Workload-Driven Fragment Allocation for Partially Replicated Databases Using Linear Programming

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*—In replication schemes, replica nodes can process read-only queries on snapshots of the master node without violating transactional consistency. By analyzing the workload, we can identify query access patterns and replicate data depending to its access frequency. In this paper, we define a linear programming (LP) model to calculate the set of partial replicas with the lowest overall memory capacity while evenly balancing the query load. Furthermore, we propose a scalable decomposition heuristic to calculate solutions for larger problem sizes. While guaranteeing the same performance as state-of-the-art heuristics, our decomposition approach calculates allocations with up to 23% lower memory footprint for the TPC-H benchmark.**

***Index Terms*—database replication, allocation problem, linear programming**

I. Introduction

Increasing demand for analytical processing capabilities can be managed by scale-out approaches. Master replication – a single master node processes all OLTP workload while OLAP queries are load-balanced across replica nodes – is a common scale-out approach, e.g., supported in SAP HANA [1], Postgres-R [2], and as replication middleware [3]–[6]. Using full replication, the memory consumption increases linearly with the number of nodes. In addition, all replicas have to continuously synchronize all data modifications to the database, i.e., inserts, updates, and deletes.

Although analytical requests access large data sets, the majority of queries require a limited set of tuples and attributes. When scaling out to cope with increasing query load, we can analyze the workload and identify frequently accessed data portions or query tree patterns [7]. The derived knowledge can be used to replicate only subsets of the data, while still being able to balance the query load evenly.

Partial replication consists of two steps [8]. First, the data set is partitioned horizontally and/or vertically into disjoint data fragments, the units of replication. Second, the individual fragments are allocated to one or multiple database nodes. It is widely accepted to separate fragmentation and allocation to better deal with the complexity of the problem [8]. We focus on the allocation part. The calculation of optimal fragment allocations is an NP-hard problem. Note, the data allocation and the distribution of workload are mutually dependent and have to be optimized simultaneously.

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

In this paper, we present the following contributions:

- Workload-driven LP models to calculate both *optimal* fragment allocations as well as near-optimal heuristic solutions for *large-scale* problem instances.
- Reproducible analytical evaluations with the TPC-H benchmark to show that our model outperforms state-of-the-art allocation approaches.

The remainder of this paper proceeds with a description of the addressed fragment allocation problem in Section II. In Section III, we discuss related work. Our LP models are described in Section IV and Section V. Section VI concludes this paper.

II. Fragment Allocation Problem

The problem is a coupled data placement and workload distribution problem. We assume a database consisting of N disjoint fragments/partitions. The fragmentation scheme and data model are not relevant to the problem, even though our examples focus on vertical fragmentation and the relational model. The size of a fragment i is determined by a_i, $i = 1, ..., N$. We assume K nodes, where data can be replicated. We assume 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, $j = 1, ..., Q$. The costs of query j are independent of the executing node k, $k = 1, ..., K$, and determined by c_j, $j = 1, ..., Q$. Query costs are numerical and can be modeled in several ways, e.g., as average processing times.

We have to decide (i) on which node to put which fragments and (ii) which query is executed at which node to which extent. The main constraints are: First, query j can only be executed at node k, if *all* relevant fragments are stored on node k. Second, we look for a *balanced* workload share of $1/K$ at each node. The objective is to *minimize* the total amount of required data.

III. Related Work

The fragment allocation problem belongs to the more general problem of allocating resources in a network [8]. This class of problems is old and well-studied for varying system architectures and optimization goals. Because it was early proven that various different formulations of allocation problems are NP-complete [9], a lot of research has tried to find good heuristic solutions [8].

However, because the system architectures and optimization goals for specific allocation problems differ [10], there is no general applicable heuristic. Further, Özsu et al. underline that heuristics are tied to specific assumptions about the system and, thus, only applicable to certain specific formulations of allocation problems [8].

In the field of partially replicated databases, our work is closely related to the work of Rabl et al. [11]. We maximize throughput by balancing the load evenly among replicas and minimize the overall memory consumption of a cluster. As an LP model for an optimal allocation is only feasible for small problem sizes, Rabl et al. propose a greedy heuristic, which starts to assign queries having high query costs and accessing large amounts of data, because these queries potentially cause the highest data duplication if they are assigned late. Although the motivation behind the heuristic is plausible, the heuristic has two shortcomings. First, when choosing the next query to assign, the accessed fragments of the query are not regarded (only their sizes). Second, the information about the remaining queries, which have to be assigned later, is not regarded.

In contrast to rule-based heuristics, our decomposition approach preserves the structure of the problem and, thus, is able to find close to optimal solutions for the coupled data placement and load distribution problem.

IV. Optimal Solution

In this section, we propose a model to derive optimal allocations, which is applicable to small problem instances. The linear programming model is described in Section IV-A Numerical benchmark examples are given in Section IV-B.

A. Linear Programming Model

In our model, we use the following decision variables:

- $x_{i,k} \in \{0,1\}$, $i = 1,...,N$, $k = 1,...,K$, is allowed to be zero or one, indicating whether fragment i is allocated to node k (1) or not (0).
- $y_{j,k} \in \{0,1\}$, $j = 1,...,Q$, $k = 1,...,K$, is allowed to be zero or one, indicating whether query j can run on node k (1) or not (0).
- $z_{j,k} \in [0,1]$, $j = 1,...,Q$, $k = 1,...,K$, is allowed to be continuously between zero and one, indicating the workload share of query j executed at node k. The sum of shares is normalized to one for all queries.

Next, we give an LP formulation of the basic allocation problem. We seek to minimize data redundancy such that all nodes do not exceed a certain workload limit L, $0 \leq L \leq 1$. Like Rabl et al. [11], we assume that z can be chosen without regard to query frequencies and costs, which may be discrete. Hence, x, y, and z have to be chosen such that the objective

$$\begin{gathered} \underset{\substack{x_{i,k}, y_{j,k} \in \{0,1\}, z_{j,k} \in [0,1], \\ i=1,...,N, j=1,...,Q, k=1,...,K}}{minimize} \\ \sum\nolimits_{i=1,..,N,k=1,...,K} x_{i,k} \cdot a_i \qquad (1) \end{gathered}$$

is minimized and the following constraints are satisfied:

$$y_{j,k} \cdot |q_j| \leq \sum\nolimits_{i \in q_j} x_{i,k}, \quad j = 1,...,Q, k = 1,...,K \qquad (2)$$

$$z_{j,k} \leq y_{j,k}, \quad j = 1,...,Q, k = 1,...,K \qquad (3)$$

$$\sum\nolimits_{j=1,...,Q} f_j \cdot c_j / C \cdot z_{j,k} \leq L, \quad k = 1,...,K \qquad (4)$$

$$\sum\nolimits_{k=1,..,K} z_{j,k} = 1, \quad j = 1,...,Q \qquad (5)$$

Constraint (2) guarantees that a query j can only be executed at node k, if all relevant fragments are available. The cardinality term $|q_j|$ expresses the number of fragments used in query j. Constraint (3) ensures that a query j can only have a positive workload share on node k if it can be executed at node k. If $y_{j,k} = 0$ then $z_{j,k} = 0$ follows; if $y_{j,k} = 1$ the shares $z_{j,k}$ are not restricted. Constraint (4) guarantees that all nodes k do not exceed the workload limit L. In this context, we use the total workload costs denoted by

$$C := \sum\nolimits_{j=1,...,Q} f_j \cdot c_j$$

to normalize the workload in each node. Constraint (5) ensures that a query's workload shares on nodes k sum up to one.

Note, constraint (3) couples the binary variables y and the continuous variables z in a linear way. This formulation qualifies the model by Rabl et. al [11], where the coupling was expressed in a non-linear way using an *if condition*, cf. equation (40) in [11].

In our basic model, the objective and all constraints are linear in the decision variables. The total number of variables ($N \cdot K + Q \cdot K$ binary and $Q \cdot K$ continuous) and the total number of constraints ($2 \cdot Q \cdot K + K + Q$) increase in the number of fragments N, the number of queries Q, and the number of nodes K, respectively. Problem (1) - (5) is a linear mixed integer problem and can be solved using off-the-shelf solvers (see, e.g., NEOS solver) as long as the size of the problem is not too large.

B. Numerical Results

We illustrate the results of our problem formulation (1) - (5) using the $Q = 22$ queries of the TPC-H benchmark and use vertical partitioning with each column as individual fragment. The TPC-H schema consists of $N = 61$ columns.

To *reproduce* the calculation of replication factors, Table I shows the query costs. Further, we use scale factor 1 and make the following assumptions to derive fragment sizes: (i) table *LINEITEM* contains exactly 6M tuples; (ii) *date*, *identifier*, and *integer* require 4 bytes; (iii) *decimal* requires 8 bytes; (iv) *variable* and *fixed text* require maximum length bytes.

TABLE I
Exemplary TPC-H query processing times in seconds, i.e., costs c_j; the query frequencies are identical, i.e., $f_j = 1 \; \forall j = 1,...,22$.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
6.20	0.14	2.60	2.20	1.30	0.50	1.80	0.70	1.70	1.70	0.20
Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
1.30	8.10	0.40	0.30	1.10	1.30	1.70	3.60	1.20	7.60	0.70

TABLE II
OPTIMAL SOLUTION OF PROBLEM (1) - (5): PROBLEM COMPLEXITY, DATA REPLICATION FACTOR W^*/V, AND SOLVING TIME; DISTRIBUTION OF 22 TPC-H QUERIES TO K NODES WITH SINGLE COLUMNS AS FRAGMENTS.

K	# variables	# constraints	W^*/V	solve time
2	210	112	1.2889	0.29 s
3	315	157	1.4245	0.56 s
4	420	202	1.6495	13.55 s
5	525	247	1.7617	33.23 s
6	630	292	2.0537	158.42 s
7	735	337	2.1858	268.59 s
8	840	382	2.4555	2 260.95 s

We used CPLEX (version 12.7.0.0, 4 threads) via NEOS (https://neos-server.org). By W/V, we denote the *replication factor*, where the total amount of data used

$$W := \sum\nolimits_{i=1,..,N,k=1,...,K} x_{i,k} \cdot a_i$$

is normalized by the minimal amount of used data

$$V := \sum\nolimits_{i \in \bigcup_{j=1,...,Q: f_j > 0} \{q_j\}} a_i \tag{6}$$

For different numbers of nodes K Table II summarizes the problem complexity, the replication factor of the optimal solution, as well as the computation time. The optimal allocations achieve a perfect workload distribution $L^* = 1/K$. The optimal replication factor W^*/V is small compared to K, i.e., the replication factor for full replication. Optimal allocations for $K \geq 9$ could not be computed within a reasonable amount of time (termination after 8 hours), see also [11]. The reason is the increasing problem size characterized by the number of variables and constraints. To approach larger problems, heuristics have to be used, e.g., greedy approaches [11]. Although the optimal solution approach is limited to smaller problems, it is extremely useful as it enables to analytically measure any heuristic's performance.

V. DECOMPOSITION-BASED HEURISTIC

In Section V-A, we present a heuristic decomposition approach to compute near-optimal allocations. In Section V-B, we demonstrate that our heuristic enables to address large-scale problems and outperforms state-of-the-art approaches.

A. Linear Programming Model

In this section, we propose a decomposition-based heuristic to solve problem (1) - (5). The key idea is solve smaller subproblems by iteratively splitting the workload into different shares/chunks using a tree structure with K leaves. Figure 1 illustrates such a tree for $K = 2 \times 2 = 4$ with one additional level of decomposition for a given specific workload characterized by queries and fragments (cf. top left corner of the figure).

In each step, we split the workload using a general LP approach (similar to (1) - (5)) which can be applied in any node of the tree: Consider an arbitrary node on a certain level. Assume, the node's workload share is supposed to be split into B subnodes (within one step). Each subnode b represents n_b

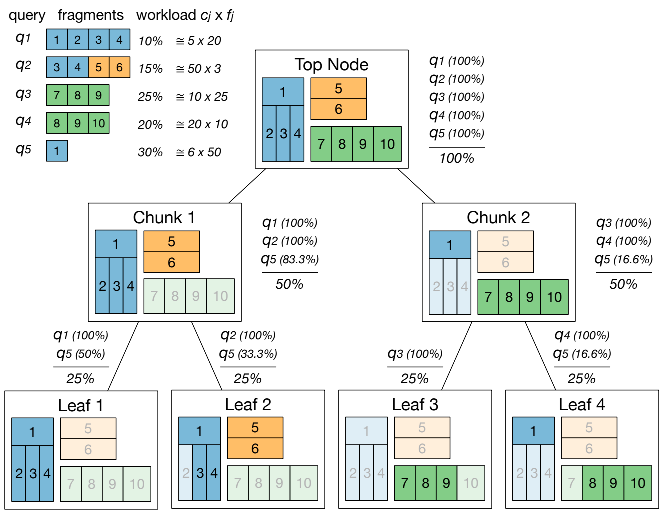

Fig. 1. Iteratively splitting data allocations and workload shares, using a tree decomposition principle; illustration of the case of $B = 2$ subnodes at two levels leading to $K = 2 \times 2$ nodes for a workload with $Q = 5$ queries and $N = 10$ fragments. Transparent fragments visualize memory savings.

final nodes, $b = 1, ..., B$. Hence, a subnode b has to take the workload share $w_b := n_b/K$, $b = 1, ..., B$. Further, we assume the remaining workload in this node is specified by the parameter $\bar{x}_i := 1$, $\bar{y}_j := 1$, $\bar{z}_j \in (0, 1]$, for subsets $i \in I \subseteq \{1, ..., N\}$ and $j \in J \subseteq \{1, ..., Q\}$.

The top node represents the total workload characterized by $\bar{x}_i := 1$, $\bar{y}_j := 1$, $\bar{z}_j := 1$, for all $i = 1, ..., N$ and $j = 1, ..., Q$. A node's share of the workload is denoted by $\bar{w}$. For the top node, the workload is undivided, i.e., $\bar{w} := 1$. A chunk node aggregating the load of n leaf nodes has the load $\bar{w} = n/K$. For each chunk node, we assign the data allocation and the workload distribution to its B subnodes with $\sum_{b=1,..,B} w_b = \bar{w}$ using the following generalized LP, cp. (1) - (5):

$$\underset{\substack{x_{i,b}, y_{j,b} \in \{0,1\}, z_{j,b} \in [0,1], L \geq 0, \\ i=1,...,N, j=1,...,Q, b=1,...,B : \bar{x}_i = 1, \bar{y}_j = 1}}{minimize}$$

$$\sum\nolimits_{i=1,..,N,b=1,...,B: \bar{x}_i = 1} x_{i,b} \cdot a_i + \alpha \cdot L \tag{7}$$

$$s.t. \quad y_{j,b} \cdot |q_j| \leq \sum\nolimits_{i \in q_j} x_{i,b}, \quad \begin{array}{l} j = 1, ..., Q : \bar{y}_j = 1 \\ b = 1, ..., B \end{array} \tag{8}$$

$$z_{j,b} \leq y_{j,b}, \quad \begin{array}{l} j = 1, ..., Q : \bar{y}_j = 1 \\ b = 1, ..., B \end{array} \tag{9}$$

$$\sum_{j=1,...,Q: \bar{y}_j = 1} f_j \cdot c_j / C / w_b \cdot z_{j,b} \leq L, \quad b = 1, ..., B \tag{10}$$

$$\sum\nolimits_{b=1,...,B} z_{j,b} = \bar{z}_j, \quad j = 1, ..., Q : \bar{y}_j = 1 \tag{11}$$

Note, in the balance constraint (10) L can be used as a fixed parameter L (cf. (4)) or as a *continuous variable* while adding a penalty term $\alpha \cdot L$ in the objective (7). This formulation has advantages as for solvers it is easier to identify a suitable starting solution. If α is chosen sufficiently large, i.e., $\alpha >> V$, cf. (6), the targeted workload limit is attained.

TABLE III
PERFORMANCE COMPARISON: DATA REPLICATION FACTORS OF THE RULE-BASED HEURISTIC BY [11] (W^S) VS. OUR HEURISTIC (W), CF. PROBLEM (7) - (11), WITH ONE DECOMPOSITION LEVEL OF B CHUNKS AND ASSOCIATED n_b NODES, $b = 1, ..., B$, SUMMING UP TO K; TPC-H.

K	chunks B	nodes n_b	W/V	W/W^*	solve time	W/W^S
4	2	2+2	1.6959	+2.82%	0.97 s	-19.37%
5	2	3+2	1.8546	+5.27%	1.33 s	-12.54%
6	2	3+3	2.0619	+0.40%	0.83 s	-10.71%
7	2	4+3	2.2162	+1.39%	1.22 s	-11.83%
8	2	4+4	2.4979	+1.73%	1.84 s	-18.42%
9	3	3+3+3	2.7725	/	6.88 s	-14.09%
10	4	4+2+2+2	2.7047	/	14.39 s	-12.09%
11	3	4+4+3	2.8604	/	2.50 s	-16.44%
12	4	3+3+3+3	3.1830	/	14.39 s	-9.92%
13	4	4+3+3+3	3.3886	/	13.89 s	-22.85%
14	4	4+4+3+3	3.7710	/	7.30 s	-17.13%
15	4	4+4+4+3	3.9540	/	23.83 s	-14.08%
16	4	4+4+4+4	4.0306	/	33.98 s	-11.28%

Let the optimal solution of the linear program (7) - (11) be denoted by x^*, y^*, and z^*. Further, the remaining workload of each subnode b on the next level is characterized by $\bar{x}_{\bar{i}} := 1$, $\bar{i} \in \{i = 1, ..., N : x^*_{i,b} = 1\}$, $\bar{y}_{\bar{j}} := 1$, and $\bar{z}_{\bar{j}} := z^*_{j,b}$, $\bar{j} \in \{j = 1, ..., Q : y^*_{j,b} = 1\}$, $b = 1, ..., B$, i.e., on the next level the program (7) - (11) can be applied again.

Note, from level to level the number of relevant fragments and queries decreases. Thus, the number of variables and constraints gets smaller (for constant B). On each level, for each node, the number of subnodes B and their workload weights w_b, $b = 1, ..., B$, can be chosen arbitrarily. This way, all integer numbers K can be utilized. An optimal solution also guarantees an even workload distribution, cf. $L = 1/K$.

The number of subnodes B can be used to *control* the problem complexity. The smaller B the faster is the computation (on each level) but overall data redundancy of the heuristic might increase compared to optimal allocations. To obtain minimal computation times, it is advantageous to start with $B = 2$ on the highest level.

The problem complexity gets quickly smaller with each decomposition from the root to the leaves. However, if the number of fragments and queries are large, the LP approach might take too long even if the number of chunks is small. In such cases, the corresponding subproblems can be simplified – e.g., by (i) clustering similar query classes, (ii) considering only large fragments, or (iii) considering only most expensive query classes with a large workload share – or solved heuristically. In particular, the heuristic proposed by Rabl et al. [11] can be used to decompose large subproblems (at a tree's root). If subproblems are sufficiently small the LP approach can be used henceforth (towards a tree's leaves).

B. Numerical Results

In this section, we illustrate the results of our decomposition approach using the TPC-H benchmark. We compare the replication factors of our heuristic to optimal solutions (if applicable) and to solutions of the state-of-the-art heuristic proposed in [11]. To this end, we implemented Rabl's algorithm, see also [12]. Their results are denoted by W^S.

Table III summarizes the solutions of the heuristics and the computation times of the decomposition heuristic for numbers of K between 4 and 16 for one decomposition level. Note, the results were achieved using AMPL (single threaded on a laptop) and a student version of CPLEX 12.5.1.0, restricted to 500 variables and 500 constraints. In our penalty formulation, we used the (normalized) factor $\alpha := 1\,000 \cdot V$, cf. (6).

We observe that results of the decomposition heuristic are *near-optimal*. Compared to the optimal data redundancy factors W^*/V, see Table II, the corresponding factors W/V are just $1\% - 5\%$ larger. The workload limits are also $L = 1/K$.

Remark 1 *The optimal solution for $K = 4, ..., 8$ (TPC-H example), cf. Table II, enables to determine the quality of our solution: Compared to Rabl's solution (W^S, cf. Table III), we observe that our decomposition approach (W) requires $10\% - 23\%$ less data, which compared to optimal solutions reduces the optimality gap by $(W^S - W)/(W^S - W^*) \approx 89\%$. Moreover, the computation times are just single seconds.*

VI. CONCLUSIONS

In this paper, we investigated an NP-hard workload-driven fragment allocation problem, minimizing the overall memory consumption of a replication cluster while maximizing throughput by balancing the load evenly. Our novel heuristic approach decomposes the problem into subproblems which preserve the structure of the original problem but can be solved efficiently using linear programming.

We compared the solutions of our heuristic with optimal solutions (as long as applicable) and solutions of a state-of-the-art heuristic for the TPC-H benchmark. Our heuristic calculates close to optimal allocations, reducing the memory consumption of the cluster significantly compared to the heuristic of Rabl et al. [11].

REFERENCES

[1] J. Lee *et al.*, "Parallel replication across formats in SAP HANA for scaling out mixed OLTP/OLAP workloads," *PVLDB*, vol. 10, no. 12, pp. 1598–1609, 2017.
[2] B. Kemme and G. Alonso, "Don't be lazy, be consistent: Postgres-R, A new way to implement database replication," in *VLDB*, 2000, pp. 134–143.
[3] E. Cecchet, G. Candea, and A. Ailamaki, "Middleware-based database replication: the gaps between theory and practice," in *SIGMOD*, 2008, pp. 739–752.
[4] E. Cecchet *et al.*, "C-JDBC: Flexible database clustering middleware," in *FREENIX@USENIX*, 2004, pp. 9–18.
[5] J. M. Milán-Franco *et al.*, "Adaptive middleware for data replication," in *Middleware*, 2004, pp. 175–194.
[6] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso, "MIDDLE-R: consistent database replication at the middleware level," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 375–423, 2005.
[7] M. Boissier *et al.*, "Analyzing data relevance and access patterns of live production database systems," in *CIKM*, 2016, pp. 2473–2475.
[8] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.
[9] K. P. Eswaran, "Placement of records in a file and file allocation in a computer," in *IFIP*, 1974, pp. 304–307.
[10] L. Dowdy and D. Foster, "Comparative models of the file assignment problem," *ACM Comput. Surv.*, vol. 14, no. 2, pp. 287–313, 1982.
[11] T. Rabl and H. Jacobsen, "Query centric partitioning and allocation for partially replicated database systems," in *SIGMOD*, 2017, pp. 315–330.
[12] S. Halfpap and R. Schlosser, "A comparison of allocation algorithms for partially replicated databases," in *ICDE*, 2019.