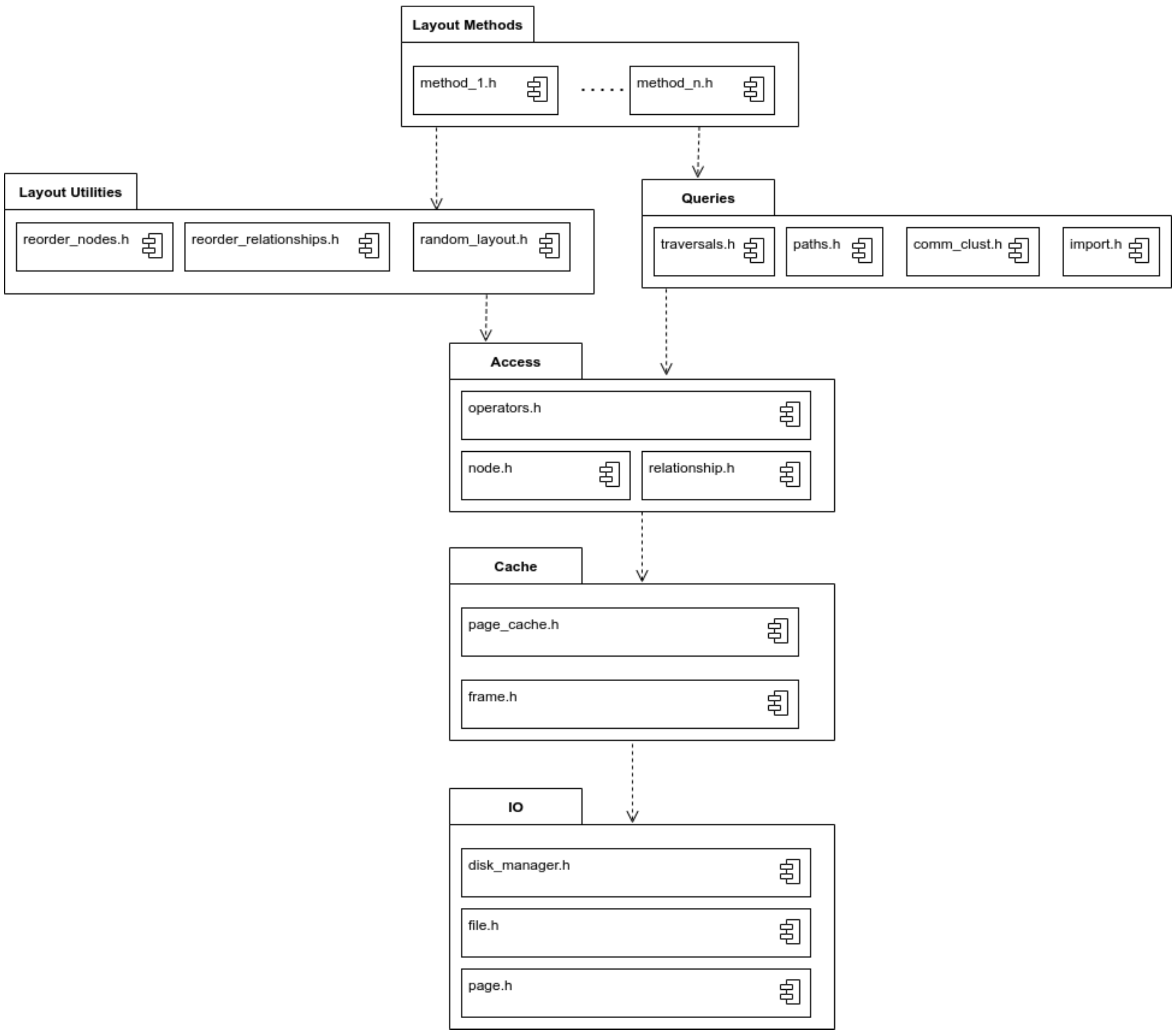
3 Correctness


Implement new layout method
Import data
Rearrange records according to layout produced by method X
Run queries and monitor IOs, cache & prefetch hits
[no more methods left to compare against]
visualize results
[more methods to compare against]

2.3 Software Design

2.3.1 Overview

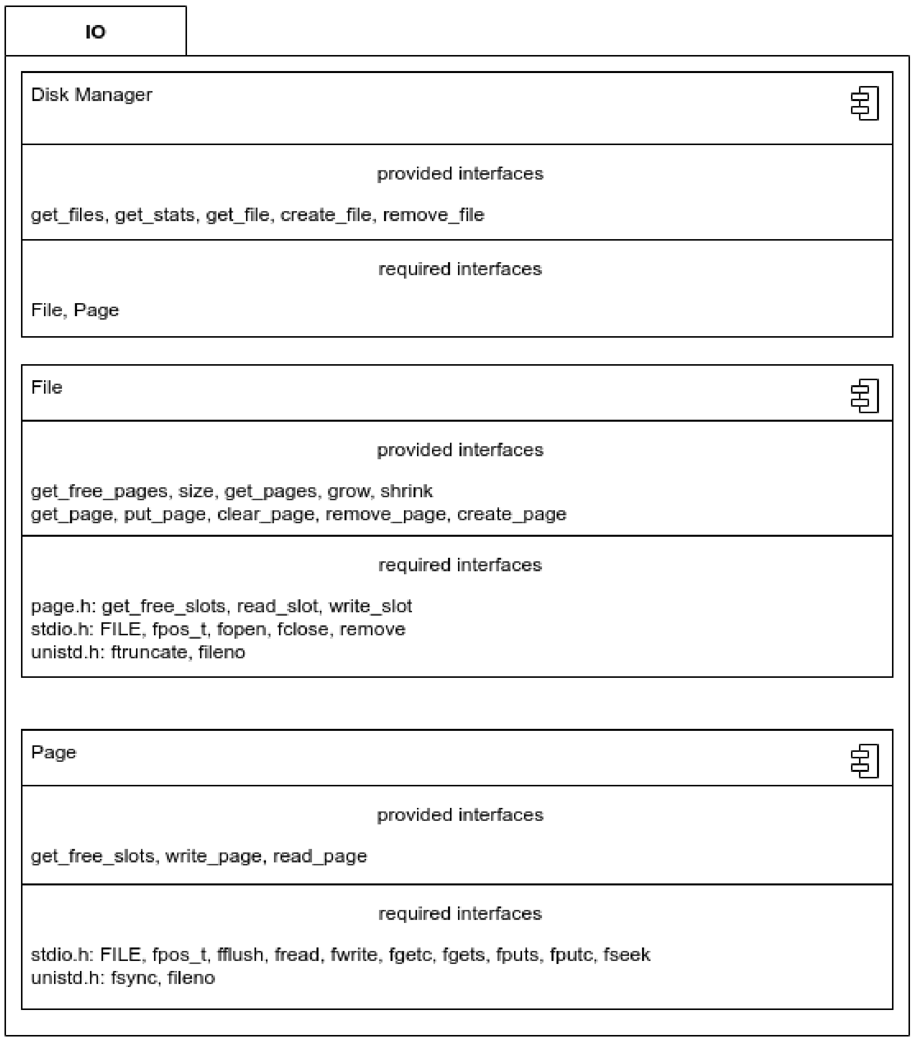
The graph record layout research environment consists of the lower layers of a database and a set of functions to assist in reordering records, measuring the number of IOs of a certain layout, visualizing the results, queries with certain access patterns, and an interface to implement new layout methods easily into the environment. The database layers are implemented in the IO, Cache and Access layers. The Query package does not contain a complete query language and evaluator but just a set of algorithms to be executed on the graph.



2.3.2 Subsystem decomposition

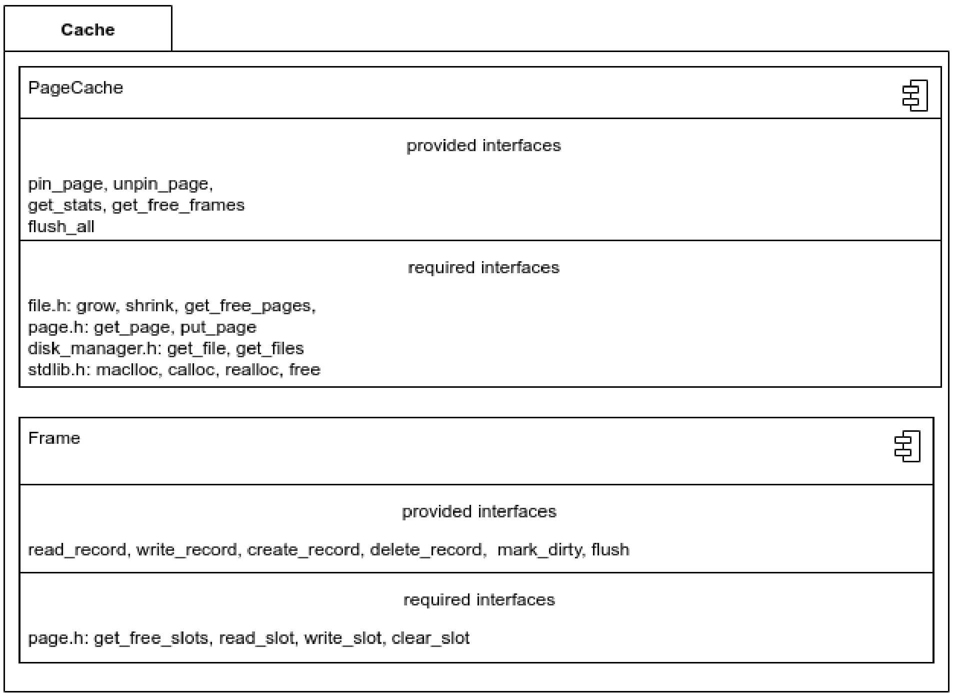
IO

The IO package shall provide the concepts of a file, a page and a disk manager.
Each page consists of slots that can hold records. The page keeps track of the usage of these slots with a bitmap. It also provides functions to read and write a slot.
Each file consists of a set of pages. The file accounts for the usage of pages with a page directory [10]. It also provides functions to read and write a page.
The database stores its data in the slots contained in the pages contained in the files. The files are created, grown and shrinked by the disk space manager.


IO
Disk Manager
provided interfaces
get_files, get_stats, get_file, create_file, remove_file
required interfaces
File, Page
File
provided interfaces
get_free_pages, size, get_pages, grow, shrink
get_page, put_page, clear_page, remove_page, create_page
required interfaces
page.h: get_free_slots, read_slot, write_slot
stdio.h: FILE, fpos_t, fopen, fclose, remove
unistd.h: ftruncate, fileno
Page
provided interfaces
get_free_slots, write_page, read_page
required interfaces
stdio.h: FILE, fpos_t, fflush, fread, fwrite, fgetc, fgets, fputs, fputc, fseek
unistd.h: fsync, fileno

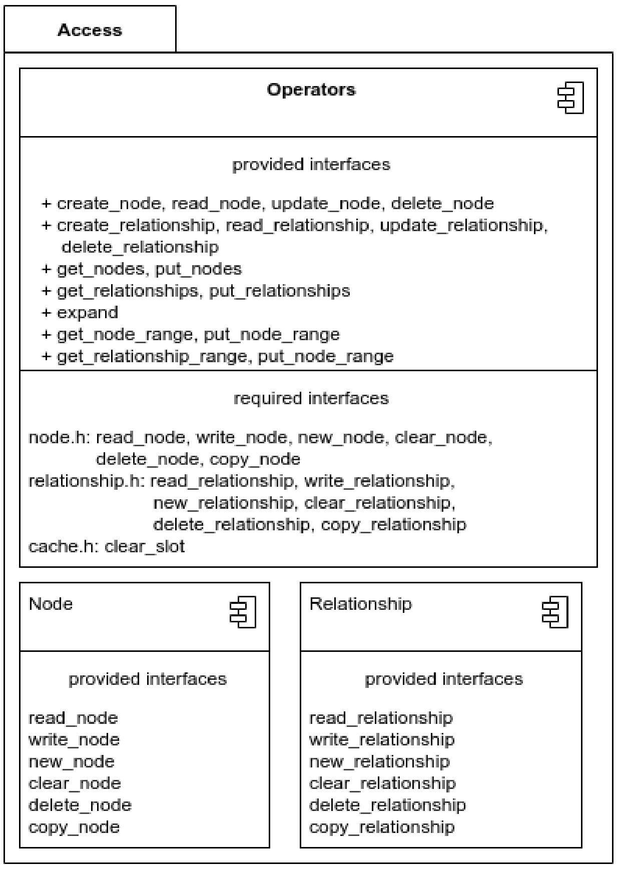
Page Cache

This packages provides a cache for pages, that is it loads and stores pages from or to disk in a fixed amount of RAM. To do this it splits the available main memory into frames of the size of a page and maintains a mapping from frames to pages, a list of free frames and evicts pages from main memory to free up space for pages to be loaded.


Cache
PageCache
provided interfaces
pin_page, unpin_page,
get_stats, get_free_frames
flush_all
required interfaces
file.h: grow, shrink, get_free_pages,
page.h: get_page, put_page
disk_manager.h: get_file, get_files
stdlib.h: maclloc, calloc, realloc, free
Frame
provided interfaces
read_record, write_record, create_record, delete_record, mark_dirty, flush
required interfaces
page.h: get_free_slots, read_slot, write_slot, clear_slot

Access

The Accesss package provides the record structures along with methods to create, read, update and delete nodes and relationships. The records structs are inspired by the ones used in Neo4 (see the documentation on that). It futher provides these functions for all records and ranges of records at once. Finally it implements the expand operator as described by Grossniklaus and Hoelsch [6].


Access
Operators
provided interfaces
+ create_node, read_node, update_node, delete_node
+ create_relationship, read_relationship, update_relationship, delete_relationship
+ get_nodes, put_nodes
+ get_relationships, put_relationships
+ expand
+ get_node_range, put_node_range
+ get_relationship_range, put_node_range
required interfaces
node.h: read_node, write_node, new_node, clear_node, delete_node, copy_node
relationship.h: read_relationship, write_relationship, new_relationship, clear_relationship, delete_relationship, copy_relationship
cache.h: clear_slot
Node
provided interfaces
read_node
write_node
new_node
clear_node
delete_node
copy_node
Relationship
provided interfaces
read_relationship
write_relationship
new_relationship
clear_relationship
delete_relationship
copy_relationship

Queries

The query package is a collection of algorithms to be executed using the lower levels of the graph database. These include traversals, shortest path algorithms, community detection methods, but also a utility to import datasets from the stanford network analytics platform project [7]. Currently implemented are:

- Traversals
  - BFS [12]
  - DFS [8]
  - Random Walk [9]
- Shortest Path
  - Dijksta's Algorithm [2]
  - A* [5]
  - ALT [4]

- Community Detection
  - Louvain Method [1]

Layout Utilities

The layout utilities package provides functions to reorder the records given a mapping from the current record IDs to the desired ones. It also provides means to generate a random mapping and to sort the incidence list of the relationships after reorganizing the records.

Layout Method Interface

The layout method interface is to be implemented, such that it returns a map from the current IDs to the desired record IDs, where the record IDs encode the position in the file. The name of the method can be arbitrary, it is just of importance that the arguments and the return type are as specified, as well as the invariants.

```
unsigned long[] layout_nodes(graph_database db) {
  require db != null;
  require db.num_nodes > 0;
  ensure db.num_nodes == result array length
  ensure unique node IDs
}

 unsigned long[] layout_relationships(graph_database db) {
  require db != null;
  require db.num_nodes > 0;
  require db.num_relationships > 0;
  ensure db.num_relationships == result array length
  ensure unique relationship IDs
}
```

Visualization Utilities

Finally a python script provides means of visualizing the number of IOs and the cache hit rate as monitored by the page cache package. It relies on matplotlib and numpy.

Bibliography

[1] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[2] Edsger W Dijkstra et al. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[3] *GitHub - neo4j/neo4j: Graphs for Everyone*. Dec. 9, 2020. URL: https://github.com/neo4j/neo4j (visited on 12/09/2020).

[4] Andrew V Goldberg and Chris Harrelson. "Computing the shortest path: A search meets graph theory." In: *SODA*. Vol. 5. Citeseer. 2005, pp. 156–165.

[5] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[6] Jürgen Hölsch and Michael Grossniklaus. "An Algebra and Equivalences to Transform Graph Patterns in Neo4j". In: *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference (EDBT/ICDT 2016)*. Ed. by Themis Palpanas and Kostas Stefanidis. CEUR Workshop Proceedings 1558. 2016. URL: http://ceur-ws.org/Vol-1558/paper24.pdf.

[7] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[8] Édouard Lucas. *Récréations mathématiques: Les traversees. Les ponts. Les labyrinthes. Les reines. Le solitaire. la numération. Le baguenaudier. Le taquin*. Vol. 1. Gauthier-Villars et fils, 1891.

[9] Karl Pearson. "The problem of the random walk". In: *Nature* 72.1867 (1905), pp. 342–342.

[10] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2000.

[11] *The Open Group Base Specifications Issue 7, 2018 edition, IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*. June 1, 2021. URL: https://pubs.opengroup.org/onlinepubs/9699919799/functions/contents.html (visited on 06/01/2021).

[12] Konrad Zuse. "Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben". In: *Archiv der Mathematik* 1.6 (1948), pp. 441–449.