LNCS 11860 Journal Subline

Transactions on Large-Scale Data- and Knowledge-Centered Systems XLII

Abdelkader Hameurlain · Roland Wagner
Editors-in-Chief

Springer

Lecture Notes in Computer Science 11860

Founding Editors

Gerhard Goos
Karlsruhe Institute of Technology, Karlsruhe, Germany
Juris Hartmanis
Cornell University, Ithaca, NY, USA

Editorial Board Members

Elisa Bertino
Purdue University, West Lafayette, IN, USA
Wen Gao
Peking University, Beijing, China
Bernhard Steffen
TU Dortmund University, Dortmund, Germany
Gerhard Woeginger
RWTH Aachen, Aachen, Germany
Moti Yung
Columbia University, New York, NY, USA

More information about this series at http://www.springer.com/series/8637

Abdelkader Hameurlain · Roland Wagner (Eds.)

Transactions on Large-Scale Data- and Knowledge-Centered Systems XLII

Springer

Editors-in-Chief
Abdelkader Hameurlain
IRIT, Paul Sabatier University
Toulouse, France

Roland Wagner
FAW, University of Linz
Linz, Austria

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISSN 1869-1994 ISSN 2510-4942 (electronic)
Transactions on Large-Scale Data- and Knowledge-Centered Systems
ISBN 978-3-662-60530-1 ISBN 978-3-662-60531-8 (eBook)
https://doi.org/10.1007/978-3-662-60531-8

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer-Verlag GmbH, DE
part of Springer Nature
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

Preface

This volume contains five fully revised selected regular papers, covering a wide range of very hot topics. The focus of the papers is on privacy-preserving top-k query processing, permissioned distributed ledgers, a tensor data model for logical data independence and data impedance mismatch, question answering based on mutual information and reinforced co-occurrence, and a scalable on-line algorithm that manages configuration changes while providing read-after-write consistency.

We would like to sincerely thank the Editorial Board and the external reviewers for thoroughly refereeing the submitted papers and ensuring the high quality of this volume.

Special thanks go to Gabriela Wagner for her high availability and her valuable work in the realization of this TLDKS volume.

September 2019

Abdelkader Hameurlain
Roland Wagner

Organization

Editorial Board

Reza Akbarinia	Inria, France
Dagmar Auer	FAW, Austria
Djamal Benslimane	Lyon 1 University, France
Stéphane Bressan	National University of Singapore, Singapore
Mirel Cosulschi	University of Craiova, Romania
Dirk Draheim	Tallin University of Technology, Estonia
Johann Eder	Alpen Adria University Klagenfurt, Austria
Anastasios Gounaris	Aristotle University of Thessaloniki, Greece
Theo Härder	Technical University of Kaiserslautern, Germany
Sergio Ilarri	University of Zaragoza, Spain
Petar Jovanovic	Universitat Politècnica de Catalunya, BarcelonaTech, Spain
Dieter Kranzlmüller	Ludwig-Maximilians-Universität München, Germany
Philippe Lamarre	INSA Lyon, France
Lenka Lhotská	Technical University of Prague, Czech Republic
Vladimir Marik	Technical University of Prague, Czech Republic
Jorge Martinez Gil	Software Competence Center Hagenberg, Austria
Franck Morvan	IRIT, Paul Sabatier University, France
Torben Bach Pedersen	Aalborg University, Denmark
Günther Pernul	University of Regensburg, Germany
Soror Sahri	LIPADE, Descartes Paris University, France
A Min Tjoa	Vienna University of Technology, Austria
Shaoyi Yin	Paul Sabatier University, Toulouse, France

Additional Reviewers

Abdelmalek Benzekri	IRIT, Paul Sabatier University, France
Ladjel Bellatreche	ENSMA, Poitiers, France
Guillaume Cabanac	IRIT, Paul Sabatier University, France
Julius Köpke	Alpen Adria University Klagenfurt, Austria
Riad Mokadem	IRIT, Paul Sabatier University, France

Contents

Privacy-Preserving Top-k Query Processing in Distributed Systems 1
Sakina Mahboubi, Reza Akbarinia, and Patrick Valduriez

Trust Factors and Insider Threats in Permissioned Distributed Ledgers: An Analytical Study and Evaluation of Popular DLT Frameworks 25
Benedikt Putz and Günther Pernul

Polystore and Tensor Data Model for Logical Data Independence and Impedance Mismatch in Big Data Analytics 51
Éric Leclercq, Annabelle Gillet, Thierry Grison, and Marinette Savonnet

A General Framework for Multiple Choice Question Answering Based on Mutual Information and Reinforced Co-occurrence 91
Jorge Martinez-Gil, Bernhard Freudenthaler, and A Min Tjoa

Rejig: A Scalable Online Algorithm for Cache Server Configuration Changes. 111
Shahram Ghandeharizadeh, Marwan Almaymoni, and Haoyu Huang

Author Index 135

Privacy-Preserving Top-k Query Processing in Distributed Systems

Sakina Mahboubi[1], Reza Akbarinia[2(✉)], and Patrick Valduriez[2]

[1] University of Batna, Batna, Algeria
[2] INRIA & LIRMM, Univ. Montpellier, Montpellier, France
{reza.akbarinia,patrick.valduriez}@inria.fr

Abstract. We consider a distributed system that stores user sensitive data across multiple nodes. In this context, we address the problem of privacy-preserving top-k query processing. We propose a novel system, called SD-TOPK, which is able to evaluate top-k queries over encrypted distributed data without needing to decrypt the data in the nodes where they are stored. We implemented and evaluated our system over synthetic and real databases. The results show excellent performance for SD-TOPK compared to baseline approaches.

Keywords: Privacy preserving · Top-k query · Distributed system

1 Introduction

Top-k queries are important for many centralized and distributed applications such as information retrieval [32], sensor networks [38], data stream management systems [35], crowdsourcing [7], spatial data analysis [29], social networks [16], etc. A top-k query allows the user to get the k data items that are most relevant to the query.

We consider a distributed system where users can outsource their *sensitive* data and issue top-k queries. The user data are encrypted (for privacy reasons) and distributed (for performance reasons) across multiple nodes. In this context, we address the problem of privacy-preserving top-k query processing.

Privacy preserving top-k query processing is a critical requirement for some distributed applications that outsource *sensitive* data. For example, consider a university that outsources the students database in a public cloud, in Infrastructure-as-a-Service (IaaS) mode, with non-trusted nodes. The database is vertically partitioned (for performance reasons) and encrypted. Then, an interesting top-k query over the encrypted distributed data is the following: return the k students that have the worst averages in some given courses.

There are different approaches for processing top-k queries over *plaintext* (non encrypted) data. One of the best known approaches is TA [14] that works on sorted lists of attribute values. However, there is no efficient solution capable of evaluating efficiently top-k queries over encrypted data in distributed systems with non-trusted nodes.

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
A. Hameurlain and R. Wagner (Eds.): TLDKS XLII, LNCS 11860, pp. 1–24, 2019.
https://doi.org/10.1007/978-3-662-60531-8_1

A naive solution is to retrieve the encrypted database from the distributed system to the user machine that keeps the secret keys, decrypt it, and then evaluate the top-k query over *plaintext* data. This solution is not efficient, because it does not allow us to take advantage of the distributed system power for evaluating queries.

In this paper[1], we propose a system, called SD-TOPK (Secure Distributed TOPK), that encrypts and stores user data in a distributed system, and is able to evaluate top-k queries over the encrypted data. SD-TOPK comes with a novel top-k query processing algorithm that finds a set of encrypted data that is proven to contain the top-k data items. This is done without having to decrypt the data in the nodes where they are stored. In addition, we propose a powerful filtering algorithm that removes the false positives as much as possible without data decryption. We implemented and evaluated the performance of our system over synthetic and real databases. The results show excellent performance for SD-TOPK compared to TA-based approaches. They show the efficiency of our filtering algorithm that eliminates almost all false positives in the distributed system, and reduces significantly the communication cost between the distributed system and the user.

The rest of the paper is organized as follows. Section 2 gives the problem definition. Section 3 describes the architecture of SD-TOPK system. In Sect. 4, we present two TA-based algorithms for top-k query processing over encrypted data. In Sect. 5, we describe the SD-TOPK system, and analyze its security in Sect. 6. Section 7 presents the performance evaluation results. Section 8 discusses related work, and Sect. 9 concludes.

2 Background and Problem Definition

In this section, we give a background about top-k queries, and define the problem which we address.

2.1 Top-k Queries

By a top-k query, the user specifies a number k, and the system should return the k most relevant answers. The relevance degree of the answers to the query is determined by a *scoring function*. A common method for efficient top-k query processing is to run the algorithms over *sorted lists* (also called *inverted lists*) [14]. Let us define them formally.

Let D be a set of n data items, then the sorted lists are m lists $L_1, L_2, ..., L_m$, such that each list L_i contains every data item $d \in D$ in the form of a pair $(id(d), s_i(d))$ where $id(d)$ is the identification of d and $s_i(d)$ is a value that denotes the *local score* (attribute value) of d in L_i. The data items in each list L_i are sorted in descending order of their local scores. For example, in a relational table, each sorted list represents a sorted column of the table where the local score of a data item is its attribute value in that column.

[1] This journal paper is a major extension of [23].

Let f be a scoring function given by the user in the top-k query. For each data item $d \in D$ an *overall score*, denoted by $ov(d)$, is calculated by applying the function f on the local scores of d. Formally, we have $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$.

The result of a top-k query is the set of k elements that have the highest overall scores among all elements of the database. In this work, we assume that the scoring function is in the class of linear functions with positive coefficients (denoted by $LFPC$).

The sorted lists model can be used for top-k query processing in many applications. For example, suppose we want to find the top-k tuples in a relational table according to some scoring function over its attributes. To answer such query, it is sufficient to have a sorted list for the values of each attribute, and return the k tuples whose overall scores in the lists are the highest.

2.2 Distributed System and Adversary Model

We suppose that the sorted lists are stored in the nodes of a *distributed system*. We make no specific assumption about the distributed system architecture which can be very general, *e.g.*, a cluster of nodes. Formally, let P be the set of the nodes in the distributed system. Each sorted list L_i is kept in a node $p \in P$. We call p the *owner* of L_i.

We consider the *honest-but-curious* adversary model for the nodes of the distributed system. In this model, the adversary is inquisitive to learn the sensitive data without introducing any modification in the data or protocols. This model is widely used in many privacy preserving solutions [21].

2.3 Problem Statement

The problem we attack in this paper is top-k query processing over encrypted data in distributed systems.

Let D be a database composed of n data items. We want to encrypt the data items contained in D, and store the encrypted data items in a distributed system. Then, our goal is to develop a distributed algorithm A that given any top-k query q (including a scoring function f) returns the k data items that have the highest overall scores with regard to f. This should be done without decrypting the data items in the nodes of the distributed system, while minimizing the response time and the communication cost of the query execution.

3 SD-TOPK System Architecture

The architecture of SD-TOPK has two main components (see Fig. 1):

– **Trusted client.** It is responsible for encrypting the user data, decrypting the results and controlling the user accesses. The security keys used for data encryption/decryption are managed by this part of the system. When a query

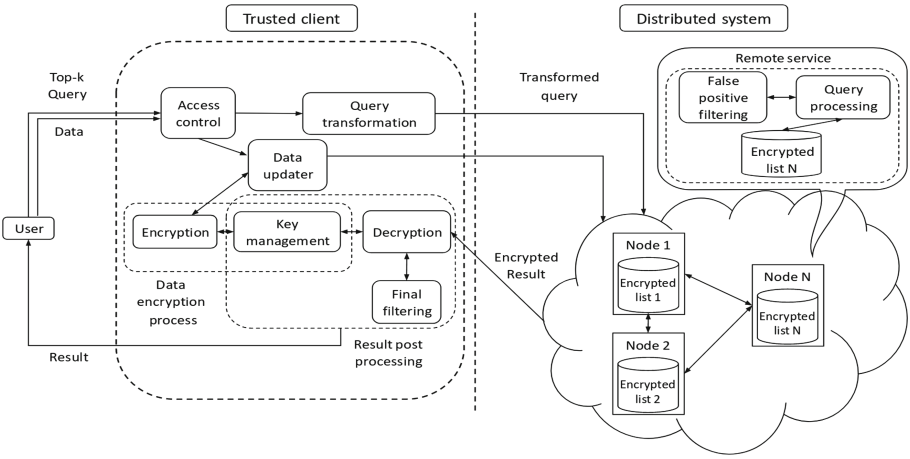


Fig. 1. SD-TOPK architecture

is issued by a user, the trusted client checks the access rights of the user. If the user does not have the required rights to see the query results, then her demand is rejected. Otherwise, the query is transformed to a query that can be executed over the encrypted data.
Note that the trusted client component should be installed in a trusted location, *e.g.*, the machine(s) of the person/organization that outsources the data.

– **Remote service.** It is installed in the nodes of the distributed system, and is responsible for storing the encrypted data, executing the queries provided by the trusted client, and returning the results. This component does not keep any security key, thus cannot decrypt the encrypted data in the distributed system.

In the trusted client of SD-TOPK, we use two types of schemes. *Deterministic* encryption is an encryption scheme that, for two equal inputs, generates the same ciphertexts. With a *probabilistic* encryption, for the same plaintexts different ciphertexts can be generated, but the decryption function returns the same plaintext for them [26]. The details of data encryption in SD-TOPK are described in Sect. 5.1.

4 TA-Based Algorithms

In this section, we present two basic approaches based on the TA algorithm for top-k query processing over encrypted data in distributed environments: Remote-TA and Block-TA. Our main contribution, i.e. SD-TOPK, will be presented in the next section.

4.1 Data Encryption

To be able to execute TA-based algorithms over encrypted data, the trusted client stores the database in the nodes of the distributed system as follows. It encrypts the pairs $\langle d, s_i(d)\rangle$ of the sorted lists using two encryption schemes: (1) *deterministic* to encrypt data identifier d; (2) *probabilistic* to encrypt local score of the data, *i.e.*, $s_i(d)$. The encrypted pairs are sorted in the same order as their initial order. After encrypting the pairs, the trusted client sends each encrypted sorted list to one node of the distributed system, which is called the *owner* of the list.

4.2 Remote-TA

Given a top-k query containing a number k and a scoring function f, Remote-TA proceeds as follows:

1. The trusted client asks the list owners to return the encrypted pairs (encrypted data id and score) which are in position j of the lists (initially $j = 1$). The list owners return the asked data.
2. The trusted client decrypts the received encrypted scores and calculates a threshold TH by applying the scoring function on the decrypted scores.
3. Let S be the set of encrypted data items returned from the position j of the lists. The trusted client demands the list owners to return the encrypted scores of data items in S. Each list owner does random access in its list to find the encrypted scores of each data item in S, then sends them to the trusted client.
4. Trusted client decrypts each returned data item d and calculates its overall score $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$. Then, it checks if among the yet received data items there are at least k data items that have an overall score greater than or equal to TH. If this is the case, then it stops the algorithm and returns the k received data items that have the highest overall scores to the user. Otherwise, it increases j by one and restarts from step 1.

4.3 Block-TA

Block-TA is an improvement of the Remote-TA algorithm where the encrypted data items are read block by block. Indeed, in order to minimize the communication cost, in the first step of Block-TA, the trusted client asks blocks of predefined size from the list owners. After decrypting the data items of the block, the trusted client computes the threshold by applying the scoring function on the scores of the last data items in the blocks, and stops if among yet received data items there are at least k data items with overall scores higher than or equal to the threshold. Otherwise it retrieves the next block, and so on.

Example. Consider Table 1 that represents a database composed of three encrypted lists. There are three sorted lists, and each list L_i is stored in a node.

Table 1. Example of an encrypted database

List 1		List 2		List 3	
encrypted data item	encrypted local score	encrypted data item	encrypted local score	encrypted data item	encrypted local score
$E(d_3)$	$E(30)$	$E(d_3)$	$E(29)$	$E(d_6)$	$E(27)$
$E(d_1)$	$E(27)$	$E(d_6)$	$E(28)$	$E(d_3)$	$E(25)$
$E(d_6)$	$E(26)$	$E(d_2)$	$E(26)$	$E(d_2)$	$E(22)$
$E(d_5)$	$E(24)$	$E(d_1)$	$E(24)$	$E(d_5)$	$E(21)$
$E(d_8)$	$E(20)$	$E(d_7)$	$E(21)$	$E(d_1)$	$E(20)$
$E(d_2)$	$E(15)$	$E(d_4)$	$E(19)$	$E(d_9)$	$E(18)$
$E(d_4)$	$E(14)$	$E(d_5)$	$E(16)$	$E(d_8)$	$E(17)$
$E(d_7)$	$E(12)$	$E(d_9)$	$E(13)$	$E(d_7)$	$E(14)$
$E(d_9)$	$E(11)$	$E(d_8)$	$E(10)$	$E(d_4)$	$E(11)$
...	...	...	...	...	...

The user asks for the top-4 data items in the database with SUM as scoring function.

Let us run Block-TA on the database of Table 1 by using blocks of size 4. The trusted client asks the list owners to return the 4 first data items in each list. The owner of the list L_1 returns d_3, d_1, d_6 and d_5. It also returns $E(24)$ which is the encrypted score of the last data item in the block. This is used later to calculate the threshold. The owner of L_2 returns d_3, d_6, d_2, d_1 and $E(24)$. The owner of L_3 returns d_6, d_3, d_2, d_5 and $E(21)$. Then, the trusted client decrypts the received data and calculates the threshold $TH = 24 + 24 + 21 = 69$. The client asks for the encrypted score of the returned data items d_1, d_2, d_3, d_5 and d_6. When it gets the asked scores, it decrypts them and calculates the overall score of each data item: $ov(d_1) = 71$, $ov(d_2) = 63$, $ov(d_3) = 84$, $ov(d_5) = 61$, $ov(d_6) = 81$. The client finds that only d_1, d_3 and d_6 have an overall score greater than or equal to the threshold TH, so it asks the next block of four data items in each list. The owner of L_1 returns d_8, d_2, d_4, d_7 and $E(12)$. The owner of L_2 returns d_7, d_4, d_5, d_9 and $E(13)$, and that of L_3 returns d_1, d_9, d_8, d_7 and $E(14)$. The trusted client calculates the threshold $TH = 12 + 13 + 14 = 39$. Then, the client asks each node to return the encrypted score of the data items and calculates their overall score: $ov(d_4) = 44, ov(d_7) = 47, ov(d_8) = 47, ov(d_9) = 42$. The trusted client finds that there are at least 4 data items with overall scores greater than or equal to TH. Thus, it stops communicating with the list owners, and returns to the user the data items d_1, d_2, d_3, d_6 that have the biggest overall score among the data items received from the list owners.

5 SD-TOPK

The TA-based algorithms, presented in the previous section, evaluate correctly the top-k queries over encrypted data. But, as shown by our experiments reported in Sect. 7, there may be a high number of false positives which are sent from the distributed system nodes to the client, and this renders these algorithms inefficient in practice. In this section, we present the SD-TOPK system, designed for efficient processing top-k queries over encrypted data in distributed systems. As shown by our experiments, SD-TOPK is much more efficient than the TA-based algorithms.

The rest of this section is organized as follows. We first describe SD-TOPK's method for encrypting the data items and storing them in the distributed system. Afterwards, we propose an efficient algorithm for processing top-k queries over the encrypted data. Then, we propose an algorithm for removing the false positives from the results of the top-k query processing algorithm, without decrypting the data. Afterwards, we discuss how we can update the encrypted data in the distributed system. Finally, we propose a technique to enforce the security of data outsourcing in SD-TOPK.

5.1 Data Encryption and Outsourcing

Before outsourcing a database, SD-TOPK creates sorted lists for all important attributes, *i.e.*, those that may be used in the top-k queries. Then, each sorted list is partitioned into buckets. There are several methods for partitioning a sorted list, for example dividing the attribute domain of the list to almost equal intervals or creating buckets with equal sizes. In the current implementation of our system, we use the latter method, *i.e.*, we create buckets with almost the same size where the bucket size is configurable by the system administrator.

Let $b_1, b_2, ..., b_t$ be the created buckets for a sorted list L_j. Each bucket b_i has a lower bound, denoted by $min(b_i)$, and an upper bound, denoted by $max(b_i)$. A data item d is in the bucket b_i, if and only if its local score (attribute value) in the list L_j is between the lower and upper bounds of the bucket, *i.e.*, $min(b_i) \leq s_j(d) < max(b_i)$.

We use two types of encryption schemes (methods) for encrypting the data item ids and the local scores of the sorted lists: *deterministic* and *probabilistic*. The *deterministic* scheme is used to encrypt the ID of the data items. This allows us to have the same encrypted ID for each data item in all sorted lists.

We use the *probabilistic* scheme to encrypt the local scores (attribute values) of data items. By using *probabilistic* encryption, if two data items have the same local scores in a sorted list, their encrypted scores may be different. This protects the scores against the frequency attacks [24] that decrypt order-preserved encrypted lists given auxiliary information about the database, such as the distribution of the plaintext values.

After encrypting the data IDs and local scores of each list L_i, the trusted client puts them in their bucket (chosen based on the local score). Then, the trusted client sends the buckets of each sorted list to one node in the distributed

system. The buckets are stored in the nodes according to their lower bound order. However, there is no order for the data items inside each bucket, *i.e.*, *the position of the data items inside each bucket is chosen randomly.* This prevents the nodes to know the order of data items inside the buckets.

5.2 Top-k Query Processing Algorithm

The main idea behind top-k query processing in SD-TOPK is to use the bucket boundaries and a new technique to decide when to stop reading the encrypted data from the lists.

For each top-k query, one of the nodes of the distributed system performs the coordination between the nodes to execute the query. The coordinator may be the node that initially receives the user's query or it can be randomly chosen among the system nodes.

Let us describe our top-k query processing algorithm. Given a top-k query with a number k and a scoring function f that is linear with positive coefficients, *i.e.*, it is in the form of $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$. SD-TOPK chooses a node as *coordinator*, and then the following steps are performed to answer the query:

1. The coordinator broadcasts the query in parallel to the nodes, and asks each node to return the buckets that contain the k first data items in its list. Each node returns the encrypted identifier of the first k data items, as well as the lower bound of their including buckets.
2. For each returned data item d, the coordinator calculates its *minimum overall score* defined as follows: $ov_{min}(d) = f(v_1(d), v_2(d), ..., v_m(d))$ where $v_i(d)$ is the lower bound of the bucket that contains d in the list L_i. If d has not been returned to the coordinator by the owner of a list L_j then $v_j(d) = 0$.
3. The coordinator sorts the received data items according to their minimum overall score, and chooses the data item d' that has the k^{th} minimum overall score denoted by δ. Then, it uses the minimum overall score of d' to calculate a threshold θ as follows: $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$ where $a_1, \ldots, a_m$ are the coefficients in the scoring function.
4. The coordinator broadcasts θ in parallel to the nodes. Each node returns to the coordinator the buckets that have upper bounds greater than or equal to θ.
5. Let Y be the set of all data items that are sent to the coordinator by at least one node. We call Y the set of *candidate items.* The coordinator sends the encrypted id of all data items contained in Y to the nodes, and they return the encrypted score of each data item contained in Y.
6. Finally, the coordinator returns to the trusted client the candidate items and their encrypted local scores.

When the trusted client receives the candidate items, it decrypts them using the secret keys. Then, it calculates for each candidate d its overall score, extracts the k data items that have the highest overall scores, and returns them to the user.

The following theorem shows that the output of the above algorithm contains the encrypted top-k data items.

Theorem 1. *Given a top-k query with a scoring function f that is linear with positive coefficients, then the output of the top-k algorithm of SD-TOPK contains the encrypted top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$. Let Y be the output of the algorithm, *i.e.*, the set of candidate items. To prove the theorem, it is sufficient to show that each data item d that has not been sent to the coordinator in the 4th step of the algorithm, has an overall score less than or equal to the overall score of at least k data items in Y. Let θ be the threshold value that is sent to the nodes in the 4th step of the algorithm. For each list L_i, let s_i be the local score of d in the list L_i. The overall score of d is computed as $ov(d) = a_1s_1 + \cdots + a_ms_m$. Since d has not been sent to the coordinator, from the 4th step of the algorithm we know that $s_i < \theta$. Thus, we have $ov(d) < a_1 \times \theta + \cdots + a_m \times \theta = \sum_{i=1}^{m} a_i \times \theta$. From the 3rd step of the algorithm, we know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have $ov(d) < \delta$. In other words, the overall score of d is less than the minimum overall score of the data item d' that is the k^{th} data item found in the 3rd step of the algorithm. Therefore, the overall score of d is less than at least k data items found by the top-k algorithm of SD-TOPK, so d cannot be among the top-k results.

5.3 Filtering Algorithm

In the set Y returned by the top-k query processing algorithm, in addition to the top-k results there may be false positives. Below, we propose a filtering algorithm to eliminate most of them in the distributed system nodes, without decrypting the data items.

Given the set of candidate data items Y, the filtering algorithm executed by the coordinator proceeds as follows:

1. Calculate the *minimum overall score* of all candidate data items, sort them according to their minimum overall score, and take the k^{th} *minimum overall score* denoted by δ_2.
2. Calculate the *maximum overall score* of all candidate data items, and eliminate those with *maximum overall score* less than $< \delta_2$. The *maximum overall score* of a data item d is computed as follows: $ov_{max}(d) = f(v_1(d), v_2(d), ..., v_m(d))$ where $v_i(d)$ is the upper bound of the bucket that contains d in the list L_i. If d has not been returned to the coordinator by the node that keeps L_i then $v_i(d)$ is equal to the lower bound of the last bucket received from that node.

The above algorithm eliminates almost all false positives (see the experimental results on filtering rate in Sect. 7.8), and by doing that it improves significantly the response time of the queries because the eliminated false positives do not need to be communicated to the trusted client and should not be decrypted.

To prove the correctness of the filtering algorithm, we need the following lemmas.

Lemma 1. *Given a top-k query with a scoring function f that is linear with positive coefficients. The minimum overall score of any data item d is less than or equal to its overall score.*

Proof. The minimum overall score of a data item d is calculated by applying the scoring function on the lower bound of the buckets in which d is involved. Let b_i be the bucket that contains d in the list L_i. Let s_i be the local score of d in L_i. Since $d \in b_i$, its local score is higher than or equal to the lower bound of b_i, i.e. $min(b_i) \leq s_i$. Since f is monotonic, we have $f(min(b_1), ..., min(b_m)) \leq f(s_1, ..., s_m)$. Therefore, the minimum overall score of d is less than or equal to its overall score.

Lemma 2. *Given a top-k query with a scoring function f that is linear with positive coefficients, then the maximum overall score of any data item d is greater than or equal to its overall score.*

Proof. The proof can be done in a similar way as Lemma 1

The following theorem shows that the filtering algorithm works correctly, i.e., the removed data are only false positives.

Theorem 2. *Given a top-k query with a scoring function f that is linear with positive coefficients, then any data item removed by the filtering algorithm cannot belong to the top-k results.*

Proof. The proof can be done by considering the fact that any removed data item d has a maximum overall score that is lower than the minimum overall score of at least k data items. Thus, according to Lemmas 1 and 2 the overall score of d is less than or equal to that of at least k data items that are not removed. Therefore, we can eliminate d.

5.4 Example

To illustrate SD-TOPK, we use the database shown in Table 2 with a top-4 query and SUM as scoring function. In this example, we suppose that a node n_0 is the coordinator. It sends a messages to all list owners (e.g., n_1, n_2, n_3) and asks for the 4 first data items in each list (since $k = 4$), and the minimum of the buckets in which they are. The node n_1 returns $\langle d_1, 24.6\rangle$ $\langle d_3, 24.6\rangle$ $\langle d_6, 24.6\rangle\langle d_2, 14.8\rangle$. The node n_2 returns $\langle d_6, 25.5\rangle$ $\langle d_3, 25.5\rangle$ $\langle d_2, 25.5\rangle$ $\langle d_1, 18\rangle$ and the node n_3 returns $\langle d_2, 21.9\rangle$ $\langle d_3, 21.9\rangle$ $\langle d_6, 21.9\rangle$ $\langle d_5, 17.7\rangle$. The coordinator calculates the minimum overall score of the returned data items by using the minimum of their buckets. It finds that $ov_{min}(d_1) = 24.6 + 18 = 42.6$, $ov_{min}(d_2) = 14.8 + 25.5 + 21.9 = 62.2$, $ov_{min}(d_3) = 24.6 + 25.5 + 21.9 = 72$, $ov_{min}(d_5) = 17.7$ and $ov_{min}(d_6) = 72$. After sorting the minimum overall scores, the coordinator finds that the 4th

Table 2. Encrypted database, with 3 data items in each bucket. The encrypted scores inside buckets are not sorted. The boundaries (minimum and maximum) of buckets are shown below

List 1			List 2			List 3		
bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score
B_{11}	$E(d_1)$	$E(27)$	B_{21}	$E(d_6)$	$E(28)$	B_{31}	$E(d_2)$	$E(22)$
B_{11}	$E(d_3)$	$E(30)$	B_{21}	$E(d_3)$	$E(29)$	B_{31}	$E(d_3)$	$E(25)$
B_{11}	$E(d_6)$	$E(26)$	B_{21}	$E(d_2)$	$E(26)$	B_{31}	$E(d_6)$	$E(27)$
B_{12}	$E(d_2)$	$E(15)$	B_{22}	$E(d_1)$	$E(24)$	B_{32}	$E(d_5)$	$E(21)$
B_{12}	$E(d_8)$	$E(20)$	B_{22}	$E(d_7)$	$E(21)$	B_{32}	$E(d_1)$	$E(20)$
B_{12}	$E(d_5)$	$E(24)$	B_{22}	$E(d_4)$	$E(19)$	B_{32}	$E(d_9)$	$E(18)$
B_{13}	$E(d_4)$	$E(14)$	B_{23}	$E(d_5)$	$E(16)$	B_{33}	$E(d_8)$	$E(17)$
B_{13}	$E(d_9)$	$E(11)$	B_{23}	$E(d_9)$	$E(13)$	B_{33}	$E(d_7)$	$E(14)$
B_{13}	$E(d_7)$	$E(12)$	B_{23}	$E(d_8)$	$E(10)$	B_{33}	$E(d_4)$	$E(11)$
...	...	...	...	...	...	...	...	...

Table 3. Bucket boundaries

List 1			List 2			List 3		
bucket ID	min	max	bucket ID	min	max	bucket ID	min	max
B_{11}	24.6	32	B_{21}	25.5	31	B_{31}	21.9	28
B_{12}	14.8	24.1	B_{22}	18	24.1	B_{32}	17.7	21.5
B_{13}	10.7	14.2	B_{23}	9	16.5	B_{33}	10	17.3

minimum overall score is 42.6 (that of d_3), so it sets $\delta = 42.6$. Then, it calculates $\theta = \delta/3 = 14.2$. Afterwards, it asks each node to return the encrypted id of the data items that are in buckets b_i such that $max(b_i) \geq \theta$. The data items received from list owners are called candidate data items. The coordinator calculates for the candidate items their minimum overall score: $ov_{min}(d_1) = 60.3$, $ov_{min}(d_2) = 62.2$, $ov_{min}(d_3) = 72$, $ov_{min}(d_4) = 38.7$, $ov_{min}(d_5) = 41.5$, $ov_{min}(d_6) = 72$, $ov_{min}(d_7) = 38.7$, $ov_{min}(d_8) = 33.8$ and $ov_{min}(d_9) = 37.4$. It also calculates $\delta_2 = 60.3$, that is the kth minimum overall score among candidate data items (defined in the filtering algorithm). Then, it calculates the maximum overall scores of the candidate data items: $ov_{max}(d_1) = 77.6$, $ov_{max}(d_2) = 83.1$, $ov_{max}(d_3) = 91$, $ov_{max}(d_4) = 55.6$, $ov_{max}(d_5) = 62.1$, $ov_{max}(d_6) = 91$, $ov_{max}(d_7) = 55.6$, $ov_{max}(d_8) = 57.9$ and $ov_{max}(d_9) = 52.2$. According to the filtering algorithm, the coordinator eliminates the data items that have a maximum overall score less than 60.3. Then, it remains five data items in the set of candidate items $Y = \{d_1, d_2, d_3, d_5, d_6\}$. The coordinator asks the list owners to return all encrypted scores of the candidate items and sends them to the trusted client. When the client receives the data items, it decrypts them and calculates their real overall score: $ov(d_1) = 71, ov(d_2) = 63, ov(d_3) = 84, ov(d_5) = 61, ov(d_6) = 81$. Finally,

the trusted client finds that the top-4 data items are d_1, d_2, d_3 and d_6. It returns them to the user who had issued the query (Table 3).

5.5 Update Management

In our system, updating a data item d in the nodes is done by deleting the old encrypted scores (attribute values) of d and then inserting its new scores.

To delete the old encrypted scores of d from the outsourced database, the trusted client encrypts the ID of d using the key that has been used for encrypting the data IDs, and then asks the nodes of the distributed system to find the encrypted ID in the buckets of their lists and then remove the pairs encrypted score and encrypted ID of d from the lists.

Inserting the new scores of d is done as follows. The trusted client uses the metadata of the buckets (*i.e.*, the lower and upper bounds), and for each list L_i, it calculates the bucket of the list to which the data score s_i should be stored. Let b_i be the corresponding bucket of s_i. The trusted client encrypts the ID and scores of d by using the encryption schemes that are used for encrypting the ID and scores. Then, it asks the nodes to put the encrypted ID and encrypted scores of d in the corresponding buckets.

5.6 Obfuscating Bucket Boundaries

A drawback of the basic version of SD-TOPK, presented until now, is that the limits (i.e., lower and upper bounds) of the buckets are disclosed to the nodes of the distributed system. To strengthen the security of our system, we change the bucket limits as follows. We choose two random numbers a and c. These numbers must be kept secret in the trusted client. Before encrypting the database, the lower and upper bounds of each bucket b_i are obfuscated (modified) as follows:

$$min(b_i) := min(b_i) \times a + c \tag{1}$$

$$max(b_i) := max(b_i) \times a + c \tag{2}$$

Thus, the trusted client multiplies the lower (upper) bounds by the secret number a, and then adds the secret number c to the result. These obfuscated bucket limits are sent to the nodes of distributed system together with the encrypted IDs and scores. By the above strategy, we can hide the limits of the buckets from the nodes.

The following theorem proves that SD-TOPK works correctly if it uses the obfuscated lower bounds.

Theorem 3. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Eq. 1, then the output of SD-TOPK will involve the top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$. Let Y be the output of SD-TOPK algorithm, *i.e.*, the set of candidate items. We show that

each data item d that has not been sent to the coordinator by the list owners, has an overall score that is less than or equal to the overall score of at least k data items involved in Y. Let b_i be the bucket that contains the data d in the list L_i, and thus $max(b_i) * a + c$ is the new (modified) upper bound of b_i. From the 4^{th} step of Sd-TOPK, we know that in each list L_i we have $max(b_i) * a + c < \theta$. Thus, we have $a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c) < \sum_{i=1}^{m} a_i \times \theta$. We know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have the following equation:

$$a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c) < \delta \tag{3}$$

Now, let d' be the data item that has the k^{th} minimum overall score in the 3rd step of SD-TOPK. In each list L_i, let b'_i be the bucket that contains d' in L_i, and thus $min(b'_i) * a + c$ is the new (modified) lower bound of b'_i. From the 3rd step of the algorithm, we know that the minimum overall score of d' (computed by using the obfuscated buckets) is equal to δ. Thus, we have $a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) = \delta$. Thus, we have the following equation:

$$a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) = \delta \tag{4}$$

By comparing Eqs. 3 and 4, we have: $a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) > a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c)$. Since the numbers a and c are positive, we can write: $a_1 \times (min(b'_1) + \cdots + a_m \times min(b'_m) > a_1 \times max(b_1) + \cdots + a_m \times max(b_m)$. This means that the minimum overall score of the data item d' is higher than the maximum overall score of d. In other words, the data item d could not be among the top-k results.

The following theorem shows that the filtering algorithm works correctly, if it uses the obfuscated bucket limits.

Theorem 4. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Eq. 1, then the filtering algorithm does not remove any top-k result.*

Proof. Let Y be the output of SD-TOPK algorithm, and d' be the data item that has the k^{th} minimum overall score among the data items in the 3rd step of SD-TOPK. In each list L_i, let b'_i and s'_i be the bucket and local score of d'_i in the list.

We do the proof by contradiction. We assume a top-k data item d has been removed by the filtering algorithm, and show that this assumption yields to a contradiction. Let b_i and s_i be the bucket and local score of d_i in the list L_i. Since, d has been removed from the list, its maximum overall score using the modified limits is lower than or equal to minimum overall score of d'. Thus, we have: $a_1 \times (max(b_1) \times a + c), ..., a_m \times (max(b_m) \times a + c) \leq a_1 \times (min(b'_1) \times a + c), ..., a_m \times (min(b'_1) \times a + c)$. Since the parameters a and c are positive, we have: $a_1 \times max(b_1), ..., a_m \times max(b_m) \leq a_1 \times min(b'_1), ..., a_m \times min(b'_1)$. This means that the maximum overall score of the data item d is lower than the minimum overall score of d'. Thus, d cannot be a top-k result.

6 Security Analysis and Improvement

In this section, we analyze the different types of information that can be leaked to the adversary (the nodes of the distributed system), and for each type of leakage, we propose some techniques to reduce the risk of disclosing sensitive data.

6.1 Partial Order Leakage

In SD-TOPK, we use the bucketization technique for managing the data in the distributed system. Inside the buckets, no information is leaked because the data items are not ordered and the local scores are encrypted using a probabilistic scheme. But a partial order is leaked about the data items that are in different buckets (since the buckets are ordered).

Even a partial order leakage may help the adversary to obtain rough information about the sensitive data of individuals if he has some background information about the data. For example, if the adversary A knows that the age of a target person u is very high, then A may find the bucket containing u in the list corresponding to age (*i.e.*, the first bucket of the list). Then, by guessing the ID of u in the bucket (e.g., if the size of the bucket is too small), A may find the bucket of u in the salary's list, and then estimate her salary with some confidence probability. *We show that this probability (i.e., the risk of privacy violation) is very low, when the size of buckets is not small.*

Let u be an individual (data item) in the database, and assume that the adversary A knows the value of u in some attribute a. We want to compute the confidence probability that A finds the bucket containing u's value in a sensitive attribute s. Let us denote this confidence probability by $P(b_{s,u}|a)$. We assume that if A finds the bucket of u in the list representing s, then he can make a good estimation of u's value, *e.g.*, using some background knowledge about the values of attribute s.

To find the bucket of u in the sensitive attribute s, the adversary A needs to perform the following steps: (1) guessing the lists that represent a and s; (2) finding the bucket of u in the list representing a; (3) guessing the ID of u in the found bucket; (4) searching u's ID in the list representing s, and finding its bucket.

Let $P(L_1 = a \wedge L_2 = s)$ be the probability that A guesses correctly the lists representing the attributes a and s. Let m be the number of lists in the database. In our system, the metadata of sorted lists (*e.g.*, their identification) is encrypted, and they have the same size and format. Thus, the probability of finding the correct list of an attribute is $\frac{1}{m}$. Therefore, the probability of correctly guessing the lists representing both attributes a and s is:

$$P(L_1 = a \wedge L_2 = s) = \frac{1}{m \times (m-1)} \tag{5}$$

If the adversary A finds correctly the list representing a, then we assume that A is able to find the bucket containing u by using the background knowledge

about the value of u in a (and some statistical information). After finding the bucket, say b, the adversary needs to guess the ID of u. Let $size(b)$ be the number of encrypted values in the bucket b. Then, the probability of finding u' ID in the bucket b, denoted as $P(ID = u)$, is:

$$P(ID = u) = \frac{1}{|size(b)|} \tag{6}$$

If A guesses correctly the ID of u in the bucket b, then he can find the bucket containing the ID in the list representing the attribute s (if he guesses correctly the list of s), and then he can roughly estimate the u's value in s.

The following theorem provides a formula that calculates the probability that an adversary finds the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a.

Theorem 5. *Let s be a sensitive attribute, $size(b)$ be the size of the buckets, and m be the number of sorted lists (attributes). Let $P(b_{s,u}|a)$ be the probability that the adversary detects correctly the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a. Then, $P(b_{s,u}|a)$ is:*

$$P(b_{s,u}|a) \leq \frac{1}{size(b) \times m \times (m-1)} \tag{7}$$

Proof. The proof can be done using Eqs. 5 and 6.

The above theorem shows that when the size of the buckets is not small, the probability of privacy violation is negligible.

When the bucket size is one (which is equivalent of preserving the total order), the risk of privacy violation is the highest. We advise at least the size of 10 for the buckets, in order to make the privacy violation risk very low. For example, if the number of sorted lists (*i.e.*, attributes) is $m = 5$, with the bucket size of 10, the privacy violation risk $P(b_{s,u}|a)$ is less than 0.005.

However, note that choosing very big buckets increases the response time of query processing (see Sect. 7.5). Therefore, the size of the buckets should be taken based on the user privacy requirements (*e.g.*, the maximum acceptable probability of privacy violation) and performance (*e.g.*, response time).

6.2 Obfuscating the Number of Asked Results

In the basic version of our approach, the information about the number of asked results, *i.e.*, k, is disclosed to the cloud provider. We can obfuscate this information as follows. The trusted client generates a random integer s between 0 and a predefined (small) value, and adds it to k. Then, it sends $k' = k + s$ to the cloud as the number of required results. After receiving the encrypted results from the cloud, the trusted client filters the result set and sends only k results to the user.

6.3 Obfuscating the Database Size

Another information which is leaked is the database size (*i.e.*, the number of tuples). This leakage can be avoided by adding dummy tuples to the database before sending it to the cloud. But, we have to be careful not to add dummy tuples which could be returned to the user as a result of top-k queries. For this, we can proceed as follows. Let n' be the number of dummy data items that we want to add to the lists. In each list L_i, let s_i be the last local score in the list. We generate n' random data IDs. Then, for each list L_i, we generate n' random scores smaller than s_i, and assign them randomly to the n' data IDs. Afterwards, we add the generated data IDs and their local scores to the sorted lists. Since the local scores of the dummy data items are smaller than any real data item, they have no chance to be returned as a result of top-k queries.

7 Performance Evaluation

In this section, we first describe the experimental setup, and then present the performance evaluation results.

7.1 Setup

We implemented SD-TOPK and performed experiments on real and synthetic datasets. As in some previous work on privacy (*e.g.*, [21]), we use the Gowalla database, which is a location-based social networking dataset collected from users locations. The database contains 6 million tuples where each tuple represents user number, time, user geographic position, etc. In our experiments, we are interested in the attribute time, which is the second value in each tuple. As in [21], we decomposed this attribute into 6 attributes (year, month, day, hour, minute, second), and then created a database with the values of those attributes. In addition to the real dataset, we have also generated random datasets using uniform and Gaussian distributions.

We compared SD-TOPK with the two TA-Based algorithms: *Remote-TA* and *Block-TA*.

In the experiments, the number of nodes is equal to the number of lists, *i.e.*, each node stores one of the lists. The coordinator of SD-TOPK is one of the nodes of the system (randomly chosen).

We study the effect of several parameters: (1) n: the number of data items in the database; (2) m: the number of lists; (3) k: the number of required top items; (4) $bsize$: the number of data items in the buckets (or blocks) in SD-TOPK and Block-TA. The default value for n is 2M items. Unless otherwise specified, m is 5, k is 50, and $bsize$ is 10. The default database is the synthetic uniform database, and the latency of the messages is around 50 ms.

To evaluate the performance of SD-TOPK, we measured the following metrics:

- **Response time:** includes top-k query processing time, communication time, filtering time, and the result post-processing time (*e.g.*, decryption).

– **Filtering rate:** the number of false positives eliminated by the filtering algorithm in the distributed system.
– **Communication cost:** we measure two metrics: (1) the number of messages communicated between the nodes to answer a top-k query; (2) the total number of bytes communicated to answer a top-k query.

7.2 Effect of Database Size

In this section, we compare the response time of SD-TOPK, Remote-TA and Block-TA, while varying the number of data items, *i.e.*, n.

Figure 2 shows how response time evolves, with increasing n, while the other parameters are set as default values described in Sect. 7.1. Note that the results are shown in logarithmic scale. The response time of all approaches increases with increasing the database size. SD-TOPK is the best; its response time is at least two orders of magnitude better than the other algorithms. This high difference between SD-TOPK and TA-based algorithms is mainly due to the high number

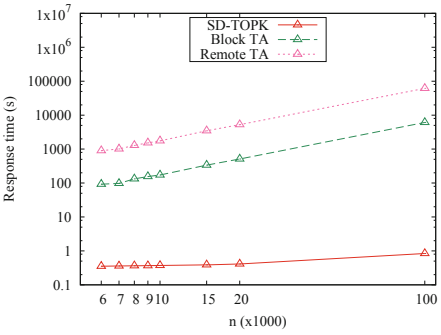


Fig. 2. Response time vs. number of database tuples

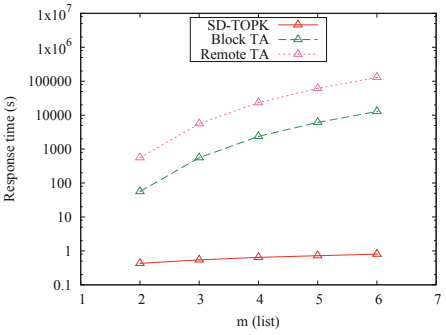


Fig. 3. Response time vs. number of lists

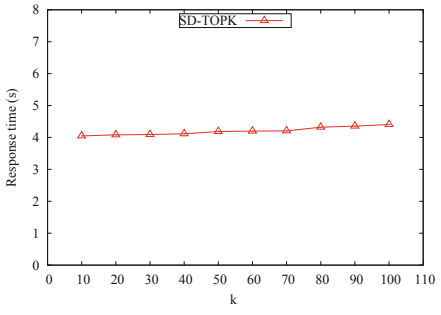


Fig. 4. Response time vs. k

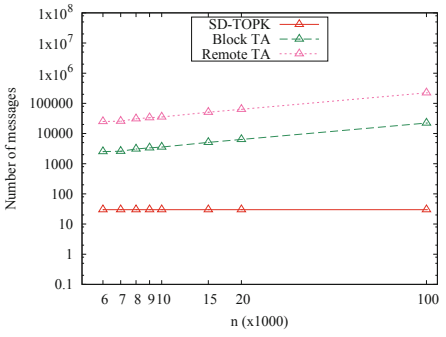


Fig. 5. Number of communicated messages vs. number of database tuples

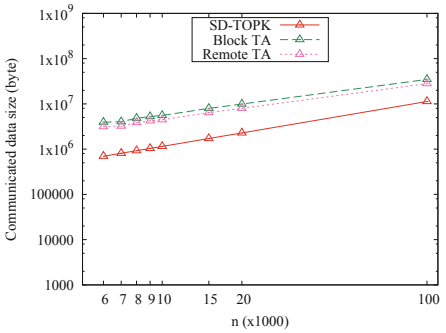


Fig. 6. Size of communicated data (in bytes) vs. number of database tuples

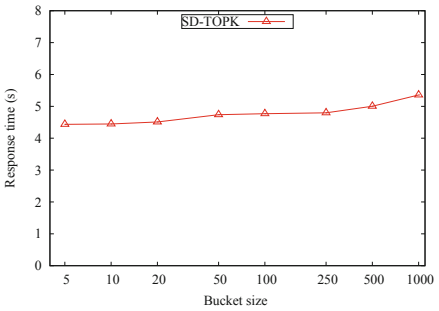


Fig. 7. Response time vs. bucket size

of encrypted data items that should be decrypted by TA-based algorithms in the trusted client, and also the messages needed for communicating them. Block-TA performs better than Remote-TA, because of reading the lists in blocks, thus it needs less number of messages.

7.3 Effect of the Number of Lists

Figure 3 shows the response time of SD-TOPK and TA-based algorithms when varying m (i.e., the number of attributes in the scoring function), and the other parameters set as default values. We observe that the response time of SD-TOPK increases slightly comparing to Remote-TA and Block-TA when the number of lists increases. The reason is that when we increase the number of lists, more data (sent by the nodes) should be processed by the coordinator for finding the candidate items.

7.4 Effect of k

Figure 4 shows the response times of SD-TOPK with increasing k, and the other parameters set as default values. We observe that with increasing k the response time increases slightly. The reason is that when k increases, SD-TOPK needs to get more data items from the list owner nodes in each step. In addition, increasing k augments the number of data items that the trusted client needs to decrypt (because at least k data items are decrypted by the trusted client).

7.5 Effect of Bucket Size

Figure 7 reports the response time of SD-TOPK when varying the size of buckets, and the other parameters set as default values. We observe that the response time increases slightly when the bucket size increases. The reason is that increasing the bucket size increases the number of data items to be considered in the different steps of SD-TOP algorithm. It also increases the number of false positives to be removed by the filtering algorithm.

7.6 Communication Cost

We measure the communication cost of SD-TOPK, Remote-TA and Block-TA in terms of the total number of messages exchanged between the different nodes of the distributed system and the size of the exchanged data.

Figure 5 shows the number of communicated messages while increasing the number of tuples and fixing the other parameters to the default values. We observe that SD-TOPK needs to exchange a small number of messages comparing to the others approaches. The reason is that SD-TOPK runs in only some rounds of communication, and does not depend on the database size. But for the TA-based algorithms, the number of messages depends on the position where they stop in the lists, and that position depends on the database size.

Figure 6 illustrates the size of the communicated data in bytes, while increasing the number of tuples in the database and setting the other parameters to the default values. We note that the size of the communicated data increases with the database size. The amount of data transferred by SD-TOPK is less than that of Remote-TA and Block-TA. The reason is that SD-TOPK uses the obfuscated bucket boundaries to check the top-k data items and these boundaries have a size less than the encrypted scores used by other algorithms.

7.7 Performance over Different Datasets

We study the effect of the datasets on the performance of SD-TOPK, Remote-TA and Block-TA using different datasets: synthetic datasets with uniform and Gaussian distributions, and real dataset (Gowalla). Figure 8 shows the response time of the approaches over different datasets, while other parameters are set as default values. We see that the performance of all approaches over the Gaussian

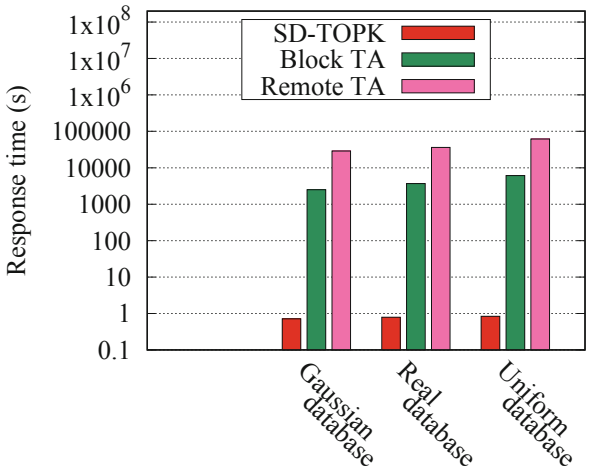


Fig. 8. Response time using different databases

database is better than real and uniform databases. The reason is that with the latter databases, the algorithms need to go deeper into the lists to be sure that they have found the top-k results.

7.8 Filtering Rate

We study the efficiency of the filtering algorithm of SD-TOPK by using different datasets. The results are shown in Table 4. The results show that the filtering algorithm is very efficient over all the tested datasets. However, there is a little difference in the filtering rates because of the local score distributions. For example, in the Gaussian distribution, the local scores of many data items are very close to each other, thus the filtering rate decreases in this dataset.

Table 4. False positive elimination by the filtering algorithm of SD-TOPK over different databases

	Uniform dataset	Real dataset	Gaussian dataset
filtering Rate	100%	99.995%	99.991%

8 Related Work

Efficient processing of top-k queries is important for many applications such as information retrieval [32], sensor networks [38], data stream management systems [28,35], crowdsourcing [7], string matching [34], spatial data analysis [29], temporal databases [25], graph databases [16,19], uncertain data ranking [11,30], etc.

One of the most efficient algorithms for top-k query processing is the TA algorithm [13], which models the general problem of top-k using lists of data items sorted by their local scores and proposes a simple and efficient algorithm. Several TA-based algorithms have been proposed for processing top-k queries in different environments, e.g. [1,3,8]. However, all these algorithms assume that the data scores are available as plaintext, and not encrypted.

In the literature, there has been some research work to process keyword queries over encrypted data , *e.g.*, [5,31]. For example [5] and [31] propose matching techniques to search words in encrypted documents. However, the proposed techniques cannot be used to answer top-k queries. There have been also some solutions proposed for secure kNN similarity search, *e.g.*, [6,10,12,22,36]. The problem is to find k points in the search space that are the nearest to a given point. This problem should not be confused with the top-k problem in which the given scoring function plays an important role, such that on the same database and with the same k, if the user changes the scoring function, then the output may change. Thus, the proposed solutions proposed for kNN cannot deal with the top-k problem.

The bucketization technique (*i.e.*, creating buckets) has been used in the literature for answering range queries over encrypted data, *e.g.*, [17,18,27]. For example, in [18], Hore et al. use this technique, and propose optimal solutions for distributing the encrypted data in the buckets in order to guarantee a good performance for range queries. However, the techniques for range queries no longer apply to top-k queries, because in a top-k query, we should find the k tuples that are most relevant to the query using a given scoring function. In a range query, the objective is simpler, i.e. only to find the tuples whose attribute values are in a given range.

In [20], Kim et al. propose an approach for preserving the privacy of data access patterns during top-k query processing. In [33], Vaidya et al. propose a privacy preserving method for top-k selection from the data shared by individuals in a distributed system. Their objective is to avoid disclosing the data of each node to other nodes. But, their assumption about the nodes is different from ours, because they can trust the node that stores the data (this is why the data are not crypted), but in our system we trust no node of the distributed system.

When we think about top-k query processing on encrypted data, the first idea that comes to mind is the utilization of a fully homomorphic encryption cryptosystem, e.g. [15], which allows one to do arithmetic operations over encrypted data. Using this type of encryption would allow to compute the overall score of data items over encrypted data. However, fully homomorphic encryption is still impractical for query processing over large databases, particularly due to the prohibitive computational time [2,9].

CryptDB [26] is a system designed for processing SQL like queries over encrypted data. It is capable to execute several types of queries, *e.g.*, exact-match, join and range queries. However, top-k queries are not supported by CryptDB.

The Three Phase Uniform Threshold (TPUT) [4] is an efficient algorithm to answer top-k queries in distributed systems. Like our SD-TOPK algorithm, it is done in few round-trips between the nodes of the distributed system. However, TPUT can be used only with the queries in which the scoring function is SUM, whereas our algorithm can be used for a large range of scoring functions. In addition, our algorithm finds top-k results over encrypted data, while TPUT can be used only over plaintext data.

Meng et al. [37] propose a solution for processing top-k queries over encrypted data in clouds. They assume the existence of two non-colluding nodes, one of which can decrypt the data (using the decryption key) and execute a TA-based algorithm. Our assumptions about the nodes of the distributed system are different, as we do not trust any node.

9 Conclusion

In this paper, we proposed SD-TOPK, an efficient system to encrypt and outsource user data in a distributed system. SD-TOPK is able to evaluate top-k queries over encrypted data, without decrypting them in the nodes of the

system. We evaluated the performance of our solution over synthetic and real databases. The results show excellent response time and communication cost for SD-TOPK. They show that the response time of SD-TOPK can be several order of magnitude better than that of the TA-based algorithms. This is mainly due to its optimized top-k query processing and filtering algorithms. The results also show a significant gain in communication cost of SD-TOPK compared to the other algorithms. They also show the efficiency of the filtering algorithm that eliminates almost all false positives in the distributed system.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for efficient top-k query processing. Inf. Syst. **36**(6), 973–989 (2011)
2. Barhamgi, M., Bandara, A.K., Yu, Y., Belhajjame, K., Nuseibeh, B.: Protecting privacy in the cloud: current practices, future directions. IEEE Comput. **49**(2), 68–72 (2016). https://doi.org/10.1109/MC.2016.59
3. Bast, H., Majumdar, D., Schenkel, R., Theobald, M., Weikum, G.: IO-Top-k: Index-access optimized top-k query processing. In: Proceedings of International Conference on Very Large Databases (VLDB), pp. 475–486 (2006)
4. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: Proceedings of ACM PODC, pp. 206–215 (2004)
5. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_30
6. Choi, S., Ghinita, G., Lim, H., Bertino, E.: Secure kNN query processing in untrusted cloud environments. IEEE TKDE **26**(11), 2818–2831 (2014)
7. Ciceri, E., Fraternali, P., Martinenghi, D., Tagliasacchi, M.: Crowdsourcing for top-k query processing over uncertain data. IEEE TKDE **28**(1), 41–53 (2016)
8. Das, G., Gunopulos, D., Koudas, N., Tsirogiannis, D.: Answering top-k queries using views. In: Proceedings of International Conference on Very Large Databases (VLDB), pp. 451–462 (2006)
9. Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., Garofalakis, M.N.: Practical private range search revisited. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, 26 June–01 July 2016, pp. 185–198 (2016)
10. Ding, X., Liu, P., Jin, H.: Privacy-preserving multi-keyword top-k similarity search over encrypted data. IEEE TDSC **99**, 1–14 (2017)
11. Dylla, M., Miliaraki, I., Theobald, M.: Top-k query processing in probabilistic databases with non-materialized views. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 122–133 (2013)
12. Elmehdwi, Y., Samanthula, B.K., Jiang, W.: Secure k-nearest neighbor query over encrypted data in outsourced environments. In: Proceedings of IEEE ICDE, pp. 664–675 (2014)
13. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of ACM PODS (2001)
14. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)

15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: ACM STOC, pp. 169–178 (2009)
16. Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: Top-k interesting subgraph discovery in information networks. In: IEEE ICDE, pp. 820–831 (2014)
17. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. J. VLDB **21**(3), 333–358 (2012)
18. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proceedings of International Conference on Very Large Databases (VLDB), pp. 720–731 (2004)
19. Khemmarat, S., Gao, L.: Fast top-k path-based relevance query on massive graphs. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 316–327 (2014)
20. Kim, H.-I., Kim, H.-J., Chang, J.-W.: A privacy-preserving top-k query processing algorithm in the cloud computing. In: Bañares, J.Á., Tserpes, K., Altmann, J. (eds.) GECON 2016. LNCS, vol. 10382, pp. 277–292. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61920-0_20
21. Li, R., Liu, A.X., Wang, A.L., Bruhadeshwar, B.: Fast range query processing with strong privacy protection for cloud computing. PVLDB **7**(14), 1953–1964 (2014)
22. Liao, X., Li, J.: Privacy-preserving and secure top-k query in two-tier wireless sensor network. In: Global Communications Conference (GLOBECOM), pp. 335–341 (2012)
23. Mahboubi, S., Akbarinia, R., Valduriez, P.: Privacy-preserving top-k query processing in distributed systems. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018. LNCS, vol. 11014, pp. 281–292. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96983-1_20
24. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 644–655. ACM (2015)
25. Pilourdault, J., Leroy, V., Amer-Yahia, S.: Distributed evaluation of top-k temporal joins. In: ACM SIGMOD, pp. 1027–1039 (2016)
26. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: processing queries on an encrypted database. Commun. ACM **55**(9), 103–111 (2012)
27. Sahin, C., Allard, T., Akbarinia, R., Abbadi, A.E., Pacitti, E.: A differentially private index for range query processing in clouds. In: ICDE Conference (2018)
28. Shen, Z., Cheema, M.A., Lin, X., Zhang, W., Wang, H.: Efficiently monitoring top-k pairs over sliding windows. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 798–809 (2012)
29. Shi, J., Wu, D., Mamoulis, N.: Top-k relevant semantic place retrieval on spatial RDF data. In: ACM SIGMOD, pp. 1977–1990 (2016)
30. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 896–905 (2007)
31. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P, pp. 44–55 (2000)
32. U, L.H., Mamoulis, N., Berberich, K., Bedathur, S.J.: Durable top-k search in document archives. In: Proceedings of ACM International Conference on Management of Data (SIGMOD), pp. 555–566 (2010)
33. Vaidya, J., Clifton, C.: Privacy-preserving top-k queries. In: Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, pp. 545–546. IEEE (2005)

34. Wang, J., Li, G., Deng, D., Zhang, Y., Feng, J.: Two birds with one stone: an efficient hierarchical framework for top-k and threshold-based string similarity search. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 519–530 (2015)
35. Wang, X., Zhang, Y., Zhang, W., Lin, X., Huang, Z.: SKYPE: top-k spatial-keyword publish/subscribe over sliding window. PVLDB **9**(7), 588–599 (2016)
36. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure kNN computation on encrypted databases. In: ACM SIGMOD, pp. 139–152 (2009)
37. Xianrui Meng, H.Z., Kollios, G.: Top-k query processing on encrypted databases with strong security guarantees. In: ICDE Conference (2018)
38. Yang, H., Chung, C., Kim, M.: An efficient top-k query processing framework in mobile sensor networks. Data Knowl. Eng. **102**, 78–95 (2016)

Trust Factors and Insider Threats in Permissioned Distributed Ledgers

An Analytical Study and Evaluation of Popular DLT Frameworks

Benedikt Putz(✉) and Günther Pernul

Department of Information Systems, University of Regensburg, Regensburg, Germany
{benedikt.putz,guenther.pernul}@ur.de

Abstract. Permissioned distributed ledgers have recently captured the attention of organizations looking to improve efficiency, transparency and auditability in value network operations. Often the technology is regarded as trustless or trust-free, resulting in a false sense of security. In this work, we review the various trust factors present in distributed ledger systems. We analyze the different groups of trust actors and their trust relationships to the software layers and inherent components of distributed ledgers. Based on these analyses, we investigate how insiders may conduct attacks based on trust in distributed ledger components. To verify practical feasiblity of these attack vectors, we conduct a technical study with four popular permissioned distributed ledger frameworks: Hyperledger Fabric, Hyperledger Sawtooth, Ethereum and R3 Corda. Finally, we highlight options for mitigation of these threats.

Keywords: Trust frameworks · Distributed systems security · Distributed ledger technology · Insider threat

1 Introduction

Distributed ledger technology (DLT) offers great potential to decentralize operations in collaborative business networks and may even enable new business models [46]. Benefits include cost reduction and increased transparency in information sharing between organizations. However, great potential also entails great risks and potential security issues. Recent reviews regarding the future of blockchain technology have pointed out the need to study security and trust aspects of DLT [13,51].

Blockchain and DLT are often described as trustless or trust-free alternatives to currently established centralized systems (see [29,30,36]). In this work, we take a closer look at the usage of permissioned distributed ledgers and examine whether it can really be considered "trustless". The term "trustless" originates from the decentralization of control in distributed ledger networks [29], which aims to replace trusted third parties. The goal of this work is to establish a framework for reasoning about trust elements in permissioned distributed

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
A. Hameurlain and R. Wagner (Eds.): TLDKS XLII, LNCS 11860, pp. 25–50, 2019.
https://doi.org/10.1007/978-3-662-60531-8_2

ledgers. These trust elements can also be exploited by insiders, who are aware of them and in control of crucial components of the trust system.

Insider threats are a tough cybersecurity problem, which remains difficult to detect and prevent due to abuse of legitimate access permissions by the attacker. According to the roadmap of cybersecurity research by the US department of Homeland Security, insider threats are one of the "hard problems" of information security research [50]. Similarly, the European cybersecurity agency ENISA's threat landscape report lists insider threats among the top 10 information security threats, with 77% of companies' data breaches caused by insiders [23].

Insider threats are particularly relevant for distributed ledgers operated by a network of independent organizations. These networks are called *permissioned*, since they are operated by a restricted set of authenticated member nodes. In this scenario, intra-organizational insiders are supplemented with external insiders [25] from other organizations, who also have access to information shared on the network. According to a recent survey on enterprise adoption of DLT, there are at least 50 corporations with valuations larger than $1 billion looking to implement DLT to trade digital assets [11]. Many of these are financial institutions looking to trade high-value assets, leading to an attractive target for insider attacks.

To appropriately assess trust in distributed ledgers, our trust definition is based on software trust as defined by Amoroso and Watson [3]: *"Software trust is the degree of confidence that the software will be acceptable for 'one's needs'. It is established after one has become convinced, presumably based on a meaningful set of information, that the software does not include flaws that will prevent it from meeting its requirements."*

Besides trust in software components, the second form of trust is related to assessments of the human agents that collectively control the distributed system (hereafter referred to as trust actors). We follow Gambetta's definition of trust [28]: *"Trust (or, symmetrically, distrust) is (...) the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (...) and in a context in which it affects his own action."*

The remainder of this work is structured as follows: In Sect. 2 we give a short overview of related work concerning trust and insider threats with regard to distributed ledgers. Subsequently, we analyze trust actors, layers and components of permissioned distributed ledgers in Sect. 3. Based on our assessment of trust factors we identify relevant attack vectors for the different groups of insiders in Sect. 4. In Sect. 5 we perform a threat analysis for 5 popular distributed ledger frameworks, examining how insiders might exploit these vectors in practice. Finally, we wrap up by giving recommendations for future research in Sect. 6 and summarize our results in Sect. 7.

2 Related Work

While research on trust in blockchain systems is still scarce, the Global Blockchain Benchmarking Study by the University of Cambridge points out

that blockchains always require some degree of trust [32]. Recent blog posts have highlighted trust factors in public permissionless blockchains [47]. Permissioned blockchains rely on similar trust primitives: trust in application code, network/cryptographic protocols and hardware. We aim to expand upon these notions by exploring the trust factors in more detail.

Overall, only partial aspects of trust in blockchain networks have been studied. Locher et al. [38] create a formal model to examine whether a distributed ledger may fully replace a trusted third party. In the process, they also evaluate previously proposed use cases of DLT that still require trust in other organizations and third parties. Hawlitschek et al. [30] review the conceptualization of trust in the blockchain environment. They argue that it is difficult to assess whether a system is actually trust-free or not. Correspondingly, another study claims that blockchain shifts trust from central authorities towards algorithms [39]. However, for this shift to be successful, the algorithms need to be trusted. Smart contracts represent the application-level algorithms, and their control flow immutability and independence of third parties have been shown to be lacking [26]. In addition to algorithmic properties, researchers have also studied user trust of different stakeholder groups from an HCI perspective [45]. In summary, research has established the existence of trust factors that contradict the claim of a trust-free system [30], but a comprehensive model of trust relationships is still missing.

Despite the severity of insider threats as pointed out by government agency assessments [23], research regarding insider threats in distributed ledger consortia is still scarce. Numerous surveys on the security of blockchain systems have been carried out [17,35], but none of them have focused on insiders in particular. We intend to fill this research gap and provide direction for future research.

In particular, we go beyond existing work by presenting a novel model of trust actors, their relationships and trusted DLT software components. We use this model to derive insider threats that organizations face when implementing permissioned DLT. By analyzing frameworks popular in both industry and research, we show that these attacks are applicable in practice. To protect against these threats, we outline technical and organizational options to mitigate the insider threats at hand.

3 Trust Factors in Permissioned Distributed Ledgers

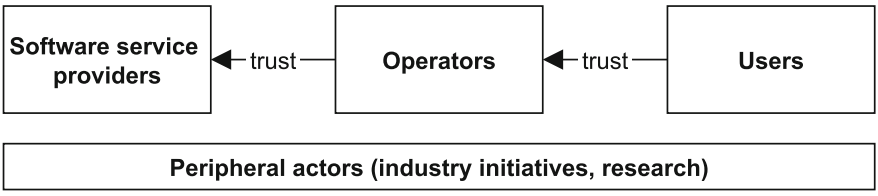


Fig. 1. Trust actors and relationships.

As noted by other researchers, the requirement for trust does not disappear simply by employing a distributed ledger. Instead, trust shifts from trust in other organizations to trust in the technology and its operation [38]. In this section we focus on analyzing the different components of a distributed ledger system to establish trust actors, layers and components.

Before examining the trust factors in detail, trust actors need to be identified. There are four types of actors in the DLT ecosystem, three of which directly contribute to trust relationships in a consortium: software service providers, operators and users [32]. Peripheral actors (i.e. industry initiatives and researchers) do not directly interact with consortium networks, as they are not involved with building or operating DLT software. Nevertheless, they contribute by developing standards, methods and paradigms to solve current technical challenges [32].

An overview of the resulting trust hierarchy is shown in Fig. 1. **Software service providers** (SSPs) develop the software components of a distributed ledger consortium. They are trustees responsible for creating trust in the technology by developing secure and reliable applications. **Operators** represent distinct groups of actors responsible for running the distributed ledger overlay network and applications built on top of it. They act as both trustors and trustees: they trust the chosen DLT software to operate as expected and are also trusted by their users to provide reliable operations. Finally, **Users** place their trust in these applications and rely on them to work as advertised, without knowledge of the lower layers.

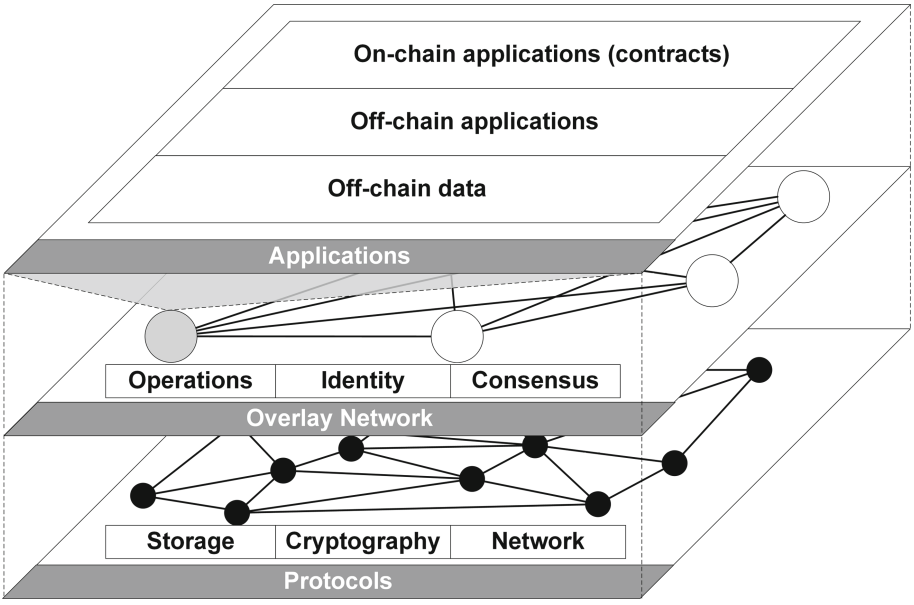


Fig. 2. Distributed ledger software layers and components.

The trust actors in Fig. 1 interact using a permissioned distributed ledger network, which consists of several software layers. Figure 2 shows the layers and the software components on each layer. They are derived from the three-layer view of Component Based Systems: platform, middleware and application [43]. The platform in this case consists of various underlying *protocols* responsible for storage, cryptography and network communication. Also part of the platform, but out of scope for this work, are the operating system and hardware layers. The middleware is represented by an *overlay network*, which provides configurable functionality for operations, identity management and distributed consensus. The *applications* layer provides replicated application logic (on-chain) and external logic and data (off-chain). These off-chain applications integrate with the framework by reading/writing data through its APIs. Each of the components within these layers requires software trust: it should be working correctly and not be maliciously exploitable. Since layers are built on top of each other, software bugs or vulnerabilities may propagate upwards to affect higher layers. In the following subsections we elaborate in detail on the layers' components and how they are involved in creating trust in the distributed ledger.

Figure 3 integrates trust actors from Fig. 1 with the layers from Fig. 2 by illustrating which trust actors govern each system layer. While software service providers are involved in the development of all three layers, operators do not interact with the underlying protocols. They merely configure the network framework and develop applications on top of it. Meanwhile, users only interact directly with the application layer.

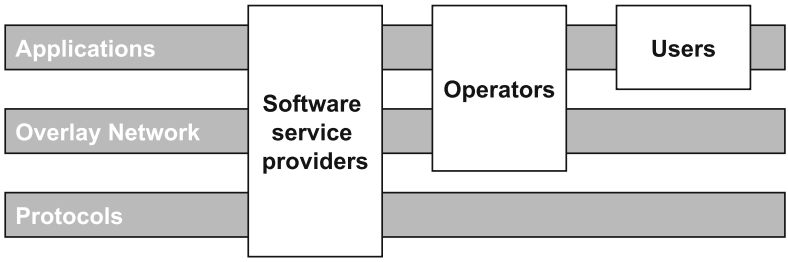


Fig. 3. Intersection of trust actors and layers.

3.1 Protocols

Storage, cryptographic and network protocols carry out the low-level tasks instructed by the distributed ledger framework and its applications. For this reason, they form the trust basis of the network.

Storage. A key property of blockchain-based systems is the goal of maintaining immutability of the underlying chain of blocks. Transaction and block metadata are stored in relational or key-value databases locally on each node. Replication

integrity is assured through distributed consensus. The claimed immutability is a key factor in enabling trust in the technology, but it only holds if storage and network protocols can be trusted.

Cryptography. A manifold of cryptographic protocols are involved in distributed ledger operation. They include:

- hash functions for integrity assurance (hash chains, Merkle trees and proofs)
- public key cryptography (authentication of consensus protocol messages and user-submitted transactions)
- zero knowledge proofs (privacy-preserving transactions)
- symmetric encryption (on-chain confidentiality)

All of these protocols are trusted to not have design or implementation flaws. Many frameworks assemble their cryptography from a variety of sources, including standard libraries, external libraries and custom implementations (see Sect. 5). While some developers such as the Hyperledger open-source project perform third-party security audits [33], even the most diligent audits may miss vulnerabilities. Operators and users must trust protocol design and implementation, often without the ability to verify due to lack of cryptographic expertise.

Network Protocols. A fundamental trust factor for distributed ledger node communication is untampered operation of the underlying network. Distributed ledger networks are overlay networks, so they rely on P2P routing algorithms and message dissemination protocols for communication. Since all peers are equal, any single peer may cause disruption in the network by sending anomalous or malicious traffic. This may cause unexpected behavior and violate the aforementioned trust assumption.

3.2 Overlay Network

A distributed ledger network is a permissioned overlay network that consensually maintains a replicated ledger. In this overlay network, independent operators deploy a software framework previously agreed upon (i.e. Hyperledger Fabric). There are several tasks that each operator is trusted to fulfill by other participants: carry out operations tasks, maintain identity and access privileges and participate in consensus. The network layer also provides virtualization capabilities for replicated deterministic application execution in the application layer.

Operations. These independent organizations trust each other to perform node setup and operation without malicious manipulation. While there is some fault tolerance built into distributed ledgers (see Sect. 3.2), more than two-thirds of operators must behave honestly if byzantine-fault tolerant protocols are used.

Operation includes consensual admission/removal of members, on-chain application upgrades and framework upgrades. For the latter, all operators must agree on a coordinated time for system maintenance. Provided that all partners agree on the schedule, some partners might not be able to upgrade successfully [20]. Even if the failure to upgrade is not of malicious intent, it might pose

considerable challenges to all involved parties, like setting up a new network and migrating data. Overall, the process requires significant trust in other organizations that cannot be mitigated by technology.

Identity and Access Control. Since the network is made up of independent organizations, each entity must be able to manage its users independently. This means that every organization must trust the others to properly manage identities and access rights. Since internal screening and job rotation processes are usually opaque to others in collaborative business networks [25], this can be considered blind trust.

Fundamental to permissioned distributed ledgers is an access control mechanism that ensures only authorized operators are part of the network. This is usually realized by assigning each node a public key, which is known to the other nodes and used to authenticate and secure communication. While admission/removal of node operators is based on majority consensus, other participants must be trusted to only accept legitimate new members. Additionally, compromise of a single node's credentials may undermine the trust assumption of a closed network.

Consensus. The consensus protocol is the distributed agreement protocol at the core of a permissioned distributed ledger, allowing all nodes to share a single replicated state. However, due to fundamental limitations underlying deterministic replicated state machines, less than one-third of participants may be malicious at the same time [12]. This limit means that operators must trust one another to act honestly and to not manipulate the consensus protocol.

3.3 Applications

Applications implement the business logic specific to each network. Next to on-chain smart contracts, off-chain applications and data are often required to implement all functionality and integrate with other enterprise systems.

On-Chain Applications. On-chain applications are generally referred to as smart contracts, although this depends on their implementation (see Sect. 5.7). They promise to replace trust through replicated and verifiable deterministic execution. Since both code and state can be inspected by anyone with access to the ledger, their execution is predictable to these parties. However, a number of smart contract vulnerability studies have shown that code is not always law and may be exploited to an attacker's advantage. For example, in 2016 a symbolic execution tool found almost half of all Ethereum contracts at the time to be vulnerable [40]. In another study, 2 out 5 deployed Ethereum contracts were shown to require trust in at least one third party, since parts of their control flow may be changed after deployment [26].

Off-Chain Applications. One example for off-chain applications are web applications for user interaction with the distributed ledger. Without a way to verify what is going on behind the scenes, users must blindly trust that the application does not manipulate any data sent through it. While this is also the case for

Table 1. Summary of distributed ledger trust components by layer.

Protocols	Overlay network	Applications
Storage	Operations	On-chain applications
Cryptography	Identity	Off-chain applications
Network	Consensus	Off-chain data

traditional web applications, a distributed system aiming to create trust should provide stronger guarantees.

Off-Chain Data. Full replication of the blockchain data structure mandates parsimony w.r.t. transaction sizes. Distributed ledger applications rely on off-chain storage solutions to manage larger data volumes. In fact, a recent study found that a majority of DLT operators only include hashes in on-chain transactions [32], which point to off-chain data and serve as integrity timestamps. Operators must trust their peers to maintain sustained availability, since off-chain data is not fully replicated. If off-chain data is access protected, the storage operator must also be trusted to maintain correct access privileges.

Besides referenced data, external data may also be needed as input for computation (i.e. currency exchange rates). Since external data sources must return deterministic results, distributed ledgers rely on trusted external content providers (oracles) - hereby reintroducing trust elements.

A summary of all identified trust components is shown in Table 1. Overall, the complexity of the DLT software layers results in a high degree of obscurity. It becomes increasingly difficult to verify correctness and security of the software stack. The trust actors (operators, users and software service providers) must trust both software components and each other to act as expected. In the next section, we focus on how insiders can exploit these trust assumptions.

4 Insider Threats

Given the aforementioned trust elements required for operating a permissioned distributed ledger network, insider attacks may pose a significant threat. With the emergence of business networks and blockchain consortia for data sharing, the partners' information systems infrastructures are no longer isolated environments with a protectable logical perimeter. Access to an organization's resources is implicitly simplified for outsiders that are part of the consortium. This is a direct consequence of sharing information with other organizations.

As a result, a holistic security model must also include actors from network partners. This group of threat actors is also known as **external insiders**. External insiders are characterized by having limited access to an organization's network as a result of some business relationship [25]. In a distributed ledger context, the relationship may be the result of a collaborative network with multiple organizations.

Table 2. Overview of insider threats and consequences.

Insider type	Threat	MO	DE	DI	DU
Software service provider	Vulnerability injection	x	x	x	x
Operator	Denial of service				x
	Data manipulation	x	x		x
	Credential compromise	x		x	x
	Malicious misconfiguration			x	x
User	Unauthorized operations	x			x
All	Vulnerability abuse	x	x	x	x
	Information leakage			x	

Subsequently, we describe the various insider threats to distributed ledgers by analyzing each group of trust actors. Irrespective of an insider's group, there are generally four types of consequences an insider might achieve in an attack [37]:

- Modification (MO)
- Destruction (DE)
- Disclosure (DI)
- Denial of use (DU)

Insiders may exist in any of the three groups of trust actors stated in Fig. 1. Depending on the group, there are different ways to exploit their privileges. Table 2 lists the major categories of insider threats and their consequences in distributed ledger consortia. While many of these threats are also applicable to existing information systems, distributed ledgers are particularly vulnerable due to the large number of software components and cross-organizational users.

Permanent modification or destruction of data are generally difficult to achieve with DLT due to built-in fault tolerance and replication. Nevertheless, collusion-based data manipulation or software vulnerabilities may cause data manipulation on all nodes. Disclosure of information and denial of use are the more likely consequences of an insider attack on a distributed ledger. They are significantly easier to accomplish and may be achieved with user or operator level permissions. We elaborate on these threats in detail hereafter.

4.1 Software Service Providers

If an insider is in the role of an internal software developer with full code access, there is significant threat potential for any of the four consequences to happen. Collins et al. [15] have surveyed a variety of methods that programmers acting as malicious insiders have used in the past. Common methods are code modification or injection of malicious code, causing vulnerabilities. Characteristic for this type of manipulation is a time delay between injection and impact of the attack, since software builds go through testing and deployment phases. In large software

projects without strict code review procedures, these types of manipulations may easily go under the radar of other developers. Vulnerabilities can then be abused by the programmer or colluding operators/users. They may corrupt the integrity and availability guarantees that a distributed ledger provides, leading to manipulation or loss of information. Intentionally timed bugs may cause network unavailability. Backdoors (either for outsiders or insiders) could be inserted that lead to disclosure of confidential information.

The potential attack vectors depend on the developer's area of responsibility (protocol/framework/application level). Currently, distributed ledger protocols and overlay-level frameworks are often provided by open-source initiatives. If an open-source project is subject to peer-review and security audits, these parts of the software are unlikely to be affected by insider attacks threatening a single organization. However, applications built on these frameworks must be customized to specific business requirements—either by employees or third party developers. For this reason, the application level is more prone to software development insider threats. Business networks may collaboratively develop applications, which extends this attack vector to external insiders.

4.2 Operators

Overall operators have the largest number of insider attack options at their disposal, since they directly control a network node. Even though a single operator controls only one node, malicious behavior may have powerful denial-of-service effects on networks with few participants, as mentioned in Sect. 3.2.

In larger networks, operators could take advantage of their knowledge about the current consensus leader. A malicious operator may launch a targeted denial of service attack to cause network interruptions (see Sect. 5.6). Additionally, collusion of operators from other organizations (i.e. by external insiders) might result in denial of service, if it leaves the network unable to reach consensus. Generally, network operators have a common goal and should not be inclined to collude against others. This might change if goals shift, or partners feel that they contribute more to the network than they gain in return.

An insider might attempt to accomplish modification of stored data by coordinating an attempt to replace ledger data in collusion with other nodes. This type of attack is known as a 51%-attack for permissionless blockchains [35]. Generally this is only feasible if consensus is stochastic, while permissioned networks usually rely on deterministic consensus. Nevertheless, availability of off-chain data with low replication factors is still at risk.

System administrator insiders have access to all relevant credentials for node operation. These credentials can be leaked, or misused by adding/removing users or manipulating access control privileges at will.

Another way to subvert data integrity is configuration manipulation. Successful configuration manipulation requires collusion, since such changes need to be approved by a majority of operators in properly configured networks. Without automated punishment mechanisms, a single misbehaving node may however still cause temporary service disruption.

4.3 Users

Users have only a limited number of options for exploiting the distributed ledger network. Nevertheless, due to external insiders the number of potential attackers is higher than in intra-organizational applications. Improperly managed access rights for these users may enable leakage of confidential information.

Another attack vector are vulnerabilities in custom-developed contracts. They could enable insiders to carry out unauthorized asset transfers or even shut down an application. Insiders might have increased knowledge about application internals such as access to source code and technical documentation, enhancing their ability to discover programming flaws and carry out unauthorized operations.

It is important to note that threats are not strictly restricted to a specific group of trust actors. For example, operators may also impersonate users if they control the identity component. Conversely, users may gain operator-level privileges through improper access right management.

In fact, some threats are exploitable by any actor with access to the distributed ledger network. Whether intentional or inadvertent, vulnerabilities can be abused by any insider with the required knowledge and skills. Since all nodes of the network likely run the same software, remote code execution vulnerabilities may lead to irreversible manipulation or loss of data. Similarly, any participant with access to distributed ledger data may leak data to parties outside the network. The extent of information leaked depends on the insider's access privileges.

5 Insider Threat Analysis of Popular Frameworks

To demonstrate applicability of the identified insider threats to permissioned blockchain frameworks, we conduct a technical threat analysis of several popular blockchain frameworks. Frameworks were selected based on a survey of industry and research support conducted in July 2019. Regarding industry support, the Ethereum Enterprise Alliance (240 members), the Hyperledger project (249 members) and R3 (92 members and 294 partners) were the largest enterprise consortia working on open-source permissioned DLT frameworks. To gauge research interest, we searched several literature databases for mentions of permissioned distributed ledger frameworks. We used fulltext search since some framework names are ambiguous and might be used with a different meaning in a distributed systems context (i.e. Quorum). The search result counts totaling close to 1000 academic publications are shown in Fig. 4. As a result of this survey we decided upon the four frameworks detailed below.

We briefly describe each framework below and summarize the technological components in Table 4. To future-proof our analysis, we mainly analyze threats resulting from architectural design choices, which are unlikely to change in the future. We assume that operators strive for a secure configuration, which includes a byzantine-fault tolerant (BFT) consensus algorithm to prevent byzantine manipulations.

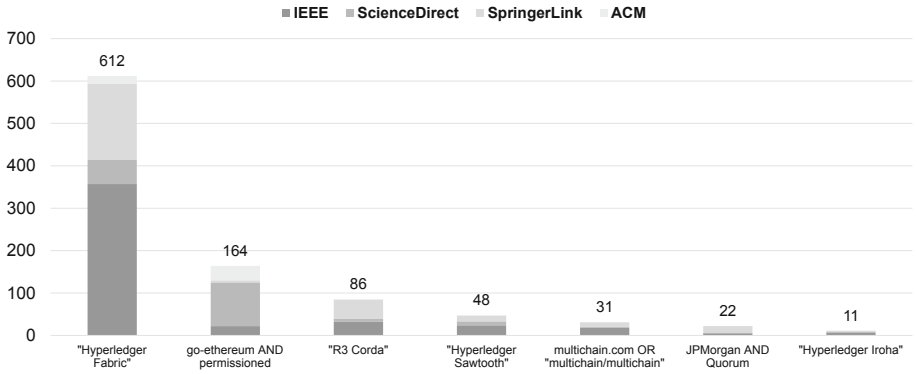


Fig. 4. Research popularity of distributed ledger frameworks (search result count by search term and academic database).

Hyperledger Fabric is a blockchain framework relying on a novel execute-order-validate architecture. This architecture was created to rule out source of non-determinism during consensus and improve performance [4]. We assume that the BFT-SMaRt consensus algorithm[1] [9] is used for ordering service consensus, since it is to our knowledge the only currently available BFT consensus module for Fabric.

Hyperledger Sawtooth [49] is a blockchain framework, which modularizes transaction processing with so-called transaction families. They include predefined families for permissioning and on-chain settings management. Consensus is also modular, but we assume that the Practical Byzantine Fault Tolerance (PBFT)[2] [12] module is used.

Ethereum [18,24] is a popular permissionless blockchain framework, which runs smart contracts written in the Solidity language in an isolated environment called the Ethereum Virtual Machine. The go-ethereum client can also be set up as a permissioned network with Clique Proof-of-Authority (PoA) consensus. PoA is a leader-based consensus protocol with stochastic consensus, which only provides eventual consistency as opposed to strong consistency provided by BFT algorithms. It only requires 50% for a consensus majority, trading consistency for availability.

R3 Corda [31,44] is based on a DAG data structure and only shares data with other nodes when needed. To prevent double spending, mutually agreed upon notary service clusters are used for consensus. We assume that BFT-SMaRt consensus is used among notaries, since it is the only built-in consensus algorithm which tolerates byzantine faults.

All of these frameworks have unique differences in their architecture and the way applications are built on them. While some threats are applicable to all frameworks, others apply only to specific frameworks due to architectural

[1] github.com/bft-smart/fabric-orderingservice.
[2] github.com/hyperledger/sawtooth-pbft.

Table 3. Mapping of insider threats to abused distributed ledger trust components.

Threat	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9
Vulnerability injection		x		x		x	x	x	
Denial of service				x		x			
Data manipulation					x	x			x
Credential compromise					x				
Malicious misconfiguration				x	x	x			
Unauthorized operations							x	x	
Vulnerability abuse	x		x				x	x	
Information leakage	x				x				x

choices. Subsequently, we survey each framework's trust components and analyze where insiders may abuse architectural flaws.

Table 3 highlights which framework components that each type of threat exploits. The columns represent the subsections corresponding to the affected trust components. Corresponding to the trust actors that each threat applies to (Table 2) and their ability to access various trust layers (Fig. 3) some cells are marked gray (not applicable). Vulnerability injection does not result in a vulnerability abuse threat if the vulnerability can only be effectively abused by the software service provider. Hereafter, we explain per component how each threat may occur. Threats are italicized when they refer to a table entry.

5.1 Storage

The surveyed distributed ledger frameworks mostly rely on existing key-value databases for data storage (i.e. LevelDB and CouchDB). None of these databases offer encryption-at-rest, which means anyone with access to the database can read all historical data contained in the ledger. Corda marks the exception: it relies on relational databases, some of which offer encryption. Nevertheless, the database itself is an attractive attack vector for operator insiders, who may circumvent framework-level access control by directly accessing the underlying database *(information leakage)*.

5.2 Cryptography

Currently, distributed ledgers almost exclusively use public key cryptography for authentication. The reviewed frameworks use NIST-recommended ECDSA curves in combination with SHA2 for digital signatures, with some also offering EdDSA and RSA. These algorithms are vulnerable to quantum attacks based on Shor's algorithm [6] *(vulnerability abuse)*. Such attacks threaten the authenticity of transactions and network messages (see Sect. 5.5). If symmetric keys were encrypted using public key cryptography, this may also result in *information leakage*. Once quantum computers reach sufficient computational power, current

ledgers will have to be rebuilt from scratch with new identity schemes. Instead of relying on a single cryptographic primitive, developers should instead adopt a more future-proof approach. For example, quantum-proof hash-based signature schemes use hash combiners, which remain secure if at least one of the input hash functions is secure [6]. In our review, only Corda offers such a scheme with SPHINCS256[3]. Still, no framework provides guidance for migration between signature schemes.

Due to the unique challenges of trust in distributed environments, some distributed ledger frameworks rely on new variants of cryptographic protocols with unproven implementations (especially novel non-interactive zero-knowledge proofs such as zk-STARKs [8]). While we are not aware of any cryptographic flaws in the reviewed permissioned frameworks, there are examples among permissionless blockchains. Recently, a zero-knowledge proof vulnerability in the permissionless blockchain Zcash was publicized, which had been kept secret by the development team for more than 11 months [48]. In this case the developer that discovered the bug did not have malicious intentions and worked on fixing the bug instead of exploiting it. But the incident shows that open-source code is not immune to longstanding hidden vulnerabilities. These may even be inserted into the code intentionally by members of the development team *(vulnerability injection)*. If discovered by malicious actors, they could be kept secret for continued exploitation. Overall, trust in the security of cryptographic protocols is not guaranteed and may be undermined at any time.

5.3 Network Protocols

The reviewed frameworks rely on different network protocols for node-to-node communication. The ZeroMQ protocol[4] used in Sawtooth and the AMQP protocol[5] used in Corda have experienced denial of service and remote code execution vulnerabilities in the past[6]. External insiders, who know about the underlying protocols, may abuse these vulnerabilities to cause damage to specific competitors in the network *(vulnerability abuse)*.

Consensus protocols require constant network communication between all involved nodes. For this reason, they are vulnerable to network-partitioning attacks such as Border Gateway Protocol (BGP) hijacking and Eclipse attacks [35]. These attacks have so far mainly been observed and studied on permissionless blockchains. In permissioned blockchains, manipulations of the routing protocol or network traffic interception can also lead to network partitions [22]. If none of the partitions are large enough to reach consensus, the network will stop processing incoming transactions (deterministic algorithms) or create competing forks (stochastic algorithms) [21]. For some consensus algorithms, these network partitions can even allow malicious double-spending transactions (see Sect. 5.6).

[3] sphincs.cr.yp.to.
[4] zeromq.org.
[5] www.amqp.org.
[6] cve.mitre.org.

Table 4. Overview of software components used in popular distributed ledger frameworks

	Hyperledger Fabric v1.4	Hyperledger Sawtooth v1.1	Go-Ethereum v1.9	Corda v4.1
On-chain contracts	Chaincode (Go, nodeJS, Java)	Transaction Processor (see below)	Smart Contract (Solidity)	CorDapps (Java)
Off-chain applications	Go, Java, nodeJS, Python	Go, Java, nodeJS, Python, C++, C#, Swift	Web3 (nodeJS)	Java
Off-chain data	–	–	Swarm	Oracles
Operations	ReST Operations Service, CLI	Settings TP, CLI	RPC CLI	RPC CLI
Identity	Membership Service Provider	Identity Transaction Processor	Accounts	Hierarchical PKI, Doorman Service
Consensus	Endorsements (custom), Ordering (Kafka, BFT-SMaRt)	Journal (PoET, Raft, PBFT)	PoA, IBFT	Notary (Raft, BFT-SMaRt)
Storage	LevelDB, CouchDB	LMDB	LevelDB, RocksDB	H2, Postgres, SQLServer
Cryptography	ZKP: idemix Signature: ECDSA P256/384, Hash: SHA256, SHA3 Encryption: AES	Hash: SHA256/512 Signature: libsecp256k1	Hash: Keccak Signature (using SHA256/384/512, AES): ECDSA P256, P384, P521, S256, bn256	Hash: SHA256 Signature (using SHA256/512, AES): RSA; ECDSA secp256r1, secp256k1; EdDSA-ed25519; SPHINCS256
Network	GRPC, Gossip[a]	ZeroMQ	devP2P[a]	AMQP

[a]custom protocol

5.4 Operations

Regarding operational tools, the frameworks offer little in terms of monitoring capabilities. Only command-line interfaces (CLI) and transaction types for on-chain settings (Hyperledger Fabric and Sawtooth) are offered to retrieve metrics and update settings. Without significant effort by the operators, this may lead to manipulations of configuration settings going undetected *(malicious misconfiguration)*. Additionally, intentional manipulations of consensus network traffic are nearly impossible to detect without proper monitoring. For example, TCP or UDP flooding attacks reduce transaction throughput to a small fraction of peak throughput [14]. Lack of monitoring facilities effectively allows operator (external) insiders to control network throughput by launching attacks when desired *(denial of service)*.

Despite their initial immutability, smart contracts can be upgraded in all surveyed frameworks[7]. If only bytecode is available for inspection, there is no

[7] Ethereum only allows upgrades if the contract has been set up in a modular way.

easy way to tell what part of the contract was changed. Since smart contracts may contain vulnerabilities or require feature extension, upgrades cannot be regarded as unusual per default. Individual operator insiders may abuse this fact by upgrading a contract with malicious functionality *(vulnerability injection)*. Requiring signatures of multiple operators for a successful upgrade is a potential mitigation, but the contract needs to be set up with multiple owners for this to apply (Ethereum, Fabric). Corda requires all participants that share a state to sign contract upgrades, but contract signature constraints also permit custom rules that require less signatures for an upgrade [44]. Sawtooth offers no mechanism for coordinated upgrades. Transaction processors run as independent processes next to the validator network and are upgraded by each operator individually.

5.5 Identity and Access Control

All permissioned networks must admit identities through some form of a gatekeeper. Distributed ledger frameworks attempt to decentralize admission of validating nodes by voting on new members. This does not apply to users, who must receive a certificate through a validating node or its certificate authority. In Ethereum and Sawtooth, there is no certificate infrastructure integration: users either create accounts themselves or request them from a node administrator. However, account pseudonyms need to be mapped to real-world identities for many applications. A lack of certificates complicates permission revocations, which then need to be performed on the application level. Lack of a single source of truth for identity also leads to excessive access rights over time, which enable insider abuse [27]. A potential consequence of unchecked access rights is *information leakage*.

Conversely, Hyperledger Fabric and Corda rely on traditional hierarchical PKIs with a root authority. In these systems, each node operates its own certificate authority. Certificates are then shared across the network through a federation service. As a result, each validator node recognizes the identities issued by its peers. From an insider perspective, this means that any employee with access to the gatekeeper of a participating organization can create valid identities for the entire network. By subverting the local certificate authority, operator insiders may replace associated node's certificate and impersonate it *(credential compromise)*. Accordingly, a collusion between CA operators can subvert multiple node identities and overtake the network, thus enabling *data manipulation*.

Fundamentally, the identity component relies on the underlying cryptographic signature protocols. If a cryptographic primitive is broken, credentials can be forged by issuing fake signatures. Fake signatures in turn enable *data manipulation* and *malicious misconfiguration* through malicious transactions.

5.6 Consensus

The surveyed permissioned distributed ledger frameworks rely on crash fault-tolerant (CFT) or byzantine fault-tolerant (BFT) algorithms. CFT algorithms

do not tolerate any malicious activity and are built only to tolerate crashes [10]. As a result they are prone to manipulation by any one operator and not well suited for usage in semi-trusted environments like business networks. Despite this, most frameworks in our survey recommend CFT protocols and mark BFT consensus implementations as experimental.

If operators use BFT algorithms, up to f malicious nodes among $n = 3f + 1$ total nodes are tolerated without ceasing operation. Byzantine failures encompass all possible failure modes of a system. The performance of most BFT algorithms is however heavily impacted by the presence of failures and no consensus is reached with more than f failures. As a result, a single operator can significantly decrease throughput in PBFT-based networks with $n < 7$ independent nodes ($3f + 1 = 7 \mid f = 2$) by flooding the network with messages [14]. Collusion between two operators can even shut down network consensus and prevent new transactions *(malicious misconfiguration)*. Therefore, smaller permissioned networks are especially at risk of *denial of service* by a minority of participants. In addition to flooding-based denial of service, malicious consensus leaders can also degrade performance in most BFT consensus protocols [5].

The PoA algorithm used by permissioned Ethereum networks is vulnerable to the Attack of the Clones [22]. In the attack, a single malicious node can double spend with high probability. By cloning itself and intercepting messages, a network partition is created. The victim partition is deceived by submitting a conflicting transaction to the other partition, which is later accepted as the canonical transaction *(data manipulation)*. Based on the authors' assessment, the only viable countermeasures are switching to a BFT algorithm or requiring a two-thirds majority instead of the current 50%.

In addition to protocol flaws, *vulnerability injection* in the consensus protocol implementation may lead to nodes accepting invalid or malicious transactions. This could be abused by SSP insiders to circumvent on-chain access permissions and transfer assets or tokens.

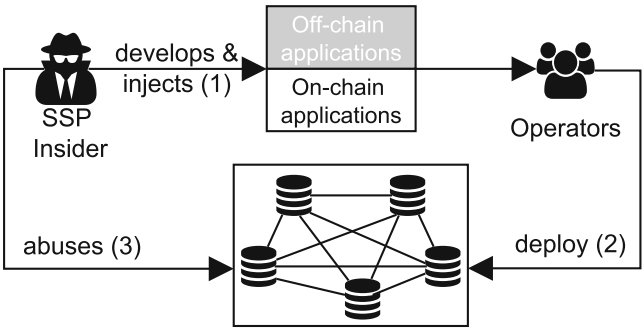


Fig. 5. Illustration of injection and delayed abuse of a vulnerability by a SSP insider.

5.7 On-Chain Applications

Transparency is an often-cited advantage of smart contracts. In the surveyed frameworks, contract code is rarely transparent to all operators, and never to users. In Hyperledger Fabric, chaincode source code is only known to the peers specified in its endorsement policy. For Sawtooth, bytecode is deployed when using the Seth (Solidity), Sabre and Java SDKs. For transaction processors based on the Python and nodeJS SDKs the source code is transparent, since they are interpreted languages. In Ethereum, only compiled Solidity bytecode is visible to blockchain node operators. Corda's CorDapps are only shared by peers concerned with the application, who need to compute the state changes for notary consensus. A lack of transparency can lead to undetected manipulations, for example through covert contract upgrades.

To demonstrate how this might occur in practice, Fig. 5 illustrates the three steps of *vulnerability injection* and subsequent *vulnerability abuse*. First, the SSP insider injects a vulnerability into an on-chain or off-chain application (1). With the next scheduled operational software upgrade, operators deploy the vulnerable version of the software to the production network (2), permitting the insider to abuse the vulnerability (3).

For Hyperledger Fabric, the two most popular SDKs on GitHub are based on nodeJS and Go. Both languages allow package imports from public version control sites such as GitHub. This method could be abused by a software service provider to conceal malicious functionality in the chaincode or insert a backdoor. By changing the code of a self-controlled dependency, the developer gains the ability to manipulate dependent code sections, while obscuring the changes from the client. Additionally, existing vulnerabilities in packages may be knowingly included by a SSP insider to be exploited later on. Due to the large number of transitive dependencies, the nodeJS dependency management system npm is especially prone to attacks based on existing vulnerabilities [19]. For Ethereum smart contracts, many vulnerability classes are known [35] due to public scrutiny and open source application bytecode. For Sawtooth and Corda there are not many production deployments yet, so to the best of our knowledge we are not aware of any vulnerabilities.

To summarize, chaincode and smart contract vulnerabilities may manipulate the output state of a contract *(data manipulation)*, prevent consensus by introducing non-determinism *(denial of service)*, and leak secrets by sending confidential data to parties outside the network *(information leakage)*. Additionally, users may be able to conduct *unauthorized operations* due to a smart contract permission management vulnerability.

5.8 Off-Chain Applications

Production deployments of DLT must include a client software, since direct interaction with a blockchain node is not user-friendly and requires command line skills. This client software relies on Software Development Kits (SDKs) published by software service providers. Table 4 shows the diverse programming

languages that these SDKs use. For SDKs using package managers, the same attack vector from Sect. 5.7 applies *(vulnerability injection)*. SSP insiders may abuse SDKs or client software to hijack user identities and thus gain access to the distributed ledger network *(credential compromise)*.

5.9 Off-Chain Data

For referenced off-chain data, the reduced replication factor exposes data availability to collusion attacks. Depending on the replication factor r of items stored off-chain, r operators may collude to irreversibly delete a key-value pair stored off-chain *(data manipulation)*.

Off-chain events providing external data are not natively supported by the surveyed frameworks. The requirement for code determinism runs counter to uncertain responses from external sources. Consequently, external data is often integrated via trusted applications that attempt to guarantee response integrity. These applications are referred to as validation oracles [52]. They act as automated arbitrators that sign transactions referencing external data on demand. Corda is the only framework that provides built-in integration with centralized oracle services for this purpose. They are accepted as an authoritative source of data by a set of peers. An insider may compromise that service and manipulate transactions at will, without needing access to the distributed ledger *(data manipulation)*. Depending on the application, such attacks can be hard to discover, since the oracle service is not transparent to all operators. Alternative proposals that avoid relying on a centralized provider include secure hardware architectures such as Town Crier [53], and cryptocurrency-based decentralized blockchain oracles such as Astraea [1].

6 Mitigations and Future Research

The previous sections have shown how various types of trust actors in a distributed ledger system may abuse trust components and carry out insider attacks. Based on these insights, we now elaborate how DLT adopters can better assess which components they trust, and how they can mitigate resulting insider threats.

6.1 A Realistic View of Trust in Distributed Ledgers

Instead of regarding blockchain as "trustless", DLT adopters should be aware of the technological components that their trust relies on. First and foremost, software trust management processes should be established to ensure that trust is warranted. The inherent failure modes and consequences should be integrated into organizational risk management.

Regarding trust in the various software components of distributed ledgers, software trust research has established trust principles and an ordered set of classes for software trust measurement [3]. The classes range from Untrusted

(T0) to Trusted (T5) and require a progressively larger set of trust principles to be fulfilled. Classes T4 and T5 aim to prevent malicious activity and could be used to certify components for inclusion in trusted distributed ledgers.

To reduce the implicit trust resulting from allowing others to manage identities and access rights, trust-based distributed access control models could be used. Such frameworks include risk assessment processes that dynamically adapt to users' behavior [7]. Additionally, next generation decentralized blockchain-based identity management could enable consensus-based trust in external users, instead of relying on federated membership schemes.

To increase trust in smart contracts, a number of approaches have been proposed. Business and legal smart contract specification languages based on formal reasoning can help reduce ambiguity for programmers and lock down edge cases [2].

To make user interaction with distributed ledgers more transparent, "decentralized applications" (DApps) may be used. DApps are web applications relying solely on an on-chain application backend. DApp frontends should be distributed as client-only applications, with code only served from, but not executed on a centralized web server. This ensures that code execution is fully transparent to end-users. Transaction submission to the blockchain can be explicitly authorized using open source browser extensions (i.e. MetaMask [42]).

6.2 Insider Threat Mitigation

Techniques for insider threat mitigation have been studied extensively in the past [16]. They require an interdisciplinary mitigation approach, combining insights from computer science, psychology and other fields. Correspondingly, mitigation techniques can be classified as technical or organizational. We have analyzed mitigation techniques in the literature and applied them to the threats mentioned in Table 2. The result of this analysis is shown in Table 5. We emphasize technical measures, but organizational mechanisms are sometimes required and often beneficial.

To counter injection of vulnerabilities or flaws by **software service providers**, *code review* processes help assure sufficient oversight and reduce the probability for malicious activity to go unnoticed. From a framework perspective, *transparency* should be assured by embedding on-chain application code into the distributed ledger framework. Participants should be able to retrieve source code for a contract at any given time. Re-compiling source-code from a repository is too time-consuming and error-prone to serve as a verification method for smart contract bytecode.

Additionally, dedicated *vulnerability scanners* exist specifically for distributed ledger on-chain applications [40]. Whether these scanners are suitable for detecting insider-induced intentional manipulations of program flow has yet to be shown. Future empirical research may also analyze specific programming techniques that insiders use to attack distributed ledgers, similar to the study conducted by Collins et al. [15].

Table 5. Overview of potential mitigations for insider threats. O: organizational, T: technical.

	Threat	Mitigation	O/T
Software service provider	Vulnerability injection	Software trust management	O
		Code review and transparency	O, T
		Vulnerability scanners	T
Operator	Denial of service	Legal agreements	O
		Robust consensus algorithms	T
	Data manipulation	Automated detection and punishment	T
	Credential compromise	Credential revocation mechanisms	T
	Malicious misconfiguration	Configuration integrity checks	T
User	Unauthorized operations	Activity monitoring	O, T
		Anomaly detection	T
All	Information leakage	Granular Identity and Access Management	T
		Granular encryption	T
	Vulnerability abuse	Software update management	T

Intentional *denial of service* caused by a collusion of **operators** may be mitigated through legal agreements and incentives. While research on collusion in distributed ledger networks is still scarce, trust-based reputation algorithms could help. They may prevent collusion attempts or at least minimize their impact on network availability by punishing colluding peers. In the past, reputation systems based on peer-to-peer networks have also faced the issue of collusion [41]. Future research could thus transfer insights on collusion prevention from peer-to-peer reputation systems research and related areas to distributed ledgers.

Regarding consensus attacks targeting network availability and throughput, *robust consensus algorithms* such as RBFT [5] can help. Robust protocols sacrifice some throughput compared to traditional algorithms [14], but maintain high availability and throughput regardless of attacks.

Activity monitoring tools and corresponding organizational processes assist with timely detection of data manipulation. The threats listed in this work can serve as guidance for activities to monitor. To prevent insider abuse by unchecked node administrators, monitoring should be part of IS security management and organizationally separated from operational IS administration. Similarly, certificate authority and DLT node should be managed by separate entities. If any entity attempts to manipulate node settings, automated *configuration integrity*

checks should raise alerts in security monitoring units across the network. In case of external insiders, attack attempts should be punished either through legal agreements or a built-in incentive system.

Transaction *anomaly detection* could help spot unauthorized operations. If anomalies are detected in time, further abuse of permissions can be prevented. For traditional database systems, tools have been developed that automatically determine profiles of normal activity based on application profiles [34]. Similar security analytics tools can be developed to detect anomalous smart contract transactions.

One of the main countermeasures for information leakage by **users** is *granular identity and access management* (IAM) [27]. If users cannot send transactions or access ledger data without first being authorized by an application owner, the attack surface becomes significantly smaller. For authorized users, automated detection and timely removal of access privileges helps limit permission buildup and impact of insider attacks. Still, it remains challenging to ensure that each operator of the network keeps its individual permissions and access rights updated. Future work could examine how to enforce granular IAM network-wide, for example through automated checks or incentives.

Another tool to limit malevolent information disclosure is *granular encryption* of data, ensuring that users are only able to view data they need to access. Organizations must correctly decide where to encrypt data on the application level. Additionally, they should be parsimonious regarding confidential data stored on the ledger. Symmetric encryption keys may eventually be leaked, but on-ledger data cannot be deleted.

Overall, we observe that a holistic security monitoring concept is necessary for each organization participating in a distributed ledger network. A set of standardized monitoring metrics is a necessary prerequisite to detect manipulations of the various trust components. Due to unique differences in distributed ledger architectures, the metrics need to be customizable to the specific application context. One metric should be vulnerabilities in framework components, with automated *software update processes* attempting to minimize the number of vulnerabilities. Future work should focus on creating and evaluating such a security management framework for permissioned distributed ledger networks.

7 Conclusion

While distributed technology offers great benefits, organizations planning to take advantage of it should be aware of the trust relationships they enter. In this work, we established key trust actors and components for distributed ledgers to provide a better understanding of hidden trust factors and security risks. On the one hand, software trust in the components of a distributed ledger system is a key factor. If there are vulnerabilities or bugs, trust assumptions may be violated with grave consequences. Additionally, operators must still trust one another to some degree to cooperatively run a network. If this trust turns sour during operation, the distributed ledger network may become subject to denial of service or collusion attacks.

These attacks may be especially severe if carried out by insiders. They possess unique access to organizational resources that may facilitate subversion of distributed ledger trust assumptions. Insiders involved in application development may willingly inject vulnerabilities or malicious code. Node administrator insiders have elevated access rights to credentials and configuration. Malicious manipulation of these components may result in denial of service for the entire network. Modification or destruction of data are also possible in some cases (see Sect. 4), despite the integrity guarantees that distributed ledgers normally provide. Both internal and external insiders may leak data or abuse vulnerabilities in the distributed ledger software stack.

Due to the current lack of productive deployments of distributed ledger networks, this work focused on analyzing the potential impact of insider attacks from a theoretical perspective. To reinforce these claims, we elaborated on how insiders may exploit the architecture of popular distributed ledger frameworks. Future research may conduct case studies with real deployments of these frameworks to validate the listed insider threats and to further develop mitigations.

References

1. Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A.: ASTRAEA: a decentralized blockchain oracle. In: 2018 IEEE International Conference on Blockchain (2018). https://doi.org/10.1109/Cybermatics_2018.2018.00207
2. Al Khalil, F., Butler, T., O'Brien, L., Ceci, M.: Trust in smart contracts is a process, as well. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 510–519. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_32
3. Amoroso, E., Nguyen, T., Weiss, J., Watson, J., Lapiska, P., Starr, T.: Toward an approach to measuring software trust. In: Proceedings of 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 198–218 (1991). https://doi.org/10.1109/RISP.1991.130788
4. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, pp. 30:1–30:15. ACM, New York (2018). https://doi.org/10.1145/3190508.3190538
5. Aublin, P.L., Mokhtar, S.B., Quema, V.: RBFT: redundant byzantine fault tolerance. In: Proceedings of International Conference on Distributed Computing Systems, pp. 297–306 (2013). https://doi.org/10.1109/ICDCS.2013.53
6. Bansarkhani, R.E., Geihs, M., Buchmann, J.: PQChain: strategic design decisions for distributed ledger technologies against future threats. IEEE Secur. Priv. (2018). https://doi.org/10.1109/MSP.2018.3111246
7. Baracaldo, N., Joshi, J.: A Trust-and-risk aware RBAC framework: tackling insider threat. In: SACMAT 2012: Proceedings of the 17th ACM symposium on Access Control Models and Technologies (2012)
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Eprint.Iacr.Org (2018). https://doi.org/10.1016/j.bspc.2009.02.004
9. Bessani, A., Sousa, J., Alchieri, E.E.P.: State machine replication for the masses with BFT-SMART. In: DSN, vol. 6897, pp. 355–362, December 2014. https://doi.org/10.1109/DSN.2014.43

10. Cachin, C., Vukolic, M.: Blockchain consensus protocols in the wild. In: Richa, A.W. (ed.) 31st International Symposium on Distributed Computing (DISC 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 91, pp. 1:1–1:16. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2017). https://doi.org/10.4230/LIPIcs.DISC.2017.1
11. del Castillo, M.: Blockchain 50: Billion Dollar Babies (2019). https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain-50-billion-dollar-babies
12. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002). https://doi.org/10.1145/571637.571640
13. Chia, V., et al.: Rethinking blockchain security: position paper. In: 2018 IEEE International Conference on Blockchain (2018). http://arxiv.org/abs/1806.04358
14. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making byzantine fault tolerant systems tolerate Byzantine faults. In: NSDI 2009: Proceedings of the 6th USENIX symposium on Networked systems design and implementation (2009). https://doi.org/10.1145/1989727.1989732
15. Collins, M., Cappelli, D.M., Caron, T., Trzeciak, R.F., Moore, A.P.: Spotlight on: programmers as malicious insiders-updated and revised. Technical report, Software Engineering Institute, Carnegie Mellon University (2013)
16. Colwill, C.: Human factors in information security: The insider threat - who can you trust these days? Information Security Technical Report (2009). https://doi.org/10.1016/j.istr.2010.04.004
17. Dasgupta, D., Shrein, J.M., Gupta, K.D.: A survey of blockchain from security perspective. J. Bank. Financ. Technol. (2019). https://doi.org/10.1007/s42786-018-00002-6
18. De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V.: PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. CEUR Workshop Proceedings, vol. 2058, pp. 1–11 (2018)
19. Decan, A., Mens, T., Grosjean, P.: An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empir. Softw. Eng. (2019). https://doi.org/10.1007/s10664-017-9589-y
20. Deventer, M.O., et al.: Techruption Consortium Blockchain: what it takes to run a blockchain together. In: Proceedings of 1st ERCIM Blockchain Workshop 2018, Amsterdam, Netherlands 8–9 May 2018. European Society for Socially Embedded Technologies (EUSSET) (2018)
21. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: BLOCKBENCH: a framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD 2017, pp. 1085–1100. ACM, New York (2017). https://doi.org/10.1145/3035918.3064033. http://doi.acm.org/10.1145/3035918.3064033
22. Ekparinya, P., Gramoli, V., Jourjon, G.: The attack of the clones against proof-of-authority. CoRR (2019). http://arxiv.org/abs/1902.10244
23. ENISA: ENISA threat landscape report 2018. Technical report, ENISA (2019). https://doi.org/10.2824/622757
24. Ethereum Foundation: Go-Ethereum Website (2019). https://geth.ethereum.org
25. Franqueira, V.N.L., van Cleeff, A., van Eck, P., Wieringa, R.: External insider threat: a real security challenge in nterprise value webs. In: 2010 International Conference on Availability, Reliability and Security, pp. 446–453, February 2010. https://doi.org/10.1109/ARES.2010.40

26. Fröwis, M., Böhme, R.: In code we trust? In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) ESORICS/DPM/CBT -2017. LNCS, vol. 10436, pp. 357–372. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67816-0_20
27. Fuchs, L., Pernul, G.: Minimizing insider misuse through secure Identity Management. Secur. Commun. Netw. (2012). https://doi.org/10.1002/sec.314
28. Gambetta, D.: Can we trust trust? In: Trust: Making and Breaking Cooperative Relations, pp. 213–237. Blackwell (1988)
29. Glaser, F.: Pervasive decentralisation of digital infrastructures: a framework for blockchain enabled system and use case analysis. In: HICSS 2017 Proceedings, pp. 1543–1552 (2017). https://doi.org/10.1145/1235
30. Hawlitschek, F., Notheisen, B., Teubner, T.: The limits of trust-free systems: a literature review on blockchain technology and trust in the sharing economy. Electron. Commer. Res. Appl. **29** (2018). https://doi.org/10.1016/j.elerap.2018.03.005
31. Hearn, M.: Corda: a distributed ledger (2016). https://docs.corda.net/head/_static/corda-technical-whitepaper.pdf
32. Hileman, G., Rauchs, M.: 2017 Global Blockchain Benchmarking Study (2017)
33. Huseby, D.: Security Code Audits - Hyperledger Wiki (2019). https://wiki.hyperledger.org/display/HYP/Security+Code+Audits
34. Hussain, S.R., Sallam, A., Bertino, E.: DetAnom: detecting anomalous database transactions by insiders. In: CODASPY 2015 - Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (2015). https://doi.org/10.1145/2699026.2699111
35. Li, X., Jiang, P., Chen, T., Luo, X., Wen, Q.: A survey on the security of blockchain systems. Futur. Gener. Comput. Syst. (2017). https://doi.org/10.1016/j.future.2017.08.020. http://www.sciencedirect.com/science/article/pii/S0167739X17318332
36. Litke, A., Anagnostopoulos, D., Varvarigou, T.: Blockchains for supply chain management: architectural elements and challenges towards a global scale deployment. Logistics **3**(1) (2019). https://doi.org/10.3390/logistics3010005
37. Loch, K.D., Carr, H.H., Warkentin, M.E.: Threats to information systems: today's reality, yesterday's understanding. MIS Q. (1992). https://doi.org/10.1163/18781527-00401005
38. Locher, T., Obermeier, S., Pignolet, Y.A.: When can a distributed ledger replace a trusted third party? In: IEEE International Conference on Blockchain (2018). http://arxiv.org/abs/1806.10929
39. Lustig, C., Nardi, B.: Algorithmic authority: the case of Bitcoin. In: Proceedings of the Annual Hawaii International Conference on System Sciences (2015). https://doi.org/10.1109/HICSS.2015.95
40. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS 2016 (2016). https://doi.org/10.1145/2976749.2978309
41. Marti, S., Garcia-Molina, H.: Taxonomy of trust: categorizing P2P reputation systems. Comput. Netw. (2006). https://doi.org/10.1016/j.comnet.2005.07.011
42. MetaMask Contributors: MetaMask (2019). https://metamask.io/
43. Muskens, J., Chaudron, M.: Integrity management in component based systems. In: Proceedings of 30th Euromicro Conference, pp. 611–619 (2004). https://doi.org/10.1109/EURMIC.2004.1333429
44. R3: Corda Documentation (2019). https://docs.corda.net/releases/release-V4.1/

45. Sas, C., Khairuddin, I.E.: Exploring trust in Bitcoin technology: a framework for HCI research. In: Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction - OzCHI 2015 (2015). https://doi.org/10.1145/2838739.2838821
46. Schaffers, H.: The relevance of blockchain for collaborative networked organizations. In: Camarinha-Matos, L.M., Afsarmanesh, H., Rezgui, Y. (eds.) PRO-VE 2018. IAICT, vol. 534, pp. 3–17. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99127-6_1
47. Schneier, B.: Blockchain and Trust - Schneier on Security (2019). https://www.schneier.com/blog/archives/2019/02/blockchain_and_.html
48. Swihart, J., Winston, B., Bowe, S.: Zcash Counterfeiting Vulnerability Successfully Remediated - Zcash (2019). https://z.cash/blog/zcash-counterfeiting-vulnerability-successfully-remediated/
49. The Linux Foundation: Hyperledger Sawtooth Documentation (2019). https://sawtooth.hyperledger.org/docs/core/releases/1.1.5/contents.html
50. United States Department of Homeland Security: A Roadmap for Cybersecurity Research (2009). https://doi.org/10.1016/j.biortech.2007.06.061
51. Vo, H.T., Wang, Z., Karunamoorthy, D., Wagner, J., Abebe, E., Mohania, M.: Internet of blockchains: techniques and challenges ahead. In: 2018 IEEE International Conference on Blockchain. IEEE (2018). https://doi.org/10.1109/Cybermatics_2018.2018.00264
52. Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A.B., Chen, S.: The blockchain as a software connector. In: Proceedings of 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, pp. 182–191. IEEE, April 2016. https://doi.org/10.1109/WICSA.2016.21
53. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town crier: an authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016). https://doi.org/10.1145/2976749.2978326

Polystore and Tensor Data Model for Logical Data Independence and Impedance Mismatch in Big Data Analytics

Éric Leclercq, Annabelle Gillet, Thierry Grison, and Marinette Savonnet(✉)

LIB EA 7534 - University of Bourgogne, 21078 Dijon, France
{eric.leclercq,thierry.grison,marinette.savonnet}@u-bourgogne.fr
annabelle.gillet@depinfo.u-bourgogne.fr

Abstract. This paper presents a Tensor based Data Model (TDM) for polystore systems meant to address two major closely related issues in big data analytics architectures, namely logical data independence and data impedance mismatch. The TDM is an expressive model that subsumes traditional data models, it allows to link different data models of various data stores, and which also facilitates data transformations by using operators with clearly defined semantics. Our contribution is twofold. Firstly, it is the addition of the notion of a schema for the tensor mathematical object using typed associative arrays. Secondly, it is the definition of a set of operators to manipulate data through the TDM. In order to validate our approach we first show how our TDM model is inserted into a given polystore architecture. We then describe some use cases of real analyses using our TDM and its operators in the context of the French Presidential Election in 2017.

Keywords: Polystore · Data model · Logical data independence · Impedance mismatch · Tensor

1 Introduction and Motivations

In a globalized economy, driven by digital technologies, data become an element of added value and wealth. In addition to their volume, data are also becoming increasingly diverse in their production mode and their use. As all sectors of the economy are undergoing deep transformations of their activities, we observe an increase of demands to manipulate and analyze heterogeneous data (structured, semi-structured, and unstructured data). For example, the agriculture industry needs to forecast product demand and analyze weather conditions in order to decide when to sow, etc. Large cities are being converted into smart cities, with applications covering a wide range of citizen needs, such as public transportation or optimization of traffic. Insurances and financial institutions

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
A. Hameurlain and R. Wagner (Eds.): TLDKS XLII, LNCS 11860, pp. 51–90, 2019.
https://doi.org/10.1007/978-3-662-60531-8_3

have significant needs for domain specific applications of big data analytics in order to be able to trace and detect potential risks for their clients, products and customers. In the energy sector, the implementation of sensors, remote control tools and smart-grids generate data which can be used to optimize and adapt network delivery to user needs. In the field of health, sociodemographic and health data available from different sources can be used to support diagnosis, to identify disease risk factors, or to assist practitioners with the choice of treatments. In marketing, social networks data are used to discover communities, opinion leaders or to detect and study the propagation of fake news. Even though all these domains use data from various sources, two main underlying principles can be observed in the previous examples: the first is to store data as events (i.e., actions, measures, logs) and the other is to store data as entities described by properties with operations that implement state changes.

In order to study these different kinds of data in depth, i.e., to go through data, information and knowledge layers, complex analytics processes are defined. Nevertheless, data analysis tasks are conditioned by objectives and therefore such tasks require compliant data structures, e.g., time series, grids, cubed sphere, complex networks, multi-layer networks, co-occurrence matrices, etc. These data structures can be completely different from models supported by storage systems.

Thus, big data analytics infrastructures must support all corresponding activities and not just the execution of algorithms which have different theoretical foundations such as graph theory, linear algebra, statistical models, Markov models. In this context, ETL (Extract Transform Load) processes are commonly used. Their role is not limited to the data ingestion phase but they are also used for extracting data from data storage systems to feed algorithms. As pointed out by Byron Ruth in a presentation entitled "ETL: The Dirty Little Secret of Data Science" at the OSCON conference in 2014, ETL processes are expensive to define and to set up. Moreover the data transformation scripts usually include implicit knowledge that hampers their reuse.

The logical data independence property in database systems insulates programs from the logical database schema. It was introduced to cope with changes of the logical database design or to maintain schema structure required by legacy applications [61]. This property has also been studied in the context of database interoperability [41] and federated database architectures [33,72,84]. The same property is also of considerable importance in the analysis of big data, and more particularly in the analysis of scientific data. The subjects, objects or entities on which analyses are carried out are not necessarily well-defined. This is especially true when the aim of analyses is scientific research or when the phenomenon being analyzed is poorly understood. Consider, for example, complex networks or the emerging science of networks [11] where an interdisciplinary community of researchers tries to elaborate tools and theories to study complex systems. These characteristics are not typically observed in Business Intelligence where enterprise data semantics is usually well-defined (e.g., there is no ambiguity about a person who is a customer and about its attributes). Thus, ensuring logical independence is a key issue for systems that analyze complex data (e.g., scientific data) using

multiple types of algorithms (and *a priori* unknown), as it makes model transformations explicit and can therefore limit the role of ETLs.

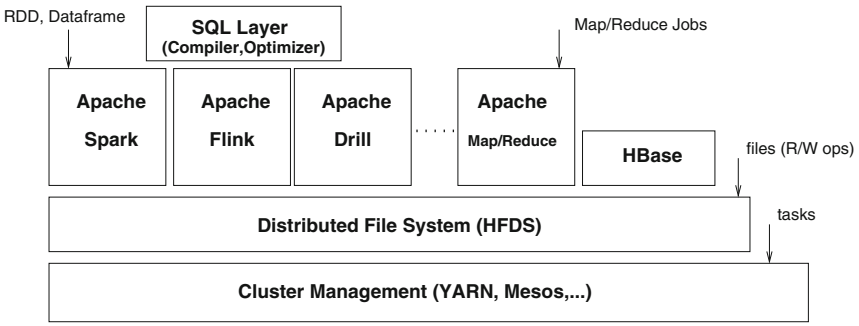


Fig. 1. Examples of programming abstractions used in Hadoop eco-system

The impedance mismatch describes mutual incompatibility between programming paradigms and database capabilities, mostly at the data model level [18,53]. As stated by Gray in [30], the impedance mismatch has made it difficult to map numerous science applications into conventional relational database systems. This problem has also been identified in big data platforms [57], where lots of different systems are used in a large software analytics stack to efficiently address specific issues. Before looking at algorithm data structures, it should be noted that different big data analysis tools use different programming and language paradigms. In [85] authors have identified eight main classes of programming paradigms. Figure 1 provides examples of programming abstractions used in the Hadoop eco-system. MapReduce forces the developer to decompose, if possible, algorithms into a sequence of mapping and reducing tasks. Streaming systems, such as Apache Kafka[1] or Flink[2], require algorithms to perform repeatedly simple operations on streams with or without a shared state. It turns out that incremental algorithms [27,62,73] are well-suited for this programming model. The GPU computing requires to organize computation in a SIMD[3] like approach, and linear algebra algorithms fit well with this paradigm. Algorithms have also their own internal data structures that amplify the problems of impedance mismatch, e.g., graph algorithms can use adjacency lists, adjacency matrices or laplacian matrices. Thus, the impedance mismatch problem between storage systems and analytics tools is at least existing at three different levels: at the computing paradigm level, at the programming language level, and at the data structure level. In order to overcome the impedance mistmatch, developers, same as in the case of the problem of logical data independence, set up complex

[1] https://kafka.apache.org/.
[2] https://flink.apache.org/.
[3] Single Instruction Multiple Data.

model transformations to extract data from storage systems, while conforming to the programming paradigm and to the data structure required by algorithms.

We propose to revisit the issues of logical data independence and impedance mismatch in the context of polystore. We develop an approach that integrates analysis tools, such as Spark, R, Drill, TensorFlow, with storage systems in a loosely coupled architecture. Our approach relies on a definition of a Tensor based Data Model (TDM) that subsumes traditional data models (relational, NoSQL, graph, etc.) and proposes a set of operators for defining virtual and materialized views to be able to dynamically transform data into appropriate data structures required by algorithms. The use of the tensor mathematical object as the foundation for a data model brings additional benefits. Tensor decompositions are powerful tools for extracting latent information [69]. TDM and its operators allow users to quickly feed algorithms with data originating from several storage systems, thus helping to reduce the complexity of ETL tools by specifying models and data transformations as sequences of operations in TDM. At the same time, it also reduces the impedance mismatch among components by establishing specific views. Moreover, storage systems included in the polystore architecture can continue to be used directly by programs written in native languages of such systems (e.g., SQL, Cypher, REST API, etc.).

The remainder of the paper is organized as follows. Section 2 describes and discusses multi-model data management approaches including multi-model database systems, multi-paradigm systems and polystore. Section 3 discusses DBMS architectures and describes our polystore architecture. Sections 4 and 5 present TDM a Tensor based Data Model and define its main operators. Section 6 relates experiments with the software architecture and provides two different use cases. Experiments have been carried out within the TEP 2017 project which studies the use of Twitter during the French Presidential Election in 2017. Section 7 studies two aspects of performance of TDM: (1) in terms of space complexity for data structures and time complexity for operators, and (2) in terms of experimental evaluation.

2 Related Work on Multi-model Data Management

During the 1990s, the need for non-relational data processing has been identified in many application areas, and thus a number of database systems have focused on a specific type of applications, e.g., object-oriented databases, XML databases, spatial databases or RDF databases. In the early 2010s, a large number of NoSQL systems have been created to tackle specific issues without maintaining all the properties of database management systems. A number of research projects are now shifting the focus to building systems for support of multiple data models. We can distinguish two existing approaches to definition, manipulation and querying multi-model data.

Multi-model database systems manage different data models with a fully integrated backend i.e., within a single system. They thus contribute to the logical data independence, as such a system can handle complex and costly data

transformations (i.e., feature selection, export and conversion of data) before applying analysis algorithms.

Multi-paradigm systems include multiple dedicated DBMSs under the umbrella of an integrated platform, i.e., a mediator. These systems are designed as an answer to some pragmatic considerations. First, limiting heterogeneous data in order to fit them into a single data model could be a hindrance for establishment of complex data analyses and could lead to performance problems. Second, as stated by Stonebraker in [76,77], "one size fits all" is not an appropriate solution for modern data-intensive applications. Likewise, Ghosh [28] explains that storing data in the same way as within an application simplifies programming and makes it easier to decentralize data processing.

The study of these two approaches is essential for developing a multi-model storage system architecture. We detail and discuss these two approaches below.

2.1 Multi-model Databases

The first category of approaches, called multi-model databases, aims at building or extending a single database system to manage multi-model data. Multi-model databases offer a unifying query language and an API that cover all data models and allow a joint use of different models within a single query. Currently, many database systems claim to be multi-model databases. However, the level of support for multi-model data varies widely, offering different capabilities to write queries across multiple distinct models, and to optimize query plans with different models.

A Taxonomy. A database system can acquire multi-model properties either by offering layers for additional data models or by supporting different data models directly in its storage engine.

The first wave of multi-model databases has appeared after the year 2000 with the object-oriented paradigm and the emergence of XML. Major relational DBMSs were extended to store and to process various types of data including XML, geographical, textual and object. SQL was extended by the SQL/XML standard or its variants. More recently, these database systems were often enhanced to support the JSON format. For this type of multi-model databases, only the relational model is the first class citizen, which means that all other models are developed on the top of the relational technology. The second wave has appeared after the year 2010 with the era of Big Data and the appearance of NoSQL databases; the need to process a wide variety of data was well identified and some NoSQL databases have proposed to natively support different models.

Representative Systems. We present some representative multi-model database systems by classifying them according to the original data types supported.

The first original database type is relational. Oracle8 was released in 1997 with an object-relational layer. Oracle 9i introduced in 2001 the ability to store

XML and use it in queries. Oracle 12c was designed in 2013 for the cloud, and it featured an in-memory column store, the JSON support for documents, as well as RDF data with the Oracle Graph module. Oracle provides in-database analytics queries implemented as packages, including functions available in SQL. Since 2006, PostgreSQL, a classical relational DBMS, supports storage of key/value pairs in a specific data type `hstore` which maps keys to string values or to other `hstore` values. Since 2013, PostgreSQL supports storage of the JSON format with specific data types `json` and `jsonb`, which can be queried using SQL extensions for JSON. HPE Vertica, a commercial version of C-Store [78], stores data tables as columns (i.e., not as rows). It was also extended with `flex tables`[4] which do not require schema definitions and allow to store semi-structured data (XML, JSON). Creating flex data is similar to creating classical tables. Vertica implicitly adds a column `_raw_` which stores the loaded semi-structured data. Vertica also adds an auto-incrementing column `_identity_` for `flex table` without an additional column definition. The loaded data are stored in an internal map data format `VMap` which is a set of key/value pairs. Vertica supports standard SQL interface enhanced by a variety of in-database analysis algorithms, including linear regression, logistic regression, k-means clustering, naive Bayes classification, random forest decision trees, and it also supports vector machine regression and classification.

The second type of systems has its roots in the NoSQL wave. ArangoDB[5] is an open source document database which was from the beginning (2011) created as a native multi-model database. In ArangoDB, documents are represented in the JSON format and grouped into collections. A document collection has a primary key and in the absence of other attributes the document collection behaves like a simple key/value store. More recently, ArangoDB supports graph data model. It has extended its original storage strategy with special edge collections. They include two special attributes, `_from` and `_to`, which allow to create relations between documents (vertices). ArangoDB query language (AQL) is similar to SQL with graph traversal. OrientDB[6] supports graph, key/value, document, and object models. An element of storage is a record having a unique id corresponding to a document (formed by a set of key/values), and a vertex or an edge. Relationships are managed as in graph databases with direct connections between records. OrientDB supports schema-less, schema-full and schema-hybrid modes. OrientDB allows to query data with graph traversal language Gremlin or SQL extended for graph traversal where the navigation across relations is possible using the dot notation.

Another type of system differentiates itself by the use of an hybrid database engine suitable for different kind of applications, e.g., OLTP, OLAP and stream processing. In OctopusDB[7] [23], all database operations create logical log-entries.

[4] https://www.vertica.com/docs/9.2.x/HTML/Content/Authoring/FlexTables/FlexTableHandbook.htm.
[5] https://docs.arangodb.com/3.4/Manual/index.html.
[6] https://orientdb.com/graph-database/.
[7] https://bigdata.uni-saarland.de/projects/octopusdb.php.

Each log entry contains a unique log sequence number, the database operation performed, and the input parameter. Based on this log, several physical representations of the log (called storage views SV) are defined: for instance, OctopusDB can create `RowSV` for transactional queries, or `ColumnSV` for read-only queries. As another example, SAP Hana [25] is a distributed in-memory database with features for the integration of OLTP and OLAP and the unification of structured, semi-structured (graph) and unstructured (text) data. All data are kept in memory as long as there is enough space available, otherwise data are unloaded from memory. SAP Hana provides rich data analytics functionalities through multiple query language interfaces, such as SQL, R and temporal queries.

Table 1. Classification of multi-model databases

Data model	DBMS	Supported formats	Storage strategy	Query languages
Relational	Oracle	XML, JSON, RDF, object	row store	SQL, XML based or JSON extension
Relational	PostgreSQL 9.4	key/value, XML, JSON, object	row store	Extended SQL
Relational	Vertica	JSON, XML	column store, flex tables + Vmap	SQL-like
Document	ArangoDB	JSON, key-value, graph	document store allowing references	SQL-like AQL
Graph	OrientDB	JSON, key-value, graph	key/value pairs + object-oriented links	Gremlin, SQL extended with path traversal

In summary, since 2017 all leading DBMSs offer multiple data models. The supported data models include relational, document, XML, graph and object. Cross-model languages are based on the extension of SQL, XML query languages, and graph languages. Table 1 presents various multi-model databases and criteria retained. Lu et al. [59] in their survey analyze multi-model databases using different criteria which also correspond to *"one size fits a bunch"* viewpoint [5].

2.2 Multi-paradigm Storage Systems

The problem of access to heterogeneous data sources has been addressed for many years by research communities using schema integration and multi-database systems [67]. As a result, several research projects have been inspired by previous work on distributed databases in order to take advantage of a federation of specialized storage systems with different models[8]. Multi-paradigm data storage relies on multiple data storage technologies, chosen according to the way data is used by applications and/or by algorithms [71]. However, the problem of accessing multiple heterogeneous and autonomous data sources is amplified by

[8] http://wp.sigmod.org/?p=1629.

some other issues: (1) NoSQL systems do not always have a well-established separation between their logic model and physical model, so the traditional export schema [72] can be difficult to define; (2) to achieve flexibility, new systems do not necessarily provide a well-defined schema but rather an intricate application oriented schema that does not promote reuse.

A Taxonomy. In [79] authors propose a survey of such systems and a taxonomy in four classes: (1) **federated database systems** as a collection of homogeneous data stores and a single query interface; (2) **polyglot systems** as a collection of homogeneous data stores with multiple query interfaces; (3) **multistore systems** as a collection of heterogeneous data stores with a single query interface; (4) **polystore systems** as a collection of heterogeneous data stores with multiple query interfaces.

In order to obtain more meaningful groups of systems we adopt a slightly different classification that replaces federated database systems by a more specific class of pragmatic systems characterized by a common query language. Thus, our updated classification is based on data models and query languages by: (1) considering multi-database query language approach [58] instead of federated systems to better represent the autonomy of data sources and the existing enterprise-oriented systems; (2) replacing homogeneity of data model by relational-like models, for example for JSON and the relational model [10,22]; (3) instead of using architecture elements such as query interface or query engine as a criterion we prefer query language. According to these criteria our classification is (Table 2): multi-database query language (unique language), polyglot system including relational-like data models (with multiple languages), multistore, and polystore. For each of these classes we describe some of the most significant representative systems.

Table 2. Classification of multi-paradigm storage approaches

Data model	Query language	
	Single	Multiple
Single or "relational-like"	Multi-database	Polyglot
Multiple	Multistore	Polystore

Representative Systems. Spark SQL[9] is the major representative of multi-database query language. It allows us to query structured data from relational-like data sources (RDBMS, JSON, Parquet, etc.) in Spark programs using SQL. Connections to major RDBMS are established using JDBC. Apache Drill[10] is similar to Spark, but without having a very large support of analysis algorithms

[9] https://spark.apache.org/sql/.
[10] https://drill.apache.org/.

as Spark has with MLlib and GraphX. In Icarus [82], multiple data store engines (row- or column-store) are queried by PolySQL, a SQL dialect used as a common query language. All data are stored in all stores, and the per-query routing mechanism is able to select, for each query, the best data store. The performance benchmark of Icarus is based on the Gavel benchmark. The joint use of different stores specialized in OLTP or OLAP has shown a speedup of up to a factor of three. Giannakouris et al. [29] propose MuSQLE, a multi-database system built using Spark. It supports a multi-engine execution cost estimation for each SQL engine.

According to our classification, CloudMdsQL [51] is a polyglot system, rather than a multistore system as suggested by the title of one of the articles of these authors published before the first taxonomy proposal. CloudMdsQL is a functional SQL-like language, designed for querying multiple data store engines (row- or column-store) within a query that may contain sub-queries to each data store's native query interface. SQL++, which is a part of the FORWARD platform[11], is a semi-structured query language that encompasses both SQL and JSON [64].

HadoopDB [2] coupled to Hive[12] is a multistore that uses the map-reduce paradigm to push data access operations on multiple data stores. D4M (Dynamic Distributed Dimensional Data Model) [44] is a multistore that provides a well-founded mathematical interface to row stores. D4M allows matrix operations and linear algebra operators composition and applies them to the row stores. D4M reduces the autonomy of data stores to achieve a high level of performance [45].

The BigDAWG system [24,26] is a polystore allowing to write multi-database queries with reference to islands of information, each island corresponding to a type of data model (PostgreSQL, SciDB and Accumulo). Myria [83] supports multiple data stores, as well as different data computing systems such as Spark. Myria supports SciDB for array processing, RDBMS, and HDFS. The RACO (Relational Algebra COmpiler) acts as a query optimizer and processor for MyriaL language. Myria also supports user data functions in different other languages such as Python. Morpheus [4] is a polystore approach, implemented using Spark, that focuses on Cypher query language instead of SQL and takes advantage of graph analysis algorithms implemented in Neo4j. ESTOCADA [15,16] combines multiple data stores with different data models; it allows to query these stores using their native languages and provides an unified API to combine partial answers relying on query reformulation techniques under constraints. In QUEPA [60], users are able to query the polystore without knowing the exact structure of each data store. Using record linkage, QUEPA returns records enriched with relevant tuples from other stores. MISO [55] has focused on the tuning of the physical design of polystores by optimizing the placement of data on data stores with the objective to reduce data movement during query processing.

[11] http://forward.ucsd.edu/.
[12] https://hive.apache.org/.

2.3 Discussion and Other Approaches

In multi-model databases, the independence mismatch is not really solved, as data still need to be restructured before applying algorithms or functions. In the in-database analytics approach, algorithms are implemented as black-boxes and naturally not all new algorithms could be implemented. Recent algorithms are rarely implemented in DBMSs, and matrix operations and associated factorizations are not directly supported by traditional storage systems. In the case of graph analysis tools, only a few NoSQL systems like Neo4j support a small set of algorithms[13]. However, Neo4j does not allow to manage very large amount of data with attributes as the column-oriented systems would do [37]. The situation for machine learning algorithms and tools is almost similar. Only some recent systems such as Vertica[14] or SciDB[15] support standard machine learning algorithms as black-boxes. Extending the capabilities of the system by adding new algorithms require specific and complex development using User Defined Functions (UDF).

Polystores are designed to make the best use of the data models, combining systems by unification with languages. They do not need to re-implement data management functionalities and can instead reuse proven storage systems. Data exchange between storage systems is a rather expensive operation and administration of multiple separate storage systems increases the management costs compared to multi-model databases. Moreover, polystores add complexity because they do not possess a global schema and they use different languages. Developers must have an implicit knowledge of different schemas on which the application is based. Recently, new work publications trying to solve this problem have appeared. Data Civilizer [21,65] has a linkage graph computation module to build a linkage graph for the data and a data discovery module which utilizes the linkage graph to help users to identify data for their analyses. The linkage graph is also used to determine possible join path in the queries. SemTK [63] provides capabilities to interact with a knowledge graph to facilitate essential tasks such as ingesting, querying and manipulating data. It simplifies user tasks by providing the users with a single logical interface for retrieving and integrating data from multiple repositories. In [36] the authors propose a hypergraph-based approach for representing the catalog of metadata in a polystore system. They also provide a simple query rewriting algorithm using this catalog.

Several kinds of data analytics platforms have also been defined in the last few years [74]. They are usually an aggregation of existing technologies and can be classified into computation-centric architecture or data-centric architecture. Two main typical architectures are data analytics stacks and data lakes.

New data analytics stacks have emerged as a suitable infrastructure for providing access to data stores and enabling data processing workflows to execute

[13] https://neo4j.com/developer/graph-algorithms/.
[14] https://www.vertica.com/product/database-machine-learning/.
[15] https://www.paradigm4.com/.

data analytics operations. Frameworks such as TensorFlow[16], Theano[17], Keras[18] or MLlib in Spark[19] have been developed to design machine learning tools using data structures linked to algorithms. As a result, these systems require the development of complex, hard to reuse and often error-prone programs for loading and transforming data [1,35]. These systems supply neither storage mechanism nor simple tools for connecting to data sources. The Berkeley Data Analytics Stack (BDAS) from the AMPLAb project[20] is a multi-layer architecture that provides data processing using the Apache Spark ecosystem and a multiple storage layer (multistore) using Alluxio[21].

The IT industry uses the metaphor of data lake to define shared data environment consisting of multiple repositories. A data lake provides data to a variety of processing systems including streaming. The solutions are mature and there are products available on the market, such as the Microsoft Azure data lake[22], and the IBM data lake[23]. Alluxio included in BDAS is also a data lake system. The value that can be extracted directly is low compared to investments required to implement data mining and business intelligence solutions. Indeed, the polystores differ from the data lake in two aspects. The first aspect concerns the objective, data stored in a polystore are meant for a short-term analysis, while those in a data lake are annotated for medium-term or long-term use. The second aspect concerns performance, as polystores are designed to make the best use of the most suitable data storage models by combining them, while data lakes store (most of the time) data in their original format.

3 Architecture

In this section, we present our polystore architecture, before we recall the founding principles for RDBMS in terms of functional and logical architectures.

3.1 Software Architectures of RDBMS

As for operating systems or network protocols, Härder and Reuter [31,34] have proposed a decomposition of the functional architecture of a DBMS in five layers: (i) file management that operates on blocks and files, (ii) propagation controls that define and manage segments and pages, (iii) records and access path management that works on access path and physical records, (iv) record oriented navigational access that describes records, sets, hierarchies and (v) non-procedural or algebraic access that defines tuples, relations, views and operators

[16] https://www.tensorflow.org/.
[17] http://deeplearning.net/software/theano/.
[18] https://keras.io/.
[19] https://spark.apache.org/mllib/.
[20] https://amplab.cs.berkeley.edu/software/.
[21] http://www.alluxio.org/.
[22] https://azure.microsoft.com/en-us/services/data-lake-analytics/.
[23] https://www.ibm.com/analytics/data-lake.

for logical schema description and data retrieval. Yet most of RDBMS rather use fewer layers following System R [7] which defines two layers: (1) the Relational Storage System (RSS) with a Relational Storage Interface (RSI) handles access to tuples and manages devices, space, allocation, storage buffers, transaction consistency and locking as well as indexes; (2) the Relational Data System (RDS) with a Relational Data Interface (RDI) provides authorization, integrity enforcement, and support views of data, as well as a definition, manipulation and query language. The RDS also maintains the catalogs of names to establish correspondences with internal names in RSS.

From a logical point of view, the ANSI/SPARC architecture [14] characterizes classical data management systems (relational, object-oriented) by proposing a 3-layer decomposition that reflects the abstraction levels of data: (i) the external data schemas describing the different external views over data dedicated to end-users or applications; (ii) the logical or conceptual schema describing entities and relationships among them, including integrity constraints and (iii) the physical schema describing the storage and the organization of data.

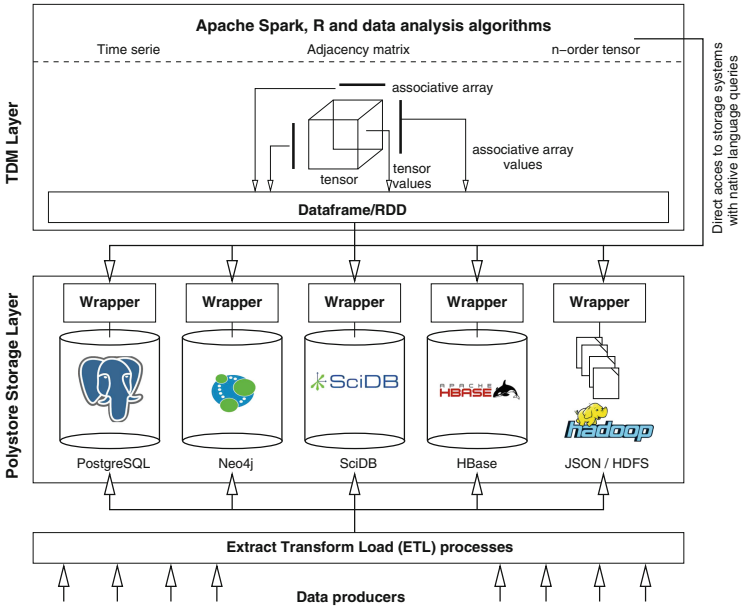


Fig. 2. Architecture of the polystore system using TDM

NoSQL systems, beyond their model differences, exhibit a common characteristic with respect to the architecture: the external and logic levels disappear [80]. As a consequence, the applications are close to the physical level with no real *logical independence* between programs and data schema. Moreover, due to the nature of schema-less NoSQL systems, the source code contains implicit

assumptions about the data schema. These drawbacks can lead to maintenance and performance problems and can make it difficult to set up a data curation process. The ingestion phase can be accomplished quite easily (i.e., a feature puts forward by data lake vendors) but the data transformation, schema integration, data cleaning and entity consolidation are heavily hindered by the lack of logical and external schemata. All these systems leave aside good principles of the RDBMS which go beyond the relational model and SQL language.

3.2 TDM and Polystore Architecture

The architecture setup includes a polystore storage layer built on the top of PostgreSQL, HDFS, and Neo4j and an abstraction layer developed using R and Spark connectors (Fig. 2). The abstraction layer comprises the Tensor Data Model and can include different analysis frameworks such as R, Spark, Drill and TensorFlow. TensorFlow is similar to libraries supporting tensors in R or Spark, but it has been designed with a workflow orientation, rather than with a data model orientation. In this particular case, a tensor is a structure of exchange among processes of a complex workflow, rather than a model to represent real/native data.

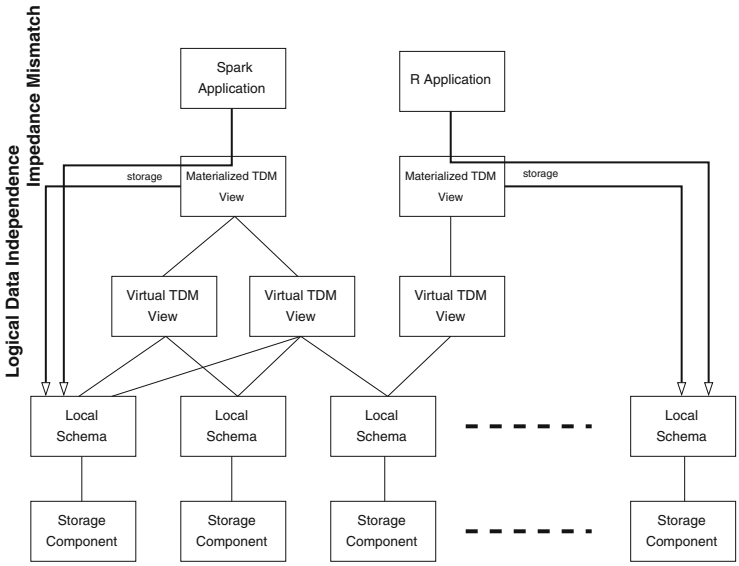


Fig. 3. Logical architecture of the polystore system using TDM

Our objective is to preserve the local autonomy of the storage systems without considering updates of the data except those consisting in materializing results of model transformations or results of analyses. According to the polystore approach, it is possible to use either the native language of each system or

our Tensor-based Data Model to express queries. TDM supports the *logical data independence* using TDM operators to define views (virtual views or materialized views). The *impedance mismatch problem* is addressed at the application level using the most appropriate TDM view and its materialization in a storage system (Fig. 3). For example, a virtual TDM view is used to define an adjacency matrix that represents a co-occurrence graph of hashtags that is then stored in SciDB to be used by a R program.

The first class citizen of TDM is the tensor mathematical object. A tensor is often defined as a generalized matrix, 0-order tensor is a scalar, 1-order tensor is a vector, 2-order tensor is a matrix, tensors of order 3 or higher are called higher-order tensors. Tensor dimensions/orders and tensor elements are represented by associative arrays. Queries for tensor construction are submitted to the *wrappers*; the *wrappers* send back $N + 1$ attributes where the first N attributes are dimensions and the last attribute serves as a value for the elements of the tensor (obtained with `GROUP BY`-like queries on attributes that represent the dimensions). This feature allows us to implement *wrappers* having all the same structure and thus to simplify model transformations. For example, Fig. 4 presents SQL queries submitted to PostgreSQL, and where the resulting tensor represents tweets published by users by time slice. For the R language, the *wrappers* are implemented using the packages R DBI, RNeo4j[24], RMongo, RCassandra et RHBase[25]. In Spark, we work with the SQL layer, data frame and the RDD (Resilient Data Sets) abstractions.

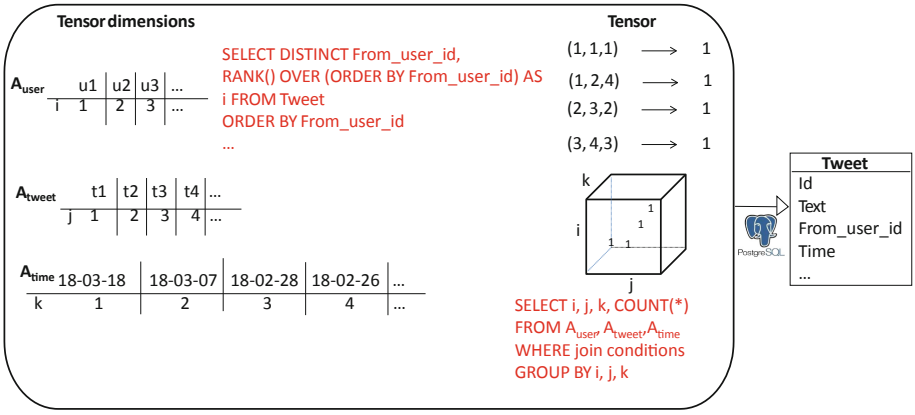


Fig. 4. Tensor construction from relational table

[24] https://github.com/nicolewhite/RNeo4j.
[25] https://github.com/RevolutionAnalytics/rhbase.

4 Core Concepts of TDM

This section addresses the definition of Tensor Data Model (TDM), starting with the tensor mathematical object, adding the notion of typed schema based on associative arrays and then defining a set of data manipulation operations. We also study mappings between TDM and other data models.

Tensors are very general abstract mathematical objects which can be considered according to various points of view. A tensor can be seen as a multi-linear application, as the result of the tensor product, as an hypermatrix. We will use the definition of a tensor as an element of the set of the functions from the product of N sets $I_j, j = 1, \ldots, N$ to $\mathbb{R} : \boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, where N is the number of dimensions of the tensor or its order or its mode. In a more general definition, a tensor is a family of values indexed by N finite sets, thus, a tensor is often treated as a generalized matrix. Tensor operations, by analogy with operations on matrices and vectors, are multiplications, transposition, unfolding or matricization and factorizations (also named decompositions) [19,50].

In the rest of the article, we use the boldface Euler script letters to indicate a tensor $\boldsymbol{\mathcal{X}}$, boldface capital letters $\mathbf{M}$ for matrices, boldface lowercase letters to indicate a vector $\mathbf{v}$, and an element of the tensor or a scalar is noted in italic, for example x_{ijk} is ijk-i-th element of 3-order tensor $\boldsymbol{\mathcal{X}}$.

4.1 TDM's Data Model

In TDM, tensor dimensions and values are represented by associative arrays. In the general case, an associative array is a map from a key space to a value space and can be implemented using a hash table or a tree.

Definition 1 (Associative Array). *An associative array is a map that associates keys to values as follows* $A : K_1 \times \cdots \times K_N \rightarrow \mathbb{V}$ *where* $K_i, i = 1, .., N$ *are the sets of keys and* $\mathbb{V}$ *is the set of values.*

The definition given in [40] restricts $\mathbb{V}$ to have a semi-ring structure and the associative array to have a finite support. In TDM we use associative arrays in three different cases.

First, we use different associative arrays denoted by $\mathbf{A}_i$ for $i = 1, .., N$ to model dimensions of a tensor $\boldsymbol{\mathcal{X}}$, in this case the associative array has only one set of keys associated with integers $\mathbf{A}_i : K_i \rightarrow \mathbb{N}$ and $\mathbf{A}_i$ represents a bijection function. For example $\mathbf{A}_1 : String \rightarrow \mathbb{N}$ associates integers to user names. The user name values are obtained by native queries sent to storage systems.

Second, an associative array can be used to represent values of a sparse N-order tensor by associating compound keys from dimensions to values (real, integer) $\mathbf{A}_{vst} : K_1 \times \cdots \times K_N \rightarrow \mathbb{V}$.

Third, for tensors with non-numerical values, two associative arrays are used as an indirection, one array to map key dimensions to a set integer keys ($\mathbf{A}_{vst}$) and another array to map the integer keys to non-numerical domains values (one integer is associated with each different value).

Definition 2 (Named Typed Associative Array). *A named and typed associative array is a triple* $(Name, \mathbf{A}, T)$ *where* $Name$ *is a unique string that represents the name of a dimension,* $\mathbf{A}$ *is the associative array, and* T *is the type of the typed associative array, i.e.,* $K \rightarrow \mathbb{N}$.

The schema of a named typed associative array is $Name : K$, application A is implicit. Dom_{Name} is the domain of values taken by the keys of A, i.e., a subset of K.

Definition 3 (Typed Tensor). *A typed tensor* $\boldsymbol{\mathcal{X}}$ *is a tuple* $(Name, D, V, T)$ *where* $Name$ *is the name of the tensor,* D *is a list of named typed associative arrays, i.e., one named typed associative array per dimension,* V *is the values of the tensor and* T *is the type of the tensor, i.e., the type of its values.*

For example, if a tensor represents the number of times a hashtag is published by a user during one hour, the schema will be $\boldsymbol{\mathcal{UHT}}(U : String, H : String, T : Integer) : Integer$. Figure 4 shows the tensor $\boldsymbol{\mathcal{UTT}}(U : String, T : Integer, T : Integer) : Integer$ that represents tweets published per user and per hour.

Remark 1 (Tensor and domain). The domain of a tensor $Dom_{\boldsymbol{\mathcal{X}}}$ is the set of tensor values. Another way of seeing the values of the tensor, by analogy with the formalization of the relational model proposed by [42], is to consider it as the subset of the Cartesian Product of the dimension domains and the proper domain of $\boldsymbol{\mathcal{X}}$, i.e., $Dom_{d_1} \times Dom_{d_2} \times \cdots \times Dom_{d_N} \times Dom_{\boldsymbol{\mathcal{X}}}$ or $K_1 \times K_2 \times \cdots \times K_N \times Dom_{\boldsymbol{\mathcal{X}}}$. D refers to the set of named typed associative arrays representing the dimensions or domains of $\boldsymbol{\mathcal{X}}$. Thus the Cartesian Product of the domains can also be written:

$$\left(\prod_{d \in D} Dom_d\right) \times Dom_{\boldsymbol{\mathcal{X}}} \quad \text{or} \quad \left(\prod_{i=1,\ldots,N} Dom_{d_i}\right) \times Dom_{\boldsymbol{\mathcal{X}}}$$

The schema of a typed tensor is $Name(S) : T$ where S is the list of schemas of its dimensions, i.e., associative arrays of D. More strictly and by analogy with the relational model, the formal schema of a tensor is the list of names of dimensions to which the name of the tensor is added. A TDM schema is a set of typed tensor schemas.

If we consider the representation of tensor values, V handles the sparsity of tensors. Sparse tensors have a default value (e.g., 0) for all the entries that do not explicitly exist in the associative array.

An associative array refers to the general mathematical concept of a map or a function. As we want to conform to the separation of logical and physical levels, the associative arrays in the model are abstract data types that can be implemented using various representation techniques. For example, Kuang et al. [52] describe a unified tensor model for representing unstructured, semi-structured, and structured data. These authors also propose a tensor extension operator for representing various types of data as sub-tensors merged into a unified tensor. Lara [38,39] proposes a logical model and an algebra using an associative array (called associative table) with a set of operations for unifying

different data models, such as relational, array and key-value. The authors show how to use Lara as a middleware algebra, and how their approach is directed towards operator translation and optimization. However their model is not very suitable for expressing high-level data transformations as tensors can do with their capacity of modeling complex relationships (i.e., not being only binary). Moreover, tensors support complex and powerful factorization or decomposition operators, such as CP or Tucker, that have proved their efficiency for data mining [69]. A lot of work has been directed towards scaling up tensor decompositions by exploiting their sparsity. For example, Kang et al. [43] propose a method for avoiding intermediate data explosion induced by ALS and benefiting from map-reduce paradigm. The MLog system [56] is an hybrid approach which defines a tensor data model with operators. The authors study optimization techniques for queries over TensorFlow.

4.2 Mapping Between TDM and Other Data Models

In this subsection, we establish mappings between TDM and other data models with the assumption that associative arrays are invariant to permutation of keys.

1. **Relation:** A relation R is a set of tuples $(v_1, v_2, \ldots, v_k)$, where each element v_j is a member of a domain Dom_j, so the set-theoretic relation R is a subset of the cartesian product of the domain $Dom_1 \times Dom_2 \times \cdots \times Dom_k$. We can write a typed tensor $\boldsymbol{\mathcal{X}}$ using the name of each associative array in D as domain $Dom_i, i = 1, \ldots, N$ for R and by adding an attribute whose domain is the name of the tensor. The values of a tuple are those corresponding to keys of each D associated with the values of $\boldsymbol{\mathcal{X}}$. The names of D form a compound key for R. The reverse mapping from a relation R to typed tensors produces a set of tensors $\boldsymbol{\mathcal{X}}_i$ where the dimensions are the n attributes that form together the key of R and for the $k-n$ remaining attributes we create a tensor for each. The keys of each D are formed of different values of each attribute domains. In Fig. 5(a), we show an example of relation R(A, B, C, D), which has a key consisting of two attributes A and B, with three tuples. The key is represented by two associative arrays in bold in Figs. 5(b) and (c), and for the remaining two attributes we create two tensors. In Fig. 5(b) we represent the attribute C with $\boldsymbol{\mathcal{X}}_1(A : String, B : String) : Integer$ and in Fig. 5(c) we represent the attribute D with $\boldsymbol{\mathcal{X}}_2(A : String, B : String) : Integer$. As the domain of D is a string, we use another associative array as an indirection. If an attribute is a foreign key, its corresponding associative array is then re-used.
2. **Key-value:** Most of key-value stores save data as ordered $(key, value)$ pairs in a distributed hash table [8]. As typed tensors are described by associative arrays, there is a straight mapping between this type of NoSQL store and our Tensor Data Model.
3. **Column:** A column store system, like Vertica or Cassandra, uses a relational-like schema, so their mapping to Tensor Data Model is the same as for the relation.

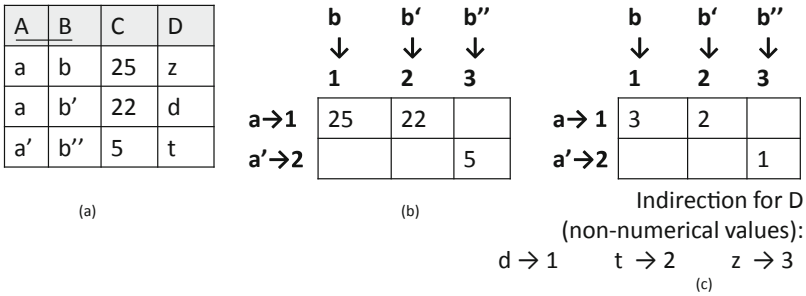


Fig. 5. An example of relational model - TDM mapping

4. **Graph:** A simple graph $G = (V, E)$, where V is the set of vertices and $E \subset V \times V$ is the set of edges, can be represented by its adjacency or incidence matrices, i.e., as a 2-order tensor. Matrices can also represent oriented, weighted graphs. A hypergraph is a (V, E) pair where V is a set of nodes and E is a set of hyperlinks. Each hyperlink $e \in E$ can contain multiple nodes, so E is defined as a subset of 2^V. A hypergraph can be represented by its incidence matrix A_{ie} with integer coefficients belonging to the set $\{0, +1, -1\}$ such that each column corresponds to a hyperlink e, and each row to a node i of H. If a hyperlink r arrives at the i node, then $A_{ie} = 1$; if a hyperlink r leaves the i node, then $A_{ie} = -1$; otherwise $A_{ie} = 0$. Hypergraphs can also be taken into consideration in a tensor model. Kivela et al. [47] show different mapping strategies for tensors. In the case of a multigraph, i.e., a graph with different types of links $E = \{E_1, E_2, \ldots, E_k\}$ where E is a partitioned set of links, the multigraph can be represented by a 3-order tensor where one dimension is used to specify different types of edges. In Fig. 6, we show an example of a multigraph with four nodes and three types of links (denoted by E_1, E_2, E_3), we obtain a 3-order tensor $\boldsymbol{\mathcal{X}}$ of size $4 \times 4 \times 3$. (i, j, k) is 1 if the i^{th} node is connected to the j^{th} node by the k^{th} type of link and each frontal slice $\boldsymbol{\mathcal{X}}_{::k}$ represents a type of link.
Recently, several research studies have proposed to improve and generalize the existing graph model approaches to take into account complex networks that include several subsystems and different levels of connectivity between systems. Although the terminology varies widely, depending on each specific case, such networks can generally be called multi-layer networks [47].
The construction of a higher-order tensor for multi-layer networks is studied in [20,47]. In the most general case, a $(2(M + 1))$-order tensor is required to model multi-layer networks without constraints. It considers edges between two vertices (2-order tensor) and adds all combinations of each vertex into the M different layers $2 \times M$-order, giving a $(2 + 2M)$-order tensor.
Graph databases handle in a different way theoretical models of graphs [6], most of them, except perhaps the nested-graph, can be generalized by a multi-layer graph and specified using tensors.

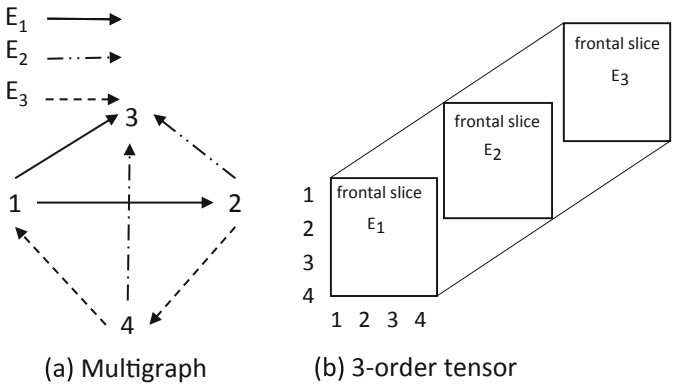


Fig. 6. An example of a mapping between a multigraph and TDM

All the above models are structurally equivalent to our TDM. The most appropriate storage system can be chosen based on the nature of the data and the cost models associated with the preferred operations. Moreover, in specific cases part of data can be duplicated. We have studied some real examples of such equivalences in [54].

5 TDM's Operators

In order to carry out a wide range of queries, it is desirable to define an analogy for several of the standard operators of the relational algebra in terms of tensor operations. In [39,40] the authors define a model and operators over associative arrays to unify relational, array, and key-value algebras. LaraDB and D4M models rely essentially on binary relations. The universality of binary relations makes it possible to represent almost all models. However, operations to reconstruct complex relationships are expensive and such models do not benefit from the potentiality of tensor decomposition operations.

Our operators are defined to provide programmers with views achieving logical data independence, i.e., to bridge the gap between the variety of analysis algorithms and storage systems. Our operators define a closed subset over tensors and works on typed tensors at two different levels: at the associative array level (i.e., the schema) and at the tensor value level. We focus on the following subset of operators on typed tensors: selection, projection, restriction, union, intersection, join, nest and some analytic operators such as aggregation and tensors decomposition. The definitions proposed in the following section are based on the formalism of [42] used to define the relational model. Our definitions are constructed as follows:

- clause (1) describes the operator's behavior on the schema, i.e., restrictions on operand schemas and specification of the result schema;
- clause (2) gives the operator's semantics on values.

We note that $\alpha(\boldsymbol{\mathcal{X}})$ refers to the schema of the typed tensor $\boldsymbol{\mathcal{X}}$.

5.1 Data Manipulation Operators

Projection operators are usual operators in tensor algebra and they produce tensors with specific names. A fiber of a tensor $\boldsymbol{\mathcal{X}}$ is a vector obtained by fixing all but one $\boldsymbol{\mathcal{X}}$'s indices: $\boldsymbol{\mathcal{X}}_{:jk}$, $\boldsymbol{\mathcal{X}}_{i:k}$ and $\boldsymbol{\mathcal{X}}_{ij:}$. Fibers are always assumed to be column vectors, similar to matrix rows and columns. A slice of a tensor $\boldsymbol{\mathcal{X}}$ is a matrix obtained by fixing all but two of $\boldsymbol{\mathcal{X}}$'s indices: $\boldsymbol{\mathcal{X}}_{i::}$, $\boldsymbol{\mathcal{X}}_{:j:}$ et $\boldsymbol{\mathcal{X}}_{::k}$. A project operator can be generalized by using the mode-n product $\times_n$ (mode-n product behavior is detailed in [50]).

Definition 4 (Project). *The projection of a N-order typed tensor $\boldsymbol{\mathcal{X}}$, noted as $\pi_{[expr]}\boldsymbol{\mathcal{X}}$, where expr is a pair (d, c), and where $d \in D$ and $c \in Dom_d$ which translates an equality between the name of an associative array (dimension) and a constant value, reduces the dimensions of the tensor by keeping the values over the others dimensions.*

(1) $\alpha(\pi_{[expr]}\boldsymbol{\mathcal{X}}) = \alpha(\boldsymbol{\mathcal{X}}) - \{d_k\}$ *where expr is* $(d_k, :)$
(2) $\pi_{[expr]}\boldsymbol{\mathcal{X}} = \{(x, v), x \in Dom_{d_1} \times \cdots \times Dom_{d_k|c} \times \cdots \times Dom_{d_N}, v \in Dom_{\boldsymbol{\mathcal{X}}}\}$[26]

The project operator can be computed by using the n-mode product with a boolean vector that contains 1 for the elements of the mode(s) to retain: $\boldsymbol{\mathcal{X}} \times_n \mathbf{b}$.

For example, consider a 3-order tensor, with the dimensions users, hashtags and time used to store the number of times a hashtag is used in tweets by a user per time slice (e.g., 1 h), the schema is $\boldsymbol{\mathcal{UHT}}(U : String, H : String, T : Integer) : Integer$. For notation purpose we assume that the names $\boldsymbol{\mathcal{UHT}}$ and $\boldsymbol{\mathcal{X}}_1$ are equivalent. The expression $\pi_{[U='u1']}\boldsymbol{\mathcal{X}}_1$ produces a 2-order tensor with the following schema $\boldsymbol{\mathcal{HT}}_1(H : String, T : Integer) : Integer$ and where tensor values represent the number of times each hashtag is used by a user $u1$, for all time slices (Fig. 7b). This number can be computed using $\boldsymbol{\mathcal{HT}}_1 = \boldsymbol{\mathcal{X}}_1 \times_1 \mathbf{b}$ with $b_i = 1, b_j = 0, \forall j \neq i$ (in our case $i = 1$ for the user $u1$, 1-mode product is used because U is the first dimension in the definition of the tensor).

A selection operator can be defined at two different levels: (1) on the values contained in the tensor or (2) on the values of the dimensions, i.e., typed associative arrays $\mathbf{A}_i$, $i = 1, .., N$.

Definition 5 (Select on tensor values). *The operator $\sigma_{[expr]}\boldsymbol{\mathcal{X}}$ selects values of the tensor that satisfy expr*[27] *and thus produces a new tensor with the same schema.*

[26] The notation | is the restriction applied to sets, $A|B = A - (A - B)$.

[27] *expr* is a logical expression to compare values of $\boldsymbol{\mathcal{X}}$ to constants. Its form is as follows: expr ::= <condition> | <condition> <logical operator> <condition> |¬ <condition> | (<condition>)
Logical operators are $\{\wedge, \vee\}$
<condition> ::= values of $\boldsymbol{\mathcal{X}}$ (implicit) <comparison operator> constant
Comparison operators are $\{<, \leq, =, \neq, \geq, >\}$.

(1) $\alpha(\sigma_{[expr]}\boldsymbol{\mathcal{X}}) = \alpha(\boldsymbol{\mathcal{X}})$
(2) $\sigma_{[expr]}\boldsymbol{\mathcal{X}} = \{(x, v), x \in Dom_{d_1} \times \cdots \times Dom_{d_N}, v \in Dom_{\boldsymbol{\mathcal{X}}} \text{ such as } expr(v)\}$
$\cup\{(x, z), x \in Dom_{d_1} \times \cdots \times Dom_{d_N},$
$z \in Dom_{null} \text{ such as } \exists(x, v), v \in Dom_{\boldsymbol{\mathcal{X}}}, \neg expr(v)\}$

For example, $\sigma_{[>10]}\boldsymbol{\mathcal{X}}_1$ selects users that have emitted the same hashtag during one time slice more than 10 times (Fig. 7(c)).

Remark 2 (Select and null values). When evaluating this operator, values not satisfying *expr* can be replaced by 0. If all the values associated with a key of a dimension are equal to 0, then the value of this key is removed from the corresponding typed associative array (represented in Fig. 7(c) by the blue dotted line). In the previous definition, Dom_{null} is reduced to a singleton, e.g., it may be 0 or another value interpreted as *null*.

Definition 6 (Restriction on dimensions values). *The operator $\rho_{[expr]}\boldsymbol{\mathcal{X}}$ restricts the tensor shape by selecting some values of the dimensions contained in the propositional formula expr*[28]*.*

(1) $\alpha(\rho_{[expr]}\boldsymbol{\mathcal{X}}) = \alpha(\boldsymbol{\mathcal{X}})$
(2) $\rho_{[expr]}\boldsymbol{\mathcal{X}} = \{(x, v), x \in Dom'_{d_1} \times \cdots \times Dom'_{d_N}, v \in Dom'_{\boldsymbol{\mathcal{X}}}$
$with\ Dom'_{d_i} = \{y, y \in Dom_{d_i},\ such\ as\ expr(y)\}\ for\ i = 1, \ldots, N$
$and\ Dom'_{\boldsymbol{\mathcal{X}}} \subseteq Dom_{\boldsymbol{\mathcal{X}}}\}$

The schema of a tensor is affected neither by the restriction operator nor by the schema of its typed associative arrays.

The example, in Fig. 7d selects for each hashtag used by the user $u1$ for time slices between 19-02-28 and 19-03-08, from the tensor $\boldsymbol{\mathcal{X}}_1$, the number of hahstag uses:

$$\rho_{[U=\text{'u1'} \wedge T \geq \text{'19-02-28'} \wedge T \leq \text{'19-03-08'}]}\boldsymbol{\mathcal{X}}_1$$

Definition 7 (Union). *The union of two typed tensors $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$ having the same schema, denoted $\cup_\theta$, where $\theta \in \{+, -, \times, \div, max, min\}$ is a typed tensor $\boldsymbol{\mathcal{X}}_3$ with the same schema and $Dom_{d_i}^{\boldsymbol{\mathcal{X}}_3} = Dom_{d_i}^{\boldsymbol{\mathcal{X}}_1} \cup Dom_{d_i}^{\boldsymbol{\mathcal{X}}_2}$, for $i = 1, \ldots, N$. Values of $\boldsymbol{\mathcal{X}}_3$ are values from $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$ except for keys in common for which the operator θ is applied.*

(1) $\alpha(\boldsymbol{\mathcal{X}}_1 \cup_\theta \boldsymbol{\mathcal{X}}_2) = \alpha(\boldsymbol{\mathcal{X}}_1)$ *and* $\alpha(\boldsymbol{\mathcal{X}}_1 \cup_\theta \boldsymbol{\mathcal{X}}_2) = \alpha(\boldsymbol{\mathcal{X}}_2)$

[28] *expr* allows to compare keys of the dimensions with constants. Its shape is the same as for the operator σ except for
<condition> ::= name of a dimension <comparison operator> constant.

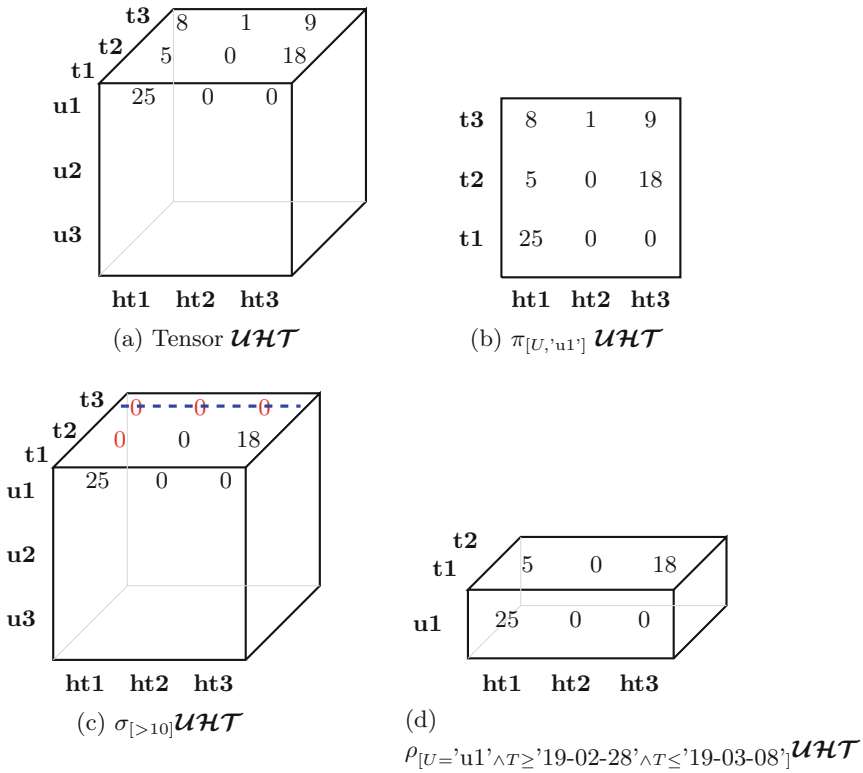


Fig. 7. Project, select and restriction operators applied to the tensor $\mathcal{UHT}$ for the values of **u1** (other values are not shown in the example) (Color figure online)

$$(2)\ \mathcal{X}_1 \cup_\theta \mathcal{X}_2 = \{(x, v), x \in Dom'_{d_1} \times \cdots \times Dom'_{d_N}$$

$$with\ Dom'_{d_i} = Dom^{\mathcal{X}_1}_{d_i} \cup Dom^{\mathcal{X}_2}_{d_i}\ for\ i = 1, \ldots, N$$

$$v = \begin{cases} v_1, & if\ \exists (x, v_1), x \in Dom^{\mathcal{X}_1}_{d_1} \times \cdots \times Dom^{\mathcal{X}_1}_{d_N}, \\ & x \notin Dom^{\mathcal{X}_2}_{d_1} \times \cdots \times Dom^{\mathcal{X}_2}_{d_N}, v_1 \in Dom_{\mathcal{X}_1} \\ v_2, & if\ \exists (x, v_2), x \in Dom^{\mathcal{X}_2}_{d_1} \times \cdots \times Dom^{\mathcal{X}_2}_{d_N}, \\ & x \notin Dom^{\mathcal{X}_1}_{d_1} \times \cdots \times Dom^{\mathcal{X}_1}_{d_N}, v_2 \in Dom_{\mathcal{X}_2} \\ v_1 \theta v_2, & if\ \exists (x, v_1), x \in Dom^{\mathcal{X}_1}_{d_1} \times \cdots \times Dom^{\mathcal{X}_1}_{d_N}, v_1 \in Dom_{\mathcal{X}_1}, \\ & \exists (x, v_2), x \in Dom^{\mathcal{X}_2}_{d_1} \times \cdots \times Dom^{\mathcal{X}_2}_{d_N}, v_2 \in Dom_{\mathcal{X}_2} \} \end{cases}$$

Figure 8 shows the operator $\cup_+$ applied to two typed tensors representing the number of hashtags used by Twitter and Instagram users. When users exist in both social networks (i.e., having the same key in the corresponding associative arrays), the operator + is applied, otherwise the values from either tensor can be retained.

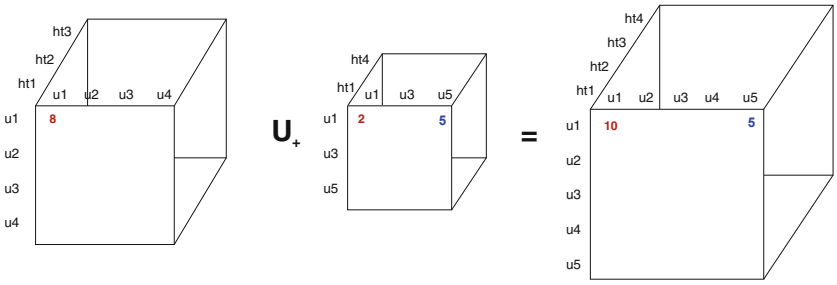


Fig. 8. ∪ operator applied on two tensors

Definition 8 (Intersection). *The intersection of two typed tensors* $\boldsymbol{\mathcal{X}}_1$ *and* $\boldsymbol{\mathcal{X}}_2$ *having the same schema, denoted* $\cap_\theta$*, where* $\theta \in \{+, -, \times, \div, max, min\}$ *is a typed tensor* $\boldsymbol{\mathcal{X}}_3$ *with the same schema and whose dimension key sets are intersection of the operand dimension key sets:* $Dom_{d_i}^{\boldsymbol{\mathcal{X}}_3} = Dom_{d_i}^{\boldsymbol{\mathcal{X}}_1} \cap Dom_{d_i}^{\boldsymbol{\mathcal{X}}_2}$*, for* $i = 1, \ldots, N$*. Values of* $\boldsymbol{\mathcal{X}}_3$ *are values associated to common keys of each dimension on which the operator* θ *is applied.*

(1) $\alpha(\boldsymbol{\mathcal{X}}_1 \cap_\theta \boldsymbol{\mathcal{X}}_2) = \alpha(\boldsymbol{\mathcal{X}}_1)$ *and* $\alpha(\boldsymbol{\mathcal{X}}_1 \cap_\theta \boldsymbol{\mathcal{X}}_2) = \alpha(\boldsymbol{\mathcal{X}}_2)$
(2) $\boldsymbol{\mathcal{X}}_1 \cap_\theta \boldsymbol{\mathcal{X}}_2 = \{(x, v), x \in Dom'_{d_1} \times \cdots \times Dom'_{d_N}$

$$\text{with } Dom'_{d_i} = Dom_{d_i}^{\boldsymbol{\mathcal{X}}_1} \cap Dom_{d_i}^{\boldsymbol{\mathcal{X}}_2} \text{ for } i = 1, \ldots, N, \text{ and}$$
$$v = v_1 \theta v_2, \exists (x, v_1), x \in Dom_{d_1}^{\boldsymbol{\mathcal{X}}_1} \times \cdots \times Dom_{d_N}^{\boldsymbol{\mathcal{X}}_1}, \text{ and}$$
$$\exists (x, v_2), x \in Dom_{d_1}^{\boldsymbol{\mathcal{X}}_2} \times \cdots \times Dom_{d_N}^{\boldsymbol{\mathcal{X}}_2}\}$$

Figure 9 shows a use of the operator $\cap_+$ applied to the two previous typed tensors. It represents the number of hashtags shared between Twitter and Instagram users. We thus only obtain the existing users in both social networks and the operator + is applied for these users.

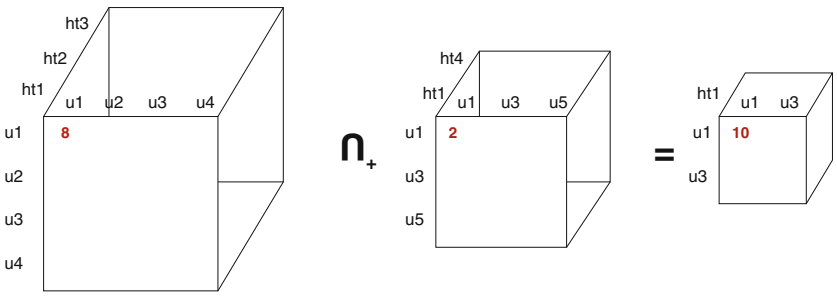


Fig. 9. ∩ operator applied on two tensors

Definition 9 (Natural-Join). *The natural-join of two typed tensors* $\mathcal{X}_1$ *and* $\mathcal{X}_2$ *having at least one common dimension (i.e., at least two associative arrays with the same type), denoted* $\bowtie$*, is a typed tensor* $\mathcal{X}_3$ *whose schema is the union of the two tensor schemata. The keys retained in the common dimensions are those which are equal, values of* $\mathcal{X}_3$ *are those of the first operand.*

(1) $\alpha(\mathcal{X}_1 \bowtie \mathcal{X}_2) = \alpha(\mathcal{X}_1) \cup \alpha(\mathcal{X}_2)$
(2) $\mathcal{X}_1 \bowtie \mathcal{X}_2 = \{(x, v), x \in \prod_{d \in DD} Dom_d,$

$$DD = \{Dom_{d1}^{\mathcal{X}_1}, d1 \in D^{\mathcal{X}_1}\} \cup \{Dom_{d2}^{\mathcal{X}_2}, d2 \in D^{\mathcal{X}_2}\}$$
$$v \in Dom_{\mathcal{X}_1}, \exists (y, z, v) \text{ such as } x = (y, z)$$
$$\text{and } (y, v) \in \left(\prod_{d \in D^{\mathcal{X}_1}} Dom_d\right) \times Dom_{\mathcal{X}_1}$$
$$\text{and } z \in \prod_{d \in D^{\mathcal{X}_2} - D^{\mathcal{X}_1}} Dom_d\}$$

For example, consider following two tensors $\mathcal{X}_1$ and $\mathcal{X}_4$. $\mathcal{X}_4$ dimensions are hashtag, category and percentage that specifies for each hashtag its relative part in its category or categories. The expression $\mathcal{X}_1 \bowtie \mathcal{X}_4$ produces a tensor with dimensions user, hashtag, time, category, the values are the number of uses of hashtags per user, per time slice, and per category. The element-wise product of $\mathcal{X}_1 \bowtie \mathcal{X}_4$ by $\mathcal{X}_4 \bowtie \mathcal{X}_1$ gives a tensor with weighted values (Fig. 10).

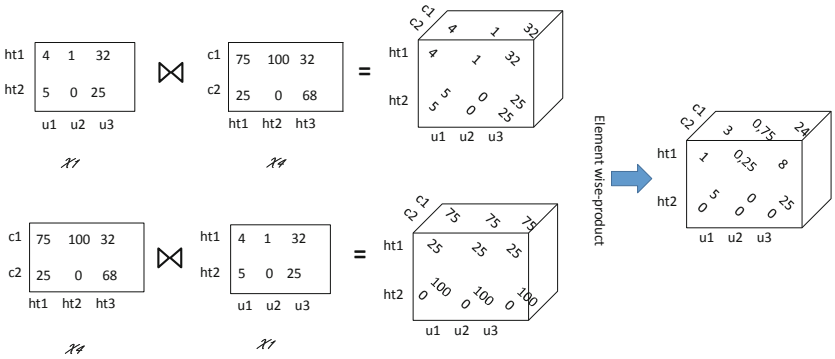


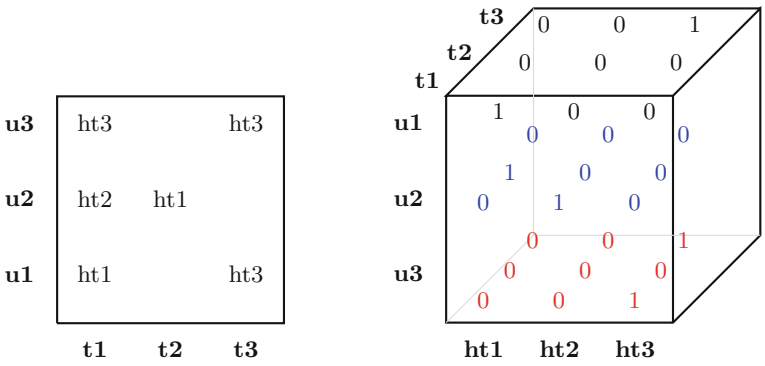
Fig. 10. $\bowtie$ operator applied on two tensors

The nest operator allows to transform values into dimensions. Thus, the nest operator increases the number of dimensions and produces a boolean tensor.

Definition 10 (Nest). *The nest operator applied to a N-order typed tensor* $\mathcal{X}$ *creates a new* $N+1$*-order typed tensor* $\mathcal{X}_1$ *by appending to the dimensions of* $\mathcal{X}$ *an additional dimension* d' *(*$D' = D \cup \{d'\}$*). The keys of the associative array* d' *is the list of unique values in the tensor* $\mathcal{X}$*, where values of* $\mathcal{X}_1$ *are 1 for the dimensions corresponding to those* $\mathcal{X}$ *and* d'*, otherwise 0.*

Boolean tensors are a specific class which can be used: (1) to represent complex relationship among data by focusing on the existence of the relationship rather than on a measure of its intensity and (2) to perform joins between other tensors, they can be compared to association tables in a relational schema.

Figure 11 shows an example of the *nest* operator applied to a 2-order tensor whose values are strings and produces a 3-order boolean tensor.



(a) $\mathcal{UT}$ (b) $\mathcal{UHT} = nest(\mathcal{UT})$

Fig. 11. *nest* operator applied to 2-order tensor

5.2 Analytical Operators

"Group by like" operations [48,66] can be defined on typed tensors applying aggregation function with selection of tensor dimension values to aggregate.

Definition 11 (Aggregation). *The aggregation on a typed tensor* $\mathcal{X}$ *is denoted* $\mathcal{F}_{op(d)}(Dim)$*, where* $op \in \{SUM, AVG, COUNT, MIN, MAX\}$ *and* d *is the name of an associative array in* D*,* Dim *is a list of names of associative arrays in* D*. It produces a typed tensor with a schema specified by* Dim*. The values are given by the application of the operator op on the specified dimension* d *for values of keys in* Dim*.*

Note that for aggregation operator, all the dimensions must be present either in d or in Dim.

For example, this operator is useful for transforming a time series represented by a 3-order tensor (i.e., with the dimensions user, hashtag, time) to a 2-order tensor with user and hashtag dimensions representing the number of times each hashtag was used by a user: $\mathcal{F}_{SUM(T)}(U, H)\mathcal{X}_1$

Tensor decompositions such as CANDECOMP/PARAFAC (CP), Tucker, HOSVD can be used to perform dimensionality reductions and/or to extract latent relationships between dimensions [50]. Since tensor representations of data may be multiple and their semantics are not explicit, the results of tensor decompositions can be complex to interpret. In [69] the authors give guidelines for each

type of decomposition. Researches on tensor decomposition constitute an important field in applied mathematics. These researches focus on different problems such as robustness of numerical methods, scalability and parallel computation, as well as value reconstruction [9,12,43]. In order to be used in a real high performance analysis platform, cost models of the manipulation and analysis tensors operators need to be defined and studied carefully with respect to sparsity. The tensor decompositions in TDM rely on the underlying libraries such as Spark or R.

6 Use Cases and Experiments

We validate our approach by a *proof-of-concept*, showing how TDM is used in a polystore architecture with real data from a multi-disciplinary project (TEP 2017) involving collaborations with social scientists and communication science researchers. The main research objective of TEP 2017 is to study the structure and the dynamics of political discourse on Twitter during the French Presidential Election in March-May 2017.

6.1 Polystore and TDM Views

The architecture setup to carry out our experiments includes a polystore built on the top of PostgreSQL, HDFS, and Neo4j connected to three analysis frameworks, namely R, Apache Spark and TensorFlow. The data set is a tweets corpus harvested during the period from 2017-02-27 to 2017-05-18. The data set contains 49 million tweets emitted by more than 1.8 million users which used 288,221 different hashtags. Of the 49 million tweets, 36 million are retweets. Raw data were stored in the JSON file format in HDFS (720Go), most important attributes (around fifty) of tweets were stored in a relational database (PostgreSQL) in a unique table (50Go), and then in another database with a normalized schema (55Go). The links between main entities (i.e., tweets, users, hashtags) were stored in Neo4j (23Go).

We have defined the following views using TDM operators:

- a view for selecting Twitter accounts that have been retweeted and their number of retweet during the period. This 1-order tensor has the schema $\boldsymbol{\mathcal{NRT}}(U : String) : Integer$ corresponding to the following Cypher query to retrieve data from Neo4j:

```
MATCH (u:USER)-[rt:RT]->(t:TWEET)<-[p:Publish]-(u1:USER)
WHERE t.date>"2017-02-27" AND t.date<="2017-05-18"
RETURN u1.name, COUNT(RT) AS NRT
```

- a view giving the number of retweets between users, this 2-order tensor as the schema $\boldsymbol{\mathcal{RT}}(U : String, U_1 : String) : Integer$ corresponding to the following Cypher query:

```
MATCH (u:USER)-[rt:RT]->(t:TWEET)<-[p:Publish]-(u1:USER)
WHERE t.date>"2017-02-27" AND t.date<="2017-05-18"
RETURN u.name, u1.name, COUNT(RT) AS RT
```

– a view ($\mathcal{UHT}$) for selecting the number of times each user emitted an hashtag per time slice (e.g., 1 h). This view is obtained from the PostgreSQL normalized schema and has the schema $\mathcal{UHT}(U : String, H : String, T : Integer) : Integer$ corresponding to the following simplified SQL query (where 414717 define the beginning of the period) :

```
SELECT U.from_user, H.hashtag, floor(time/3600)-414717 T,
count(*) nbocc FROM UHTTP2, User U, Hashtag H
WHERE UHTTP2.user_id=U.user_id AND UHTTP2.ht_id=H.ht_id
GROUP BY U.from_user, H.hashtag, floor(time/(3600)) - 414717;
```

– a view named *popular accounts* ($\mathcal{PA}$) for selecting Twitter accounts that have been retweeted at least n times: $\mathcal{PA} = \sigma_{[>n]}\mathcal{NRT}$. We note that according to Definition 5, users with a number of retweets less than n do not appear in the results.
– a view ($\mathcal{AU}$) for selecting active users (i.e., which have emitted more than r retweets) which also retweet popular accounts: $\mathcal{AU} = \sigma_{[>r]}(\mathcal{RT} \bowtie \mathcal{PA})$

6.2 Study Viral Tweets and Role of Bots

In order to study the possible influence of robots on the circulation of viral tweets, we sought to detect robots among Twitter accounts that had retweeted at least 1,000 times over the period between the two rounds of this election (from 23^{rd} April until 7^{th} May 2017), thus reducing the corpus of 49M of tweets to one thousand. In order to reduce the number of accounts to analyze, only accounts having tweeted more hundred times in the two-week period are retained. 1,077 accounts were selected in this way. This corresponds to the hypothesis that robots are tweeting intensively during this final period of the election campaign. The second hypothesis is that robots do not tweet at random so we retrieve hashtags contained in these tweets.

Using $\mathcal{UHT}$ and $\mathcal{AU}$ we built a 3-order tensor for these accounts during the two-week period. We obtained a tensor containing potentially $1,077 \times 568 \times 336$ items. In the first approach the tensor construction from the relational data set of 49M of tweets shows that the time required for production of tensors data lies between 1 and 2 min for tensor size of 205M values. One explanation may come from the sparsity of data, and we give details on performance question in the next section.

We performed a CP decomposition in order to reduce the user space based on the user behavior; this produces n groups of three 1-order tensors, here vectors $\mathbf{a}$, $\mathbf{h}$, $\mathbf{t}$. We then applied the k-means clustering algorithm to identify groups of users having similar behavior during the period. The k-means algorithm determines four groups of users: a group of one account previously detected as a robot and suspended by Twitter, a group of three accounts, a group of about thirty accounts

and the last group containing other users. The first two groups are shown in the far left column with label from_user in Fig. 13; for confidentiality reasons we hide other accounts. The group of three accounts, revealed after manual study, are to be merged (same behavior and hashtags) and they are probably assisted by an algorithm that retweets messages against the Macron candidate. Each of these accounts have tweeted between 1,000 and 1,800 times during the period with a vast majority of retweets.

In order to obtain a deeper insight on these data, we also used the community discovery Louvain algorithm [13] to detect accounts which retweet or are retweeted frequently by the other accounts and which tend to share the same retweets. The retweet graph represented by its adjacency matrix is obtained from a 3-order tensor $\boldsymbol{\mathcal{UUT}}(U : String, U : String, T : Integer) : Integer$ by using the aggregation operator: $\mathcal{F}_{SUM(T)}(U, U)\boldsymbol{\mathcal{UUT}}$. The result of the CP decomposition is confirmed (Fig. 12): yellow community corresponds to the user detected as a robot, pink community corresponds to the group of three users, and the third group are users of the blue and green communities. The biggest nodes are accounts from the four clusters obtained by the CP decomposition and the k-means algorithm.

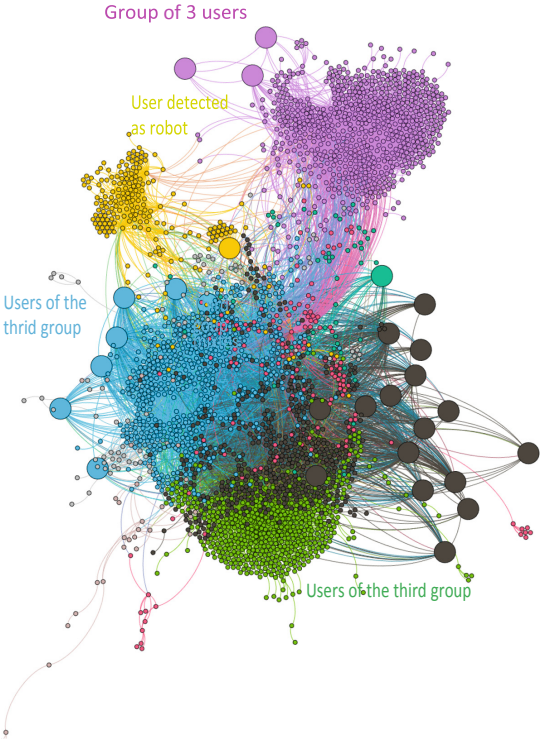


Fig. 12. Communities obtained from retweet graph using Louvain algorithm (Color figure online)

Our result is consistent with the use of OSoMe API Botometer[29] which provides an overall probability score that an account studied is automated. It uses 1,150 criteria and machine learning techniques to calculate the probability [81]. The group of about thirty accounts comprises more than half users with a probability of being a robot greater than 0.42 (sixth column with label proba in Fig. 13). But we note that the values of probabilities were not sufficiently significant to detect robots during the studied period. One of the assumptions was that there were hybrid accounts of users assisted by algorithms. However, simple criteria such as the maximum number of tweets published in one hour make it possible to unambiguously find some of the accounts with automated behavior, and later confirmed by the manual study.

from_user	user_id	nbrt	nbrtjour	nbtweetp2	proba	retrycount	maxrateh	htnb	cluster
Manuel_Hollande	639	1073	76	1078	0.66	1	113	155	1
Macron_Jamais	622	1481	105	1706	0.61	1	88	242	1
Degage_Hollande	277	1394	99	1464	0.66	0	118	189	1
	724	1594	113	5688	0.5	0	188	276	3
	933	3405	243	3727	0.31	1	149	512	4
	477	2037	145	2370	0.39	0	96	437	4
	761	2254	161	2263	0.3	1	93	381	4
	1071	2289	163	2538	0.39	2	153	387	4
	197	1988	142	2415	0.37	0	75	274	4
	932	1967	140	2041	0.35	0	90	372	4
	158	1932	138	1951	0.38	0	149	308	4
	1070	1814	129	1820	0.54	0	174	177	4
	979	1472	105	2396		9	133	236	4
	712	1365	97	1428	0.48	0	131	231	4
	968	4513	322	4558	0.39	1	214	590	4
	248	6466	461	6672	0.5	0	141	869	4
	345	6134	438	6549	0.34	0	242	911	4
	446	5228	373	5336	0.34	0	120	722	4
	112	1291	92	1395	0.69	0	95	125	4
	834	7001	500	7105		8	106	1090	4
	877	1223	87	1243	0.42	4	114	241	4
	324	1108	79	1111	0.4	0	147	108	4
	347	3525	251	3908	0.49	0	79	136	4
	896	1509	107	1509	0.62	0	201	212	4
	191	3337	238	3361	0.27	0	129	505	4
	172	2917	208	2954	0.3	0	140	459	4
	644	3431	245	3431	0.42	1	124	285	4
07_mai_2017	2	7233	516	7233		8	516	1822	6

(28 rows)

Fig. 13. Results given by Botometer (sixth column with label proba) and cluster number obtained by the CP decomposition (far right column with label cluster)

6.3 Study of Influence in Multi-relational Networks

Influence on Twitter is defined as the potential of a user's action to initiate a follow-up action by another user. The term "action" means various possible interactions between users. Hence, measuring influence on Twitter isn't that simple as the network provides several forms of interactions: *retweet*, *mention*, *reply*. In [3,70] the authors have produced state-of-the-art studies on the estimation of influence in social networks, including Twitter. Three main types of approaches have been identified: they are based on popularity measures, network topology and information fusion. Topology-based approaches include algorithms such as PageRank [68] for ranking most influential users.

[29] https://botometer.iuni.iu.edu/.

The main idea of PageRank is that "The most important pages (of websites) are likely to receive most links from other pages". PageRank assumes that the importance of a web page is determined by the quantity and quality of the pages associated with it. Initially, each node (i.e., page) receives a PR value. Then, each node uniformly distributes its PR value to its neighbors through the outgoing links. To guarantee convergence, a teleportation factor has been introduced, assuming that the user navigates between web pages following the links with the probability s, and leaves the current page for a random page with a probability of $1 - s$, $s \in [0.1]$. The probability s is usually empirically fixed around 0.85, the PR formula is as follows:

$$PR_i(t) = s \sum_{j=1}^{n} a_{ji} \frac{PR_j(t-1)}{k_j^{out}} + (1-s)\frac{1}{n}$$

where n is the total number of nodes in the network, $\mathbf{A}$ is the adjacency matrix such that $a_{ji} = 1$ if j is connected to i, otherwise $a_{ji} = 0$, and where k_j^{out} (out-degree of j) represents the number of edges outgoing from j. The above iteration stops when the PR values of all nodes reach a steady state.

We wish to find out the influence of the eleven candidates in the first round of the French presidential election in 2017 (candidates are on the x-axis in Fig. 14). To do this, we considered the candidates and users who interacted with the candidates according to the three relations *Retweet*, *Mention* and *Reply*. We obtained 320,803 accounts; 2,708,751 retweets; 29,652 mentions and 349,663 replies. Then, we applied the PageRank algorithm to the three graphs, PageRank scores are on y-axis and three relations are represented by a color, the results being presented in Fig. 14. These graphs are represented by their adjacency matrix that can be obtained from a 3-order tensor $\boldsymbol{\mathcal{UUR}}(U : String, U : String, R : String) : Integer$ by selecting each relation corresponding to keys of R independently. The PageRank scores obtained by the candidates reflect the interest they elicited during the campaign: "small" candidates have low scores compared to the candidates supported by major parties or compared to the novelty carried by the Macron candidacy.

7 Performance Evaluation

In this section, we study two aspects of TDM's performance: (1) in terms of space complexity for tensor data structure, time complexity for operators including a comparison with the relational model, and using hashtable structures (Subsect. 7.1); (2) with real data using existing algorithms implemented in different programming paradigms (Subsect. 7.2).

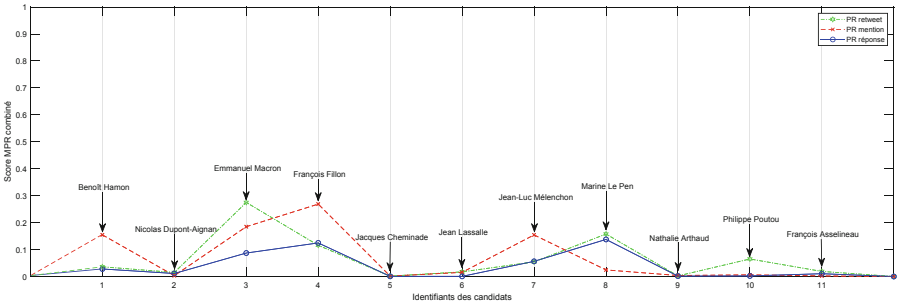


Fig. 14. Ranking of French presidential candidates by using PageRank (Color figure online)

7.1 Theoretical Evaluation

The storage of tensors is highly correlated with the data modeling choices such as array, matrix or graph. As we work with the closed world assumption, i.e., only what is known to be true is stored, we often obtain sparse data. For example, a graph is considered to be sparse if $nnz = O(n)$, where nnz is the number of edges and n is the number of vertices, for an array or a matrix nnz is the number of non-zero values.

It is often more convenient to represent tensors as matrices. The unfolding of a tensor performs a reordering of elements of a N-order tensor into a matrix [19]. Compressed Sparse Column (CSC) or Compressed Sparse Row (CSR) [17] are common data structures used to store sparse matrices and to perform linear algebra operations on sparse matrices. The total space complexity of CSC is $O(n + nnz)$. CSC allows fast access to columns of a matrix, yet it is very slow when accessing rows. Storing CSR along with CSC can overcome this problem, but it is rarely used as it doubles the storage size. These structures have been extended to represent sparse tensors with the Compressed Sparse Fiber (CSF) format [75]. Nevertheless, all of these structures promote only a few operators such as multiplication and thus are not suitable for supporting TDM operators.

We assume that tensor values are stored as tuples. We note by $T_{\boldsymbol{\mathcal{X}}}$ the representation of a N-order tensor $\boldsymbol{\mathcal{X}}$ as tuples with $N+1$ attributes and nnz values. As tensors are in most cases sparse, tuples store only non-null values. The schema of $T_{\boldsymbol{\mathcal{X}}}$ is the following $T_{\boldsymbol{\mathcal{X}}}(d_1, \ldots, d_N, \boldsymbol{\mathcal{X}})$ where $d_i \in D$ for $i = 1, \ldots, N$. Table 3 gives an example of $T_{\mathcal{UHT}}$ (Fig. 7a). For sparse tensor a tuple representation has a space complexity of $O(nnzN + nnz)$, which is the same as for a relation with $N + 1$ attributes.

In order to study time complexity, we will identify the asymptotic upper bound of TDM operators using Relational Algebra operators as a reference with the assumption of a tuple representation. In order to specify the behavior of TDM operators in more details, we study another representation assumption using associative arrays implemented as hash tables and linked lists [49]. Thus, for each operator we provide two descriptions, the first one is the comparison

Table 3. Tensor $\mathcal{UHT}$ stored as tuples

U	H	T	$\mathcal{UHT}$
u1	ht1	t1	25
u1	ht1	t2	5
u1	ht1	t3	8
u1	ht2	t1	0
u1	ht2	t2	0
u1	ht2	t3	1
u1	ht3	t1	0
u1	ht3	t2	18
u1	ht3	t3	9

with Relational Algebra expressions, while the other one is the cost of operators with hash tables.

For a representation based on hash tables, we assume that the hash function is uniform. The density α and the estimation of the number of elements for dimension μ_{d_i} (e.g., a slice for a 3-order tensor) are given by:

$$\alpha = \frac{nnz}{|Dom_{d_1}| \times \cdots \times |Dom_{d_N}|} \text{ and } \mu_{d_i} = \frac{nnz}{|Dom_{d_i}|}$$

Table 4 gives for each of operators of TDM their complexity in terms of access.

Table 4. Complexity of main TDM's operators

Operator	Relational algebra expression	Complexity
$\pi_{[expr]}\boldsymbol{\mathcal{X}}$	$\pi(\{d_1, \ldots, d_N, \boldsymbol{\mathcal{X}}\} - d_k)\sigma(d_k = c)T_{\boldsymbol{\mathcal{X}}}$	$O(1 + \mu_d)$
$\sigma_{[expr]}\boldsymbol{\mathcal{X}}$	$\sigma(expr)T_{\boldsymbol{\mathcal{X}}}$	$O(1 + nnz)$
$\rho_{[expr]}\boldsymbol{\mathcal{X}}$	$\sigma(expr)T_{\boldsymbol{\mathcal{X}}}$ Let $T_{\boldsymbol{\mathcal{X}}_3} := \sigma(d_i^{\boldsymbol{\mathcal{X}}_1} = d_i^{\boldsymbol{\mathcal{X}}_2})(T_{\boldsymbol{\mathcal{X}}_1} \times T_{\boldsymbol{\mathcal{X}}_2})$	$O(1 + nnz)$
$\boldsymbol{\mathcal{X}}_1 \cup_\theta \boldsymbol{\mathcal{X}}_2$	$((T_{\boldsymbol{\mathcal{X}}_1} \cup T_{\boldsymbol{\mathcal{X}}_2}) - \pi(d_i^{\boldsymbol{\mathcal{X}}_1}, \boldsymbol{\mathcal{X}}_3)T_{\boldsymbol{\mathcal{X}}_3})$ $\cup\ \pi(d_i^{\boldsymbol{\mathcal{X}}_1}, \boldsymbol{\mathcal{X}}_1\theta\boldsymbol{\mathcal{X}}_2)T_{\boldsymbol{\mathcal{X}}_3}$	$O(2 + nnz(\boldsymbol{\mathcal{X}}_1) + nnz(\boldsymbol{\mathcal{X}}_2))$
$\boldsymbol{\mathcal{X}}_1 \cap_\theta \boldsymbol{\mathcal{X}}_2$	$\pi(d_i^{\boldsymbol{\mathcal{X}}_1}, \boldsymbol{\mathcal{X}}_1\theta\boldsymbol{\mathcal{X}}_2)T_{\boldsymbol{\mathcal{X}}_3}$	$O(2 + nnz(\boldsymbol{\mathcal{X}}_1) + nnz(\boldsymbol{\mathcal{X}}_2))$
$\boldsymbol{\mathcal{X}}_1 \ltimes \boldsymbol{\mathcal{X}}_2$	$\pi(\bigcup_{\substack{j=1,2 \\ i=1,\ldots N_{\boldsymbol{\mathcal{X}}_j}}} d_i^{\boldsymbol{\mathcal{X}}_j}, \boldsymbol{\mathcal{X}}_1)T_{\boldsymbol{\mathcal{X}}_3}$	$O(\sum_{d \in Dom_{\boldsymbol{\mathcal{X}}_2}} \lvert Dom_d \rvert + nnz(\boldsymbol{\mathcal{X}}_1))$

In conclusion, for the hypothesis of a representation with tuples, the operators select, project, restriction and natural join have all their costs of the same order as those of the relational algebra. For union and intersection, applying the θ operation on the common elements induces a costly Cartesian Product. For representation with hash tables and linked lists, union and intersection operations have a moderate cost.

7.2 Empirical Evaluation

In order to evaluate the performance of the TDM in real analytics use cases, two experiments were carried out. The first experiment uses a 3-order tensor to detect potential robots among active Twitter users, and the second experiment executes the Multiplex PageRank [32], again on a 3-order tensor to measure the influence of individual Twitter users. For each of these experiments, we compare the construction times of tensors and execution times of algorithms in the unoptimized case, as well as in our optimized TDM environment. We also assume that intermediate tensors and results are stored in memory. We perform experiments on a single type of storage system, i.e., PostgreSQL.

The goal of the first experiment is to compare the performance of TDM operators against SQL transformations exclusively executed within the database. Selection, natural join and aggregation TDM operators are implemented in R. In the robot detection experiment, we seek to build a tensor $\boldsymbol{\mathcal{UHT}}(U : String, H : String, T : Integer) : Integer$ that represents the number of emissions of hashtags by users each hour. As we are interested in users who could potentially be robots, we retrieve values for users who have retweeted at least 100 times suspected viral tweets (namely tweets that have been retweeted at least 50 times).

With the SQL transformations we build successively: (1) a table containing suspected viral tweets; (2) a table containing users who have retweeted at least 100 times the potentially viral tweets; and (3) a table containing the values of the $\boldsymbol{\mathcal{UHT}}$ tensor for the users obtained at Step 2.

With TDM operators in R, we perform two tasks at the storage system level and four tasks at the TDM level: (1) retrieve the potential viral tweets; (2) retrieve all the tweets which are retweets, including the user who emitted them, (3) perform a join in R between the values of the previous two steps, in order to retrieve users who have retweeted at least one of the viral tweets; (4) perform a selection in order to retain only users who have retweeted at least 100 times the viral tweets; (5) retrieve $\boldsymbol{\mathcal{UHT}}$ values for all users; and (6) perform a join between users from Step 4 and $\boldsymbol{\mathcal{UHT}}$ to retain values only for those users we are interested in.

The time needed to build the final tensor is shown in Fig. 15. The number of tweets retained in Step 1 is the parameter being varied. We progress from 2,000 up to 22,000 tweets, in increments of 2,000. With TDM operators implemented using R, it takes less time to retrieve the data in comparison to in-database SQL operations, yet more data are obtained directly from the database. Thus, TDM operators implementation is more efficient for tensor sizes exceeding 300B values. A coarse grained parallel version of the tensor construction has also been tested for exploiting the independence of certain of the above six steps.

The goal of second experiment is to evaluate the overhead induced by the TDM abstraction layer. For this experiment we do not need complex data transformations. As for the PageRank experiment in the previous section, we build a 3-order tensor $\boldsymbol{\mathcal{UUI}}(U : String, U : String, I : String) : Integer$. The first two dimensions correspond to a matrix representing users, and the third dimension represents the interaction types: *retweet*, *mention*, *reply* (Fig. 6b). Three methods

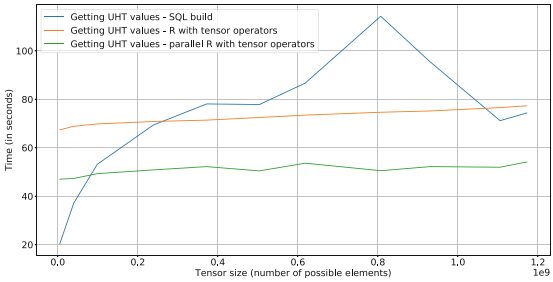


Fig. 15. Time needed to obtain values for tensor $\mathcal{UHT}$ with relational database tables and with TDM operators in R

are compared: (1) the naive one, in which CSV files are produced using queries on the relational database and then used to fill the tensor; (2) the Scala TDM wrapper, which directly connects to the database to execute queries needed for the three interactions layers; and (3) the Scala TDM parallel wrapper, where queries for retrieving values of different interactions are sent in parallel. Once the tensor is built, the Multiplex PageRank algorithm is performed to measure influence of individual users. We tried out three different implementations of the algorithm. The reference algorithm is written in Matlab[30] and takes advantage of optimized and multi-threaded libraries. The second implementation is a rewrite of the reference algorithm in Scala[31] using Breeze library[32] (linear algebra for Scala), while the third implementation is a parallel version of the second implementation.

In order to evaluate the performances with the Multiplex Page Rank, we build our tensor with a number of users varying from 10,000 to 300,000 in increments of 10,000. We tried out the three implementations, and the results obtained are shown in Fig. 16. The wrapper based extraction or CSV extraction takes almost the same time, yet the use of a wrapper simplifies the workflow. The parallel wrapper outperforms the other two methods and it seems to be very effective in building tensors for analysis of multi-layer networks. The execution efficiency of the Multiplex PageRank in Scala is also comparable to that of the Matlab execution, especially for the multi-threaded version.

In concluding we note that these experiments show that the TDM unification layer does not induce noticeable overhead in the data abstraction layer and can profit from an abstract representation that can also help parallelization.

[30] https://github.com/ginestrab/Multiplex-PageRank.
[31] https://github.com/AnnabelleGillet/Multiplex-PageRank.
[32] https://github.com/scalanlp/breeze.

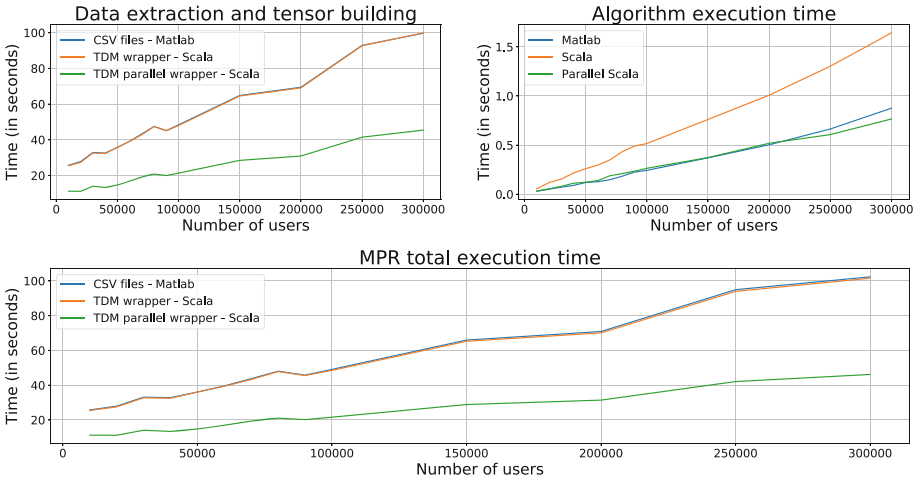


Fig. 16. Multiplex PageRank applied with TDM compared to a reference implementation using Matlab and CSV files

8 Conclusion

In this article, we described a Tensor based Data Model for polystore systems to achieve logical data independence and to solve data impedance mismatch problems in big data analytics architectures. We studied its ability to generalize major types of data models, such as relational, column, key-value, graph (including multi-layer networks). Using associative arrays we added the notion of schema to the tensor mathematical object. We defined a set of manipulation and analytics operators.

We described our experiments on Twitter data set from the French presidential election carried out to evaluate the ease of use of the operator toolkit as well as to study their performance. We detected the possible impact of robots on the circulation of viral tweets and evaluated the influence of candidates. Our results have been validated by researchers in the communication science. The experiments demonstrated the TDM capabilities by showing the ease of data transformations in our analyses.

Our upcoming work will concentrate on the storage of tensors as materialized views in SciDB through a matricization process [46]. As there are multiple ways of performing matricization, one particular way must be chosen based on the privileged operators and it may be necessary to specify normal forms to guide matricization. We are working on an implementation of each operator in Apache Spark and R using partition hashing. We did not consider the tensor algebra operators as a foundation for a user query language; instead we are implementing them, in a functional programming point of view, as Scala operators associated these objects. We also plan to study the behavior of operators in relation to each other in order to integrate them into a query optimizer such as Spark Catalyst.

We are selecting use case queries to carry out a performance evaluation study for operators and storage systems in order to provide users with needed guidelines.

On a more theoretical side, we need to add additional operators to the tensor algebra to manipulate dimensions and to allow renaming. We also need to study the expressive power of the set of operators including operators for manipulation of values and structures, as the first order logic is not being sufficient.

Acknowledgement. This research was partially supported by the project I-SITE UBFC COCKTAIL. We thank George Becker for comments that have greatly improved the manuscript and Arnaud Da Costa for the maintenance of the server infrastructure.

References

1. Abo Khamis, M., Ngo, H.Q., Nguyen, X., Olteanu, D., Schleich, M.: In-database learning with sparse tensors. In: ACM SIGMOD/PODS Symposium on Principles of Database Systems, pp. 325–340 (2018)
2. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Rasin, A.: HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc. VLDB Endowment **2**(1), 922–933 (2009)
3. Al-Garadi, M.A., et al.: Analysis of online social network connections for identification of influential users: survey and open research issues. ACM Comput. Surv. (CSUR) **51**(1), 1–37 (2018)
4. Allen, D., Hodler, A.: Weave together graph and relational data in apache spark. In: Spark+AI Summit. Neo4j (2018). https://vimeo.com/274433801
5. Alsubaiee, S., et al.: AsterixDB: a scalable, open source BDMS. Proc. VLDB Endow. **7**(14), 1905–1916 (2014)
6. Angles, R.: A comparison of current graph database models. In: IEEE International Conference on Data Engineering Workshops (ICDEW), pp. 171–177 (2012)
7. Astrahan, M.M., et al.: System R: relational approach to database management. ACM Trans. Database Syst. (TODS) **1**(2), 97–137 (1976)
8. Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M.: Workload analysis of a large-scale key-value store. ACM SIGMETRICS Perform. Evaluation Rev. **40**, 53–64 (2012)
9. Austin, W., Ballard, G., Kolda, T.G.: Parallel tensor compression for large-scale scientific data. In: IEEE International Parallel and Distributed Processing Symposium, pp. 912–922 (2016)
10. Baazizi, M.A., Lahmar, H.B., Colazzo, D., Ghelli, G., Sartiani, C.: Schema inference for massive JSON datasets. In: Extending Database Technology (EDBT), p. 222, 233 (2017)
11. Barabási, A.L., et al.: Network Science. Cambridge University Press, Cambridge (2016)
12. Battaglino, C., Ballard, G., Kolda, T.: A practical randomized CP tensor decomposition. arXiv preprint arXiv:1701.06600 (2017)
13. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech: Theory Exp. **2008**(10), P10008 (2008)
14. Brodie, M.L., Schmidt, J.W.: Final report of the ANSI/X3/SPARC DBS-SG relational database task group. ACM SIGMOD Rec. **12**(4), 1–62 (1982)

15. Bugiotti, F., Bursztyn, D., Deutsch, A., Ileana, I., Manolescu, I.: Invisible glue: scalable self-tuning multi-stores. In: Conference on Innovative Data Systems Research (CIDR) (2015)
16. Bugiotti, F., Bursztyn, D., Deutsch, A., Manolescu, I., Zampetakis, S.: Flexible hybrid stores: constraint-based rewriting to the rescue. In: International Conference on Data Engineering (ICDE), pp. 1394–1397 (2016)
17. Buluc, A., Gilbert, J.: On the representation and multiplication of hypersparse matrices. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 1–11 (2008)
18. Chen, J., Huang, Q.: Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages. Monash University, Clayton (1995)
19. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.: Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind-source Separation. Wiley, Hoboken (2009)
20. De Domenico, M., et al.: Mathematical formulation of multilayer networks. Phys. Rev. X **3**(4), 041022 (2013)
21. Deng, D., et al.: The data civilizer system. In: Conference on Innovative Data Systems Research (CIDR) (2017)
22. DiScala, M., Abadi, D.J.: Automatic generation of normalized relational schemas from nested key-value data. In: Proceedings of the International Conference on Management of Data, pp. 295–310. ACM (2016)
23. Dittrich, J., Jindal, A.: Towards a one size fits all database architecture. In: Conference on Innovative Data Systems Research (CIDR), pp. 195–198 (2011)
24. Duggan, J., et al.: The BigDAWG polystore system. ACM SIGMOD Rec. **44**(2), 11–16 (2015)
25. Färber, F., et al.: The SAP HANA database-an architecture overview. IEEE Data Eng. Bull. **35**(1), 28–33 (2012)
26. Gadepally, V., et al.: The BigDAWG polystore system and architecture. In: IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–6 (2016)
27. Gama, J.: A survey on learning from data streams: current and future trends. Prog. Artif. Intell. **1**(1), 45–55 (2012)
28. Ghosh, D.: Multiparadigm data storage for enterprise applications. IEEE Soft. **27**(5), 57–60 (2010)
29. Giannakouris, V., Papailiou, N., Tsoumakos, D., Koziris, N.: MuSQLE: distributed SQL query execution over multiple engine environments. In: IEEE International Conference on Big Data, pp. 452–461 (2016)
30. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. ACM SIGMOD Rec. **34**(4), 34–41 (2005)
31. Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. (CSUR) **15**(4), 287–317 (1983)
32. Halu, A., Mondragón, R.J., Panzarasa, P., Bianconi, G.: Multiplex pagerank. PloS ONE **8**(10), e78293 (2013)
33. Hammer, M., McLeod, D.: On database management system architecture. Technical report, Massachusetts Institute of Technology, Cambridge Lab. For Computer Science (1979)
34. Härder, T.: DBMS architecture-the layer model and its evolution. Datenbank-Spektrum **13**, 45–57 (2005)
35. Hellerstein, J.M., et al.: The MADlib analytics library or MAD skills, the SQL. Proc. VLDB Endow. **5**(12), 1700–1711 (2012)

36. Hewasinghage, M., Varga, J., Abelló, A., Zimányi, E.: Managing polyglot systems metadata with hypergraphs. In: Trujillo, J.C., et al. (eds.) ER 2018. LNCS, vol. 11157, pp. 463–478. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_33
37. Hölsch, J., Schmidt, T., Grossniklaus, M.: On the performance of analytical and pattern matching graph queries in Neo4j and a relational database. In: EDBT/ICDT International Workshop on Querying Graph Structured Data (GraphQ) (2017)
38. Hutchison, D., Howe, B., Suciu, D.: Lara: a key-value algebra underlying arrays and relations. arXiv preprint arXiv:1604.03607 (2016)
39. Hutchison, D., Howe, B., Suciu, D.: LaraDB: A minimalist kernel for linear and relational algebra computation. In: ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, pp. 2–12 (2017)
40. Jananthan, H., Zhou, Z., Gadepally, V., Hutchison, D., Kim, S., Kepner, J.: Polystore mathematics of relational algebra. In: IEEE International Conference on Big Data, pp. 3180–3189 (2017)
41. Johnson, M., Rosebrugh, R., et al.: Database interoperability through state-based logical data independence. Int. J. Comput. Appl. Technol. **16**(2–3), 97–102 (2003)
42. Kanellakis, P.C.: Elements of relational database theory. In: Formal models and semantics, pp. 1073–1156. Elsevier (1990)
43. Kang, U., Papalexakis, E., Harpale, A., Faloutsos, C.: Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 316–324 (2012)
44. Kepner, J., et al.: Dynamic distributed dimensional data model (D4M) database and computation system. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5349–5352 (2012)
45. Kepner, J., et al.: Achieving 100,000,000 database inserts per second using Accumulo and D4M. In: High Performance Extreme Computing Conference (HPEC), pp. 1–6. IEEE (2014)
46. Kim, M.: TensorDB and tensor-relational model (TRM) for efficient tensor-relational operations (2014)
47. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. J. Complex Netw. **2**(3), 203–271 (2014)
48. Klug, A.: Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM **29**(3), 699–717 (1982)
49. Knuth, D.: The Art of Computer Programming, Vol. 1: Fundamental Algorithms. Addison-Wesley, Boston (1978)
50. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
51. Kolev, B., Bondiombouy, C., Valduriez, P., Jiménez-Peris, R., Pau, R., Pereira, J.: The CloudMdsQL multistore system. In: International Conference on Management of Data (SIGMOD), pp. 2113–2116 (2016)
52. Kuang, L., Hao, F., Yang, L.T., Lin, M., Luo, C., Min, G.: A tensor-based approach for big data representation and dimensionality reduction. IEEE Trans. Emerg. Top. Comput. **2**(3), 280–291 (2014)
53. Lämmel, R., Meijer, E.: Revealing the X/O impedance mismatch. In: Backhouse, R., Gibbons, J., Hinze, R., Jeuring, J. (eds.) SSDGP 2006. LNCS, vol. 4719, pp. 285–367. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76786-2_6

54. Leclercq, E., Savonnet, M.: TDM: A tensor data model for logical data independence in polystore systems. In: Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2018 Workshops, Poly and DMAH, pp. 39–56 (2018)
55. LeFevre, J., Sankaranarayanan, J., Hacigumus, H., Tatemura, J., Polyzotis, N., Carey, M.J.: MISO: souping up big data query processing with a multistore system. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1591–1602 (2014)
56. Li, X., Cui, B., Chen, Y., Wu, W., Zhang, C.: MLog: towards declarative in-database machine learning. Proc. VLDB Endow. **10**(12), 1933–1936 (2017)
57. Lin, J., Ryaboy, D.: Scaling big data mining infrastructure: the Twitter experience. SIGKDD Explor. Newsl. **14**(2), 6–19 (2013)
58. Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., Vigier, P.: MSQL: a multidatabase language. Inf. Sci. **49**(1–3), 59–101 (1989)
59. Lu, J., Holubova, I.: Multi-model databases: a new journey to handle the variety of data. ACM Comput. Surv. (CSUR) **52**(3), 55 (2019)
60. Maccioni, A., Torlone, R.: Augmented access for querying and exploring a Polystore. In: 34th International Conference on Data Engineering (ICDE), pp. 77–88. IEEE (2018)
61. Maier, D., Rozenshtein, D., Salveter, S., Stein, J., Warren, D.S.: Toward logical data independence: a relational query language without relations. In: ACM SIGMOD International Conference on Management of Data, pp. 51–60 (1982)
62. McGregor, A.: Graph stream algorithms: a survey. ACM SIGMOD Rec. **43**(1), 9–20 (2014)
63. McHugh, J., Cuddihy, P.E., Williams, J.W., Aggour, K.S., Kumar, V.S., Mulwad, V.: Integrated access to big data polystores through a knowledge-driven framework. In: IEEE International Conference on Big Data, pp. 1494–1503 (2017)
64. Ong, K.W., Papakonstantinou, Y., Vernoux, R.: The SQL++ query language: configurable. Unifying and semi-structured. Technical report, UCSD (2015)
65. Ouzzani, M., Tang, N., Fernandez, R.C.: Data civilizer: end-to-end support for data discovery, integration, and cleaning. In: Making Databases Work, pp. 291–300. Association for Computing Machinery and Morgan & Claypool (2019)
66. Özsoyoğlu, G., Özsoyoğlu, Z.M., Matos, V.: Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. ACM Trans. Database Syst. **12**(4), 566–592 (1987)
67. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Springer, New York (2011)
68. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the Web. In: Proceedings of the 7th International World Wide Web Conference, pp. 161–172 (1999)
69. Papalexakis, E.E., Faloutsos, C., Sidiropoulos, N.D.: Tensors for data mining and data fusion: models, applications, and scalable algorithms. ACM Trans. Intell. Syst. Technol. (TIST) **8**(2), 16 (2017)
70. Riquelme, F., González-Cantergiani, P.: Measuring user influence on Twitter: a survey. Inf. Process. Manage. **52**(5), 949–975 (2016)
71. Sharp, J., McMurtry, D., Oakley, A., Subramanian, M., Zhang, H.: Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence. Microsoft patterns & practices (2013)
72. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. (CSUR) **22**(3), 183–236 (1990)

73. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., De Carvalho, A.C., Gama, J.: Data stream clustering: a survey. ACM Comput. Surv. (CSUR) **46**(1), 13 (2013)
74. Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. J. Big Data **2**(1), 8 (2015)
75. Smith, S., Ravindran, N., Sidiropoulos, N.D., Karypis, G.: SPLATT: efficient and parallel sparse tensor-matrix multiplication. In: IEEE International Parallel and Distributed Processing Symposium, pp. 61–70 (2015)
76. Stonebraker, M., et al.: One size fits all? Part 2: benchmarking results. In: Conference on Innovative Data Systems Research (CIDR) (2007)
77. Stonebraker, M., Cetintemel, U.: "One size fits all": an idea whose time has come and gone. In: International Conference on Data Engineering, ICDE 2005, pp. 2–11. IEEE (2005)
78. Stonebraker, M., et al.: C-store: a column-oriented DBMS. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 553–564. VLDB Endowment (2005)
79. Tan, R., Chirkova, R., Gadepally, V., Mattson, T.G.: Enabling query processing across heterogeneous data models: a survey. In: IEEE International Conference on Big Data, pp. 3211–3220 (2017)
80. Vargas-Solar, G., Zechinelli-Martini, J.L., Espinosa-Oviedo, J.A.: Big Data management: what to keep from the past to face future challenges? Data Sci. Eng. **2**(4), 328–345 (2017)
81. Varol, O., Ferrara, E., Davis, C.A., Menczer, F., Flammini, A.: Online human-bot interactions: detection, estimation, and characterization. In: Proceedings of the 11th International Conference on Web and Social Media (ICWSM), pp. 280–289 (2017)
82. Vogt, M., Stiemer, A., Schuldt, H.: Icarus: towards a multistore database system. In: IEEE International Conference on Big Data, pp. 2490–2499 (2017)
83. Wang, J., et al.: The Myria big data management and analytics system and cloud services. In: Conference on Innovative Data Systems Research (CIDR)
84. Wiederhold, G.: Mediators in the architecture of future information systems. Computer **25**(3), 38–49 (1992)
85. Wu, D., Sakr, S., Zhu, L.: Big Data programming models. In: Zomaya, A.Y., Sakr, S. (eds.) Handbook of Big Data Technologies, pp. 31–63. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49340-4_2

A General Framework for Multiple Choice Question Answering Based on Mutual Information and Reinforced Co-occurrence

Jorge Martinez-Gil[1(✉)], Bernhard Freudenthaler[1], and A Min Tjoa[1,2]

[1] Software Competence Center Hagenberg GmbH,
Softwarepark 21, 4232 Hagenberg, Austria
{jorge.martinez-gil,bernhard.freudenthaler,amin.tjoa}@scch.at
[2] Vienna University of Technology, Favoritenstrasse 9-11/188, 1040 Vienna, Austria

Abstract. As a result of the continuously growing volume of information available, browsing and querying of textual information in search of specific facts is currently a tedious task exacerbated by a reality where data presentation very often does not meet the needs of users. To satisfy these ever-increasing needs, we have designed an solution to provide an adaptive and intelligent solution for the automatic answer of multiple-choice questions based on the concept of mutual information. An empirical evaluation over a number of general-purpose benchmark datasets seems to indicate that this solution is promising.

Keywords: Expert systems · Knowledge engineering · Information retrieval · Question answering

1 Introduction

With the increasing amount of information that is available online, efficient and reliable computational techniques for accessing that information are needed. In fact, an ever-increasing amount of professionals from a wide range of disciplines agree that the information explosion that we are currently experiencing makes their work more tedious and even error-prone. The major reasons therefor are the fact that newly generated information is usually formatted in an unstructured way, and that the huge volume and speed at which that information is made available usually lead to information overload in their daily activities.

It is widely known that working with huge volumes of information has always been a major issue for computer scientists and practitioners in their efforts for applying for the latest advances in information technologies to offer solutions for the aforementioned problems. In fact, the latest advances in Big Data and Natural Language Processing have proven to be extremely useful for solving many

This manuscript is an extended version of Ref. [22].

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
A. Hameurlain and R. Wagner (Eds.): TLDKS XLII, LNCS 11860, pp. 91–110, 2019.
https://doi.org/10.1007/978-3-662-60531-8_4

problems that have traditionally affected the field of information processing. In practice, the daily operations of a wide range of professions require reading a large amount of textual material to identify the relevant documents and to locate the right piece of information needed. One step in the evolution towards the improvement of these processes emerges from Question Answering (QA) systems as a sub-field of Information Retrieval (IR). The design of QA systems is widely considered as an alternative to overcome traditional processes by providing accurate and understandable answers to specific questions, rather than presenting the user with a list of search-related documents [16].

The research community agrees that systems allowing generating automatic responses to textual questions could have a strong impact and practical implications in many diverse disciplines. In this way, efficient techniques for answering specific questions are in high demand and some systems implementing methods for answering questions have been designed to meet this need. However, QA technology faces some problems that prevent its progress. For example, typical approaches try to initially generate many possible responses for each question and then try to choose the right answer from all possible answers. However, the techniques for choosing the right answer need to be further improved. Moreover, the old assumption that answers to most of the questions are often explicitly stated somewhere, and the only remaining factor needed in addition is the access to a sufficiently large corpus have been proved to be inaccurate.

To effectively reason over knowledge derived from the text, QA systems must handle incomplete and potentially noisy knowledge. To tackle this problem, we have focused on computational techniques for mutual information exchange and reinforced co-occurrence analysis. Techniques of this kind have been widely used in various forms of research on content analysis, text mining, thesauri building, and ontology learning. Since the problem to be faced is too huge, our focus in this paper is laid in a particular sub-problem: multiple choice QA, i.e. answering questions in a scenario where the possible answers are already given beforehand [1]. This problem is very common in practice, as many people know how to determine the number of potential answers beforehand, and the fact that some potential clues are already given can also significantly help to reduce the workload. This is mainly because a QA system would be able to automatically process a huge amount of textual resources to find the answer that best matches a question, and that means that QA systems could save resources in the form of effort, costs and time in many fields where the explosion of information is causing problems.

Therefore, we propose here a novel framework intended to operate over huge text corpora to discover latent textual structures of existing textual representations that allow to automatically answer multiple-choice questions on any subject. Unlike our previous work [22], we envision our solution as a general-purpose framework so that conclusions being drawn apply to a wide range of specific domains. Therefore, with this idea in mind, we present here our research from which the following contributions can be highlighted:

- **Contribution 1:** We propose a new method for the automatic answer of multiple-choice questions based on the notion of mutual information exchange and reinforced co-occurrence. The advantages of this proposal in comparison with the existing ones are:
  (1) **Advantage 1**: Our approach can generate a ranking of answers without the need of a training phase over the data
  (2) **Advantage 2**: Unlike most existing systems, our approach does not need to consume textual corpora whereby the correct answer to the question is explicitly stated
  (3) **Advantage 3**: Our approach can explain the results so that a human operator can understand the ranking of the generated answers
- **Contribution 2:** We have empirically evaluated our approach using some of the most common benchmark datasets for the automatic answering of questions in the legal, geographical, and historical field. And we have verified that the results are in line with those of the state-of-the-art despite having the present solution presents the aforementioned advantages over the most advanced techniques.

The remainder of this work is organized as follows: Sect. 2 reports the state-of-the-art on question answering methods and tools that have proven to be successful in the past. Section 3 presents the fundamentals of our contribution concerning the computation of the reinforced co-occurrence over huge corpora of text. Section 4 reports the empirical evaluation of our novel approach over some benchmark datasets and the analysis of the results that we have achieved from that evaluation. Finally, we outline the conclusions and future lines of research.

2 State-of-the-art

QA systems are traditionally considered as groups of interacting software components intended to automatically reply questions by analyzing different sources of either structured or unstructured information. In practice, these sources are usually called Knowledge Bases (KBs) and can lead to two different approaches to address the problem depending on the nature of the information to be exploited: structured or unstructured solutions. Each of them has different advantages and disadvantages. For example, working with structured KBs allows exploiting the knowledge represented by using the so-called inference engines, to infer new knowledge and to answer questions [35]. However, at present, there is not an automatic way to introduce a new entity into the KB nor to determine with which existing entities should be related and how [24]. Therefore, finding practical solutions is considered as an important research challenge and it currently matters of intense research [13].

The fact is that, in practice, it is not easy to implement these systems, so they have been progressively replaced by another type of more efficient systems based on lighter knowledge models such as knowledge graphs [8] and other enhanced lexical semantic models [34], but in general, it is widely assumed that building a

fully structured KB is expensive in terms of resource consumption, it is subject to many errors, it is usually difficult and expensive to maintain, and last but not least, it is usually hardly reusable in other contexts.

In contrast, systems exploiting unstructured KBs have more practical benefits as most of them have been specifically designed to efficiently process huge amounts of textual data (usually represented in natural language). These huge amounts of data come from existing documents, databases, websites, and so on. For this reason, the most frequent type of QA system that is mentioned in the literature is the one that uses different collections of unstructured natural language KBs. The current generation of QA systems has evolved to extract answers from a wide range of different plain machine-readable resources. These QA systems exploit the massive set of unstructured information available on some data sources to retrieve information about any particular question. It is important to note that these QA systems are only possible mainly due to recent advances in big data [12] and natural language technologies [18]. Moreover, since these novel QA systems are capable of processing questions about different domains and topics, they are now used in a wide range of different scenarios [23].

IR-based solutions represent words in the form of discrete and atomic units. For example, given the fact that today's web search engines can successfully retrieve simple answers to many queries expressed by a human operator just by searching the Web. Therefore, the first approach could be to query the number of Google results for a specific question and a given answer together. However, this solution has brought several problems like the lack of context (not to mention very serious problems related to denial of service). Li et al. proposed the exploitation of structured lexical databases and corpus statistics [21]. However, the method is not optimized for dealing with QA scenarios. To overcome these problems, word processing models such as LSA [7] and term frequency-inverse document frequency (TF-IDF) partially solve these ambiguities by using terms that appear in a similar context based on their vector representation, and then they group the semantic space into the same semantic cluster. In this context, one of the best-known QA systems is IBM Watson [10], that it is very popular for its victory in the televised show *Jeopardy* [11]. Although in recent times, IBM Watson has become a generic umbrella that includes other business analytics capabilities.

There is a second possible classification that distinguishes QA systems between closed-domain and open-domain. If we focus strictly on QA in closed-domain, we find that this technology has been used in real information systems, and especially in knowledge management systems [2]. The logic behind these systems is that given an issue, the extraction of relevant resources and the decision whether or not to use that content to answer the question are two key steps in building a system. In recent times, this approach has delivered many successful applications, e.g. in the legal area. In the literature we can distinguish between two major approaches: (a) with structured KB. For example, Lame et al. [19] and Fawei et al. [9] using ontologies, or Xu et al. [33] by exploiting other KBs such as Freebase. And (b) exploiting unstructured KBs. For example, Brueninghaus

and Ashley with a classical IR approach [4], Bennet et al. with strong focus on scalability [2], Maxwell and Schafer paying attention to context [26], Mimouni et al. with the possibility to make use of complex queries [27], or most modern deep learning techniques from Marimoto et al. [28] and Nicula et al. [29], the latter with good results, although with issues concerning the interpretability of the results.

Concerning open-domain, several systems capable to operate in general-purpose scenarios have been proposed. For example, the open-source system Calcipher [31], or the more advanced IR solver which uses the Waterloo corpus from Clark et al. [5]. The IR solver tries to determine if the question along with an answer option is explicitly stated in the textual corpus, and returns the confidence that such a statement was found. Another outstanding system is the DrQA system[1] that is available under an open-source license. This system addresses the challenge of open domain question answering using Wikipedia as unstructured KB. This means that the system has to combine the challenges of finding the relevant Wikipedia pages with that of identifying the answers from those pages. What we present here is an open-domain system that uses unstructured KBs to face multiple-choice questionnaires about any subject. Our system benefits from features such as no need for training (typical of systems that use machine learning), no need to find explicit answers in the textual corpus which it is used as background (typical of early QA systems), and the ability to provide answers with a high degree of interpretability (as opposed to proposals based on neural models).

3 General Framework for Multiple Choice Question Answering Based on Mutual Information

Our approach is intended to automatically process massive amounts of textual information to look for evidence allowing to infer the most promising answers with regards to the huge range of questions that people can make. In this way, our contribution is a novel framework for automatically answering multiple-choice questions concerning a wide range of topics. This approach needs to fulfill two stages: first, we need to calculate alignment matrices between the question and the possible choices using textual corpora, and in the next stage, we need to normalize the results to produce a final result and associated ranking of possible answers. Next subsections introduce the technical preliminaries, the notion of reinforced co-occurrence, the normalization process, the implementation of our approach, and several running examples that show how this approach works in practice.

3.1 Technical Preliminaries

To overcome the current limitations of exiting QA approaches, we propose to automatically analyze the mutual information exchange [6] between a pre-processed version of the question and each of the possible choices in the context

[1] https://github.com/facebookresearch/DrQA.

of different corpora of unstructured text. In this context, the mutual information exchange of two random variables is a measure of the mutual dependence between the two variables, i.e. the mutual information I(Q; C) between two random variables Q and C is the amount of information that the choice C gives about the question Q. This can be formally defined as:

Let (q, c_n) be a question and a possible answer with values over the space $\mathcal{Q} \times \mathcal{C}$. If their joint distribution is $P_{(Q,C)}$ and the marginal distributions are P_C, the mutual information between them could be:

$$I(Q;C) = D_{\mathrm{KL}}(P_{(Q,C)} \| P_Q \otimes P_C) \tag{1}$$

Our framework considers a pair of entities q and c_n that belong to two discrete random variables Q and C quantifies the probability of their co-occurrence given their joint distribution and their specific distributions. It can be mathematically expressed such as:

$$\mathrm{P}(q, c_n) \equiv \log \frac{p(q, c_n)}{p(q)p(c_n)} = \log \frac{p(q|c_n)}{p(q)}. \tag{2}$$

Our approach minimizes when the information overlap between the question and the potential choice is 0 (i.e. $p(q|c) = 0$ or $p(c|q) = 0$), which means that the two variables considered are independent. On the other hand, our approach maximizes in the (rare case of) the question and the potential choice might be perfectly associated (i.e. $p(q|c) = 1$ or $p(c|q) = 1$), yielding the following bounds:

$$-\infty \leq \mathrm{P}(q, c_n) \leq \min\left[-\log p(q), -\log p(c_n)\right]. \tag{3}$$

Therefore, we treat the QA problem of ranking the choice set such that the correct hypothesis is the one associated with a higher score and therefore, it is placed on the top of the ranking. We learn a scoring function $S(H, z)$ with a normalization parameter z such that the score of the correct choice (i.e. its corresponding co-occurrence probability) is higher than the score of the other hypotheses and their corresponding co-occurrence probabilities. Some interesting properties are:

(1) If $Q \perp C, I(Q;C) = 0$ because $H(Q) = H(Q|C)$
(2) If $Q = C, I(Q;C) = H(Q)$
(3) If $Q = f(C), I(Q;C) = H(Q)$ where f is deterministic
(4) If $C = g(Q), I(Q;C) = H(C)$

As a final note, it is necessary to remark that mutual information is symmetric, i.e. $I(Q;C) = I(C;Q)$, this means that in our application we do not need to worry about one direction than the other since though mathematically they are the same. The symmetry can be proven such as:

$$H(Q) + H(C|Q) = H(Q,C) = H(C,Q) = H(C) + H(Q|C)$$
$$H(Q) - H(Q|C) = H(C) - H(C|Q)$$
$$I(Q;C) = I(C;Q)$$

3.2 Reinforced Co-occurrence

Our text mining approach works under the distributional assumption [20]. This assumption has proven to perform well for several problems in the past. We hypothesize that the most important words of the question and the answers will co-occur in a small fraction of the given textual corpora. Our goal is to identify and analyze this co-occurrence, to present to the user our suggestions based on the automatic interpretation and normalization of that co-occurrence.

As the mutual information method can measure how much the actual probability of a particular co-occurrence of the question and the possible choice differs from what we would expect it to be based on the probabilities of the individual events and the assumption of independence. The question arises when dealing with the concept of co-occurrence itself. Many authors use the same text sentence whereas others assume that a text frame of n-units should be considered. Many others applied the notion of the paragraph, and so on. Our proposal considers an intelligent aggregation of all of them. That is why we call it reinforced co-occurrence. Formally, reinforced co-occurrence takes input a set of numeric values from the different aspects to be analyzed and outputs an aggregated number that it is supposed to represent in a meaningful way some of the most important characteristics of the input set. And it can formally be expressed in the following way:

$$P_r = \prod_{i=0}^{i=n} P_i(q, c_n) \tag{4}$$

The key research question is how this aggregation should be performed to deliver the best possible results. To answer that question, we propose a software framework to experiment on how that aggregation could be carried out. At this point, it is important to remark that we handle the concept of trust in terms of physical proximity [25]. For example, if a given (pre-processed) question and potential answers appear in the same paragraph of a document, we will have, at least, low evidence of a relation between them. But if this pair seems to appear together frequently, in the same sentences, or pre-defined text frames, or even in the context of the same regular expressions from the textual corpora, then we could infer that we could have an answer for the given question. This is precisely what can be achieved through the reinforced co-occurrence. However, exact technical details have to be defined using proper fine-tuning.

Moreover, it is obvious to see that the design of such a framework in this context is far from being trivial. However, our experience in rapid prototyping and testing text mining pipelines has shown us that it is possible to reach a reasonable level of success [23]. According to our experience, a solution that works very well is a method with four levels of co-occurrence depending on the context whereby the question and the choice being evaluated can be found together. For this reason, we propose to work with various levels of co-occurrence, ranging from quite low degrees of restriction to very high degrees: text frame, regular expression, sentence, and paragraph. This way of working means that

very few co-occurrences can be found, but the key to all of this is that those co-occurrences found will be very precise. Therefore, the corpus of text to be used must be huge. Otherwise, it is quite probable that our technique will not be able to obtain the values (since the restrictions that we impose are very strict).

Finally, it is necessary to remark that the problem addressed here is based on short response models. The reason is that these models provide the potentially correct answer in the form of a number, a name, a date, or even a short phrase or text fragment. This makes the work of our text mining engine much easier. It is also important to note, that this assumes that there are different ways of asking questions, and most of them are characterized by the formulation of questions expressed by interrogative particles (i.e. what, who, why, when, where, where) or some kind of is-a or have-a association. At the same time, the aforementioned possible choices are expressed in natural language, and therefore, they need some pre-processing too.

3.3 Normalization

In cases where the decision is not clear, for example, several answers have all the cells in their associated column filled with values, we apply normalization. Normalization is the process of mitigating the impact of the outliers on the final decision. This is done using adjusting the values from different scales to a common scale. Some words are much more common than others. Therefore, their associated co-occurrence values will be always much higher than others. To mitigate this effect, we applied an exponential reduction of the values obtained, so the highest values are significantly reduced in comparison with the lower ones. More formally, $\alpha = 1 - e^{(-1/w)}$. With exponential normalization, the averaging window w includes the desired number of reinforced co-occurrence values, although the lowest values weight more.

3.4 Implementation

Although the concept seems to be not quite straight forward, there is a huge technical limitation for its development from a pure engineering perspective. This approach is limited by an important number of technical issues which should be overcome. These limitations, originally identified by [3], are inherent to the process of massive text mining and include: Limitations concerning the corpora size, variability inherent to the processing of natural language (verbal forms, plurals, etc.), issues concerning the different domain nomenclatures, degree of uncertainty on the accuracy of the contents, and language in which the information is represented.

Accordingly, IR systems are usually designed in the form of a pipeline, i.e. a workflow whereby the data is processed in a way that the output of one module is the input of the next one. Figure 1 shows us an overall view of our IR pipeline. These components are related to each other and process the textual information available on different levels until the QA process has been completed. The questions formulated which serve as input for our system are initially processed

by the question analysis component. This process is very important to transfer just meaningful data into the mining phase, whereby the calculation of the reinforced co-occurrence will take place. Answer extraction [32] will assign the proper results to each of the possible choices. Finally, it is necessary to normalize the raw data and create the final ranking to be delivered. The main modules of our QA system could be summarized in the following steps:

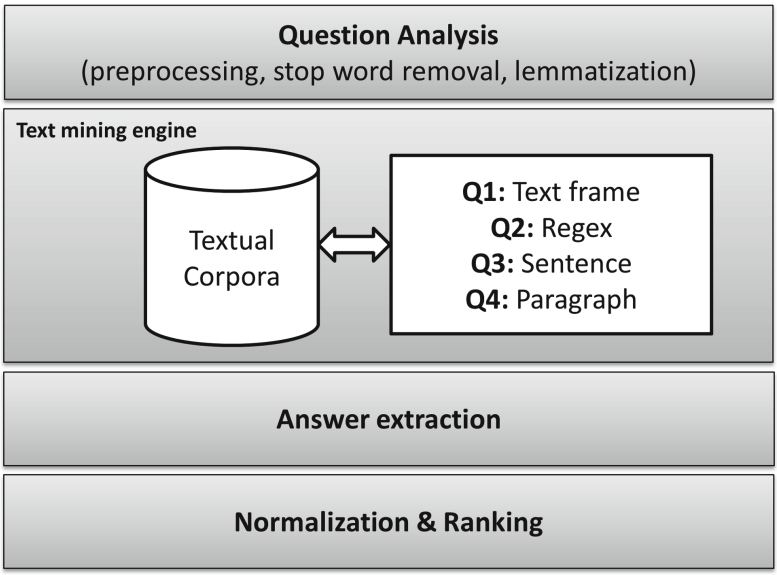


Fig. 1. Overall view of a pipeline designed to answer the multiple-choice tests. First of all, questions and answers need to be pre-processed. After this, a text mining engine is in charge of mining reinforced co-occurrence patterns. Then, these patterns are analyzed. Finally, the results are normalized and a ranking of potential choices is delivered

- *Question Analysis.* It is in charge of pre-processing both the question and the possible answers. To do that, it is necessary to remove the stop words and very common words (prepositions, adverbs, articles, etc.), to proceed with a lemmatization process, determining the root of the words to prevent irregular forms (i.e. plurals, third persons, etc.) to affect the co-occurrence.
- *Reinforced Co-occurrence Calculation.* The logic behind this module consists of counting how many times the pre-processed question and the evaluated answer co-occur together in the same text frame, in the same text expression, in the same sentence, and the same paragraph. Some parameters should be manually tuned.
- *Answer Extraction.* It consists of compiling the results and assign them to each of the possible choices. After this process, we have just raw values that need to be refined.

– *Answer Normalization and Ranking.* In this work, we usually work with exponential reductions, but other methods need to be considered in future work. The ranking consists of creating an ordered list of response according to the score obtained after normalization. Also, a heatmap is automatically created to deliver an explanation suitable for humans who need to understand how the ranking has been created.

3.5 Running Examples

To illustrate how our approach works, we have designed a running example to better understand how our pipeline processes the information. Let us see a couple of examples.

Running Example 1. Let us think in a question whereby we would like to know in which island is the volcano Etna situated. Let us think how the question could be, and how the different choices would look like.

```
On which island is Etna volcano located?
(a) Sicily
(b) Corsica
(c) Rhodes
(d) Sardinia
```

To do that, we can see in Fig. 2 the graphical summary of how this process is performed: The question and the associated choices have to be preprocessed to remove non-relevant words, perform lemmatization, etc. Then, the system continues working by conveniently dividing the information into different parts which will be transferred to the following process which is a text mining engine that looks for the reinforced co-occurrence of the question and each possible answer. As a result, we get the reinforced co-occurrence values that have to be normalized so the outliers might not behave an extreme weight in the final value. As we see, the word island presents relatively high co-occurrence values, which makes sense since the island is a very common word. One would expect to find that word many times in the corpus, so in case that the answer is not clear, this effect will be mitigated with an exponential reduction of the highest values.

After repeating this process for each of the possible choices (Sicily, Corsica, Rhodes, or Sardinia), we have must discern whether it is the correct one. After performing the corresponding pre-processing, and reinforced co-occurrence calculation, we get the Table 1 the raw results. Since just one column has all its cells with values, these results are definitive, and they give a very clear clue that the option chosen is going to be Sicily (which on the other hand is the correct answer).

Therefore, the choice that our system would select as the most promising one is (a) Sicily, also the correct answer according the ground truth. The other three possible choices has a strong relation with the word island but not to volcano

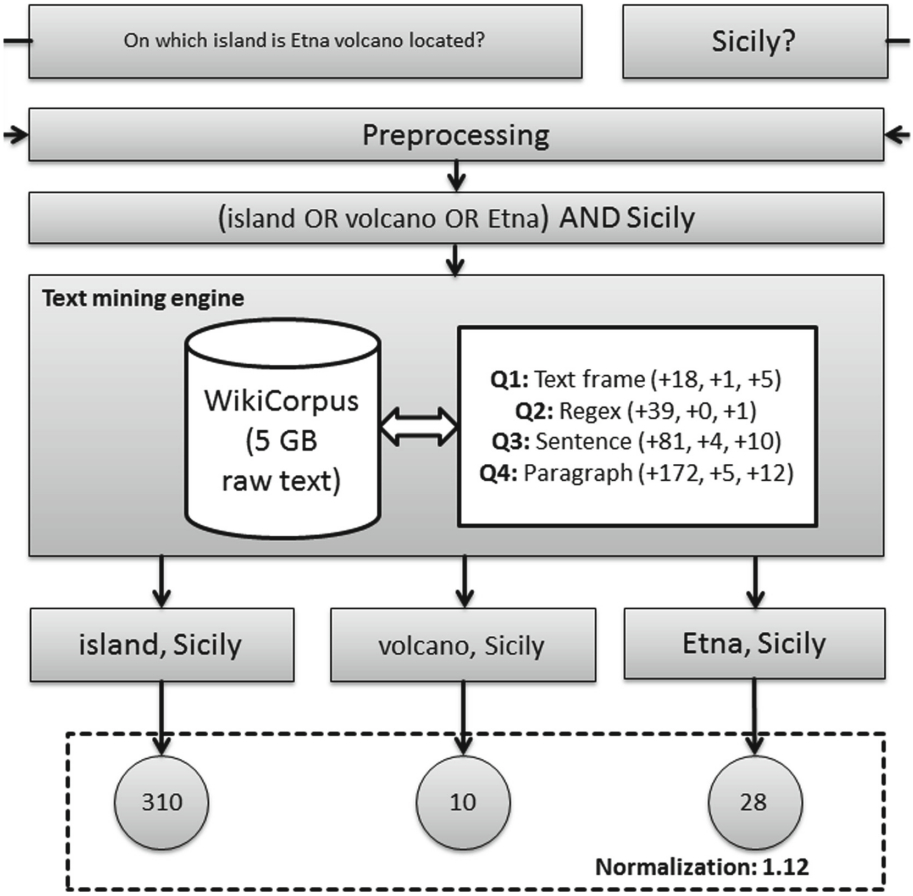


Fig. 2. Overall view of one iteration whereby a question (On which island is Etna volcano located?) and a potential answer (Sicily) is evaluated

Table 1. Raw results obtained for the reinforced co-occurrence of using WikiCorpus

Reinforced co-occur.	Sicily	Corsica	Rhodes	Sardinia
Island	310	161	262	142
Volcano	10	0	0	0
Etna	28	0	0	0

or Etna, so they would not even be considered as the final answer. In the next subsection, we also explain how a heatmap might allow to visually inspect the rationale behind the result for interpretability issues.

Running Example 2. Let us think in a question whereby we would like to know from which country did Papua New Guinea got its independence. Let us think how the question could be, and how the different choices would look like.

```
From which country did Papua New Guinea got its independence?
(a) Mozambique
(b) Australia
(c) Indonesia
(d) New Zealand
```

After the three first processing stages, i.e. Question Analysis, Reinforced Co-occurrence Calculation, and Answer Extraction; we have been able to get the values that we see in Table 2.

Table 2. Raw results obtained for the reinforced co-occurrence of using WikiCorpus

Reinforced co-occur.	Mozambique	Australia	Indonesia	New Zealand
Country	256	6028	1033	2309
Papua New Guinea	5	411	137	144
Independence	134	225	611	114

Everything seems to indicate that Australia will be the option finally chosen. Please note that Australia, Indonesia, and New Zealand as countries appear very frequently. However, in this case, normalization will reduce the impact of this fact on the final result. Therefore, the choice that our system would select as the correct one is (b) Australia, what is also the correct one according to the ground truth. The second would be (b) New Zealand. And the other two possible choices has lower values for reinforced co-occurrence, so they would not even be considered as plausible answers. As in the previous example, a heatmap allows to visually inspect the rationale behind the result for accountability and interpretability issues.

3.6 A Brief Note on the Interpretability of Our Solution

The higher the interpretability of a solution, the easier it is for a human user to understand why the predictions have been made. A solution is assumed to be more interpretable than another one if its decisions are easier for a human to understand than decisions from the other solution. For this reason, we envision the result of our process by not just choosing the most promising choice, but also we figure out how to represent the final answer in the form of a heatmap. The idea behind that it is offering a heatmap so that our solution might be more interpretable. This is mainly because, in some scenarios requiring accountability and/or interpretability, it is not just enough to provide the answer, but some reasons for that answer. By visually inspecting the heatmap, a human operator can understand how the decision has been made.

Figure 3 shows the heatmap corresponding to Running Example 1. The presence of values in each of the cells of the column Sicily indicates that the choice Sicily will be ranked first.

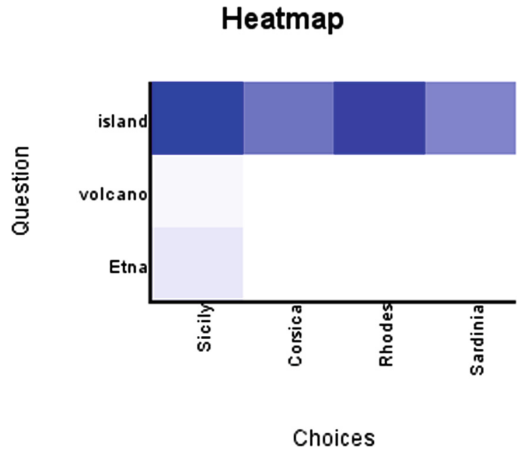


Fig. 3. Heatmap obtained for the scenario whereby an user wants to know in which island is the volcano Etna located. Higher values in the column Sicily correctly indicates that the desired answer is Sicily

Figure 4 shows the heatmap corresponding to Running Example 2. The highest values in the column Australia clearly indicates that this choice will be ranked first.

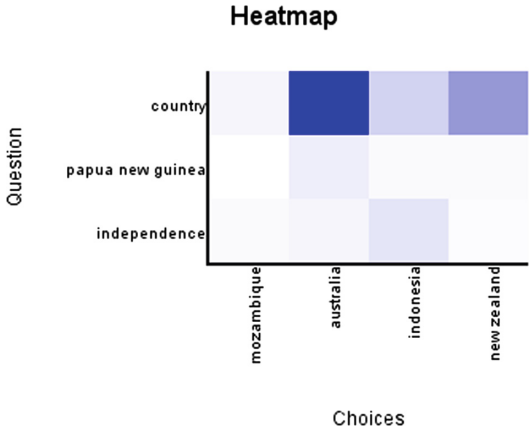


Fig. 4. Heatmap obtained for the scenario whereby an user wants to know from which country did Papua New Guinea got its independence. Higher values in the column Austria indicates that the desired answer is Australia

We want to emphasize that the heatmap can be generated in two different approaches: with raw values or with normalized values. It is up to the user to decide what it is better for a specific purpose.

4 Results

We explain here the results. It is important to remark that results are highly dependent on the base textual corpus that will be processed. Choosing a relevant, specific base corpus to evaluate each of the possible choices is important in this situation. On the other hand, the task of evaluating the system is a vital stage, as it will assess the performance, as well as the accuracy of the techniques. In this work, we have chosen the strictest methodology to evaluate systems, which consists of a binary classification. The answer was right or wrong. However, there are many proposals in this sense, being some of the most popular those that grant a score according to the ranking that the evaluated system gives to the correct answer.

4.1 Setup Configuration

We explain here the implementation decisions that we have taken to achieve a prototype for testing our hypothesis. The most important implementation details of our approach include:

- Limitations concerning the corpus size. With the emergence of new paradigms approaches for big data management, this kind of problems is losing importance. In this work, we have used WikiCorpus [30] which is a reduced version of Wikipedia. Wikicorpus is widely popular in the text mining community since it combines a great number of general-purpose articles represented in almost 5 GB of plain text.
- Variability is inherent to the processing of natural language. In this work, we have relied on the Krovetz solution [17] to proceed with the lemmatization. Besides, we have implemented some functionality to avoid processing verbs, common stop words, and nouns with a low meaning load.
- Issues concerning domain nomenclature. The problem for methods trying to exploit information extraction strategies is that they should be adapted to each different domain. It is widely assumed that meaning is usually represented by nouns (and noun phrases) so that it is common to built retrieval methods based on noun representations extracted. Since we are building a framework intended for general purpose, we have not taken design decision within this regard.
- Degree of uncertainty on the accuracy of the contents. In this work, we assume the fact that it is quite likely that the corpus to be analyzed might have some errors or inaccurate information. However, we foresee that the impact of these errors might be blurred by the overwhelming presence of correct information.

– Language in which the information is represented. To overcome this limitation, we have decided to use only English in this version of the work. WikiCorpus [30] is represented in English, so we have no problems with this regard.
– Other additional parameters: Other additional parameters are the size of the text frame (we have chosen a text frame of 5 units), and the kind of regular expressions to be considered that we have chosen are (is-a) and (have-a), the exploration of more sophisticated regular expressions is pending as future work.

4.2 Experiments

As a demonstration, we report here the results from three different datasets: legal, geographical, and historical. We have chosen 60 multiple-choice questions (20 for each dataset), and we have compared the results with those achieved by several publicly available QA systems. The legal questions have been retrieved from textbooks by the editorial Oxford University Press[2].

At the same time, the multiple-choice questionnaires on geography and history have been taken from the OpenTrivia approach [15]. But please note that since ours is a general-purpose framework, in principle, there would be no restriction to operate on other datasets. On the other hand, the proposals we are going to compare with are: the open-source system Calcipher [31], the once-outstanding solution Li et al. [21], and the classic but yet very powerful [7] using the classical configuration. Finally, since there is a 25% chance of making the right choices just by answering randomly, that result will be our baseline.

Legal Dataset. We have worked with a dataset on questions of legal nature. The reason is to check if our solution could help to alleviate the problem of information overload in the legal area, which is currently one of the professional fields that needs it the most. An example of question is

```
A procedure of peaceful settlement of international dispute is a:
(a) Conciliation (correct)
(b) Cooperation procedure
(c) Jurisdiction
(d) Resolution
```

The summary of results that we have achieved are summarized in Table 3.

Our approach is able to beat the rest of solutions by correctly answering around one third of the 20 questions. Rest of QA systems are not able to properly answer even half of the questions, although they manage to beat the baseline.

[2] http://www.oup.com.

Table 3. Comparison with other approaches regarding the legal dataset

Approach	Correct answers	Accuracy
Baseline	5/20	25%
Calcipher [31]	7/20	35%
Li et al. [21]	9/20	45%
LSA-classic [7]	9/20	45%
Our approach	13/20	65%

Geographical Dataset. The second benchmark dataset is about questions of general geography. Sometimes it is very difficult to know a certain data about geography. We now want to see if our proposal could satisfactorily help a human operator. An example of multiple-choice question is

```
What is the deepest freshwater lake on Earth?
(a) Onega
(b) Ladoga
(c) Huron
(d) Baikal (correct)
```

The results achieved by all the approaches considered are summarized in Table 4.

Once again, our solution has managed to correctly answer more questions than the rest of the proposals, which this time also fail to reach half the desired answers.

Table 4. Comparison with other approaches regarding the geographical dataset

Approach	Correct answers	Accuracy
Baseline	5/20	25%
Calcipher [31]	7/20	35%
Li et al. [21]	6/20	30%
LSA-Classic [7]	9/20	45%
Our approach	12/20	60%

Historical Dataset. The third dataset is about questions about the history of mankind, regardless of date or geographical region. It is very difficult for a human operator to store all this encyclopedic knowledge. For this reason, we want to know if our proposal could be useful in this sense. One example question is:

```
Name the first great Greek tragic playwright who is now acknowledged
as the Father of Drama
(a) Aeschylus (correct)
(b) Aesop
(c) Euripides
(d) Sophocles
```

The results that the QA systems considered have achieved are summarized in Table 5.

Table 5. Comparison with other approaches regarding the historical dataset

Approach	Correct answers	Accuracy
Baseline	5/20	25%
Calcipher [31]	5/20	25%
Li et al. [21]	8/20	40%
LSA-Classic [7]	8/20	40%
Our approach	13/20	65%

Once again, our proposal is ranked first, just ahead of LSA and Li et al. which also fail to reach half the correct answers this time. Calcipher presents the worst performance.

4.3 Discussion

QA technology is becoming an important solution in many areas overloaded by the constant generation of large amounts of information in the form of free text. In this context, being able to automatically answering specific questions correctly can contribute to alleviating the problem of dealing with those huge amounts of unstructured text. This technology, however, faces some obstacles in its development. And it requires engineering work to properly tune some of the parameters associated with the processes that intervene in the pipeline.

The lessons learned from this work can be applied in more advanced situations where the possible choices are not present. At this point, we would need a way to automatically generate possible choices, which will then be evaluated by our system. Moreover, it is important to remark that the choice of the different alternatives for answering the questions is a critical point. Therefore, it is necessary to evaluate the fairness of the choices to be evaluated. In the future, we want to use the knowledge base YAGO [14] for automatically generate candidate choices.

5 Conclusions and Future Work

Methods and techniques for automatically answering specific questions are in high demand, and as a result, many solutions for QA have been developed to respond to this need. The major reason for that is that the capability to automatically answer questions using computers could help alleviate a problem involving tedious tasks such as an extensive information search what is, in general, time-consuming. By automatically providing hints concerning a wide number of topics, lots of resources in the form of effort, costs and time can be preserved. In this work, we have presented our general framework for automatically addressing multiple-choice questions and the development of techniques for automatically finding the correct answer through mutual information and reinforced co-occurrence.

We have seen that although approaches based on structured KB often yield good results, it is difficult to use them in practice mainly due to the time and associated cost when building such structured KB (i.e. it is expensive in terms of effort, costs and time needed) and it is often very difficult to find experts for curating the KBs. In contrast, our approach is more suitable when selecting the actual right choice from a list of the possible answers due to the advances in big data processing and natural language technology. Although with some limitations, the experiments that we have performed over general-purpose datasets yields good results and seem to be promising. Moreover, in the present work, we have not yet fully explored the characteristics of many texts to utilize these features for building our QA system. For example, properties such as references between articles should be investigated more deeply as part of future work.

As additional future lines of research, we also need to work towards improving the technical limitations that we were not able to overcome within the frame of this work. This includes the capability to work with different multilingual textual corpora at the same time, the proper processing of verbs when formulating questions and preparing potential answers, the sentiment analysis of the questions and answers, and the proper aggregation of the different features through a training phase that can help to appropriately configure the complete pipeline. We think that by successfully addressing these challenges, it is possible to build solutions that can help to the many users to overcome one of the most serious problems that they have to face in their daily activities.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful suggestions to improve this work. This research has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

References

1. Aydin, B.I., Yilmaz, Y.S., Li, Y., Li, Q., Gao, J., Demirbas, M.: Crowdsourcing for multiple-choice question answering. In: AAAI 2014, pp. 2946–2953 (2014)

2. Bennett, Z., Russell-Rose, T., Farmer, K.: A scalable approach to legal question answering. In: ICAIL 2017, pp. 269–270 (2017)
3. Blohm, S., Cimiano, P.: Using the web to reduce data sparseness in pattern-based information extraction. In: PKDD 2007, pp. 18–29 (2007)
4. Brueninghaus, S., Ashley, K.D.: Improving the representation of legal case texts with information extraction methods. In: ICAIL 2001, pp. 42–51 (2001)
5. Clark, P., et al.: Combining retrieval, statistics, and inference to answer elementary science questions. In: AAAI 2016, pp. 2580-2586 (2016)
6. Church, K.W., Hanks, P.: Word association norms, mutual information and lexicography. In: 27th ACL, pp. 76–83 (1989)
7. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. JASIS **41**(6), 391–407 (1990)
8. Ding, J., Wang, Y., Hu, W., Shi, L., Qu, Y.: Answering multiple-choice questions in geographical Gaokao with a concept graph. In: ESWC 2018, pp. 161–176 (2018)
9. Fawei, B., Pan, J.Z., Kollingbaum, M.J., Wyner, A.: A methodology for criminal law and procedure ontology for legal question answering. In: JIST 2018, pp. 198–214 (2018)
10. Ferrucci, D.A.: Introduction to this is Watson. IBM J. Res. Dev. **56**(3), 1 (2012)
11. Ferrucci, D.A., Levas, A., Bagchi, S., Gondek, D., Mueller, E.T.: Watson: beyond jeopardy! Artif. Intell. 199–200, 93–105 (2013)
12. Hameurlain, A., Morvan, F.: Big Data management in the cloud: evolution or crossroad? In: BDAS 2016, pp. 23–38 (2016)
13. Hoeffner, K., Walter, S., Marx, E., Usbeck, R., Lehmann, J., Ngonga Ngomo, A.-C.: Survey on challenges of question answering in the semantic web. Semant. Web **8**(6), 895–920 (2017)
14. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. Artif. Intell. **194**, 28–61 (2013)
15. Joshi, M., Choi, E., Weld, D.S., Zettlemoyer, L.: TriviaQA: a large scale distantly supervised challenge dataset for reading comprehension. In: ACL1 2017, pp. 1601–1611 (2017)
16. Kolomiyets, O., Moens, M.-F.: A survey on question answering technology from an information retrieval perspective. Inf. Sci. **181**(24), 5412–5434 (2011)
17. Krovetz, R.: Viewing morphology as an inference process. Artif. Intell. **118**(1–2), 277–294 (2000)
18. Kumar Ray, S., Singh, S., Joshi, B.P.: Exploring multiple ontologies and WordNet framework to expand query for question answering system. In: IHCI 2009, pp. 296–305 (2009)
19. Lame, G.: Using NLP techniques to identify legal ontology components: concepts and relations. Artif. Intell. Law **12**(4), 379–396 (2004)
20. Lee, L.: Measures of distributional similarity. In: ACL 1999 (1999)
21. Li, Y., McLean, D., Bandar, Z., O'Shea, J., Crockett, K.A.: Sentence similarity based on semantic nets and corpus statistics. IEEE Trans. Knowl. Data Eng. **18**(8), 1138–1150 (2006)
22. Martinez-Gil, J., Freudenthaler, B., Tjoa, A.M.: Multiple choice question answering in the legal domain using reinforced co-occurrence. In: DEXA (1) 2019, pp. 138–148 (2019)
23. Martinez-Gil, J., Freudenthaler, B., Natschlaeger, T.: Automatic recommendation of prognosis measures for mechanical components based on massive text mining. IJWIS **14**(4), 480–494 (2018)

24. Martinez-Gil, J.: Automated knowledge base management: a survey. Comput. Sci. Rev. **18**, 1–9 (2015)
25. Martinez-Gil, J.: An overview of textual semantic similarity measures based on web intelligence. Artif. Intell. Rev. **42**(4), 935–943 (2014)
26. Maxwell, K.T., Schafer, B.: Concept and context in legal information retrieval. In: JURIX 2008, pp. 63–72 (2008)
27. Mimouni, N., Nazarenko, A., Salotti, S.: Answering complex queries on legal networks: a direct and a structured IR approaches. In: AICOL 2017, pp. 451–464 (2017)
28. Morimoto, A., Kubo, D., Sato, M., Shindo, H., Matsumoto, Y.: Legal question answering system using neural attention. In: COLIEE@ICAIL 2017, pp. 79–89 (2017)
29. Nicula, B., Ruseti, S., Rebedea, T.: Improving deep learning for multiple choice question answering with candidate contexts. In: ECIR 2018, pp. 678-683 (2018)
30. Reese, S., Boleda, G., Cuadros, M., Padró, L., Rigau, G.: Wikicorpus: a word-sense disambiguated multilingual Wikipedia corpus. In: LREC 2010 (2010)
31. Stam, M.: Calcipher system. https://github.com/matt-stam/calcipher. Accessed 01 Apr 2019
32. Sun, H., Wei, F., Zhou, M.: Answer extraction with multiple extraction engines for web-based question answering. In: NLPCC 2014, pp. 321–332 (2014)
33. Xu, K., Reddy, S., Feng, Y., Huang, S., Zhao, D.: Question answering on freebase via relation extraction and textual evidence. In: ACL1 2016 (2016)
34. Yih, W.-T., Chang, M.-W., Meek, C., Pastusiak, A.: Question answering using enhanced lexical semantic models. In: ACL1 2013, pp. 1744–1753 (2013)
35. Zhang, Y., He, S., Liu, K., Zhao, J.: A joint model for question answering over multiple knowledge bases. In: AAAI 2016, pp. 3094–3100 (2016)

Rejig: A Scalable Online Algorithm for Cache Server Configuration Changes

Shahram Ghandeharizadeh(✉), Marwan Almaymoni, and Haoyu Huang

Computer Science Department, USC, Los Angeles, CA 90089-0781, USA
{shahram,almaymon,haoyuhua}@usc.edu

Abstract. A cache server configuration describes an assignment of data fragments to cache manager instances (CMIs). A load balancer may change this assignment by migrating fragments from one CMI to another. Similarly, an auto-scaling component may change the assignment by either inserting or removing CMIs in response to load fluctuations. These changes may generate stale cache entries. Rejig is a scalable online algorithm that manages configuration changes while providing read-after-write consistency. It is novel for several reasons. First, it allows for a subset of its clients and CMIs to use different configurations. Second, its client components propagate configuration changes to one another on demand and by using CMIs. This enables Rejig to scale and support diverse application classes including trusted mobile clients accessing the caching layer. When clients have intermittent network connectivity, Rejig detects if their cached configurations may result in stale data and updates them to the latest with no performance impact on either the CMIs or other clients. Rejig's overhead is in the form of 4 extra bytes of memory per cache entry and 4 extra bytes of the network bandwidth per request from a client to a CMI.

1 Introduction

Caches such as memcached [32], Redis [35], Ignite [15], KOSAR [17], and others improve the performance of traditional database management systems with workloads that exhibit a high read to write ratio [6,7,40]. A caching layer may consist of tens of servers for a small installation and thousands of servers with a popular site such as Facebook [33].

A physical server with many cores may host several Cache Manager Instances, CMIs. Each CMI is a process that might be multi-threaded. It is assigned a fixed number of cores and some amount of memory. It is also assigned a fraction of cache entries, a *fragment*. Multiple fragments are assigned to one CMI for load balancing. A load balancer may consider factors such as imposed load and cache hit rate to adjust the assignment of fragments to CMIs to enhance a performance metric such as system throughput [1,38].

A *configuration* is an assignment of fragments to CMIs. A coordinator manages configuration changes. A configuration changes due to: (1) addition or removal of CMIs by an auto-scaling component (or a system administrator),

© Springer-Verlag GmbH Germany, part of Springer Nature 2019
A. Hameurlain and R. Wagner (Eds.): TLDKS XLII, LNCS 11860, pp. 111–134, 2019.
https://doi.org/10.1007/978-3-662-60531-8_5

(2) re-assignment of fragments to CMIs by a load-balancer, (3) re-assignment of fragments to CMIs in the presence of network partitions, (4) re-partitioning of data across fragments by a re-organization component in the form of either increasing or decreasing the number of fragments, or (5) a combination of these.

Configuration changes must preserve the application's read-after-write consistency defined as a read of a cache entry observing the value produced by the last committed write of the entry [31]. Configuration changes may compromise read-after-write consistency for two reasons. First, during the window of time when the coordinator publishes a new configuration, a few clients may have the old configuration while others have the new configuration. This discrepancy may cause two or more clients that reference the same cache entry to contact different CMIs, observing different values. If they write this cache entry then they will generate different values in different CMIs. The value observed by a subsequent read depends on whether this read is issued using the old or the new configuration, potentially compromising read-after-write consistency and correctness of an application. Second, a configuration change may re-assign a fragment to a new destination CMI without physically deleting its cache entries from the source CMI, leaving these entries to become cold at the source CMI and evicted by its cache replacement technique. In the presence of updates and a subsequent configuration change that assigns the fragment back to the source, the application may observe stale cache entries. To elaborate, consider a system that migrates F_k from CMI_i to CMI_j without deleting F_k's cache entries stored at CMI_i. Should the application update F_k's cache entries assigned to CMI_j then the value of their replicas on CMI_i become stale. If a subsequent configuration change assigns F_k back to CMI_i, references to these cold cache entries observe stale values. This violates read-after-write consistency.

An ideal solution to these two challenges should (1) be **pauseless** by processing user requests in the presence of configuration changes, (2) provide **read-after-write consistency** that guarantees data produced by a committed write is observed by all subsequent reads, (3) be **agnostic to the number of clients**, and (4) preserve as many **valid keys** as possible in the presence of configuration changes. With the latter, with v fragments assigned to CMI_i, if a configuration change assigns one fragment to a different CMI then keys of the remaining $v-1$ fragments of CMI_i should remain valid.

The primary *contribution* of this paper is Rejig[1] [16], a scalable online algorithm that satisfies the above requirements. Rejig extends existing systems [21,33]. While it allows multiple clients to use different configurations, it guarantees consensus on a fragment's replica by requiring clients that reference the fragment to use its latest CMI assignment always. Moreover, it allows cache entries of an old replica of a re-assigned fragment to become cold and evicted by the CMI's replacement technique. With those entries that remain, Rejig detects when the application references them and treats them as cache misses. Rejig may be configured to delete these cache entries to free the CMI's memory.

[1] The definition of Rejig is to rearrange or organize differently.

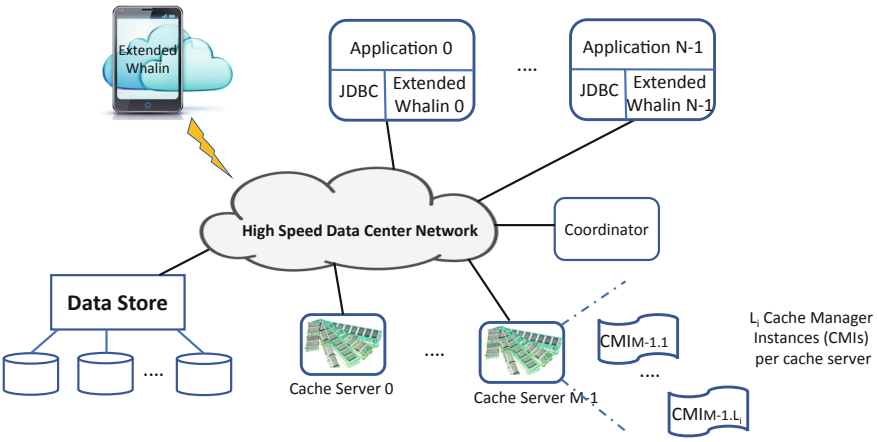
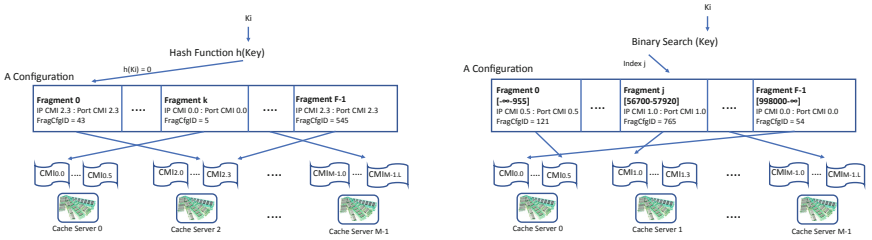


Fig. 1. A candidate architecture for Rejig. Concepts underlying Rejig are applicable to those architectures that tightly integrate the cache with the application [17] or represent the caching layer as a middleware [15,28,34].

Rejig is designed for clients and CMIs deployed in a data center. Rejig does not require the coordinator to propogate a new configuration to all CMIs. It employs clients to perform this task on demand, making it appropriate for other deployments. For example, it may be used with trusted mobile clients[2] that have intermittent network connectivity to the caching layer. It also functions with CMIs deployed across two or more geographically distributed data centers [3]. These deployments are possible because Rejig satisfies the following properties. First, CMIs are passive entities that respond to requests. Second, clients do not transmit a configuration to one another directly. They use CMIs and the coordinator intelligently to obtain the latest configuration on demand, i.e., once a Rejig client detects that its request was issued using an old configuration.



(a) Hash partitioning. (b) Range partitioning.

Fig. 2. A configuration with two partitioning techniques.

[2] Untrusted mobile clients may open possibilities for a Denial of Service (DoS) attack or data corruption.

The software architecture of Rejig hides its implementation details. Its transparency frees software developers to focus on their application and its requirements (instead of configuration changes and how to manage impacted fragments). Central to its design is a monotonically increasing global unique identifier for each configuration published by the coordinator, GlobalCfgID. Each updated and inserted cache entry is tagged with the current GlobalCfgID. Moreover, each client must piggyback the GlobalCfgID of its cached configuration with the request it issues to a CMI. Hence, Rejig imposes two types of overhead. First, it requires 4 extra bytes of memory per cache entry to store GlobalCfgID. Second, a client request consumes 4 extra bytes of the available network bandwidth per request to transmit GlobalCfgID of its configuration.

The rest of this paper is organized as follows. Section 2 details Rejig's design. Section 3 presents an implementation of this design. We evaluate this implementation using microbenchmarks and traces from Azure [10] and WorldCup '98 [4] in Sect. 4. Section 5 contains related work. Brief conclusions are presented in Sect. 6.

2 General Design of Rejig

Rejig's software architecture consists of cache manager software, a coordinator responsible for maintaining the configuration, and a client component used by the application to issue requests. Figure 1 shows an example deployment of this software architecture. A cache server hosts one or more Rejig cache manager instances, CMIs. Each CMI is identified using the combination of its IP and port number, see Fig. 2. The coordinator represents a configuration as F fragments assigned to different CMIs. We detail a space efficient implementation of a configuration in Appendix B.

Rejig supports hash and range partitioning techniques to shard the application's data across CMIs. With the former, a hash function is used to identify the fragment containing a referenced key K. With the latter, the configuration maintains the range of values assigned to each fragment. It performs a binary search to identify the fragment containing K. Rejig also supports a hybrid partitioning technique [1] that applies a hash function to map K to an order preserving space that is range partitioned across CMIs. We describe how Rejig's coordinator increases the number of fragments F with the hash partitioning strategy in Appendix B.1.

Rejig's coordinator uses a monotonically increasing integer, GlobalCfgID, to identify a configuration. Rejig's client and server components maintain their latest value of GlobalCfgID. Rejig's client component caches a local copy of the latest configuration for efficient processing of requests. When the coordinator computes a new configuration, it increments GlobalCfgID. For each impacted fragment, the coordinator informs either a subset or all impacted CMIs of the new GlobalCfgID and inserts the corresponding configuration in these CMIs. As an example of updating a subset of the impacted CMIs, consider a scenario that migrates a fragment from a source CMI to a destination CMI. It is sufficient

Table 1. Terms and their Definitions.

Term	Definition
F	Number of fragments
CMI_i	Cache manager instance i
Configuration	Mapping of F fragments to CMIs
F_k	Fragment k
GlobalCfgID	Coordinator maintained monotonically increasing value that identifies a configuration
$FragCfgID_k$	GlobalCfgID value of the configuration that either created or changed the assignment of fragment F_k
V_{cid}	Configuration id associated with a cache entry when it is either inserted or updated in a CMI
α	Number of CMIs a client inserts GlobalCfgID and configuration into once it obtains the latest configuration

for Rejig to update the source CMI with the new GlobalCfgID and insert the latest configuration in this CMI only. As an example of updating all impacted CMIs, when a CMI is removed, its fragments are assigned to different CMIs. The coordinator updates the GlobalCfgID of these CMIs and inserts the new configuration in all (Table 1).

Rejig's clients and CMIs use a distributed collaborative algorithm to update their GlobalCfgID value and cached configuration. Section 2.1 details this algorithm.

Rejig's coordinator implements a re-organization algorithm that changes the assignment of fragments to CMIs in response to an evolving workload. The efficiency of a configuration change and how it re-organizes fragments' assignment is the responsibility of this algorithm (and not Rejig). At any given time, there is one active coordinator. However, there may be multiple standby coordinators, each of which is prepared to take over if the active coordinator crashes. The active coordinator stores the latest configuration and its GlobalCfgID on an external storage system that is highly available (such as ZooKeeper [26]). The standby coordinators use the external storage system to detect failure of the active coordinator, select a new active coordinator, and recover the configuration and its GlobalCfgID.

With each configuration change, the coordinator maintains the value of GlobalCfgID for each fragment F_k impacted by that change, $FragCfgID_k$. A fragment's $FragCfgID_k$ is initialized to the GlobalCfgID value that created it. A fragment is impacted by one configuration change. However, a configuration change may impact several different fragments. For example, removal of a cache server in Fig. 2 impacts multiple CMIs and their assigned fragments. Thus, at any instance in time, different fragments may have different $FragCfgID_k$ values, identifying the GlobalCfgID value that changed their assignment to a CMI.

Table 2. Rejig and its variants.

Rejig	A client always inserts its newly obtained configuration in the next α unique CMIs
Rejig^C	A client inserts its newly obtained configuration in the next α unique CMIs only if it fetches the latest configuration from the coordinator
Rejig^T	Similar to Rejig with the following termination condition. A client stops inserting its obtained configuration in CMIs once a CMI reports it has the latest configuration

Example 2.1. In Fig. 2(b), F_j's FragCfgID is 765. Assume the current GlobalCfgID value is also 765, GlobalCfgID = 765. If the coordinator assigns a different fragment, say F_0, to a different CMI then it increments GlobalCfgID by one, GlobalCfgID = 766. It produces a new configuration with F_0's FragCfgID set to 766 along with IP and port number of its newly assigned CMI. Other fragments' FragCfgID including F_j's FragCfgID remain unchanged.■

2.1 Processing Get Requests

Algorithms 1 and 2 provide Rejig's protocol to process the get command by a client and a CMI, respectively. Appendix A provides a formal proof of this protocol. A client piggybacks its value of GlobalCfgID with every request it issues to a CMI_i, see line 3 of Algorithm 1. A cache manager instance, CMI_i, compares its latest known GlobalCfgID to the one provided by the client. There are three possibilities, see lines 1 to 8 of Algorithm 2. Either the two are equal, CMI_i's GlobalCfgID is greater than the client's GlobalCfgID, or the CMI's GlobalCfgID is less than client's GlobalCfgID. Consider each in turn.

CMI_i's GlobalCfgID Equals Client Provided GlobalCfgID: If CMI_i has the value associated with the referenced key then it returns this value including its configuration id, V_{cid}. V_{cid} identifies the configuration in which this cache entry was either inserted or updated in CMI_i (line 10 in Algorithm 2). The client compares V_{cid} with its assigned fragment's $FragCfgID_k$. If V_{cid} is greater than or equal then the value is valid and the client provides it to the application. Otherwise, the value was created in an older configuration that mapped this fragment to the same CMI and the value *may be* stale. Hence, the client discards the value and reports a cache miss. One may configure Rejig clients to delete this cache entry, freeing the available cache space of the CMI, see lines 19 to 24 in Algorithm 1.

This algorithm may incorrectly identify a value as stale if it was not updated while its fragment F_k was assigned to some other CMIs. The likelihood of this false negative is a function of the popularity of the cache entry, the mix of reads and writes in the workload, and how long F_k was mapped to a different CMI before being re-assigned. We quantify this in Sect. 4.2.

Algorithm 1: Client: get

```
   get(key)
   Input: key: byte array
   Result: cached value or null
   Let ConfigKey = the key that identifies the cache entry of a configuration.
   Let LatestConfig = Client's latest copy of the configuration.
   Let ClientGCfgID = Client's current GlobalCfgID value.
 1 fragment = getFragment(LatestConfig, key);
 2 cache = fragment.CMI;                                  // get the assigned CMI
 3 result = cache.get(ClientGCfgID, key);
 4 if result.code == RefreshAndRetry then
 5     newConfig = cache.get(ConfigKey);
 6     if newConfig.value ≠ null then
 7         LatestConfig = newConfig;
 8         ClientGCfgID = newConfig.GlobalCfgID;
 9     else
           /* the configuration cache entry may be evicted                  */
10         newConfig = coordinator.getLatestConfig();
11         LatestConfig = newConfig;
12         ClientGCfgID = newConfig.GlobalCfgID;
13     end
14     return get(key);
15 else
16     if cache is one of the next α unique CMIs then
17         cache.set(ClientGCfgID, ConfigKey, LatestConfig);
18     end
19     if result.code == hit then
20         if fragment.FragCfgID_k ≤ result.V_cid then
21             return result.value;
22         else
23             cache.delete(ClientGCfgID, key) ;          // asynchronously
24             return null;
25         end
26     else
27         if result.code == miss then
28             return null;
29         end
30     end
31 end
```

CMI$_i$'s GlobalCfgID is Greater than the Client's GlobalCfgID: This condition is satisfied when the client's cached configuration is old and CMI$_i$ is provided with a more recent configuration. CMI$_i$ returns a "*Refresh & Retry*" response to the client, see line 2 in Algorithm 2. In response, the client fetches the latest configuration from CMI$_i$ and retries its request. If CMI$_i$ evicted[3] the latest

[3] CMI$_i$ may pin the latest configuration to prevent its eviction.

configuration (i.e., reports a cache miss), the client contacts the coordinator for the latest configuration. Subsequently, it retries its request, see lines 4 to 14 in Algorithm 1. It is possible to reduce the number of roundtrips by requiring a CMI_i to piggyback the new configuration (assuming CMI_i has it) with its "*Refresh & Retry*" response.

Rejig clients disseminate the latest configuration to one another using the CMIs. A client that fetches a configuration inserts it into the next α unique CMIs that it contacts to process a request, piggybacking its GlobalCfgID value along with each insert, see line 17 in Algorithm 1. A CMI ignores this insertion when its GlobalCfgID is greater, i.e., the configuration changed and this CMI has a more recent configuration.

There are other variations of this dissemination technique [12]. We consider two variants named Rejig^C and Rejig^T, see Table 2. With Rejig^C, a client inserts its known configuration into the next α unique CMIs only if it fetches the latest configuration from the coordinator. With Rejig^T, a client stops inserting once a CMI reports that it has the latest configuration. We quantify the tradeoffs associated with Rejig, Rejig^C, and Rejig^T in Sect. 4.1.

CMI_i's GlobalCfgID is Less than the Client's GlobalCfgID: CMI_i deletes its known configuration and sets its GlobalCfgID with the one provided by the client, see lines 5 to 6 in Algorithm 2. The client may insert its known configuration in CMI_i, see line 17 of Algorithm 1.

Algorithm 2: CMI: get

```
   get(ClientGCfgID, key)
   Input: ClientGCfgID: integer, key: byte array
   Result: response code, the cached value and the cache entry's configuration id if
           found.
   // ClientGCfgID is the client's GlobalCfgID value
   Let ConfigKey = the key of the latest known configuration.
   Let CMIGCfgID = CMI's current GlobalCfgID value.
1  if CMIGCfgID > ClientGCfgID then
2  |  return RefreshAndRetry;
3  else
4  |  if CMIGCfgID < ClientGCfgID then
5  |  |  CMIGCfgID = ClientGCfgID;
6  |  |  delete(ConfigKey);
7  |  end
8  end
9  if key is cached then
10 |  return cache hit, the cached value and its associated configuration id;
11 end
12 return cache miss;
```

When multiple threads of a client receive "*Refresh & Retry*", only one thread obtains the latest configuration from CMI_i while other threads wait. Once this thread obtains the configuration, all threads use it to retry their requests. Threads referencing CMIs other than CMI_i are not blocked.

In sum, Rejig employs clients (instead of the coordinator) to propagate a new configuration to other CMIs on demand. This prevents the coordinator from becoming a bottleneck, realizing a scalable Rejig protocol.

Example 2.2. Consider the range partitioned configuration of Fig. 2(b). Cache entry K_i is assigned to Fragment F_j. F_j is assigned to CMI 1.0. Assuming GlobalCfgID = 765, a write of K_i sets its V_{cid} to 765. Assume a configuration change that assigns F_j to the CMI hosting F_0, CMI 0.5. This results in the following changes: GlobalCfgID is set to 766, F_j's FragCfgID is set to 766, F_j's CMI IP and port number are set to those of CMI 0.5. Another write of K_i is directed to CMI 0.5, creating this cache entry and setting its V_{cid} to 766. Now, the copy of K_i on CMI 1.0 is stale. Should another configuration change assign F_j back to CMI 1.0, the GlobalCfgID is incremented by one, GlobalCfgID = 767. Moreover, F_j's FragCfgID is also set to 767 and its IP and port number is set to CMI 1.0. A reference for K_i may observe the stale version. This version's FragCfgID (765) is lower than F_j's FragCfgID 767. Hence, Rejig discards this version and reports a cache miss. ■

2.2 Read-After-Write Consistency

Intuitively, Rejig provides read-after-write consistency for several reasons. First, a CMI impacted by a configuration change does not process a request by a client that does not have a GlobalCfgID pertaining to either this configuration change or a more recent one. Second, a client must update its configuration when a CMI provides a "Refresh & Retry" response to the client request, increasing the response time of this request. This increase is dictated by the amount of time required for the CMI to transmit the new configuration to the client. Third, a CMI always updates its GlobalCfgID to the latest GlobalCfgID provided by a client or the coordinator. Fourth, a cache entry that observes a hit is valid only if its configuration id[4] V_{cid} is more recent than the configuration that changed its fragment's assignment. The latter ensures replicated cache entries from a previous configuration (that have not yet been evicted and are potentially stale) are discarded. See Appendix A for a formal proof.

2.3 CMI Discarding Stale Cache Entries

Thus far, a Rejig client discards cache entries identified as potentially stale. A CMI may report a miss instead of transmitting a potentially stale cache entry to a client. To realize this, Rejig's client component is extended to provide both $FragCfgID_k$ and GlobalCfgID with every read. After a CMI verifies that a client

[4] The configuration that inserted or updated this entry.

provided GlobalCfgID is the latest, it must verify the V_{cid} of the referenced cache entry is greater than or equal to the $FragCfgID_k$. If this is the case then it provides the cache entry to the client. Otherwise, it reports a cache miss and deletes this entry. This requires extra processing by a CMI and incurs the overhead of transmitting $FragCfgID_k$ with every read request. However, if configuration changes are the norm and discarded cache entries are large in size then this approach may save network bandwidth.

2.4 Leases

Rejig assumes an architecture that uses leases to ensure consistency of data in the presence of server failures and network partitions. These leases are managed by the coordinator of Fig. 1. A lease is similar to a lock but with a fixed lifetime [25]. A CMI may process requests referencing Fragment F_k as long as the coordinator grants it a valid lease on F_k. The CMI may contact the coordinator to renew its lease on F_k prior to its expiration. Once a lease on F_k expires, the CMI stops servicing requests referencing F_k. The coordinator then assigns F_k to other available CMIs.

Similarly, before the coordinator changes the assignment of F_k from CMI_i to CMI_j, it (1) revokes F_k's lease from CMI_i to stop it from processing requests, and (2) grants a lease on F_k to CMI_j to enable it to process requests referencing F_k. Subsequently, it changes the configuration and uses Rejig to propagate the new configuration to the clients.

Leases and Rejig serve different purposes and complement one another. Both are required to implement read-after-write consistency in the presence of network partitions and configuration changes. While leases ensure data availability in the presence of network partitions, Rejig disseminates a new configuration efficiently. In particular, Rejig enables a CMI to use its eviction policy to delete cache entries of those fragments that are no longer assigned to it (lazily as the space occupied by these entries is required).

The coordinator does not publish a new configuration that impacts the assignment of a fragment from/to a CMI that is unreachable. It waits for the lease to expire (or the network connection to be restored to issue a revoke/grant lease) prior to publishing a new configuration.

When a client references a CMI for a cache entry assigned to a fragment with an expired lease, the CMI may respond with a "*Refresh & Retry*" response. This causes the client to look up the CMI for the latest configuration. If the CMI reports a miss, the client contacts the coordinator for the latest configuration. If no CMI is assigned to this fragment then the coordinator selects the least loaded CMI, assigns the lease on the fragment to it, increments its GlobalCfgID and computes a new configuration, updates the GlobalCfgID of the CMI, inserts the latest configuration in the CMI, and provides the client with the latest configuration.

2.5 Replication for High Availability

A system may construct R replicas of a fragment to enhance data availability in the presence of CMI failure(s). Below, we describe two popular approaches to maintain these replicas consistent. For each, we describe how a failure is represented as a configuration change and supported by Rejig.

The first approach requires (1) a read action to obtain r Shared (S) leases prior to reading the value of a replica and (2) a write action to obtain ω eXclusive (X) leases prior to writing all replicas [11]. While S leases are compatible, X leases conflict with one another and S leases. Conflicts cause the leases to race with the loser backing off and retrying. For read-after-write consistency, $r + \omega$ must be greater than R. With this approach, failure of a CMI is a configuration change that removes one replica of a fragment, decrementing R by one.

The second approach designates one copy of a fragment as primary and the other $R - 1$ as secondaries [22,39]. All reads and writes are processed by the primary fragment. The primary is responsible for propagating updates to the secondaries in the same order it receives them. If the primary fails, the coordinator promotes one of its secondaries to become the primary. With this design, a configuration change reflects promotion of a secondary to the primary and demotion of the primary to be a secondary. The coordinator increments GlobalCfgID. The value of FragCfgID_k for the promoted secondary is left unchanged. If this new primary buffers changes for the demoted primary, then the value of FragCfgID_k for the demoted primary is also left unchanged. Should the coordinator decide to discard the fragment of the demoted primary, then it simply sets its FragCfgID_k to the latest value of GlobalCfgID.

Coordinator inserts the new configuration into $R-1$ secondaries and updates their GlobalCfgID. A client that fails to issue a request to the failed primary CMI may contact one of the secondary CMIs for the latest configuration. If it observes a miss then the client contacts the coordinator for the latest configuration piggybacking the identity of the unavailable CMI. Hence, clients discover failed CMIs and report them to the coordinator.

2.6 Overhead

Rejig's overhead is in the form of storage space and network bandwidth. Storage overhead include (1) a 4-byte configuration id associated with a cache entry, (2) a configuration cache entry in a CMI. Network overhead include transmission of (1) a 4-byte client configuration id attached to each request, (2) at most two round-trips per client to get the latest configuration (clients first retrieve the configuration from a CMI and, if not found, fetch the configuration from the coordinator), (3) α roundtrips per client to insert a new configuration into CMIs.

We quantify Rejig's overhead based on the statistical models of Facebook's production key size and value size [5,19]. Facebook's mean key size is 35 bytes and the mean value size is 329 bytes, resulting in the average entry size of 364 bytes. Hence, the average storage overhead of the configuration id associated

with a cache entry is 1% (4/364). With a configuration consisting of F = 5000 fragments and 12-byte metadata of each fragment, the configuration size is 60,000 bytes. This is equivalent to 165 (60,000/364) of Facebook's cache entries. Rejig's network overhead is 11.5% (4/35) for get and delete requests, and 1% (4/364) for SET requests with Facebook's cache entries.

3 Implementation

We implemented a prototype of Rejig using memcached's Whalin client [41]. Its client library is implemented with around 500 lines of Java code. Client interfaces to communicate with a CMI is unchanged.

We implemented Rejig's coordinator using Google RPC [24] with 800 lines of Java code. The coordinator increases or decreases the number of CMIs based on the system load and adjusts the assignment from fragments to CMIs with the goal to ensure each CMI receives a similar number of fragments.

We extended IQ-Twemcached [21] (the Twitter extended version of memcached with Inhibit and Quarantine leases) to store GlobalCfgID and associate each cache entry with a configuration id. We extended standard APIs, e.g., get, set, and delete, to accept an optional configuration id. We also added a new API in IQ-Twemcached to allow the coordinator to update a CMI's configuration id. The extension to the IQ-Twemcached only requires 40 lines of C code.

4 Evaluation

We answer the following questions in this section: *(1) How fast can Rejig disseminate a new configuration? (2) How much stale data does Rejig prevent? (3) Does Rejig impact cache hit rate in the presence of graceful and drastic configuration changes?*

Factors that impact Rejig's dissemination rate are the number of clients and CMIs. CMIs are passive providers of a new configuration. Once all CMIs receive a new configuration, all clients receive the new configuration as soon as they issue a request to a CMI. The load imposed on the coordinator and the number of configuration insertions to CMIs are an interplay of α, Rejig and its variants, and the duration of time a CMI caches a configuration. In the worst case scenario, the size of a configuration cache entry is larger than the available memory of each CMI, causing the load imposed on the coordinator to equal the number of clients. In the best case scenario, each CMI has sufficient memory to cache the configuration and never evicts it. In this case, a larger α expedites dissemination of the configuration to all CMIs, reducing the number of clients that fetch the configuration from the coordinator. Rejig^C bounds the number of configuration fetches from the coordinator to be the number of CMIs. This is because a client must discover a new configuration from a CMI and fetches the configuration from the coordinator before the client propagates the new configuration to other CMIs. Rejig^T bounds the number of repeated configuration insertions in CMIs to

the number of clients since a client terminates the insertion once it encounters a CMI with a copy of the configuration. Section 4.1 quantifies Rejig's dissemination rate.

We use Azure virtual machine event traces [10] and WorldCup '98 request traces [4] to demonstrate the number of stale data Rejig prevents and its impact on the cache hit rate. Azure trace exhibits graceful configuration changes while WorldCup '98 trace exhibits drastic configuration changes with a diurnal pattern.

Lastly, we use YCSB [8] benchmark to evaluate the performance impact of Rejig, namely, tagging a request with the client's configuration id and returning a cache entry's configuration id along with its value upon a cache hit. Our evaluation shows that Rejig's network overhead is insignificant. The average read latency increases by less than 2% with Rejig when compared to without it.

Rejig can also be extended to support diverse migration techniques [27,33,36,42]. In its current format, once the coordinator assigns a fragment to a different CMI, the new CMI starts with an empty replica of the fragment. If migration is enabled, a cache miss on the new CMI migrates the cache entry from the original CMI.

Main lessons are as follows:

- Rejig disseminates a new configuration to all clients and CMIs efficiently and quickly.
- Collaborative dissemination reduces the number of clients that contact the coordinator for the latest configuration significantly.
- Rejig preserves read-after-write consistency in the presence of configuration changes by detecting and discarding all consistency violations.
- In all experiments, the cache hit rate remains high even though a configuration change does not migrate cache entries. With the trace driven analysis, the cache hit rate is always higher than 99%.
- With $\alpha \geq 1$, if the objective is to minimize the number of configuration fetches from the coordinator then Rejig is superior to the alternatives shown in Table 2. On the other hand, if the objective is to minimize the number of repeated configuration insertions into CMIs then Rejig^C is a superior technique. Rejig^C is superior to Rejig^T.

Below, we describe experiments in turn.

4.1 Scalable Configuration Dissemination

We design a microbenchmark to evaluate Rejig's configuration dissemination. It consists of 100 CMIs, a fixed number of clients, and one coordinator. There are 5000 fragments, F = 5000. We use Facebook's published cache entry size of 364 bytes [5]. The size of a configuration is 60,000 bytes, F×12; twelve bytes assuming 8 bytes for CMI address + 4 bytes for FragCfgID. For experiments of Sects. 4.1 and 4.1, we assume the total available memory is greater than the database size and there are no evictions. We consider limited memory in Sect. 4.1.

An experiment performs a sequence of iterations. In each iteration, each client issues a read for a randomly selected key K_i assigned to CMI_i,

i = Config[h(K_i)].CMI. The experiment starts with the coordinator publishing a new configuration that assigns a fragment from a source CMI to a destination CMI. The coordinator inserts the new configuration into the source CMI. The experiment terminates when all clients and CMIs have the latest GlobalCfgID and configuration. We report the number of iterations required for Rejig to disseminate a configuration change.

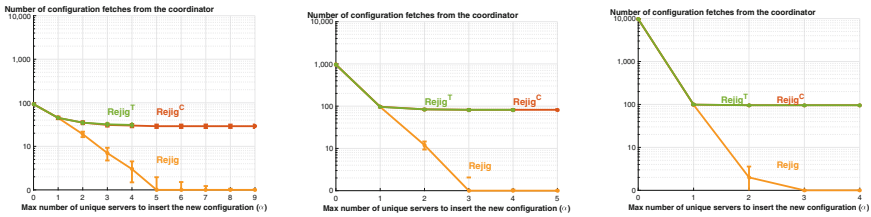
Configuration Dissemination Rate. The number of clients impacts how fast Rejig disseminates a configuration change. As we increase the number of clients from 100 to 1,000 and 10,000, Rejig requires 12 ± 1.7, 5 ± 0.5, 4 ± 0 (mean $\pm$ standard deviation) iterations, respectively. This is also an upper bound on the number of client configuration insertions into a CMI. For a CMI to receive the latest GlobalCfgID, a client with the latest GlobalCfgID must issue a request to it. With a larger number of clients, a larger number of requests are issued to CMIs in each iteration. This increases the likelihood of clients referencing all CMIs in an iteration, causing the experiment to terminate with fewer iterations. The reported number of iterations is orthogonal to the value of α because a CMI always updates its configuration id if the client provided one is more recent.

The number of configuration fetches from the coordinator depends on the value of α. With $\alpha = 0$, it is approximately the same as the number of clients: 93 ± 2.2, 962 ± 5.4, 9745 ± 16.3 with 100, 1000, and 10,000 clients, respectively. The explanation for this is as follows. Once the coordinator publishes the new configuration in one CMI, say CMI_i, it also notifies this CMI of the new GlobalCfgID. A client that references CMI_i and obtains the latest GlobalCfgID will subsequently reference another CMI_j and cause its GlobalCfgID to reflect the latest *without* providing it with the latest configuration. Those clients that reference this CMI_j observe a cache miss for the configuration and fetch the configuration from the coordinator. The number of configuration fetches is slightly less than the number of clients because those that observe a hit for the configuration using CMI_i do not contact the coordinator.

Next section discusses $\alpha \geq 1$.

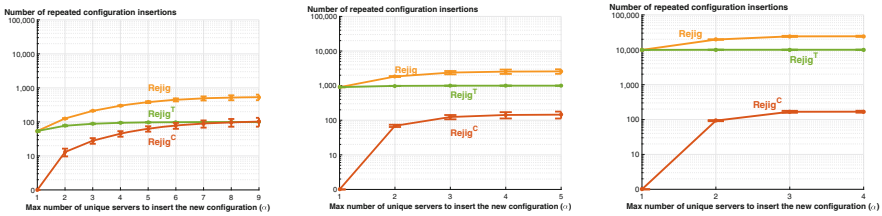
Collaborative Dissemination, $\alpha \geq 1$. In a second experiment, we evaluate the impact of requiring a client to insert a new configuration in its next unique $\alpha \geq 1$ referenced CMIs. This is the standard Rejig. We consider its variants Rejig^C and Rejig^T, see Table 2. Results of this section show Rejig^C is superior to Rejig^T.

Figure 3 shows the number of configuration fetches with these variants as a function of α. With Rejig, the number of configuration fetches from the coordinator drops 100 folds with $\alpha = 1$ and 10,000 clients when compared with $\alpha = 0$, see Fig. 3(c). When a client inserts its configuration in the next $\alpha = 1$ unique CMI it visits, other clients that reference CMI_j and observe a "Refresh & Retry" reply will now observe a cache hit for the configuration. Hence, they will *not* fetch the configuration from the coordinator. The number of configuration fetches from the coordinator continues to drop as we increase the value of α.



(a) 100 clients. (b) 1,000 clients. (c) 10,000 clients.

Fig. 3. The impact of α on the number of configuration fetches from the coordinator.



(a) 100 clients. (b) 1,000 clients. (c) 10,000 clients.

Fig. 4. The impact of α on the number of repeated configuration insertions in CMIs.

With Rejig and $\alpha \geq 4$, there are almost no client fetches from the coordinator. Clients and CMIs facilitate propagation of the configuration among themselves without further involvement of the coordinator.

With Rejig^C, the number of configuration fetches from the coordinator is higher than 10 as we increase α beyond 4. Rejig^C requires a client to insert a configuration only if it fetches the configuration from the coordinator. Hence, it is less aggressive in spreading the latest configuration when compared with Rejig. This increases the likelihood of a client observing a miss for a configuration in a CMI. Hence, the number of configuration fetches from the coordinator is higher.

A larger α causes two or more clients to insert the same configuration into the same CMI repeatedly. Figure 4 shows the total number of repeated insertions with different variants. Rejig^C performs the fewest repeated insertions because it is the least aggressive. The standard Rejig performs the most because every client that obtains the latest configuration (either from the coordinator or a CMI) will insert into the next α unique CMIs. Rejig^T is moderately aggressive by requiring a client to terminate configuration insertion once a CMI reports that it has the configuration.

These results show Rejig^C is superior to Rejig^T for two reasons. First, it performs significantly fewer configuration insertions than Rejig^T, see Fig. 4. Second, its overall number of configuration fetches from the coordinator is comparable, see Fig. 3.

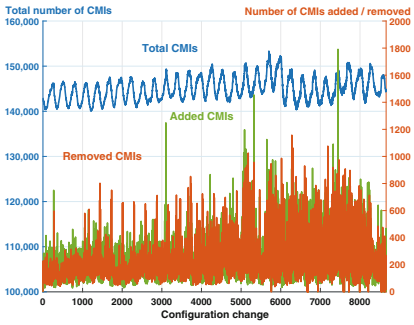
Worst Case Scenario. Consider a worst case scenario where the size of a configuration equals the amount of memory assigned to each CMI. (This is highly unlikely because the size of a configuration is in the order of hundreds of kilobytes with thousands of nodes and memory sizes are typically much larger.) Thus, the coordinator's insertion of a configuration evicts all cached entries of that CMI. Similarly, a client that inserts a cache entry upon a cache miss will evict the configuration cache entry. With $\alpha = 0$, almost all clients (9998) fetch the configuration from the coordinator. The coordinator populates at least one CMI and this copy is fetched by 1 or 2 clients. As we increase α, the number of configuration fetches drops dramatically; 9998 ± 0.15, 2197 ± 133.93, 110 ± 10.39 and 24 ± 5.26 for α values of 0, 1, 2 and 3, respectively. This is because a client's insertion of the configuration in a CMI may observe a reference by another client prior to an entry's insertion that evicts the configuration. The likelihood of this hit increases with a larger values of α, reducing the number of configuration fetches from the coordinator.

4.2 Trace Driven Evaluation

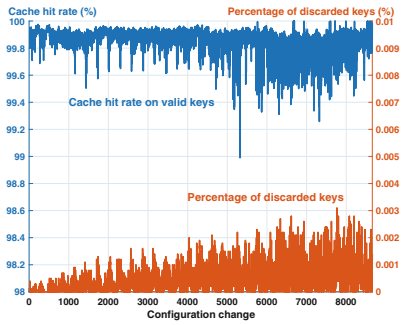
We use two traces to evaluate Rejig: Azure virtual machines (VM) trace [10] and 92 days of WorldCup 1998 request trace [4]. The first trace provides a dynamic addition and removal of VMs. We augment it with a database and use its trace to emulate addition and removal of CMIs from a configuration. WorldCup 1998 provides request traces (HTTP GET/POST on a page) with approximately 1.3 billion requests. It exhibits drastic workload fluctuations. We augment it with an auto-scaling framework that adjusts addition (Add_i) and removal (Remove_i) of CMIs based on the imposed load.

With both traces, when Add_i new CMIs are inserted in a configuration, the coordinator assigns fragments of existing CMIs to the new CMIs until the number of fragments per CMI is approximately the same. No data is migrated. Similarly, when Remove_i CMIs are removed, the coordinator assigns the orphaned fragments to other CMIs until the same number of fragments are assigned to each CMI. Once again, no data is migrated.

In all experiments, the values stored in a CMI are known and the workload generator can verify the correctness of the fetched values. We establish the following metrics: (1) the cache hit rate, (2) the number of discarded keys, and (3) the percentage of discarded keys. The percentage of discarded keys highlights the percentage of read requests that may observe stale entries. If a client produces a discarded key as an output, then the read request may violate read-after-write consistency. The number of invalid discarded entries highlights how many read requests violate read-after-write consistency with a system that does not use our techniques. Rejig eliminates these stale entries. We describe each trace and obtained results in turn.



(a) Added and removed CMIs.



(b) Hit rate and percentage of discarded keys.

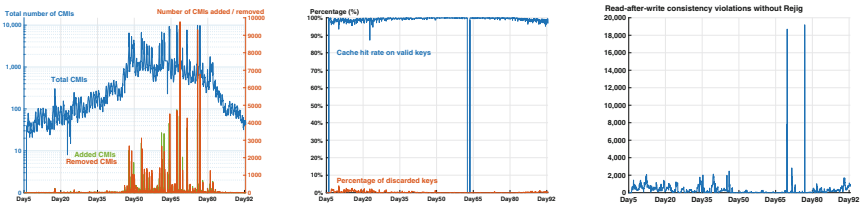
Fig. 5. Azure trace.

Azure VM Trace. Azure virtual machines (VM) trace [10] provides a representative subset of the first-party Azure VM workload in one geographical region. It monitors VMs in a consecutive 30-day period and contains a total of 2 million VMs. It provides the exact lifetime of each virtual machine with approximately 145,000 VMs running at a time. This trace has no data set. Hence, we generate a database with 10 million key-value records partitioned into 100,000 fragments for our evaluation using this trace. Before a configuration change, we randomly select 20% of entries and update their values. Since a configuration change does not migrate data, copies of these entries in the impacted CMIs are now stale.

Figure 5(a) shows the number of configuration changes we extracted from the Azure trace. It shows both the number of added CMIs, removed CMIs, and total CMIs per configuration change. Figure 5(b) shows both the observed cache hit rate and the percentage of keys discarded by Rejig with each configuration. The cache hit rate remains higher than 99% even though a configuration change may add 1800 CMIs. The variation in cache hit rate is higher with a larger number of CMIs either added or removed from a configuration. This is expected because a higher number of fragments are assigned to different CMIs that results in more cache misses. Note the percentage of discarded keys is low (less than 0.003%) in this experiment. Among all discarded keys, 99% are stale. These are read-after-write inconsistencies prevented by Rejig.

WorldCup 1998 Trace. WorldCup 1998 [4] exhibits a diurnal pattern and drastic workload fluctuations. It contains 92 days of request traces at the granularity of seconds. A large percentage (99.98%) of its approximately 1.3 billion requests are reads. Peak system load observes 10 million requests per hour. The traces reference 89,997 unique keys. We start the simulation from Day 5 (The first four days have no data).

We use the following auto-scaling framework with this trace. We assume the peak processing capacity of a CMI is $C = 1000$ requests per second. We define



(a) Added, removed, and total CMIs. (b) Hit rate and percentage of discarded keys. (c) Rejig avoided read-after-write violations.

Fig. 6. WorldCup 1998 trace.

the imposed system load as the number of requests at a given time, Load_i. The number of required CMIs is a function of these two parameters, the number of CMIs to be added $\text{Add}_i = \max(\frac{\text{Load}_i}{C} - \text{S}_{i-1},\ 0)$ and the number of CMIs to be removed $\text{Remove}_i = \max(\text{S}_{i-1} - \frac{\text{Load}_i}{C},\ 0)$. These ensure the caching layer has sufficient number of CMIs to handle Load_i at a given hour i. This auto-scaling is performed every hour of the trace for its entire 92 days.

Figure 6(a) reports the total number of added and removed CMIs per hour. As expected, more CMIs are added during the daytime. They are removed during the nighttime. The resulting drastic configuration change causes the cache hit rate to fluctuate as we do not migrate data, see Fig. 6(b). However, the percentage of discarded keys remains low. The 0% cache hit rate on Day 60 is because there is no data for a few hours on that day. The number of read-after-write consistency anomalies avoided by Rejig is still significant even though the update ratio is only 0.02%, see Fig. 6(c).

5 Related Work

Existing work [20,21,33] on cache augmented database systems focus on eliminating or minimizing inconsistency between the caching layer and the data store. IQ-framework [21] uses leases to ensure a cache entry's replica in the caching layer is consistent with its replicas in the data store. Gumball [20] minimizes inconsistency by rejecting reads and writes referencing a cache entry for a certain duration after the entry is deleted. Facebook [33] sets this duration to two seconds. Rejig complements these systems by preserving consistency in the presence of configuration changes.

The CAP theorem [23] states that a distributed data store must choose two out of the three properties: consistency, availability, and partition tolerance. Rejig does not address these properties directly. Rejig is a scalable online algorithm for cache server configuration changes. Rejig assumes an architecture that uses heartbeat messages to detect network partitions and leases to re-claim fragments assigned to unreachable nodes. While it strives to preserve consistency, one may use Rejig with an architecture that uses eventual consistency.

Two-phase commit [37] based systems (e.g., Sinfonia [2]) and Paxos [29] based systems (e.g., Spanner [9]) ensure consistency between multiple replicas. Rejig does not ensure consistency across all replicas. Instead, it detects and discards stale cache entries caused by configuration changes.

Rejig's configuration dissemination protocol is inspired by Demers et al.'s work [12] on epidemic algorithms where all sites of a data store actively participate in propagating an update made by one site. Rejig propagates a configuration made by the coordinator using both clients and CMIs. Its CMIs are passive entities that respond to client requests. Its clients actively propagate configuration to other clients using CMIs. Rejig's dissemination protocol is also applicable to a data store.

Rejig's configuration management is inspired by previous work, e.g., Google File System [22], Hyperdex [13], and Slicer [1]. Rejig is unique because it is designed for a caching solution and not a data store. With Rejig, there is a permanent copy of data elsewhere and loss of cached data does not result in data loss. Another novel feature of Rejig is that it is intended for frameworks that allow for stale cache entries to exist. Rejig detects these entries by storing a configuration id with each cache entry and its fragment, see Sect. 2. This concept is missing from prior work. Below, we provide an overview of each related system and how Rejig is different.

Google File System (GFS) [22] is a distributed file system. A GFS file contains a list of chunks. Each chunk is associated with a chunk version number. The master maintains the latest chunk version number for each chunk. Once a chunk server recovers from a failure, the master detects a stale chunk if its version number is less than the master's chunk version number. Rejig stores configuration id with each cache entry and fragment to detect stale cache entries.

Hyperdex [13] is a novel distributed key-value store that supports index structures on more than one attribute. Similar to Rejig, it employs a coordinator that manages its configuration with a strictly increasing configuration id. Upon a configuration change, the coordinator increments the configuration id and distributes the latest configuration to **all** servers. Both Hyperdex servers and its clients cache the configuration id. A client embeds its local configuration id on every request to a server and discovers its cached configuration is stale if the id does not match the server's configuration id. Hyperdex is a data store and prevents stale values for data items. Rejig is for a caching environment where stale entries may exist. It assigns a configuration id to every cache entry and uses this information to detect stale entries and discard them to provide read-after-write consistency.

Slicer [1] is Google's general purpose sharding service that is transparent to its applications. It maintains assignments using generation numbers (equivalent to Rejig's GlobalCfgID). Slicer employs leases to ensure that a key is assigned to one slicelet (equivalent to Rejig's CMI) at a time. Applications are unavailable for at most 4 s during an assignment change due to updating leases to reflect the latest generation number (and assignment) to slicelets. Also, Slicer must provision resources for a large number of distributors to disseminate a new assignment to

clients and slicelets. Rejig is designed for caches and is different in several ways. First, while its CMIs may cache a copy of configuration (Slicer's assignment), they do not use it to decide whether to process a client request or not. CMIs use GlobalCfgID (Slicer's generation number) for this purpose. Moreover, Rejig requires its coordinator to update impacted CMIs only and employs both clients and CMIs to participate in distributing a new configuration. Finally, Slicer does not either prevent or detect stale data. Rejig is novel because it detects stale cache entries by storing the configuration id with each entry and fragment.

6 Conclusion

Rejig is a scalable online algorithm for cache server configuration changes that preserves read-after-write consistency. It does not require deletion of cached entries impacted by a configuration change, leaving them to be evicted by the cache replacement technique. It serves as the building block for a fragment re-organization algorithm to balance system load, an auto-scaling framework that grows and shrinks the size of a caching layer, a data availability technique that re-assigns fragments in response to network partitions, and a persistent caching layer [18] that must recover cached entries after a failure.

Acknowledgments. We thank the anonymous reviewers of the TLDKS journal for their valuable comments and helpful suggestions.

Appendix

A Proof for Read-After-Write Consistency

This section presents a formal proof for read-after-write consistency with Rejig.

Theorem 1. *Rejig preserves read-after-write consistency for a cache entry represented as (K, V) mapped to a fragment* F_i *across N configurations* $\mathrm{Config}_i, i \in [1, N]$.

The coordinator creates F_i at Config_1. At a configuration Config_p, $p \in [1, N]$, the fragment F_i is assigned to a cache instance $\mathrm{CMI}_{i,p}$. At configuration Config_1, the coordinator's global configuration id is one and F_i's configuration id is also one. Initially, (K, V) does not exist in $\mathrm{CMI}_{i,1}$.

Lemma 1. *Rejig preserves read-after-write consistency for a cache entry (K, V) if* F_i*'s assigned* CMI *remains the same from* Config_1 *to* $\mathrm{Config}_j, j \in [1, N)$, *i.e.,* $\forall p, p \in [1, j], \mathrm{CMI}_{i,p} = \mathrm{CMI}_{i,1}$.

Proof. Since F_i remains on $\mathrm{CMI}_{i,1}$, its fragment id remains one for all configuration changes from 1 to j. Every entry (K, V) is tagged with the configuration id $\mathrm{V}_{\mathrm{cid}}$ that sets its value. When a write inserts or updates K (belonging to $\mathrm{CMI}_{i,1}$) at a configuration 1 to j, its $\mathrm{V}_{\mathrm{cid}}$ is set to the configuration id of $\mathrm{CMI}_{i,1}$. A read is able to consume (K, V) because its $\mathrm{V}_{\mathrm{cid}}$ is greater than or equal to 1, the configuration id of F_i. ■

Corollary 1. *Rejig preserves read-after-write consistency for a cache entry (K, V) in* $\text{CMI}_{i,j}$ *at* Config_j *if K does not exist in* $\text{CMI}_{i,j}$ *initially.*

Lemma 2. *Rejig preserves read-after-write consistency for a cache entry (K, V) if* F_i*'s assigned CMI changes for the first time at* Config_{j+1}*, i.e.* $\text{CMI}_{i,j} \neq \text{CMI}_{i,j+1}, \forall p, p \in [1, j], \text{CMI}_{i,p} = \text{CMI}_{i,1}$.

Proof. According to Lemma 1, Rejig preserves read-after-write consistency for K from Config_1 to $\text{Config}_j, j \geq 1$. During the configuration change from Config_j to Config_{j+1}, we have

1. the coordinator changes $\text{CMI}_{i,j}$'s configuration id to $j + 1$
2. a client's local configuration id is still $c, c \leq j$.

A client request that references K is directed to $\text{CMI}_{i,j}$. $\text{CMI}_{i,j}$ rejects the request since $j + 1 > c$. Then, the client fetches the latest configuration and issues its request to $\text{CMI}_{i,j+1}$. At configuration Config_{j+1}, Corollary 1 shows that Rejig preserves read-after-write consistency. ■

Lemma 3. *Rejig preserves read-after-write consistency for a cache entry (K, V) at* Config_q *if* $\exists o, p, q, o < p < q \leq N, \text{CMI}_{i,o} = \text{CMI}_{i,q}$ *and* $\text{CMI}_{i,o} \neq \text{CMI}_{i,p}$.

Proof. At configuration Config_q, there are two cases:

Case I: If (K, V) is inserted in $\text{CMI}_{i,o}$ at Config_o, updated at Config_p, and still exists in $\text{CMI}_{i,q}$ at Config_q. Since $\exists o, p, q, o < p < q \leq N, \text{CMI}_{i,o} = \text{CMI}_{i,q}$ and $\text{CMI}_{i,o} \neq \text{CMI}_{i,p}$, F_i's configuration id at $\text{Config}_q > \text{F}_i$'s configuration id at $\text{Config}_p > \text{F}_i$'s configuration id at Config_o. Then, the configuration id associated with (K, V) in $\text{CMI}_{i,q}$ must be lower than F_i's configuration id at Config_q. A read request that references K at Config_q discards the entry.

Case II: If K does not exist in $\text{CMI}_{i,q}$, Corollary 1 proves Rejig preserves read-after-write consistency. ■

B Physical Representation of a Configuration

A configuration consists of F fragments where F may be significantly larger than the number of CMIs. It is undesirable to repeat the CMI's IP address and port number as it increases the size of the configuration and its serialized representation, and time to serialize and deserialize a configuration. One approach to address this is to maintain the IP address and port number of each CMI in a separate array. Each element of a configuration representing a fragment stores index of the array element corresponding to its assigned CMI. Representing a CMI array element as a short (two bytes) accommodates a maximum of 65,536 CMIs. With 1000 fragments assigned to the same CMI, the memory footprint would be 2000 bytes. This is more compact than repeating IP and port numbers a thousand times. While it makes the software to serialize and deserialize a configuration more complex, it reduces network transmission time of a configuration.

B.1 Configuration Changes that Modify the Number of Fragments

A re-assignment algorithm may break a fragment into q fragments or merge p fragments into one fragment, changing the number of fragments F. These are trivial with range partitioning because they translate into breaking a sub-range into q sub-ranges and merging p adjacent ranges into one, respectively. In its extreme, breaking a fragment may result in sub-ranges that correspond to points. Each point may consist of only one data item. This is justified when the data item is extremely hot [38].

With hash partitioning, when a hash function depends on the value of F, breaking and merging of fragments must be done in a manner that is consistent with the hash function. As an example, assume a simple mod function as the hash function, $h(K_i) = K_i \% F$. Incrementing (or decrementing) the value of F by one would re-assign key-value pairs across **all** fragments. Rejig does not support these configuration changes. (Rejig supports re-assignment of fragments to CMIs only.) To use Rejig, one must modify the value of F in a manner that changes the assignment of key-value pairs for the impacted fragment only. This would be similar to extendible [14] and linear [30] hashing algorithms. For example, with the mod hash function, to break a fragment into two, F should double. This would generate a buddy for each existing fragment. The buddy of a fragment is assigned to the same CMI as the fragment. The buddy of the fragment that is broken into two is assigned to a different CMI. To merge two fragments into one, we would change the assignment of its buddy to be the same CMI as the fragment. Subsequently, we scan the array to detect if a fragment and its buddy are assigned to the same CMI. If this is the case then we halve F, $F = \frac{F}{2}$.

References

1. Adya, A., et al.: Slicer: auto-sharding for datacenter applications. In: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, pp. 739–753. USENIX Association, Savannah (2016)
2. Aguilera, M.K., Merchant, A., Shah, M., Veitch, A., Karamanolis, C.: Sinfonia: a new paradigm for building scalable distributed systems. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP 2007, pp. 159–174. ACM, New York (2007)
3. Annamalai, M., et al.: Sharding the shards: managing datastore locality at scale with Akkio. In: 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, pp. 445–460. USENIX Association, Carlsbad (2018)
4. Arlitt, M., Jin, T.: A workload characterization study of the 1998 world cup web site. Netw. Mag. Global Internetwkg. **14**(3), 30–37 (2000)
5. Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M.: Workload analysis of a large-scale key-value store. In: SIGMETRICS, pp. 53–64. ACM, New York (2012)
6. Beckmann, N., Chen, H., Cidon, A.: LHD: improving cache hit rate by maximizing hit density. In: 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, pp. 389–403. USENIX Association, Renton (2018)

7. Cidon, A., Eisenman, A., Alizadeh, M., Katti, S.: Cliffhanger: scaling performance cliffs in web memory caches. In: 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, pp. 379–392. USENIX Association, Santa Clara (2016)
8. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 143–154. ACM, New York (2010)
9. Corbett, J.C., et al.: Spanner: Google's globally-distributed database. In: 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, pp. 261–264. USENIX Association, Hollywood (2012)
10. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles, SOSP 2017, pp. 153–167. ACM, New York (2017)
11. DeCandia, G., et al.: Dynamo: Amazon's highly available key-value store. In: SOSP (2007)
12. Demers, A., et al.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 1987, pp. 1–12. ACM, New York (1987)
13. Escriva, R., Wong, B., Sirer, E.G.: HyperDex: a distributed, searchable key-value store. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM 2012, pp. 25–36. ACM, New York (2012)
14. Fagin, R., Nievergelt, J., Pippenger, N., Strong, H.R.: Extendible hashing - a fast access method for dynamic files. ACM Trans. Database Syst. **4**(3), 315–344 (1979)
15. T. A. S. Foundation. Apache Ignite (2018). https://ignite.apache.org/
16. Ghandeharizadeh, S., Almaymoni, M., Huang, H.: Rejig: a scalable online algorithm for cache server configuration changes. In: Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, p. 513. ACM, New York (2018)
17. Ghandeharizadeh, S., et al.: A demonstration of KOSAR: an elastic, scalable, highly available SQL middleware. In: Proceedings of the Posters & Demos Session, Middleware Posters and Demos 2014, pp. 23–24. ACM, New York (2014)
18. Ghandeharizadeh, S., Huang, H.: Gemini: a distributed crash recovery protocol for persistent caches. In: Proceedings of the 19th International Middleware Conference, Middleware 2018, pp. 134–145. ACM, New York (2018)
19. Ghandeharizadeh, S., Huang, H.: Hoagie: a database and workload generator using published specifications. In: Second IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications, December 2018
20. Ghandeharizadeh, S., Yap, J.: Gumball: a race condition prevention technique for cache augmented SQL database management systems. In: Proceedings of the 2nd ACM SIGMOD Workshop on Databases and Social Networks, DBSocial 2012, pp. 1–6. ACM, New York (2012)
21. Ghandeharizadeh, S., Yap, J., Nguyen, H.: Strong consistency in cache augmented SQL systems. In: Proceedings of the 15th International Middleware Conference, Middleware 2014, pp. 181–192. ACM, New York (2014)
22. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 29–43. ACM, New York (2003)
23. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002)

24. Google: Google Protocol Buffer (2018). https://developers.google.com/protocol-buffers
25. Gray, C., Cheriton, D.: Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In: Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, SOSP 1989, pp. 202–210. ACM, New York (1989)
26. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: ZooKeeper: wait-free coordination for internet-scale systems. In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, USENIXATC 2010, p. 11. USENIX Association, Berkeley (2010)
27. Hwang, J., Wood, T.: Adaptive performance-aware distributed memory caching. In Proceedings of the 10th International Conference on Autonomic Computing, ICAC 2013, pp. 33–43. USENIX, San Jose (2013)
28. IBM: IBM WebSphere (2018). https://www.ibm.com/cloud/websphere-application-platform
29. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
30. Litwin, W.: Readings in database systems. Chapter Linear Hashing: A New Tool for File and Table Addressing, pp. 570–581. Morgan Kaufmann Publishers Inc., San Francisco (1988)
31. Lu, H., et al.: Existential consistency: measuring and understanding consistency at Facebook. In: Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, pp. 295–310. ACM, New York (2015)
32. Memcached (2018). https://memcached.org/
33. Nishtala, R., et al.: Scaling Memcache at Facebook. Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, pp. 385–398. USENIX, Lombard (2013)
34. Oracle: Oracle Coherence (2018). http://www.oracle.com/technetwork/middleware/coherence/overview/index.html
35. Redis (2019). https://redis.io/
36. Redis migrate command (2019). https://redis.io/commands/migrate
37. Skeen, D., Stonebraker, M.: A formal model of crash recfovery in a distributed system. IEEE Trans. Softw. Eng. **9**(3), 219–228 (1983)
38. Taft, R., et al.: E-store: fine-grained elastic partitioning for distributed transaction processing systems. Proc. VLDB Endow. **8**(3), 245–256 (2014)
39. van Renesse, R., Schneider, F.B.: Chain replication for supporting high throughput and availability. In: OSDI (2004)
40. Waldspurger, C., Saemundsson, T., Ahmad, I., Park, N.: Cache modeling and optimization using miniature simulations. In: 2017 USENIX Annual Technical Conference, USENIX ATC 2017, pp. 487–498. USENIX Association, Santa Clara (2017)
41. Whalin, G., Wang, X., Li, M.: Memcached Whalin Client (2018). https://github.com/gwhalin/Memcached-Java-Client
42. Zhu, T., Gandhi, A., Harchol-Balter, M., Kozuch, M.A.: Saving cash by using less cache. In: 4th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2012. USENIX, Boston (2012)

Author Index

Akbarinia, Reza 1
Almaymoni, Marwan 111

Freudenthaler, Bernhard 91

Ghandeharizadeh, Shahram 111
Gillet, Annabelle 51
Grison, Thierry 51

Huang, Haoyu 111

Leclercq, Éric 51

Mahboubi, Sakina 1
Martinez-Gil, Jorge 91

Pernul, Günther 25
Putz, Benedikt 25

Savonnet, Marinette 51

Tjoa, A Min 91

Valduriez, Patrick 1