A Comparison of Allocation Algorithms for Partially Replicated Databases

Stefan Halfpap
Hasso Plattner Institute
University of Potsdam, Germany
stefan.halfpap@hpi.de

Rainer Schlosser *
Hasso Plattner Institute
University of Potsdam, Germany
rainer.schlosser@hpi.de

***Abstract*—Increasing demand for analytical processing capabilities can be managed by replication approaches. However, to evenly balance the replicas' workload shares while at the same time minimizing the data replication factor is a highly challenging allocation problem. As optimal solutions are only applicable for small problem instances, effective heuristics are indispensable. In this paper, we test and compare state-of-the-art allocation algorithms for partial replication. By visualizing and exploring their (heuristic) solutions for different benchmark workloads, we are able to derive structural insights and to detect an algorithm's strengths as well as its potential for improvement. Further, our application enables end-to-end evaluations of different allocations to verify their theoretical performance.**

I. Introduction

The demand for analytical processing capabilities is steadily growing. The increasing workload can be scaled out to additional replica nodes. Using a naive replication approach, all data is duplicated and stored on all nodes in the database cluster. However, with such full data replication (i) the memory consumption quickly increases with the number of nodes and (ii) all replicas have to synchronize all data modifications caused by inserts, updates, and deletes.

Analytical queries access large data sets. Nevertheless, analyses of real workloads show that the majority of queries require only a limited set of tuples and attributes [1]. When scaling out, this knowledge can be used to *optimize database replicas* to either process a specific subset of queries more efficiently [2] or to reduce the memory consumption. One approach to reduce the memory consumption of a database cluster is partial replication.

Partial replicas are used to answer subsets of analytical queries while not requiring the memory capacity of a full copy of the database. For this purpose, the database must be partitioned horizontally and/or vertically into disjoint data fragments [3]. Individual fragments are, then, allocated to one or multiple database nodes. The calculation of optimal fragment allocations is an NP-hard problem as the data allocation and the workload distribution are mutually dependent.

In this paper, we test and compare algorithms to assign data to multiple replicas to *evenly* balance the query load while *minimizing* the overall memory capacity. While for small problems *optimal* fragment allocations can be determined using linear programming (LP), for larger problems heuristic approaches have to be used. We consider two state-of-the-art algorithms to calculate effective allocations which can be even applied to *large-scale* problems: (i) the (greedy) allocation approach by Rabl et al. [4] and (ii) the LP-based decomposition approach described in [5].

In [4] and [5], the authors propose different concepts and derive their allocation algorithms. They investigate their solutions' complexity (solving time, number of variables and constraints, etc.) and analyze the performance of calculated allocations for single examples.

However, more general properties or structures of the solutions depending on the specific approaches cannot be derived from their numerical results. We aim to address this issue. Our contributions can be summarized as follows:

- We test different allocation concepts and visualize their solutions/allocations for the TPC-H and TPC-DS benchmark as well as configurable workloads.
- We propose a step-by-step demonstration of the algorithms and compare solution properties, which enable us to derive structural insights.
- We support end-to-end evaluations to verify the theoretical performance of allocations in practice.

The remainder of this paper proceeds with a precise problem description in Section II. In Section III, we describe the different allocation algorithms. In Section IV, we present the demo scenario. Section V concludes this paper.

II. Fragment Allocation Problem

The goal is to allocate fragments to replicas such that the workload can be shared equally and the necessary data, i.e., the replication factor, is minimized. Hence, we consider a *coupled* data placement and workload distribution problem.

We assume a database consisting of N disjoint fragments and K nodes, where fragments can be replicated. The size of a fragment i is denoted by a_i, $i = 1, ..., N$. Further, we consider a set of Q (classes of) queries j, characterized by fragments used, i.e., $q_j \subseteq \{1, ..., N\}$, $j = 1, ..., Q$. We assume a workload, where queries j occur with frequency f_j, and the costs of a query j are independent of the executing node k, $k = 1, ..., K$, and denoted by c_j, $j = 1, ..., Q$.

* The authors are listed alphabetically. All work was shared equally.

Accepted at ICDE 2019, https://doi.org/10.1109/ICDE.2019.00226.

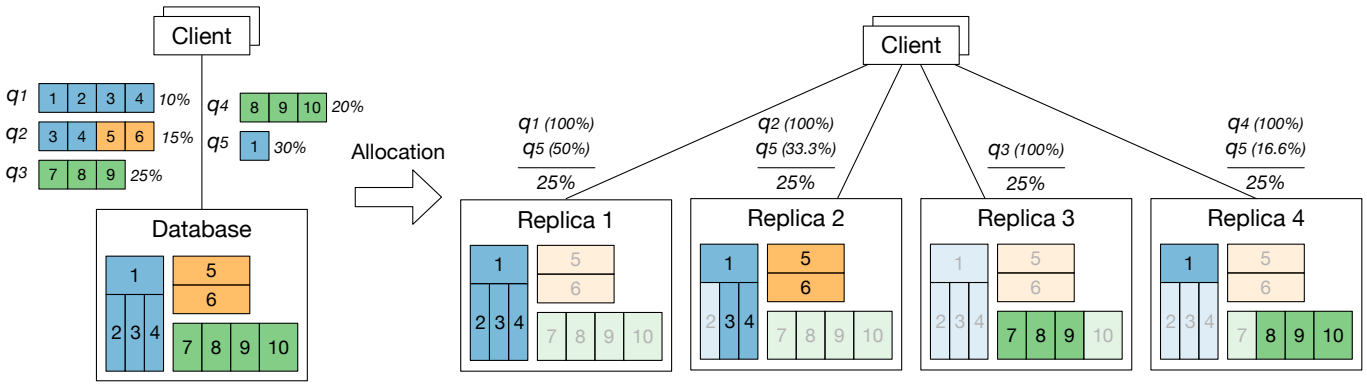

Fig. 1. Workload-driven fragment allocation. The left hand side of the figure visualizes the model input. Single tables, marked with different colors, consist of individual fragments. Queries correspond to different workload shares. Processing queries requires storing subsets of fragments. The objective is to minimize the overall memory consumption of the replication cluster while evenly balancing the load among (four) replica nodes. The right hand side of the figure illustrates a valid allocation and workload distribution. Transparent fragments visualize the memory savings per partial replica.

TABLE I
ILLUSTRATING EXAMPLE OF QUERIES, ASSOCIATED REQUIRED FRAGMENTS, AND TOTAL WORKLOAD SHARES (MEASURED BY QUERY COSTS × FREQUENCY), $Q = 5$, $N = 10$.

queries j	fragments i	costs c_j × frequency f_j	workload share
q_1	1, 2, 3, 4	5 × 20	10%
q_2	3, 4, 5, 6	50 × 3	15%
q_3	7, 8, 9	10 × 25	25%
q_4	8, 9, 10	20 × 10	20%
q_5	1	6 × 50	30%

Suitable metrics to model query costs may differ in complexity, ranging from easy to measure and widely applicable metrics, such as the average processing time of a query, to advanced cost models, e.g., considering memory hierarchies and memory access patterns.

Figure 1 illustrates the model input and an exemplary allocation for $N = 10$ fragments, $K = 4$ nodes, and $Q = 5$ queries. For the illustrating example, we assume an equal size for each fragment. Based on all query frequencies and costs, we obtain the *workload share* of each query. The input parameters for the five queries of the illustrating example are summarized in Table I, describing the accessed fragments and workload shares (cf. frequency × costs).

An algorithm has to decide (i) on which node to put which fragments and (ii) which query is executed at which node to which extent. The data placement problem and the workload distribution problem cannot be decoupled and have to be simultaneously solved. Further requirements are:

(a) A query j can only be executed at node k, if *all* relevant fragments are stored on node k.

(b) A perfect workload distribution aims at a workload share of $1/K$ at each node k.

(c) Additional factors can be taken into account: costs to synchronize replica nodes with regard to data modifications, reallocation costs to react on workload changes, and robustness of allocations to cope with node failures.

III. Allocation Algorithms

In this section, we describe the key ideas of the three state-of-the-art allocation algorithms that we compare.

A. Optimal Solution via Linear Programming

The described fragment allocation problem is NP-hard. However, it can be formulated as a linear mixed integer problem. The complexity of the LP problem *quickly* increases with the number of queries, fragments, and nodes. The problem can be *optimally* solved using off-the-shelf solvers as long as the size of the problem is sufficiently *small*. For instance, it is only possible to calculate optimal solutions (within 8 hours) for TPC-H (61 fragments, 22 queries) for up to 8 nodes [5] and TPC-DS (425 fragments, 99 queries) for up to 5 nodes.

B. Greedy Heuristic

The heuristic of Rabl et al. [4] starts to assign queries which account for a large workload share and access the most data, because these queries potentially cause the highest data duplication if they are assigned late. In specific, queries are ordered by the product of the workload share and the total size of accessed fragments. The node to assign a query is chosen by determining the largest *overlap* of already allocated fragments and those accessed by the query (nodes with no assigned queries are treated as if they have a complete overlap.) If a query's workload share exceeds the workload capacity of a node, the replica is assigned up to its limit and the remaining workload share has to be assigned later.

The algorithm runs in polynomial time. The heuristic has two shortcomings: First, when ordering the queries, the accessed fragments are not regarded (only their sizes). Second, the remaining queries are not regarded. Details of the algorithm are described in [4].

C. Decomposition-Based Heuristic

The key idea of the decomposition-based heuristic is to iteratively *split* the workload into smaller workload packages (chunks) such that the data redundancy is minimized in each

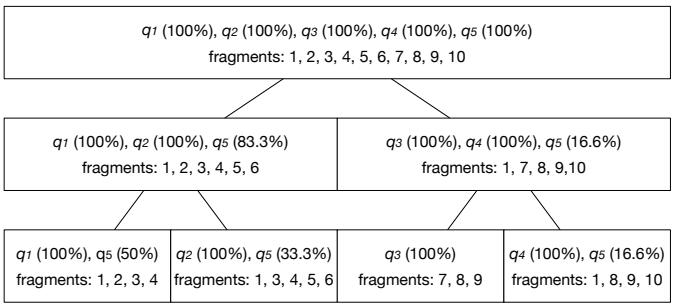

Fig. 2. Decomposition-based approach. Iteratively splitting data allocations and workload shares, cf. Table I; illustration of the case of $B = 2$ subnodes at two levels leading to $K = 2 \times 2 = 4$ nodes (using three subproblems) [5].

step, cf. [5]. The decomposition of workload is done using smaller LP *subproblems* similar to the program to calculate optimal solutions.

Figure 2 illustrates a decomposition for the problem example of Figure 1. The top node represents the total workload. The decomposition leads to $K = 4$ final nodes (leaves). The final data allocation enables a perfect workload distribution with a share of $1/K$ in each leaf node.

For each chunk node, the data allocation and workload distribution are calculated by applying a generalized LP. In particular, the number of subnodes B and their workload weights w_b, $b = 1, ..., B$, can be chosen arbitrarily. This way, the number of variables and constraints can effectively be decreased such that in each step the problem complexity is still manageable.

However, as in case of large problems early decompositions can still be costly, it is advisable to initially use small numbers of subnodes, e.g., $B = 2$. Alternatively, also a *hybrid* decomposition approach is possible: While the heuristic of Rabl et al. is applied close to the root node, the LP-based approach is used as soon as the subproblems are sufficiently small.

IV. Demonstration

We implemented the three algorithms and integrated them into an end-to-end application to calculate, visualize, and set up partial replica allocations. In this section, we describe the visualization (IV-A), summarize the algorithms' structural properties (IV-B), and explain how to conduct end-to-end evaluations for replication clusters (IV-C).

A. Visualization

Figure 3 shows an application screenshot, which visualizes a fragment allocation for six replicas. The application offers to calculate and visualize fragment allocations for *arbitrary* workloads and data sets, for which the costs per query and sizes of fragments are known. The demonstration includes query costs and fragments sizes for the benchmarks TPC-H and TPC-DS: costs for running the queries in PostgreSQL; measuring fragment sizes for scale factor 1, vertical partitioning with each column as individual fragment, and single column indices on all primary key columns.

The navigation bar shows the input parameters for the currently visualized allocation. In this case, we visualize an allocation for the TPC-H benchmark, using six replica nodes, and calculated by the decomposition approach.

The workload shares of the individual queries are visualized below the navigation bar. (Processing TPC-H query 17 and 20 did not return within 120s. That is why we omitted them in our allocations.) We can derive that the execution costs for query 1, 9, and 18 are the highest, because their segment widths are the broadest. When hovering the mouse over a query segment, the required fragments for the query are shown.

Below the navigation bar and the query list, the allocation result is visualized: (i) the replication factor and (ii) its comparison to an optimal allocation (if available). Further six segments, one for each replica node, display detailed allocation information: (iii) assigned workload shares with regard to the queries and (iv) the allocated data fragments.

The list of workload shares shows the assigned queries and visualize to which extent they contribute to the replica load. For example: the TPC-H queries 4, 12, 14, 19, and 21 are assigned to replica one. Thereby, query 21 accounts for the largest load on this replica.

The allocated fragments are visualized as hierarchical map with nested rectangles. Rectangles with the same basic color represent fragments of the same table. Transparently colored fragments are not stored, but show the memory savings. The size of the rectangles corresponds to the fragment size. The 16 blue tiles model the 16 columns of the TPC-H *LINEITEM* table, which accounts for the largest share of the data set.

Besides visualizing allocations for different input parameters, the application offers to visualize the steps taken by the algorithms: For Rabl's algorithm, these steps are the greedy query by query allocations to replicas; for the decomposition approach, the single partition steps can be examined.

In Figure 3, the final allocation to six nodes is based on an intermediate partitioning of the workload into two chunks with $4 + 2 = 6$ nodes. When hovering over the chunk information, the fragments of the selected chunk are visualized.

B. Summary

The visualization of different solutions enables to detect characteristic properties. This way, more general insights can be derived and potential for improvement can be identified. Our key findings can be summarized as follows.

First, the visualization of heuristic [5]'s solutions reveals the following structure: Queries that have *little common* data overlap are *separated* from each other; queries that share *similar* fragments are kept *together*. In solutions of [4], replicas are typically filled *after another* since queries are mostly assigned to comparably full replica as their overlap is likely to be large.

Second, compared to [4] the heuristic [5] finds more efficient allocations requiring less memory. Splitting the workload in a way that is consistent with the original problem *preserves* characteristics of optimal solutions. However, splitting the queries' workloads among fewer replicas makes the allocation less *robust* against node failures or changing workloads.

Fig. 3. Screenshot of a visualized allocation result for six replica nodes, applying the decomposition approach to 20 queries of the TPC-H benchmark.

C. End-to-End Evaluation

To calculate and setup allocations for arbitrary workloads and data sets, we offer a toolchain which extracts input parameters for the algorithms and deploys the calculated allocations on a PostgreSQL cluster. The user has to provide an SQL schema file with all table definitions, data files to load into the tables, and the workload in form of (optionally templated) SQL queries. The process works as follows:

1) We create all tables by executing the SQL schema file, load the provided data into the tables, and optionally create indexes.
2) Inputs for all allocation algorithms are: measured fragment sizes, query execution costs, query frequencies, and the number of replica nodes.
3) Result allocations can be visualized and optionally be deployed on chosen servers for end-to-end evaluations.

V. Conclusions

In this paper, we compared three workload-driven fragment allocation algorithms, minimizing the overall memory consumption of a replication cluster while maximizing throughput by balancing the load evenly. Our application visualizes the intermediate steps and final solutions of the different algorithms. The visualization enables to detect characteristic properties of the algorithms and solutions, thereby helping to derive general insights and potential for improvement.

References

[1] M. Boissier *et al.*, "Analyzing data relevance and access patterns of live production database systems," in *CIKM*, 2016, pp. 2473–2475.
[2] M. P. Consens *et al.*, "Divergent physical design tuning for replicated databases," in *SIGMOD*, 2012, pp. 49–60.
[3] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.
[4] T. Rabl and H. Jacobsen, "Query centric partitioning and allocation for partially replicated database systems," in *SIGMOD*, 2017, pp. 315–330.
[5] S. Halfpap and R. Schlosser, "Workload-driven fragment allocation for partially replicated databases using linear programming," in *ICDE*, 2019.