2017 Imperial College Computing Student Workshop

ICCSW 2017, September 26–27, 2017, London, United Kingdom

Edited by

Fergus Leahy

Juliana Franco

ICCSW17

Editors

Fergus Leahy
Department of Computing
180 Queen's Gate, London, SW7 2AZ
United Kingdom
fergus.leahy@imperial.ac.uk

Juliana Franco
Department of Computing
180 Queen's Gate, London SW7 2AZ
United Kingdom
j.vicente-franco@imperial.ac.uk

ACM Classification 1998
D.1.3 Concurrent Programming, D.2.2 Design Tools and Techniques, D.2.8 Performance measures, D.2.11 Software Architectures, D.3.3 - Language Constructs and Features, D.4.7 Distributed systems, D.4.8 Performance, E.1 Trees, F.1.2. Models of Computation - Probabilistic Computation, F.4.1. Mathematical Logic, F.4.2. Formal Languages, G.1.6 Optimization, G.3 Probability and Statistics, H.2.4 Concurrency, H.5.2 User Interfaces, I.2.6 Learning, K.0 General, K.2 History of Computing.

ISBN 978-3-95977-059-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-059-0.

Publication date
February 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.ICCSW.2017.0

ISBN 978-3-95977-059-0 **ISSN 1868-8969** **http://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

http://www.dagstuhl.de/oasics

Contents

Keynotes

Regular Papers

Abstracts

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW17

Preface

Welcome to the 2017 Imperial College Computing Student Workshop (ICCSW'17), the sixth workshop in its series. ICCSW was initiated with a "by students, for students" spirit: a workshop organised solely by students to give student speakers the opportunity to present their work. The organising students gain the valuable experience of what is involved in conference organisation, including writing a call for sponsors, taking part in the reviewing process, and chairing a session. On the other hand, the participating students benefit from the interaction with international researchers who are at a similar stage in their career and developing skills in presenting their research to a non-specialist computer science audience.

This volume contains the papers accepted for presentation at the 2017 Imperial College Computing Student Workshop. ICCSW'17 received 12 submissions, including both papers and abstracts, from 6 different countries. After the thorough reviewing process and discussion by members of the Imperial College ACM Student Chapter 6 papers and 2 abstracts were accepted, representing a 75% acceptance rate.

After a year hiatus, ICCSW was back for more, more student talks, more keynotes, more socials and a new addition of a breakfast poster session. ICCSW'17 was a great success on all fronts, with over 30 students attending a variety of interesting and novel talks, covering systems, cloud, networking, programming languages and machine learning. This year we also hosted two exciting keynotes covering Google's V8 Javascript engine (Leszek Swirski) and How to write a great paper (Simon Peyton Jones), both of which saw upwards of 50 students and staff attend, and included some really insightful Q&A. To their merit, students arose first thing in the morning ready to be quizzed on their work at our breakfast poster session, as inquisitive wanderers chowed down on croissants, coffee and enlightening conversation. For our social event, we invited the students and our Googler to explore the Sky Garden atop the Walkie-Talkie and took them on a tour of the surrounding London sights, including the Tower of London and Tower Bridge, before settling down for 3 courses of pizza-and-pasta-riffic food at Pizza Express.

ICCSW'17 has been a great success - but absolutely could not have been done without the dedication and perserverence from the ICCSW committee, hard work and patience from the student authors, assistance from our network of ambassadors in disseminating the calls, financial support from our sponsors and the gratiutous support from the department here at Imperial; all of whom we would like to thank dearly.

We wish the best of luck to the new committee. Until next year!

Juliana Franco and Fergus Leahy,
ICCSW'17 Editors,
Chair & Vice-Chair,
ACM Student Chapter 2016 – 2017.

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW17

ICCSW'17 Social Photo

Figure 1 ICCSW visits the sky garden.

Conference Organisation

Organising Committee

Juliana Franco	Imperial College London
Fergus Leahy	Imperial College London
Kyriacos Nikiforou	Imperial College London
Simon Olofsson	Imperial College London
Mengjiao Wang	Imperial College London
Pamela Bezerra	Imperial College London
Shale Xiong	Imperial College London
Oana Cocarascu	Imperial College London
Assel Altayeva	Imperial College London
Nick Pawlowski	Imperial College London
Casper da Costa-Luis	Imperial College London
	Imperial College London ACM Student Chapter `http://acm.doc.ic.ac.uk/`

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW17

Ambassadors

Jasper Schulz	Kings College, UK
Fabio Niephaus	Hasso-Plattnner-Institut, Germany
Kiko Fernandez	Uppsala University, Sweden
Phuc Vo	Uppsala University, Sweden
Daniel Hillerstrom	University of Edinburgh, UK
Kim-Anh Tran	Uppsala University, Sweden
Stephan McQuistin	University of Glasgow, UK
Marija Jegorova	University of Edinburgh, UK
Dionysis Manousakas	University of Cambridge, UK
Ana-Maria Sutii	Eindhoven University of Technology, The Netherlands
Tatjana Davidovic	Serbian Academy of Science and Arts, Serbia
Darren Matthews	Royal Holloway, University of London, UJ

Supporters and Sponsors

Supporting Scientific Institutions

Imperial College London
http://www.imperial.ac.uk/

HiPEDS: Imperial College London
http://wp.doc.ic.ac.uk/hipeds/

Platinum Sponsor

Google Inc.
http://www.google.com/

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW17

How to Write a Great Research Paper

Simon Peyton Jones

Microsoft Research Cambridge, United Kindgom

Abstract

Writing papers is a core research skill for any researcher, but they aren't easy. Writing is not just a way to report on great research; it's a way to do great research. Yet many papers are so badly written that, even if they describe excellent work, the work has much less impact than it should. In this talk I'll give you seven simple, actionable guidelines that will, I hope, help you to write better papers, and have more fun at the same time. I don't have all the answers—far from it—and I hope that the presentation will evolve into a discussion in which you share your own insights, rather than a lecture. The slides and video presentation are available online [1].

Simon Peyton Jones, FRS, graduated from Trinity College Cambridge in 1980. After two years in industry, he spent seven years as a lecturer at University College London, and nine years as a professor at Glasgow University, before moving to Microsoft Research (Cambridge) in 1998.

1998 ACM Subject Classification K.0 General

Keywords and phrases Academia, Research, Writing

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.1

Category Keynote

[1] https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 1; pp. 1:1–1:1
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW17

Optimizing the Unoptimizable: A Whirlwind Tour of JavaScript

Leszek Swirski

Google Inc., London, United Kingdom

Abstract

A whirlwind tour through the history and state-of-the-art of JavaScript execution and optimization, with a focus on the V8 engine used by Chrome and Node.js, and how a 10-day prototype became one of the most important programming languages in the world.

Leszek Swirski has been a software engineer at Google for two years, first in California, now in London, working on the performance of the Android camera and the V8 Javascript engine. Before joining Google, Leszek did a PhD in the University of Cambridge, researching gaze estimation (a.k.a. eye tracking) on stereoscopic (a.k.a. "3D") displays.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, K.2 History of Computing

Keywords and phrases Javascript, NodeJS

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.2

Category Keynote

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 2; pp. 2:1–2:1

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICCSW17

Improving the Latency and Throughput of ZooKeeper Atomic Broadcast

Ibrahim EL-Sanosi[1] and Paul Ezhilchelvan[2]

1 **Faculty of Information Technology, Sebha University, Sebha, Libya and School of Computing, Newcastle University, Newcastle Upon Tyne, United Kingdom**
i.elsanosi@sebhau.edu.ly
i.s.el-sanosin@ncl.ac.uk

2 **School of Computing, Newcastle University, Newcastle Upon Tyne, United Kingdom**
paul.ezhilchelvan@ncl.ac.uk

Abstract

ZooKeeper is a crash-tolerant system that offers fundamental services to Internet-scale applications, thereby reducing the development and hosting of the latter. It consists of $N \geq 3$ servers that form a replicated state machine. Maintaining these replicas in a mutually consistent state requires executing an Atomic Broadcast Protocol, *Zab*, so that concurrent requests for state changes are serialised identically at all replicas before being acted upon. Thus, ZooKeeper performance for update operations is determined by Zab performance. We contribute by presenting two easy-to-implement Zab variants, called *ZabAC* and *ZabAA*. They are designed to offer small atomic-broadcast latencies and to reduce the processing load on the primary node that plays a leading role in Zab. The former improves ZooKeeper performance and the latter enables ZooKeeper to face more challenging load conditions.

1998 ACM Subject Classification D.2.8 Performance measures, D.4.7 Distributed systems

Keywords and phrases Atomic Broadcast, Server Replication, Protocol Latency, Throughput

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.3

1 Introduction

Apache ZooKeeper [5] is a high-availability system that is designed to offer several fundamental services to Internet-scale distributed applications. It is a widely used, industrial-strength system because it relieves large-scale applications from having to build fundamental services themselves. Some of the services offered by ZooKeeper include: leader election (used by Apache Hadoop [9]) and failure-detection and group membership configuration (by HBase [3]).

ZooKeeper is built as a replicated system using $N, N \geq 3$, fail-independent servers. At most $f = \lfloor \frac{N-1}{2} \rfloor$ of these N servers can crash which means that ZooKeeper can continue to provide uninterrupted services to applications as long as crashed servers are replaced and at least $f + 1$ servers are operative at any given time.

ZooKeeper uses the atomic broadcast protocol, Zab [5], to ensure that ZooKeeper servers' states and its clients are kept in a consistent state. Zab is typically composed of three to seven machines which are used for replicating data in order to achieve high availability. In ZooKeeper, one of the nodes has a *leader* role and the rest have *follower* roles. The leader is responsible for accepting all incoming state changes (write requests) from the clients and replicating them to all servers in the ensemble through Zab.

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 3; pp. 3:1–3:10

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, many leader-based protocols, including Zab, have problems associated with overloading, weak writes as well as scalability and bottleneck that occur under write-intensive workloads [2, 7, 10]. In Zab, write requests always take longer to process, as they must go through the Zab and the leader replica, which requires extra tasks to propagate the requests to all followers since three communication steps are needed to broadcast a single write request. Consequently, this can add more latency to the requests and decrease performance.

In this paper, we present two atomic broadcast protocols, *ZabAC* and *ZabAA*. ZabAC accomplishes write request in two-rounds of communication, namely the *proposal* and *acknowledgement-commit* rounds. ZabAC is similar to the Zab protocol, the different is that ZabAC executes a write in two communication steps rather than three. However, ZabAC works only in a three server ensemble. ZabAA can also accomplish a write in two communication steps, and moreover unlike ZabAC, it can utilise any ensemble size, N. We discuss these two approaches in more detail in section 3.

The remainder of the paper is structured as follows. Section 2 describes the design of Zab, an atomic broadcast protocol for the ZooKeeper coordination service. Section 3 describes the protocols we developed. Section 4 provides a thorough performance evaluation of the ZabAC and ZabAA model compared to the existing Zab approach. Section 5 discusses related work. Finally, section 6 concludes the paper and the outlook for our future research.

2 ZooKeeper Atomic Broadcast Protocol

ZooKeeper is implemented using an ensemble of N, $N \geq 3$, fail-independent and fully-connected servers. In practice, N is an odd number, typically 3-7 servers [4]. The following assumptions are made by ZooKeeper.

A1 – Crash Tolerance.

Servers can crash and at least $\frac{N+1}{2}$ servers are operational at any time. Thus, up to f, $f = \lfloor \frac{N-1}{2} \rfloor$, server crashes are tolerated.

A2 – Reliable and Source-Ordered Communication.

Servers are connected by a reliable communication subsystem in which messages are never lost and are received in the order in which there are sent. More precisely, if a server sends a message m then all operative destinations receive m within some finite time; if a server sends m_1 followed by m_2, any common destination for m_1 and m_2 will receive m_1 before m_2.

ZooKeeper servers are basically replicas of each other and each maintains a copy of the application state. A Zookeeper client can submit its request or signal an event to any server. If the processing of requests or events from clients does not involve modifying the application state, then the server will respond directly to the client without involving the other servers.

If however a client request requires modifying the application state, this will be handled by all servers in a mutually consistent manner; that is, it will be identically ordered against any concurrent requests/events received at other servers before it is processed. Ensuring identical order on concurrent requests and events is accomplished through *Zab*, the ZooKeeper atomic broadcast protocol.

Zab is an asymmetric protocol in its structure: it designates one of the ZooKeeper servers as the *leader* and the rest as *followers*. As with the well-known 2-Phase commit protocol in database transactions [1], atomic broadcasting can be initiated only by the leader and

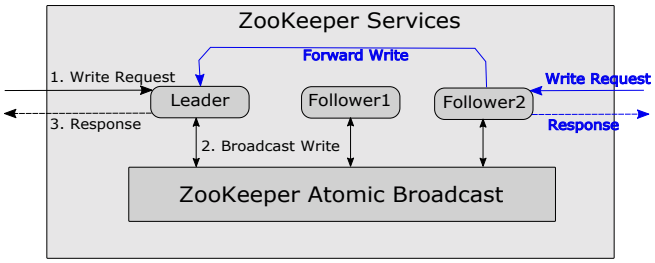


Figure 1 Write Operations in ZooKeeper.

followers respond to what they receive. Figure 1 depicts how requests and events requiring state modification are handled by ZooKeeper.

When a follower receives a write request from a client (shown in blue in Figure 1), it forwards it to the leader. Whenever the leader receives a write request that has been forwarded to it by a follower or sent to it directly by a client, it initiates a Zab execution for that request. The execution ensures that the request is delivered to all servers in the same order and only the server that received the request directly from the client returns a response.

2.1 Zab Protocol

It consists of the following steps.

- L1: Leader initiates *proposal(m)* (state change request) by proposing a sequence number $m.c$ for m and by broadcasting its *proposal(m)* to all processes, including itself;
- F1: A follower, on receiving *proposal(m)*, logs m and then sends an acknowledgement, *ack(m)*, to the leader;
- L2: Leader sends *ack(m)* to itself after logging m. On receiving *ack(m)* from a quorum, it broadcasts *commit(m)* before *commit(m′: $m'.c = m.c + 1$)* is broadcast;
- F2: A follower, on receiving *commit(m)*, delivers m.
- L3: Leader, on receiving *commit(m)* (from itself), delivers m.

2.2 Crash-Tolerance Invariant

Let Π be the set of ZooKeeper servers: $\Pi = \{p_1, p_2, ..., p_N\}$. Let $\boldsymbol{Q}$ be the set of all majority subsets or *quorums* of Π: $\boldsymbol{Q} = \{Q : Q \subseteq \Pi \wedge |Q| > f = \lfloor \frac{N-1}{2} \rfloor\}$.

For example, when $N = 3$, $\boldsymbol{Q} = \{\{p_1, p_2\}, \{p_2, p_3\}, \{p_3, p_1\}, \{p_1, p_2, p_3\}\}$.
The **invariant** is as follows: If any server delivers m_i, then all servers in some $Q \in \boldsymbol{Q}$ have logged m_i locally.

To see informally that this invariant is a requirement for crash-tolerance provisions, suppose that the leader delivers m_i and then crashes, possibly before broadcasting the commit message for m_i. Some quorum of servers, say Q', will elect the new leader and inform it of all messages proposed by leader that crashed. Suppose that the invariant holds and there is a quorum Q of servers that have logged m_i. By definition, Q and Q' must intersect. Q' cannot contain the leader that crashed. Thus, Q and Q' must have at least one server in common that is not the crashed leader. That server will instruct the new leader of the existence of m_i and of the need to complete the delivery of m_i by all followers.

3 Zab Alternatives

In this section, we present two protocols that also preserve the crash-tolerance invariant (see section 2.2). The first protocol, called *ZabAC*, works only when $N = 3$ and the second, called *ZabAA*, is developed for when $N > 3$.

3.1 ZabAC

The design of ZabAC is based on the observation that when $N = 3$, the crash-tolerance invariant is satisfied as soon as a follower locally logs the proposal received from the leader. This is because when $N = 3$, any two servers constitute a quorum and, given that the leader broadcasts its proposal only after logging it locally, the follower can deliver a proposal as soon as it logs it locally; that is, a follower does not have to wait for an explicit commit message from the leader. The letters *AC* in *ZabAC* stand for the optimisation that followers can acknowledge and commit, without having to wait for an explicit commit message from the leader.

The key stages of the ZabAC protocol are detailed below. Note that v stands for a write request.

1. **Leader Logs and Sends Atomic Broadcast** – Process a proposal $\langle v, zxid \rangle$ and broadcast it to all processes in Π.
2. **Follower Delivers a Proposal** – Receive, log a proposal $\langle v, zxid \rangle$, send an acknowledgement for $\langle zxid \rangle$ to the leader and deliver a proposal $\langle v, zxid \rangle$.
3. **Leader Delivers a Proposal** – Receive an acknowledgement for $\langle zxid \rangle$, compute a majority of ACK (acknowledgement) and deliver a proposal $\langle v, zxid \rangle$.

3.1.1 ZabAC Implementation Details

We explore the inner workings of each step of the ZabAC protocol. We describe each step in the order in which they are executed by the protocol.

1. Leader Sends Atomic Broadcast (Proposal Stage)

Prior to commencing the broadcast, a leader places a client's write operation in its Broadcast Request Pool (BRP), which holds all client write operations until they are broadcast. When BRP contains operations, a single thread, called *send thread*, is utilised for retrieving the operations from the BRP and broadcasting them to all processes in Π. Operations are retrieved from the BRP in the order in which they were originally received (FIFO). Upon retrieving an operation, the send thread creates a proposal message which includes a tuple $\langle v, zxid \rangle$ that uniquely identifies the broadcast.

Note that, before broadcasting a proposal message, the send thread places it in a list called *pending* until it receives acknowledgements from a quorum of processes. The pending list contains the proposals. Each proposal waits for a quorum of processes to send acknowledgements to the leader. In parallel, the send thread stores the proposal in *logging* list and periodically logs the list contents in persistent storage for recovery purpose.

2. Follower Delivers a Proposal (Acknowledgment and Commit Stage)

A follower, on receiving the proposal, first places it in a logging list and periodically logs the list's contents on a disk. After this, the follower must certify that the proposal has the highest zxid that has been received and precedes the last committed zxid. Since the ZabAC uses

reliable communication and FIFO when exchanging messages and the leader sends a proposal in the order according to its zxid, the proposal can always be certified, except when a crash occurs before has been certified the proposal. Once a proposal is certified, the follower, in parallel, sends an ACK message to the leader and commits the proposal, delivering it to the memory. Upon certifying a proposal, the followers deliver the proposal due to receiving ACKs from quorum of processes: a follower receives one ACK from the leader, piggybacked with the proposal, and one from itself when it acknowledges the proposal.

3. Leader Delivers a Proposal

Upon receiving an ACK, the leader delivers the proposal as it receives ACKs from a quorum of processes: it receives one from itself and one from any followers. Note that each process has a *delivered* list which stores all delivered proposals for future read requests by clients. Unlike Zab, ZabAC's leader does not need to send a commit message to the followers as each follower commits the change locally as soon as it receives ACKs from a quorum of processes. As a result we save one-third of the communication steps compared to Zab.

Moreover, there are similarities between Zab and ZabAC in the way that proposals are delivered from the perspective of the leader replica. In Zab and ZabAC, a proposal $\langle v, zxid \rangle$ is delivered as soon as the leader receives an ACK from any follower. However, ZabAC's leader does not need to process and send a commit message to Π. One major difference between Zab and ZabAC is that the followers in ZabAC always deliver a proposal before the leader does, while it is the other way round in Zab.

3.2 ZabAA

ZabAA is developed for any N and as with ZabAC it has been designed so that the leader does not have to broadcast commit messages to its followers. This is achieved by having followers broadcast acknowledgements to every server in the system (AA in $ZabAA$ stands for *Acknowledge All*). A follower commits a proposal after it (i) receives that proposal from the leader and (ii) knows that at least f followers have acknowledged that proposal. Note that (i) and (ii) ensure that the crash-tolerance invariant is preserved: committing a proposal by a follower occurs only after the leader and at least f followers have logged that proposal locally, and when any subset of $f + 1$ servers constitute a quorum in Π.

Like ZabAC, ZabAA requires two communication steps: Proposal and Acknowledgement-Commit rounds. Proposal and Commit stages remind unchanged (They are similar to ZabAC implementation). However, the number of acknowledgement messages sent between followers increases quadratically with N. Note that ZabAA does not increase the number of acknowledgements that are sent to the leader. ZabAA thus trades-off against higher message overhead for followers. The quadratic increase in follower message overhead may off-set the gain from reduced follower latencies as N increases. Yet, it is worth investigating ZabAA to study the effect of this trade-off for small values of N, particularly for $N = 5$ which is the second most typically value (after $N = 3$).

4 Experiments and Performance Evaluation

In this section, we present a comparative evaluation of Zab, ZabAC and ZabAA. We study different performance metrics: namely latency and throughput.

We used 250 simultaneous clients executed on 10 machines; with each machine operating up to 25 clients. Up to 5 machines were dedicated to run evaluated protocols, typical

ZooKeeper installations use 3-7 servers, so 5 is just smaller compared to a typical setting [5]. All machines in the experiment utilised commodity PCs of 2.80GHz Intel Core i7 CPU and 8GB of RAM, running Fedora 21 and communicating over 100 Mbps Switched Ethernet.

The evaluated protocols were implemented in Java (JDK 1.8.0) on the top of the JGroups framework; JGroups is a toolkit for reliable communication and it is used to establish a group membership where members can send messages to each other. All messages were transmitted using JGroups' reliable UDP. As well as using a reliable UDP, the JGroups' UNICAST3 protocol is used to provide node-to-node ordering as the default for each message sent. Utilising a UNICAST3 protocol provides FIFO ordering similar to a TCP protocol.

Each request consists of a read or write with 1000 bytes of maximum payload, which represents a typical request size [5]. In our experiment, 1 million requests were sent. Each client was responsible for sending $\frac{(\frac{10^6}{10 machines})}{25 clients}$ requests. To distribute the load equally on the server protocols, the clients sent the requests to the protocol in a round robin manner. Our benchmark client used the synchronous JGroups client API.

Experiments are run in failure-free scenarios. Furthermore, servers do not log a proposal message in disk (as ideally required) but only record the proposal in main-memory. Thus the performance figures we present here do not include disk write delays, but only network delays. This kind of evaluations corresponds to the 'Net-Only' category of the evaluations in [5] where several ways of logging have been considered. Since Zab and proposed protocols require logging of the proposal message exactly at the same point in the execution for every broadcast, ignoring delays due to disk writes cannot invalidate the integrity of observations made and conclusions drawn from the performance figures.

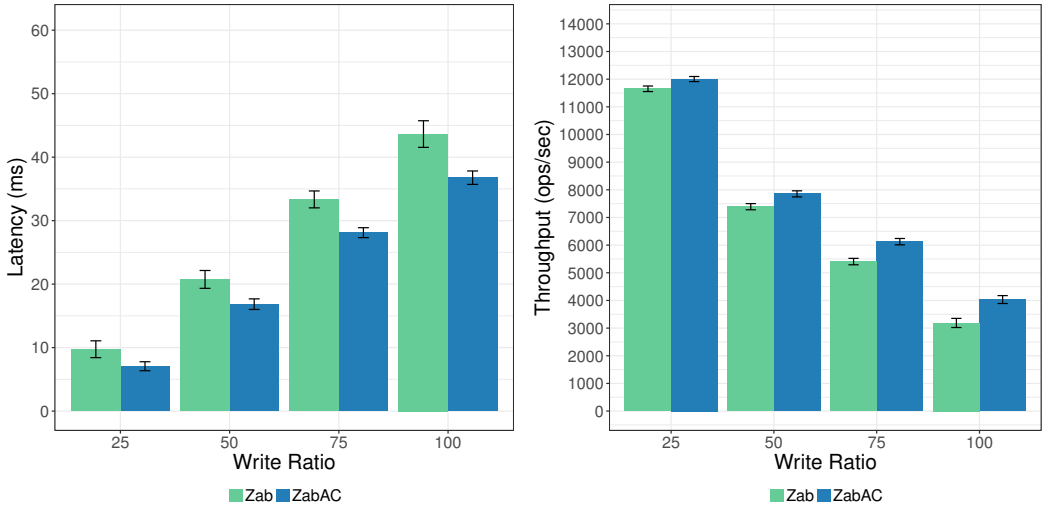
Note that the latency is defined here as $t_1 - t_0$ where t_0 is the time at which a client sends a request to a protocol's server and t_1 is the time at which the client receives a replay message from the server. So, the final latency is an average of the computed latencies for all clients. We compute the average of 1000000 such latencies and repeat the experiment 10 times for a confidence interval of 95%. Throughput is defined as the number of requests made by all servers per unit time and is computed, like latencies, with a 95% confidence interval.

4.1 Zab vs ZabAC

In this experiment, we deployed Zab and ZabAC in a three-server ensemble since ZabAC works only when $N = 3$. Figure 2a shows the latency in milliseconds (ms) of the mixed workload (the ratio of writes to reads) as the percentage of writes was increased. The figure shows that increasing the number of writes has a negative impact on the performance of Zab and ZabAC. The reason the performance is affected is because write requests must go through atomic broadcast, which requires additional processing and adds more latency to requests whereas read requests require the server to only read data from the local replica's state.

The graph also shows that the latency for ZabAC is lower than for Zab in all writes to reads ratios. For example, with a 100% writes, ZabAC's latency is approximately 37 ms whereas Zab's latency is 44 ms. This finding was expected because ZabAC has lower overheads as a result of fewer messages being broadcast generally and more specifically because it dispenses with the leader having to send commit messages to its followers.

Figure 2b shows a throughput comparison between Zab and ZabAC. Throughout the figure, ZabAC has a higher throughput compared to Zab in all cases, with the maximum difference being 846 operations per second (ops/sec) (operations size is 1000 bytes) when



(a) Latency comparison **(b)** Throughput comparison

Figure 2 Performance comparison, varying the ratio of writes to reads.

the number of writes is 100%. This is due to the fact that in ZabAC there are fewer communication patterns and less network traffic than in Zab. In other words, reducing the number of communication steps results in less computation being performed by the leader, which creates a significant throughput advantage for ZabAC.

4.2 Zab vs ZabAA

In this section, we investigate the effect of ensemble size in Zab and ZabAA.

Figures 3a and 3b show how latency varies according to the size of the ensemble and the workload (the ratio of writes to reads requests). Each figure corresponds to a different ensemble size. We can see that the larger the ensemble size, the higher latency there is in both Zab and ZabAA. This is because the leader needs to synchronize its state with a quorum of replicas, that is, the larger the ensemble size, the longer the leader has to wait before delivering a proposal (for instance, with an ensemble size of three ($N = 3$), only two acknowledgements are needed whereas with an ensemble size of five ($N = 5$) the leader must wait until it receives an acknowledgement from three replicas). Thus, increasing the ensemble size has an impact on both latency and throughput.

Moreover, each figure shows that latency increases when the workload includes more writes than reads. This could be due to the fact that write requests must go through atomic broadcast and this requires additional processing which, in turn, causes further delay, thus increasing latency.

Comparing ZabAA with Zab, we observe that ZabAA experiences lower latency than Zab for all types of workload and ensemble size. Moreover, the difference becomes more significant when the percentage of write requests increases. For example, with 100% writes and an ensemble size of three, latency is approximately 37 ms for ZabAA and 44 ms for Zab. Likewise, with 100% writes and an ensemble size of five, latency is approximately 95 ms for ZabAA and 103 ms for Zab. This difference in latency between ZabAA and Zab stems from the fact that in ZabAA only two communication steps are required to deliver a request whereas in Zab three communication stages are needed.

Unsurprisingly, when the percentage of read requests increases, small differences were found between Zab and ZabAA in term of latency, although ZabAA experiences slightly lower latency than Zab. This small difference in latency could be due to the fact that, as previously stated in section 2, reads are in-memory operations and are serviced from the local replica which means no agreement protocol needs to be run and therefore, latency are decreased and becomes less significant when comparing ZabAA with Zab.

Figure 4a and 4b shows how throughput varies with the number of servers and a mixed workload. The figures indicate that since the leader propagates a proposal to all followers, the throughput must drop as the number of servers increases. Another possible explanation for a decrease in the throughput is that as we scale the number of servers (from three to five), we saturate the network card of the leader. Therefore, the throughput of the evaluated protocols depends on the number of servers connected to the leader as well as write ratio.

Comparing the two protocols, it can be seen that at 100% writes, ZabAA's throughput is higher than that of Zab, with the maximum difference being relatively significant, approximately 520 ops/sec for $N = 3$ and 278 ops/sec for $N = 5$. There are two possible explanations for this result. First, ZabAA's leader does not process and broadcast the commit message, unlike in Zab. Second, ZabAA only requires two communication steps to complete write request whereas Zab requires three communication steps. However, the difference in throughput becomes less noticeable as the number of reads increases; the reason being once again that, no additional CPU processing or network load when servicing read requests (in both ZabAA and Zab), which in turn makes the difference between ZabAA and Zab in terms of throughput of less significant. In fact, an increase in the number of reads to writes leads to better overall performance in both protocols.

One interesting observation that has arisen from this experiment is that the ZabAA protocol has a better performance than Zab in terms of latency and throughput at 100% write when $N = 3$ and 5. We observe that broadcasting an acknowledgement to all servers in the ensemble does not seem to impair the overall performance, but it might impact on the performance if N increases to 7 or more as the number of acknowledgment broadcast increases.

5 Related Work

Leader based protocols tend to overload the leader and several authors [2, 7, 10, 6] have sought to remedy this drawback. S-Paxos [2] relieves the leader from broadcasting client requests by separating the roles of request dissemination and request ordering. Each process directly broadcasts client requests to others and request ordering is done using only request identifiers. Chain replication [10] reduces the leader load by distributing the role between two servers called the *head* and the *tail* but involves sequential transmission of message which tends to increase latencies for large N.

The benefit of sending an acknowledgement as a broadcast instead of a unicast is explored in the algorithm described in [8]. More importantly, to our knowledge, although the approach of ZabAA (changing from unicast to broadcast) is relatively simple, no previous study has evaluated it or exposed the trend, particularly in leader and quorum-based protocols.

6 Conclusion

We have presented ZabAC and ZabAA as atomic broadcast protocols that follow a leader-based approach, similar to Zab. ZabAC and ZabAA guarantee the delivery and order of

requests which means that each process has an equal opportunity of having its messages delivered in the same order. ZabAA is an alternative to ZabAC when $N > 3$.

Performance benchmarks showed that ZabAC had low latency and high throughput. Furthermore, by increasing the number of replicas, the ZabAA protocol not only becomes more fault tolerant but also achieves higher throughput and lower latency than the Zab protocol.

Further investigation needs to be accomplished. We plan to evaluate ZabAA using $N = 7$ and 9 to measure latency and throughput. Moreover, our research is currently being carried out to reduce the ZabAA's message overhead by conditioning the sending of acknowledgements on the outcomes of coin tosses.

References

1 Philip A Bernstein and Eric Newcomer. *Principles of transaction processing.* Morgan Kaufmann, 2009.

2 Martin Biely, Zoran Milosevic, Nuno Santos, and Andre Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, pages 111–120, 2012.

3 Lars George. *HBase: the definitive guide.* " O'Reilly Media, Inc.", 2011.

4 Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

5 Flavio P Junqueira, Benjamin C Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 245–256. IEEE, 2011.

6 Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

7 Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. Mencius: building efficient replicated state machines for wans. In *OSDI*, volume 8, pages 369–384, 2008.

8 Pedro Ruivo, Maria Couceiro, Paolo Romano, and Luis Rodrigues. Exploiting total order multicast in weakly consistent transactional caches. In *IEEE 17th Pacific Rim International Symp. on Dependable Computing (PRDC), 2011*, pages 99–108, 2011.

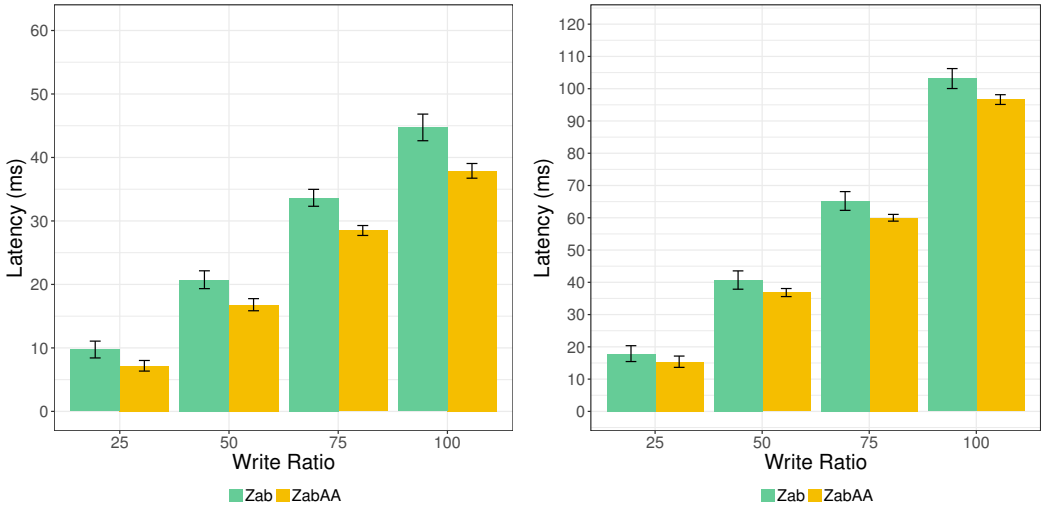
9 Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2*, pages 1–10, 2010.

10 Robbert Van Renesse and Fred B Schneider. Chain replication for supporting high throughput and availability. In *OSDI*, volume 4, pages 91–104, 2004.

A Source Code

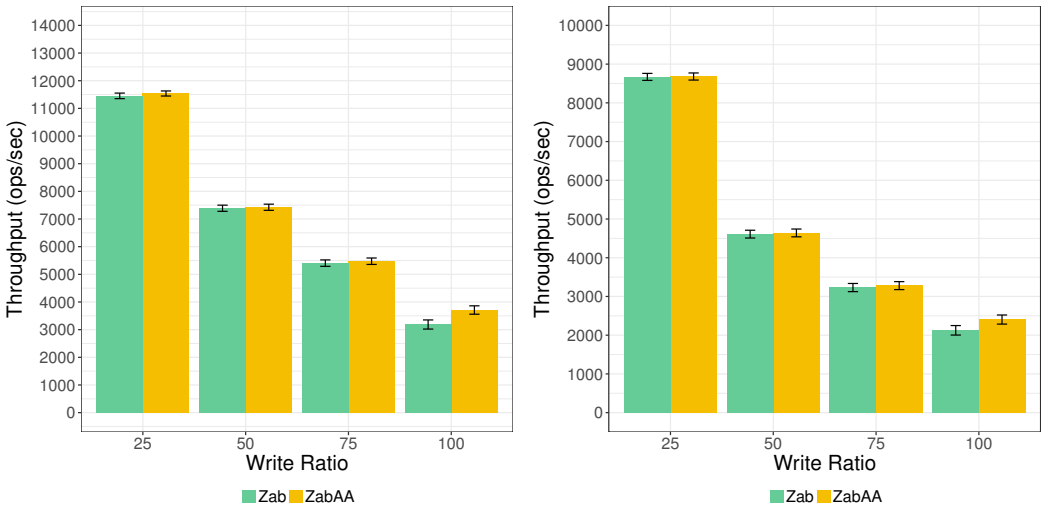
The source code for the evaluated protocols and the benchmarks are publicly available at `https://github.com/ibrahimshbat/JGroups`.

B Performance Compression for Zab and ZabAA



(a) Ensemble size $N = 3$ **(b)** Ensemble size $N = 5$

Figure 3 Zab and ZabAA latency comparison, varying the ratio of writes to reads.



(a) Ensemble size $N = 3$ **(b)** Ensemble size $N = 5$

Figure 4 Zab and ZabAA throughput comparison, varying the ratio of writes to reads.

Demand for Medical Care by the Elderly: A Nonparametric Variational Bayesian Mixture Approach

Christoph F. Kurz[1] and Rolf Holle[2]

1 **Helmholtz Zentrum München, Institute of Health Economics and Health Care Management, Neuherberg, Germany**
christoph.kurz@helmholtz-muenchen.de
1 **Helmholtz Zentrum München, Institute of Health Economics and Health Care Management, Neuherberg, Germany**

Abstract

Outpatient care is a large share of total health care spending, making analysis of data on outpatient utilization an important part of understanding patterns and drivers of health care spending growth. Common features of outpatient utilization measures include zero-inflation, overdispersion, and skewness, all of which complicate statistical modeling. Mixture modeling is a popular approach because it can accommodate these features of health care utilization data. In this work, we add a nonparametric clustering component to such models. Our fully Bayesian model framework allows for an unknown number of mixing components, so that the data, rather than the researcher, determine the number of mixture components. We apply the modeling framework to data on visits to physicians by elderly individuals and show that each subgroup has different characteristics that allow easy interpretation and new insights.

1998 ACM Subject Classification G.3 Probability and Statistics

Keywords and phrases machine learning, health care utilization, Bayesian statistics

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.4

1 Introduction

Outpatient hospital services account for a large share of health care utilization and therefore of total health care spending. To understand the variation in this major component of health care expenditures, researchers have sought to identify patient subgroups with different utilization and spending patterns.

Health care resource use data are often non-negative, right-skewed, heavy-tailed, and multi-modal with a point mass at zero. Desirable analytical approaches for these data should be sufficiently powerful and flexible to accommodate all of these features. Several authors showed that finite mixture models (FMMs) provide better model fit than single distribution generalized linear models (GLMs) and the hurdle model. [1, 2] In addition, FMMs have two advantages: first, they can easily handle multimodality. This may be important when the outcome distribution suggests decomposing resource use into different components. For example, it may be necessary to fit the tail distribution separately. Second, mixture models allow us to link the prevalence of different mixture components to different covariates. [2] Generally, mixture models distinguish between different groups of users (e.g. low- and high users) and avoid the sharp dichotomy between users and non-users.

A key question in mixture models is the optimal number of components. (Note that we use component, rather than cluster, to describe the subpopulations identified by FMMs.)

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 4; pp. 4:1–4:7
ICCSW17

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Too many components may overfit the data and impair model interpretation, while too few components limit the flexibility of the mixture to approximate the true underlying data structure. The number of different user groups can be decided either "ex-ante" by a defined value (two or three groups are common), or "ex-post", i.e. chosen by model fit after calculating different models. While the ex-ante approach is focused on feasibility and is a one-stage decision process, ex-post approaches use information which extends beyond the time at which the actual model is prepared and involves a second decision process. Both approaches introduce a decision and model selection bias.

In this paper, we present a fully variational Bayesian (VB) hierarchical mixture model, where the optimal number of components is evaluated during model fit. This one-stage process yields both the ideal number of components and allows interpretation of each component. In this Bayesian nonparametric mixture model, we let the data determine both the number and the form of the local mean functions. In contrast to frequentist nonparametric regression methods, this Bayesian approach creates a model that is only as complex as the data require. [3] In models with a fixed, finite number of parameters, there may be misfit between the complexity of the model and the amount of data available. By contrast, Bayesian nonparametric models are less subject to over- or under-fitting: the unbounded complexity of the infinite mixture mitigates under-fitting, while the Bayesian approach of computing the full posterior over parameters mitigates over-fitting. [4]

Our model uses a Dirichlet process (DP) prior for the mixing component and comprises a fully VB regression scheme. VB is an alternative to Markov chain Monte Carlo (MCMC) sampling methods for taking a fully Bayesian approach to statistical inference over complex distributions that are difficult to directly evaluate or sample from. In particular, whereas MCMC techniques provide a numerical approximation to the exact posterior using a set of samples, VB provides a locally-optimal, exact analytical solution to an approximation of the posterior. VB inference algorithms are usually faster than MCMC and suitable for large scale data sets, which are becoming more and more prevalent through the analysis of claims data and electronic health records.

In the following, we define a VB regression mixture model for counts and apply it on a data set to analyze outpatient health care utilization. The data set has already been used in [1] where Deb and Trivedi showed that a FMM with two components provides better model fit than a simple GLM. In this paper, we apply our proposed VB mixture model on the DebTrivedi data set and demonstrate that this model has good clustering and inference properties that allow new insights.

2 Model Definition

2.1 Dirichlet Process Mixtures for Generalized Linear Models

The DP is a *measure on measures* [5] or a *distribution over distributions* [6] parameterized by a base distribution G_0 and a concentration parameter α. Each draw from a DP is a distribution that is discrete with countably infinite parameters, making this a nonparametric model. Using a DP prior for the distribution of component means in mixture models does not require one to specify the number of components. Instead, a concentration parameter controls it implicitly. Suppose the sample space Ω is partitioned into measurable subsets $U_1, ..., U_k$. Let $\mathfrak{U}$ be the collection of all possible subsets of Ω. If G is a random probability measure over $(\Omega, \mathfrak{U})$ that assigns probabilities to all subsets, then $G \sim \mathrm{DP}(\alpha, G_0)$ is a measure

with property

$$G(U_1), ..., G(U_k) \sim \text{Dir}(\alpha G_0(U_1), ..., \alpha G_0(U_k)).$$

More precisely, if $\alpha > 0$ and G is an instantiation of a DP with base measure G_0, then each component k has mixture weight c_k sampled as follows:

$$\begin{aligned} G &= \sum_{k=1}^{\infty} c_k \delta(\theta = \zeta_k), \quad \text{where } \zeta_k \overset{\text{iid}}{\sim} G_0, \ k = 1, ..., \infty, \\ v_k &= \Phi(\alpha), \quad c_k = v_k \prod_{j=1}^{k-1}(1 - v_j), \quad \sum_{k=1}^{\infty} c_k = 1, \end{aligned} \tag{1}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function for the standard normal distribution and δ is the delta function. The base measure G_0 provides an initial guess at G, and α controls how close samples from the Dirichlet process are to G_0. The DP serves as a nonparametric prior on the mixture components. As the ζ's are drawn from a (discrete) DP-distributed distribution, it is very likely that they will be the same in each draw. The distinct number of ζ's defines the number of components.

The representation in Equation 2.1 is called a *stick-breaking process* and yields an infinite mixture model representation:

$$f_{\text{mix}}(x|\alpha, G_0) = \sum_{k=1}^{\infty} c_k f(x|\phi_k),$$

where f is the density function with parameters ϕ_k. Note that we define the stick-breaking process according to the probit representation [7] instead of using Beta random variables.

In addition to the usual regression parameters, these nonparametric mixture models produce several additional parameters of interest. For each mixture component k, we want to estimate the relative prevalence of the mixture component in the data and parameters of the mixture component's distribution, such as the mean, variance, and regression coefficients. The mixture weights c_k are the probabilities associated with each component and come directly from the stick-breaking proportions v_k. The features of the mixture component are in ϕ_k. In addition, for each observation, we want to estimate the mixture component from which it was most likely drawn, also called the component assignment.

2.2 The Negative Binomial Regression Model

The Negative Binomial distribution is a flexible alternative to the Poisson model for counts that accommodates over-dispersion with a longer, fatter tail. [8] Hilbe identified more than 12 different parameterizations of the Negative Binomial in the literature; [9] here, we use the definition in [1].

For $i = 1, \ldots, N$ observations and $d = 1, \ldots, D$ covariates, the data comprise an (N, D)-dimensional covariate matrix $\mathbf{X}$ with rows $\mathbf{x}_i$ and an N-vector of outcomes $\mathbf{y} = (y_1, \ldots, y_N)'$. For simplicity, we omit the subscript i in what follows. The density function for the $y \sim \text{NegBin}(\mu, \psi)$ distribution is

$$f(y) = \frac{\Gamma(y + \psi)}{\Gamma(\psi)\Gamma(y + 1)} \left(\frac{\psi}{\mu + \psi}\right)^{\psi} \left(\frac{\mu}{\mu + \psi}\right)^{y},$$

where we specify a regression model (with regression coefficients β) for the mean parameter

$$\mu = \exp(\mathbf{x}\beta)$$

and ψ is a precision parameter. In this specification, mean and variance are

$$\mathbb{E}(y|\mathbf{x}) = \mu, \quad \mathrm{Var}(y|\mathbf{x}) = \mu + \psi^{-1}\mu^2 \ ,$$

which corresponds to the NB2 model definition. [10]

2.3 Variational Inference Scheme

We assume a mixture distributions with K components, each following a negative binomial regression model. The data set consists of pairs $\{\mathbf{x}_n, y_n\}_{n=1}^N$ where x_n is a vector of length D and y_n is scalar. Therefore, for each pair of observations there exists a latent variable $\mathbf{z}_n$ indicating the component assignment. The conditional distribution of the observed data vectors given the latent variables and the component parameters can be defined as:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{z}, \boldsymbol{\beta}, \boldsymbol{\psi}) = \prod_{n=1}^{N}\prod_{k=1}^{K} \mathrm{NegBin}(y_n|\boldsymbol{x\beta}, \psi)^{z_{nk}}.$$

We define a Dirichlet prior over the mixing proportions c:

$$p(\mathbf{c}) = \mathrm{Dir}(\mathbf{c}|\boldsymbol{\alpha_0})$$

and introduce a Gaussian-Wishart prior over the mean and dispersion component:

$$p(\boldsymbol{\beta}, \boldsymbol{\psi}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\beta}_k|\hat{\boldsymbol{\beta}}_k, \psi_k^{-1}\hat{P}_k^{-1})\mathcal{W}(\psi_k|\hat{\nu}_k, \hat{\tau}_k).$$

The joint distribution over all random variables is:

$$p(\mathbf{y}, \mathbf{x}, \mathbf{z}, \boldsymbol{c}, \boldsymbol{\beta}, \boldsymbol{\psi}) = p(\mathbf{y}|\mathbf{x}, \mathbf{z}, \boldsymbol{\beta}, \boldsymbol{\psi})p(\boldsymbol{z}|\boldsymbol{\beta})p(\boldsymbol{c})p(\boldsymbol{\beta}|\boldsymbol{\psi})p(\boldsymbol{\psi}).$$

The goal of variational inference is to optimize the parameters of a fully factorized variational distribution q that minimizes the Kullback-Leibler divergence from the true intractable posterior. The optimal q maximizes the *evidence lower bound* objective. Because of intractable integrals in the variational distribution

$$q(\mathbf{z}, \mathbf{c}, \boldsymbol{\beta}, \boldsymbol{\psi}) = q(\mathbf{z})q(\mathbf{c}, \boldsymbol{\beta}, \boldsymbol{\psi}),$$

We define

$$q(\mathbf{c}, \boldsymbol{\beta}, \boldsymbol{\psi}) = q(\mathbf{c}) \prod_{k=1}^{K} q(\boldsymbol{\beta}_k, \boldsymbol{\psi}_k) = q(\mathbf{c}) \prod_{k=1}^{K} q(\boldsymbol{\beta}_k, \boldsymbol{\Sigma}_k^{-1}),$$

where $q(\boldsymbol{\beta}_k, \boldsymbol{\Sigma}_k^{-1}) = \mathcal{N}(\boldsymbol{\beta}_k, \boldsymbol{\Sigma}_k^{-1})$.

The optimization problem is therefore

$$q^*(z) = \arg\min_{q(z)\in\mathcal{D}} \mathrm{KL}(q(\mathbf{z}, \mathbf{c}, \boldsymbol{\beta}, \boldsymbol{\Sigma})\|p(\mathbf{y}, \mathbf{x}, \mathbf{z}, \boldsymbol{c}, \boldsymbol{\beta}, \boldsymbol{\psi})).$$

and we solve this by memoized online variational inference as in [11].

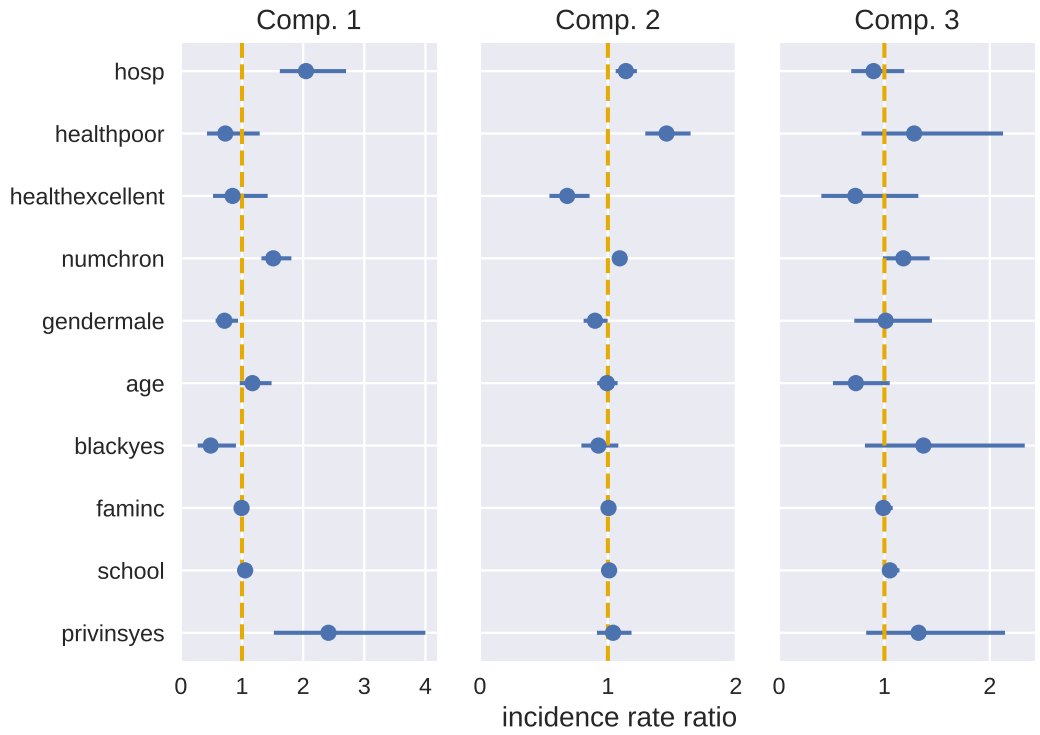


Figure 1 Parameter estimates for all three components on the DebTrivedi data set based on the negative binomial VB regression mixture model. Parameter estimates are presented as incidence rate ratios and 95% high probability density intervals. Intercept is not shown. The yellow dashed line at one marks no effect.

3 Data

We explore the model on the data set from Deb and Trivedi. [1] It contains 4406 individuals, aged 66 and over, who are covered by Medicare, a public insurance program. Originally obtained from the US National Medical Expenditure Survey (NMES) for 1987/88, the data are available in the `R` package `MixAll`. The objective is to model the demand for medical care–as captured by the number of physician/non-physician office and hospital outpatient visits–by the covariates available for the patients. Here, we adopt the number of physician office visits `ofp` as the dependent variable and use the health status variables `hosp` (number of hospital stays), `health` (self-perceived health status), `numchron` (number of chronic conditions: cancer, heart attack, gall bladder problems, emphysema, arthritis, diabetes, other heart disease), as well as the socioeconomic variables `gender`, `age`, `black` (race), `faminc` (family income), `school` (number of years of education), and `privins` (private insurance indicator) as regressors.

4 Results and Conclusion

For the DebTrivedi data set, the VB regression mixture model finds three components. The first component contains only 4.2% (186/4406) of all observations and corresponds to individuals who, on average, utilize less health care, but with higher variance. The second component corresponds to the largest proportion of individuals, 62.3%, (2745/4406)

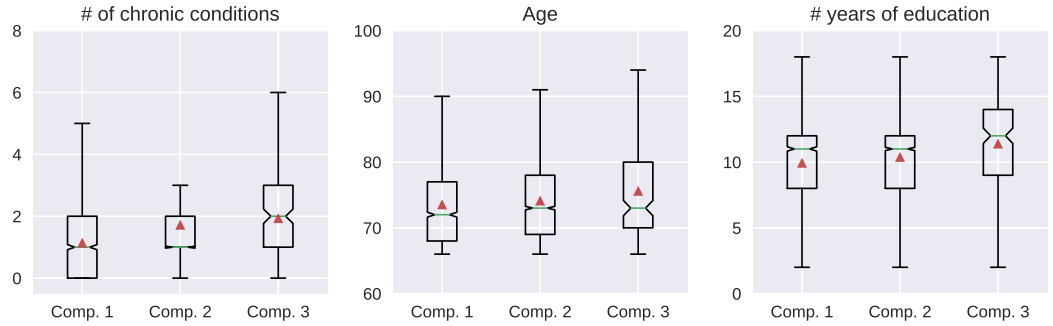


Figure 2 Boxplots for number of chronic conditions, age, and years of education for each component as modeled by the VB mixture model for the DebTrivedi data set. The red triangle marks the mean.

with medium health care utilization. The third component captures 33.5% (1475/4406) of individuals, with high utilization counts and again high variance. Figure 1 presents the parameter estimates from the regression model as incidence rate ratios (IRRs). In the first component, insurance status has the largest influence with an IRR of 2.41. This means that individuals with private insurance in the first component visit the doctor more than twice as often as those without private insurance. A similar explanation can be made for the number of hospital stays in the first component: one hospital stay accounts for 2.05 times more doctor visits, on average. This effect diminishes in the other components who contain individuals with higher utilization.

In the second component, a self perceived excellent health condition reduces the doctor visits by a factor or 0.68, while a poor health condition increases them by 1.46. This trend is slightly reduced in the third component. Most other variables show only slight effects on the number of doctor visits in the second component. In the third component, age has a protective influence on utilization, one additional year of age represents 0.73 times the utilization, on average. This seems counter-intuitive, but may be explained when comparing the age of the individuals in each cluster: Figure 2 shows that age is increasing over the components. In addition, the number of chronic diseases is also increasing in each component. That explains the highest number of doctor visits in component 3. Interestingly, the years of education also increase slightly in each component. This should be subject of further investigation as, for example, Fiscella et al. [12] found that the time spent for physical examination is lower for more educated individuals.

Regarding computational speed, the VB inference method only takes 2 seconds to analyze the data set. A comparable MCMC approach took about 45 minutes on a 2016 Core i7 CPU with 32 GB RAM. This difference is mainly due to VB providing only a solution to an approximation of the posterior, while MCMC estimates the exact posterior. While we did not find great differences in the estimates in the present case, future research should investigate this difference, for example, in simulation studies.

In conclusion, the defined VB regression mixture model provides an interesting alternative with good accuracy and speed, especially suited for large data sets.

Acknowledgements. We thank Laura Hatfield for improving the manuscript.

References

1 Partha Deb, Pravin K. Trivedi. *Demand for Medical Care by the Elderly: A Finite Mixture Approach*, Journal of Applied Econometrics, 12, 313–336, 1997

2 Borislava Mihaylova, et al. *Review of statistical methods for analysing healthcare resources and costs*, Health economics 20.8, 897-916 2011.

3 Lauren A. Hannah, David M. Blei, and Warren B. Powell, *Dirichlet process mixtures of generalized linear models*, Journal of Machine Learning Research, 1923-1953, 2011.

4 Carl Edward Rasmussen, *The infinite Gaussian mixture model*, NIPS. Vol. 12, 1999.

5 David M. Blei, Michael I. Jordan, et al.,*Variational inference for dirichlet process mixtures*, Bayesian analysis, 1(1):121–143, 2006.

6 Claude Sammut and Geoffrey I. Webb, *Encyclopedia of Machine Learning*, Springer Publishing Company, Incorporated, 1st edition, 2011.

7 Abel Rodriguez and David B. Dunson, *Nonparametric bayesian models through probit stick- breaking processes*, Bayesian Analysis, 6(1), 2011.

8 Mei-Chen. Hu, Martina Pavlicova, and Edward V. Nunes, *Zero-inflated and hurdle models of count data with extra zeros: examples from an hiv-risk reduction intervention trial*, The American journal of drug and alcohol abuse, 37(5):367–375, 2011.

9 Joseph Hilbe, *Negative Binomial Regression*, Cambridge University Press, 2011.

10 A. Colin Cameron and Pravin K. Trivedi, *Econometric models based on count data. comparisons and applications of some estimators and tests*, Journal of Applied Econometrics, 1(1):29–53, 1986.

11 Michael C. Hughes, and Erik Sudderth, *Memoized online variational inference for Dirichlet process mixture models*, Advances in Neural Information Processing Systems, 2013.

12 Kevin Fiscella, Meredith A. Goodwin, and Kurt C. Stange, *Does patient educational level affect office visits to family physicians?*, Journal of the National Medical Association 94.3, 157, 2002.

Discriminative and Generative Models for Clinical Risk Estimation: an Empirical Comparison

John Stamford[1] **and Chandra Kambhampati**[2]

1 **Department of Computer Science, The University of Hull, Kingston upon Hull, HU6 7RX, United Kingdom**
j.stamford@2014.hull.ac.uk

2 **Department of Computer Science, The University of Hull, Kingston upon Hull, HU6 7RX, United Kingdom**
c.kambhampati@hull.ac.uk

Abstract

Linear discriminative models, in the form of Logistic Regression, are a popular choice within the clinical domain in the development of risk models. Logistic regression is commonly used as it offers explanatory information in addition to its predictive capabilities. In some examples the coefficients from these models have been used to determine overly simplified clinical risk scores. Such models are constrained to modeling linear relationships between the variables and the class despite it known that this relationship is not always linear. This paper compares the conditions under which linear discriminative and linear generative models perform best. This is done through comparing logistic regression and naïve Bayes on real clinical data. The work shows that generative models perform best when the internal representation of the data is closer to the true distribution of the data and when there is a very small difference between the means of the classes. When looking at variables such as sodium it is shown that logistic regression can not model the observed risk as it is non-linear in its nature, whereas naïve Bayes gives a better estimation of risk. The work concludes that the risk estimations derived from discriminative models such as logistic regression need to be considered in the wider context of the true risk observed within the dataset.

1998 ACM Subject Classification D.4.8 Performance

Keywords and phrases Discriminative, Generative, Naïve Bayes, Logistic Regression, Clinical Risk

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.5

1 Introduction

Within the clinical domain the use of linear discriminative models such as logistic regression is a popular choice in the development of clinical risk models. Such models are able to model linear relationships between continuous variables and binary outcome events such as mortality. Generally discriminative models are preferred over generative models, however, there is a need to understand the conditions under which these models perform best [20]. Within the context of clinical risk modelling we aim to explore these conditions. This is achieved by selecting logistic regression as a linear discriminative model and comparing this to naïve Bayes which is as a linear generative model.

Logistic regression is a linear discriminative model which estimates probability directly from the inputs x and the class label y. The posterior probability is estimated directly as a function of $p(y|x)$. Logistic regression is widely applied in the clinical domain and has shown

to be effective for modelling clinical risk. The popularity of this approach in the clinical domain is the dual ability of the model to offer explanatory information about the underlying causes as well as its predictive capability. The main limitation of logistic regression is the estimated coefficients model linear relationships between the input variables and the class label.

To compare the differences between discriminative and generative models in the clinical context a naïve Bayes model is selected as it is a linear generative model. Naïve Bayes determines probability estimates from the joint probability expressed as $p(x, y)$. Probability estimates are derived by estimating the probability $p(y|x)$ for each class label based on the input, then using Bayes rule to determine the most likely class. Whilst naïve Bayes is considered a linear model, the probability estimates derived from the function $p(x, y)$ can model non-linear relationships between the inputs and the class label.

2 Background

As discussed, discriminative models are commonly applied to clinical data usually in the form of logistic regression for the use of both explanatory and predictive purposes. It is shown that logistic regression is widely used in the prediction of mortality events in addition to predicting hospital readmission, and it is able to outperform other regression techniques [18]. In other examples it is shown that logistic regression has been used to model short term mortality events including death within 30 days [2] and death within 60 days [8]. In these examples the performance was reported as area under the curve (AUC) with results of 0.86 and 0.77 respectively.

In other work, the coefficients from regression models have been used to develop simpler points based models for use in clinical practice [19][15]. The points are produced by rounding the estimated coefficients of the model to an integer value [15][10]. These integer values represent univariate risk with the sum representing a compound risk score.

Logistic regression is an extension of linear regression. With linear regression estimations been determined as

$$y = f(x, \beta) = \beta_0 + \beta_1 x_1, +...+, \beta_n x_n + \varepsilon \tag{1}$$

where β_n is some estimated coefficients and ε represents the error. This is then extended to the general linear model of logistic regression where the estimates are bound to values between 0 and 1 through the use of a logistic sigmoid function such as

$$f(y|x, \beta) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1, +...+, \beta_n x_n)}} \tag{2}$$

Regression models allow the combination of discrete and continuous input variables to be used which allows for the estimation of a linear relationship. It is useful to transform some continuous variables into categorical variables using arbitrary ranges. For example, a continuous variable could be transformed into a categorical variable with ranges representing low, medium and high. This can be seen in existing clinical risk models [19][15]. Whilst this can help model nonlinear relationships between the variable and the class it can lead to unnatural step changes in risk between each category [16].

Naïve Bayes is an example of a generative linear model which has been applied to the clinical domain. In terms of accuracy, non-linear discriminative models such as Support Vector Machines and Decision Tress have performed marginally better than naïve Bayes at identifying patients at risk of having heart failure [1][14]. However when comparing naïve

Bayes with a Multi-Layer Perceptron, which is a discriminative general non-linear model, naïve Bayes has performed better [14]. Naïve Bayes can be applied in different forms which is underpinned by the type of data which is required as the inputs. This can include multinomial naïve Bayes and Gaussian naïve Bayes. Recent work has shown that Tree Augmented naïve Bayes can perform better than other versions of naïve Bayes in the clinical domain [12].

As introduced previously, naïve Bayes estimates the probability from the joint probability expressed as $p(x, y)$. For each class the probability is estimated as

$$p(Y_1|x) = \frac{p(x|Y_1)p(Y_1)}{p(x|Y_1)p(Y_1) + p(x|Y_2)p(Y_2)} \tag{3}$$

This gives a probability estimate for class 1 and when the problem is constrained to two classes the estimation for class 0 can be $p(Y_0|x) = 1 - p(Y_1|x)$.

In this work we are using a Gaussian naïve Bayes model which determines $p(Y_n|x)$ where $x \in X$ and X conforms to a normal distribution. Within clinical dataset this requirement is commonly unsatisfied.

3 Dataset and Methodology

The dataset used in this study is a subset which is extracted from the MIMIC-III dataset [9] where heart failure patients have been identified. Patients are selected based on the use of heart failure related ICD Codes in position one, two or three of the admission diagnosis. Additionally, patients without a test for N-terminal pro-B-type natriuretic peptide (NTproBNP) was excluded as this is a valuable marker in the diagnosis of patients with heart failure [13]. The subset used contained 2,536 patients with 90 day mortality used as the class labels (Dead: 697, Alive: 1839).

Naïve Bayes and logistic regression was used for univariate analysis of the datasets. Performance of the models is measured using the Pseudo R2, the means square error (MSE) and the area under the curve (AUC).

The aim of the work is to explore how discriminative and generative models perform using real clinical data. This is achieved by comparing the commonly applied logistic regression model with naïve Bayes which are both linear models.

4 Results

The results show the performance of the two models on clinical data, specifically showing the univariate relationship between the variable and risk of death within 90 days. The results also visually show the risk estimates derived from the two models (Figure 1 and 2). Additionally, a sample range is selected from each variable with risk estimates from the models compared to the true proportion of risk in those ranges. As the outputs from the models are probabilistic estimates of risk rather than classification values the AUC is reported. R2 and MSE give representations of the distance between the probabilistic estimates and the true binary outcomes.

Risk Estimations

The distributions of the MCV and Sodium variables approximate a Gaussian distribution along with similar means and standard deviations for the two classes (Table 1 and Figure 1). In these examples linear discrimination between the two classes is not possible. This is reflected in the probabilistic risk estimates derived from the logistic regression model which

Table 1 Mean and standard deviation (SD) for MCV and Sodium for each class

	Alive		Dead	
Variable	**Mean**	**SD**	**Mean**	**SD**
MCV	89.9	6.8	90.2	7.1
Sodium	138.4	4.7	138.4	5.9

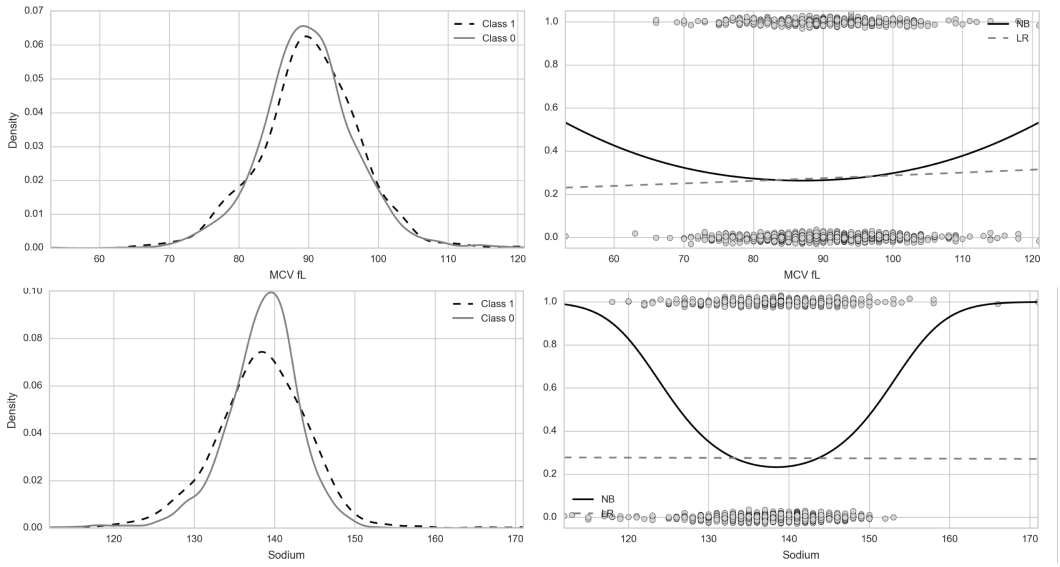


Figure 1 Density plots for both classes (left) and predictive model estimations (right) for MCV and Sodium.

shows little change in risk as values for MCV and Sodium change. Furthermore, the risk estimations for these two variables is low and when interpreting the relationship between sodium or MCV and the risk of death using a logistic regression model these variables can be said to be poor predictors. However the probabilistic estimates derived from the naïve Bayes show a different relationship between these variables and risk. It is known that levels of sodium which are too high or too low can increase the risk of death [3]. From the distributions shown in Figure 1 for both MCV and Sodium it is intuitive that risk is increased for low and high values of these variables and risk is reduced in the mid ranges. As naïve Bayes derives its estimates from the mean and standard deviation of the classes it can better approximate the relationship between theses variables and outcome when the means are similar but the standard deviations are different. When considering this in terms of detecting patients at risk, the naïve Bayes would enable a larger number of patients at risk of death to be identified (increasing sensitivity) at the cost of incorrectly identifying low risk patients as high risk (decreased specificity).

The plots for NTproBNP which has been transformed using a natural logarithm where it can be seen that the estimates from logistic regression and naïve Bayes are similar (Figure 2).

Clinical Ranges

To further explore the accuracy of the models at determining risk, three ranges (low, medium, high) where selected from each variable. The proportion of mortality events are calculated within these ranges and formed the ground truth for comparison. Risk estimations for both models are recorded on the original data and log transformed data.

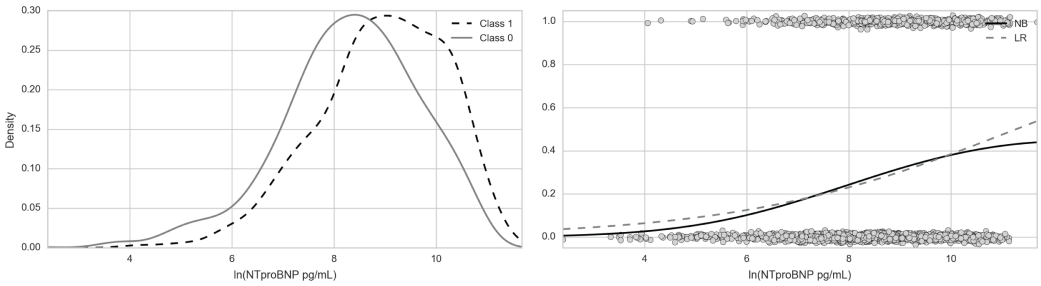


Figure 2 Density plots for both classes (left) and predictive model estimations (right) for NTproBNP.

Table 2 Estimation Accuracy using a sample of ranges.

			Raw		Log	
Variable	**Range**	**True %**	**LR**	**NB**	**LR**	**NB**
Creatinine	1 - 1.2	0.211	0.268	0.294	0.264	0.261
Creatinine	4 - 4.5	0.346	0.302	0.208	0.324	0.334
Creatinine	08 - 12	0.077	0.368	0.002	0.365	0.409
Glucose	50 - 60	0.375	0.282	0.267	0.286	0.292
Glucose	190 - 200	0.417	0.271	0.279	0.270	0.270
Glucose	350 - 400	0.304	0.257	0.108	0.262	0.269
NTproBNP	150 - 200	0.053	0.211	0.195	0.095	0.061
NTproBNP	19,000 - 20,000	0.364	0.333	0.297	0.374	0.374
NTproBNP	45,000 - 50,000	0.269	0.552	0.927	0.454	0.415
Potassium	3 - 3.2	0.360	0.259	0.292	0.260	0.309
Potassium	4.8 - 5	0.234	0.282	0.268	0.281	0.234
Potassium	9 - 10	0.714	0.349	0.987	0.314	0.770
Sodium	125 - 128	0.464	0.276	0.494	0.280	0.494
Sodium	138 - 140	0.242	0.275	0.234	0.275	0.237
Sodium	150 -155	0.714	0.273	0.604	0.269	0.536

The results for these tests show that overall the naïve Bayes model gave a closer estimate of risk to the ground truth values (Table 2). The distribution of Creatinine is positively skewed with logistic regression giving more accurate estimation. However, when the variable is transformed to a natural logarithm scale the naïve Bayes provides closer estimations. For values greater than 8 mg/dL, the naïve Bayes model gives a better representation of risk.

In variables where the data is closer to being normally distributed such as Potassium and Sodium it can be seen that the naïve Bayes model gives better estimations (Table 2). This is true for both the original data and the log transformed data where estimations from a naïve Bayes model are closer to the true values than estimates from the logistic regression model.

Metrics

Whilst it is expected that the results using Pseudo R2, MSE and AUC will be poor for a single variable, the results have been shown to explore measurable differences in the models performance.

When comparing pseudo R2 and MSE there is subtle differences between the two models for all clinical variables (Table 3). With variables such as Creatinine and NTproBNP which

Table 3 Results comparing psuedo R^2, Mean Square Error (MSE) and Area under the Curve (AUC).

	Naïve Bayes			Logistic Regression		
Variable	**R2**	**MSE**	**AUC**	**R2**	**MSE**	**AUC**
Creatinine mg/dL	-0.006	0.200	0.515	0.001	0.199	0.547
ln(Creatinine mg/dL)	0.002	0.199	0.547	0.003	0.199	0.547
MCV fL	0.000	0.199	0.534	0.000	0.199	0.519
ln(MCV fL)	0.000	0.199	0.532	0.000	0.199	0.519
NTproBNP pg/mL	-0.025	0.204	0.624	0.036	0.192	0.633
ln(NTproBNP pg/mL)	0.040	0.191	0.633	0.044	0.191	0.633
Glucose mg/dL	-0.004	0.200	0.500	0.000	0.199	0.495
ln(Glucose mg/dL)	0.000	0.199	0.494	0.000	0.199	0.495
Bicarbonate	0.007	0.198	0.551	0.003	0.199	0.538
ln(Bicarbonate)	0.004	0.198	0.552	0.005	0.198	0.538
Potassium	-0.019	0.203	0.535	0.001	0.199	0.508
ln(Potassium)	-0.003	0.200	0.535	0.000	0.199	0.508
Sodium	0.004	0.198	0.573	0.000	0.199	0.508
ln(Sodium)	0.003	0.199	0.572	0.000	0.199	0.508

are highly skewed it is shown that logistic regression has a better AUC. However, when these variables are transformed using the natural logarithm (ln) the AUC is the same for both models. In contrast, naïve Bayes performs better with variables which are approximately normally distributed as shown with MCV and Sodium.

5 Discussion

Discriminative models estimate the model parameters based on separation by minimising the negative log-likelihood classification loss against the true density [4]. Due to the nature of how the model parameters are estimated these types of models are regarded as supervised learning [5]. Discriminative models can be susceptible to cost or class imbalances in real world scenarios and in instances where the negative examples are more prevalent they are known to over fit [6]. This class imbalance problem is common in clinical datasets when developing models to predict mortality events [11].

The key feature of generative models is that the probability estimates of $p(y|x)$ are derived from the parameters of $p(x|y)$ and $p(x)$ which are estimated directly from the data. This involves internally representing the distribution of the data which adds an extra step in the probability estimations process. In the case of the naïve Bayes model $p(x|y)$ and $p(x)$ are assumed to be normally distributed, however, as shown, this is rarely the case with clinical data. Therefore, the application of generative models can be said to be more complex and it is recommended that you should not solve a more difficult problem as an intermediate step [17]. However by modelling the distribution of the input this information is incorporated into the model [5].

When comparing logistic regression and naïve Bayes it is concluded that discriminative models will perform better than generative models when considering the asymptotic errors [20]. However it has been shown that with real world datasets this may not always be the case [21]. When applying the two models to real clinical data it is shown that logistic regression models performed better when the data is highly skewed. Discriminative models

learn the model parameters from maximizing the conditional likelihood [21] therefore, with skewed data the model estimations will be skewed towards the higher density regions of the variables distributions. Once the distributions are closer to been normally distributed the estimations of naïve Bayes become more accurate and gives a better representation of the the relationship between the inputs and the class. This is especially true when the means for the class are similar but the shape of the distributions are dissimilar, as shown with MCV and Sodium. As the applied Gaussian naïve Bayes model takes its estimations from normal distributions based on the mean and standard deviation for each class it can more accurately estimate the relationship between the inputs and class when the data is normally distributed. In these cases, generative linear models can determine nonlinear relationships between the input variable and the class. In contrast the linear discriminative nature of logistic regression implies that the probability estimates must either increase monotonically, or decrease monotonically in relation to the input [7].

One of the key features of this work is the demonstration of generative and discriminative linear models on clinical data. While the use of linear discriminative models, specifically logistic regression, is favoured within the development of clinical risk models these models may not always be appropriate. This is shown when performing a univariate analysis on variables such as Sodium and MCV (Fig. 1). When looking at Sodium the results from the logistic regression model show no evidence that Sodium levels which are too high or too low have an impact on mortality, despite this relationship been known [3]. When looking at the probability estimations for Sodium using a naïve Bayes model it is shown that these estimates are similar to the ground truth (Table 2). This is a result of the two classes having similar means, however, class 1 (dead) has a greater proportion of the data points within the extremities of the distribution. This is similar to values reported in a study using a heart failure dataset when looking at the means for Sodium of the two groups of patients. It is shown that alive patients had a mean Sodium level of 138.93 and those who died had a mean of 138.45, with differences in the standard deviations, 3.00 and 3.56 respectively [11].

6 Conclusion

In this study discriminative and generative models have been applied to clinical data with the results showing that both models perform equally well in scenarios where the classes are linearly separable. It is shown that the generative model of naïve Bayes is better at modelling the relationship between the input variable and class label when distributions for each class is not linearly separable. When applying this to clinical data it is shown that when using naïve Bayes as a generative model, the assumption of the input variables been normally distributed is important. In variables which are not separable and normally distributed the generative model gave more accurate estimations compared to the ground truth. The work concludes that the choice of discriminative and generative models is related to the data and dependant on the task which is to be modelled.

This work forms the foundation for model selection within a clinical dataset and future work should explore the comparison of discriminative and generative models in a multivariate context using clinical data. This should include the use of non-linear discriminative models such as SVMs and MLPs.

References

1 Roohallah Alizadehsani, Mohammad Javad Hosseini, Zahra Alizadeh Sani, Asma Ghandeharioun, and Reihane Boghrati. Diagnosis of coronary artery disease using cost-

sensitive algorithms. In Jilles Vreeken, Charles Ling, Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu, editors, *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012*, pages 9–16. IEEE Computer Society, 2012. doi:10.1109/ICDMW.2012.29.

2 Ruben Amarasingham, Billy J Moore, Ying P Tabak, Mark H Drazner, Christopher A Clark, Song Zhang, W Gary Reed, Timothy S Swanson, Ying Ma, and Ethan A Halm. An Automated Model to Identify Heart Failure Patients at Risk for 30-Day Readmission or Death Using Electronic Medical Record Data. *Medical Care*, 48(11):981, 2010. doi:10.1097/mlr.0b013e3181ef60d9.

3 Jan Bohacik, Chandrasekhar Kambhampati, Darryl N. Davis, and John G. F. Cleland. Prediction of mortality rates in heart failure patients with data mining methods. *Annales UMCS, Informatica*, 13(2):7–16, 2013. doi:10.2478/v10065-012-0046-7.

4 Guillaume Bouchard and Bill Triggs. The Tradeoff Between Generative and Discriminative Classifier. In *16th IASC International Symposium on Computational Statistics*, pages 721–728, Prague, Czech Republic, 2004.

5 Olivier . Chapelle, Bernhard. Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, London, 2006. doi:10.1007/s12539-009-0016-2.

6 Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explorations*, 6(1):1–6, 2004. doi:10.1145/1007730.1007733.

7 Charles Elkan. Maximum Likelihood , Logistic Regression , and Stochastic Gradient Training. *Tutorial notes at CIKM*, page 11, 2012. URL: http://www.ats.ucla.edu/stat/stata/dae/mlogit.htm.

8 G. Michael Felker, Jeffrey D. Leimberger, Robert M. Califf, Michael S. Cuffe, Barry M. Massie, Kirkwood F. Adams, Mihai Gheorghiade, and Christopher M. O'Connor. Risk stratification after hospitalization for decompensated heart failure. *Journal of Cardiac Failure*, 10(6):460–466, 2004. doi:10.1016/j.cardfail.2004.02.011.

9 Alistair E.W. Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3:160035, 2016. doi:10.1038/sdata.2016.35.

10 Wayne C. Levy, Dariush Mozaffarian, David T. Linker, Santosh C. Sutradhar, Stefan D. Anker, Anne B. Cropp, Inder Anand, Aldo Maggioni, Paul Burton, Mark D. Sullivan, Bertram Pitt, Philip A. Poole-Wilson, Douglas L. Mann, and Milton Packer. The Seattle Heart Failure Model: Prediction of survival in heart failure. *Circulation*, 113(11):1424–1433, 2006. doi:10.1161/CIRCULATIONAHA.105.584102.

11 Lisa Moore. *Data Mining for Heart Failure: An investigation into the challenges un real life clinical datasets*. PhD thesis, University of Hull, 2015. doi:10.1017/CBO9781107415324.004.

12 Lisa Moore, Chandra Kambhampati, and John G. F. Cleland. Classification of a real live heart failure clinical dataset- is TAN bayes better than other bayes? In *2014 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2014, San Diego, CA, USA, October 5-8, 2014*, pages 882–887. IEEE, 2014. URL: https://doi.org/10.1109/SMC.2014.6974023, doi:10.1109/SMC.2014.6974023.

13 NICE. Chronic heart failure - Management of chronic heart failure in adults in primary and secondary care, 2010.

14 Sellappan Palaniappan and Rafiah Awang. Intelligent heart disease prediction system using data mining techniques. In *The 6th ACS/IEEE International Conference on Computer*

Systems and Applications, AICCSA 2008, Doha, Qatar, March 31 - April 4, 2008, pages 108–115. IEEE Computer Society, 2008. doi:10.1109/AICCSA.2008.4493524.

15 Stuart J. Pocock, Cono A. Ariti, John J V McMurray, Aldo Maggioni, Lars Køber, Iain B. Squire, Karl Swedberg, Joanna Dobson, Katrina K. Poppe, Gillian a. Whalley, and Rob N. Doughty. Predicting survival in heart failure: A risk score based on 39 372 patients from 30 studies. *European Heart Journal*, 34(19):1404–1413, 2013. doi:10.1093/eurheartj/ehs337.

16 Ewout W. Steyerberg. *Clinical Prediction Models*. Springer, 2009.

17 Vladimir Vapnik. *Statistical learning theory*. Wiley, 1998.

18 J. J. G. De Vries, Gijs Geleijnse, Aleksandra Tesanovic, and Ramon van de Ven. Heart failure risk models and their readiness for clinical practice. In *IEEE International Conference on Healthcare Informatics, ICHI 2013, 9-11 September, 2013, Philadelphia, PA, USA*, pages 239–247. IEEE Computer Society, 2013. doi:10.1109/ICHI.2013.26.

19 P W Wilson, R B D'Agostino, Daniel Levy, Albert M Belanger, Halit Silbershatz, and William B Kannel. Prediction of coronary heart disease using risk factor categories. *Circulation*, 97(18):1837–1847, 1998. doi:10.1161/01.CIR.97.18.1837.

20 Jing-Hao Xue and D. M. Titterington. Comment on "on discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes". *Neural Processing Letters*, 28(3):169–187, 2008. doi:10.1007/s11063-008-9088-7.

21 Jing-Hao Xue and D. M. Titterington. Comment on "on discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes". *Neural Processing Letters*, 28(3):169–187, 2008. doi:10.1007/s11063-008-9088-7.

Hey there's DALILA: a DictionAry LearnIng LibrAry

Veronica Tozzo[1], Vanessa D'Amario[2], and Annalisa Barla[3]

1 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`veronica.tozzo@dibris.unige.it`

2 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`vanessa.damario@dibris.unige.it`

3 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`annalisa.barlao@unige.it`

Abstract

Dictionary Learning and Representation Learning are machine learning methods for decomposition, denoising and reconstruction of data with a wide range of applications such as text recognition, image processing and biological processes understanding. In this work we present DALILA, a scientific Python library for regularised dictionary learning and regularised representation learning that allows to impose prior knowledge, if available. DALILA, differently from the others available libraries for this purpose, is flexible and modular. DALILA is designed to be easily extended for custom needs. Moreover, it is compliant with the most widespread ML Python library and this allows for a straightforward usage and integration. We here present and discuss the theoretical aspects and discuss its strength points and implementation.

1998 ACM Subject Classification G.1.6 Optimization, D.1.3 Concurrent Programming, D.2.2 Design Tools and Techniques

Keywords and phrases Machine learning, dictionary learning, representation learning, alternating proximal gradient descent, parallel computing

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.6

1 Introduction

Nowadays many optimisation algorithms and libraries are freely available for the most disparate machine learning applications. For example some machine learning tasks, as classification, have hundreds of algorithms and libraries implementations. This is not the case for *Dictionary Learning* (DL) [11, 12, 13, 14, 17, 19, 23] and its specialisation *Representation Learning* (RL) [3, 25], which are designed for matrix decomposition and reconstruction. They can be applied on many application domains such as signal processing [23], image processing [18], bioinformatics [2] and text-recognition [1]. Even though dimensionality reduction methods such as PCA [10] and ICA [9] may be used to solve these tasks, dictionary learning is preferable when more emphasis on the interpretability of the dictionary is required [11].

In this paper we present DALILA, a Python library for DL and RL. The optimisation is based on *alternating proximal gradient descent* [4], which allows flexibility on the minimisation

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 6; pp. 6:1–6:14

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem and therefore enables the imposition of prior knowledge. Designed to be extremely flexible and modular, DALILA can be easily extended differently from the other available libraries.

As regards the implementation, the library is compatible with the `scikit-learn` Python library and it supports the distribution of the most computationally heavy routines across different machines [6].

The remainder of this paper is organised as follows. In Section 2 we present DL, RL and the alternating proximal gradient descent algorithm. In Section 3 we give an overview on the library design and the implemented regularisers. In section 4 we comment the other available libraries on the topic. We finish with the conclusion and further work we intend to perform.

2 Theoretical background

In the following section a basic theoretical background on DL and RL problems is provided, together with the alternating proximal minimisation algorithm. The expert reader can feel free to skip it, if she/he is familiar with these mathematical concepts.

2.1 Dictionary Learning

Dictionary learning (DL) is a machine learning method that aims at finding a representation of the original data as a linear combination of basic patterns (atoms) and coefficients. The representation is completely data driven, in contrast to more generic and less adaptive methods such as Wavelet and Fourier transform [15, 26].

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ where n is the number of samples and d is the feature space dimension, the goal of DL is to decompose a dataset into two matrices:

- a dictionary $\mathbf{D} \in \mathbb{R}^{k \times d}$, a matrix of atoms that represent basic signals;
- the coefficients $\mathbf{C} \in \mathbb{R}^{n \times k}$, a matrix of weights for the atoms of the dictionary.

k represents the number of atoms composing the dictionary.

Hence, the original matrix $\mathbf{X}$ can be retrieved as a linear combination of dictionary and coefficients as in Equation (1).

$$\mathbf{X} \approx \mathbf{CD} \tag{1}$$

The recovering of the matrices $\mathbf{C}$ and $\mathbf{D}$ is solved by minimising a loss term ℓ: a a positive differentiable function that quantifies how well the multiplication of the two matrices approximates the signal.

$$\underset{\mathbf{C},\mathbf{D}}{\operatorname{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) \right] \tag{2}$$

The minimisation problem in Equation (2) not always gives us the best possible solution. In presence of noisy data, ill-posed problems or when we have prior knowledge on the problem one of the usual tricks is to add terms or constraints to the functional in order to obtain a regularised problem (Equation (3)).

$$\underset{\mathbf{C},\mathbf{D}}{\operatorname{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) + \Phi(\mathbf{C}) + \Psi(\mathbf{D}) \right] \tag{3}$$

$\Phi(\mathbf{C})$ and $\Psi(\mathbf{D})$ are two penalty functions that act respectively on the coefficients and on the dictionary. The functional in Equation (3) can be further specialised in order to include

Algorithm 1 Alternating proximal gradient descent

1: Random initialization of the matrices C and D
2: **for** $i = 0 : \text{max\_iters}$ **do**
3: $\quad \gamma_D \leftarrow \text{lipschitz\_step}_\ell(D)$
4: $\quad \gamma_C \leftarrow \text{lipschitz\_step}_\ell(C)$
5: $\quad D_{t+1} = \text{prox}_{\gamma_D \Psi}(D_t - \gamma_D \bigtriangledown_D (\ell_t))$
6: $\quad C_{t+1} = \text{prox}_{\gamma_C \Phi}(C_t - \gamma_C \bigtriangledown_C (\ell_t))$
7: $\quad$ **if** difference between iterates $< \epsilon$ **and**
8: $\quad\quad$ different between previous and current objective function $< \epsilon$ **then**
9: $\quad\quad$ **break**

constraints sets that reduce the space in which a solution is admissible. A very common example is when we impose the involved matrices to be non-negative, a problem known as Non-negative Matrix Factorization (NMF) [11].

2.2 Alternating proximal gradient descent

It is worth noting that the optimisation of Equation (3) poses some issues. In fact, due to the multiplication present in the loss function ℓ, Equation (3) is jointly non-convex. Moreover the generality of the penalty terms requires the use of a minimisation algorithm that deals with different choices of penalties without a substantial change in its flow.

All taken into account, our optimisation choice is *alternating proximal gradient descent* [4] which assures the convergence to a local minima under the following assumptions: 1) the loss function $\ell(\mathbf{X}, \mathbf{CD})$ is differentiable and partially Lipschitz continuous; 2) it is possible to compute the proximal mapping of the penalty terms in closed form or at least to approximate it. See [4] for mathematical details and theoretical proofs.

The proximal mapping [16] of a function g for a point $\mathbf{u}$ is defined as Equation (4).

$$\text{prox}_{\mu g}(\mathbf{u}) = \underset{\mathbf{v}}{\text{argmin}} \left(g(\mathbf{v}) + \frac{1}{2\mu} \parallel \mathbf{v} - \mathbf{u} \parallel_2^2 \right) \tag{4}$$

When the computation of the prox is not available in a closed form, it can be approximated via a iterative algorithm.

A general overview on the optimization algorithm is given in Algorithm 1. The algorithm alternates the steps 3, 4, 5 and 6 until convergence. In step 4 the computation of the gradient descent is performed on the dictionary keeping the coefficients fixed. On the result the proximal mapping of the dictionary learning penalty Ψ is applied. The same is done in step 6 on the coefficients.

We want to remark that the gradient is computed w.r.t. the previous iteration in both cases [17]. This allows to perform the two steps in parallel if needed.

2.3 Representation learning (sparse coding)

Representation learning is a more general form of sparse coding (SC) that similarly to DL aims at finding the best approximation of a signal $\mathbf{X}$ (Equation (1)), when the dictionary $\mathbf{D}$ is given. This leads to a problem which is easier to minimise and, given convex loss function and penalty, becomes convex. The convexity property guarantees that a global optimum can be always reached.

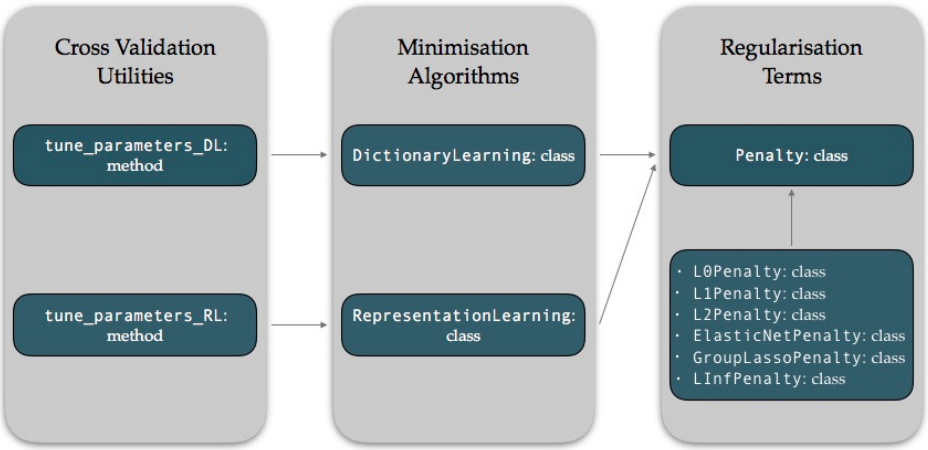


Figure 1 Diagram of DALILA library structure. The main core consists of the two classes which address the minimization problem. This depends on the class Penalty which represents a generic penalty term and that can be specialised by declaring subclasses. The library also offers cross validation utilities for the free parameters tuning.

In representation learning the choice of the penalty on the coefficients is arbitrary and dictated by the problem while in SC we assume the use of L^0 or L^1 norms. The formalisation is given in Equation (5).

$$\underset{\mathbf{C}}{\operatorname{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) + \Phi(\mathbf{C}) \right] \tag{5}$$

This optimisation problem, since involves the minimisation on only one variable, can be solved with proximal gradient descent that acts similarly to what explained in Algorithm 1 without steps 3 and 5.

3 DALILA

DALILA is a library for signal decomposition and reconstruction. Its first focus is Dictionary Learning (DL) described in Section 2. The fact that both the dictionary and the coefficients are learned from data allows for a more complete analysis of the results extracting useful information about the original signals. Moreover the possibility to impose prior knowledge on the problem using penalty terms grants that the final matrices respect certain constraints. Examples for regularised DL are: 1) image denoising where sparsity imposition forces the most important atoms to be used and the noisy ones to be discarded; 2) pattern recognition, where the atoms of the dictionary are seen as latent patterns from which the original signals are generated.

DALILA second focus is Representation Learning whose purpose is to represent the original data matrix on a new space defined by the atoms of the dictionary $\mathbf{D}$. Penalty terms can be added to impose a structure on this new representation.

The learned coefficients may be used as a new representation for further tasks such as: 1) compressed sensing that exploits sparsity reducing the size of the original signal; 2) classification where, rather than considering the original signal, the coefficients are used as new features.

3.1 Implementation

DALILA is implemented in Python. It supports different versions of Python and it is `scikit-learn` compatible. See `https://slipguru.github.io/dalila` for the full documentation and a quick start.

DALILA has a modular and easily extendible design (see Figure 1). The core of the library consists of two classes, `DictionaryLearning` and `RepresentationLearning` which respectively solve the two minimisation problems in Equation (3) and (5). These classes depend on a generic penalty term (`Penalty`) which can be specialised into different regularisers by declaring a subclass. The library is therefore easily extendible with new regularisers and flexible in the choice of the model. Cross validation utilities to tune the parameters of the model are provided, the two methods shown in Figure 1, `tune_parameters_DL` and `tune_parameters_RL`, can perform the tuning by parallel or distributed computation using `dask` library [6].

The loss function ℓ introduced in a generic form in Equation (3) and (5) is common to both `DictionaryLearning` and `RepresentationLearning` classes. It is implemented as the Frobenius norm of the difference between the original signal and its reconstruction, defined as Equation (6).

$$\ell(\mathbf{X}, \mathbf{CD}) = \|\mathbf{X} - \mathbf{CD}\|_F^2 \tag{6}$$

As regards the regularisation terms, called Φ and Ψ in Equation (3) and (5), DALILA offers many possibilities.

In this way a proper regulariser, dependent from the task, can be chosen, during the initialisation of the `DictionaryLearning/RepresentationLearning` instances. In fact `DictionaryLearning/RepresentationLearning` minimisation algorithms do not depend on the penalties chosen, as long as the penalty classes inherit from the superclass `Penalty` and reimplement the same methods (Figure 1). For a better understanding see Appendices B, C.

Available regularisation terms

The penalty terms Φ and Ψ are the product between a regularisation parameter and, typically, a norm. The norm is used to impose a structure on the matrix, while the regularisation parameter, a positive scalar, weights the regularisation term influence on the solution.

In the following we show the possible choices for Φ and Ψ applied on a generic matrix $\mathbf{M}$ whose i-th row is indicated as $\mathbf{M}_{i,:}$ and j-th column as $\mathbf{M}_{:,j}$. Its generic element is denoted by m_{ij}. With the notation $\Phi|\Psi$ we indicate that the penalty can be applied or on the dictionary or on the coefficients or on both.

- `L1Penalty` - $\boldsymbol{\ell_1}$ **norm**

  $$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \|\mathbf{M}_{i,:}\|_1 = \lambda \sum_i \sum_j |m_{ij}| \tag{7}$$

  Regularisation terms of this form, due to the geometrical meaning of the ℓ_1 norm, force the solution to be sparse and, therefore, highly interpretable [21]. If the penalty is applied on the dictionary it promotes a dictionary whose atoms have a low number of non-null components. For the coefficients, the penalisation promotes a reconstruction based only on few atoms of the dictionary, discarding the ones which give minor contribution to the original signal.

The proximal operator related to this regulariser is

$$\text{prox}_{\Phi|\Psi}(m_{ij}) = \begin{cases} m_{ij} - \lambda & \text{if } m_{ij} > \lambda \\ 0 & \text{if } m_{ij} \in [-\lambda, \lambda] \\ m_{ij} + \lambda & \text{if } m_{ij} < -\lambda \end{cases} \tag{8}$$

- `L2Penalty` - **ℓ_2 norm**

$$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \left(\sum_j m_{ij}^2 \right)^{\frac{1}{2}} \tag{9}$$

Penalties of this form, as in the previous case, can be applied to both matrices $\mathbf{C}$ and $\mathbf{D}$. The ℓ_2 regularisation term leads to the shrinkage of the components of each row, but, differently from the ℓ_1 norm, it does not lead to a sparse solution [22].
The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \max(1 - \lambda / \|\mathbf{M}_{i,:}\|_2, 0)\ \mathbf{M}_{i,:} \tag{10}$$

- `ElasticNetPenalty`

$$\Phi|\Psi(\mathbf{M}) = \sum_i \left[\alpha\lambda_1 \|\mathbf{M}_{i,:}\|_1 + (1-\alpha)\lambda_2 \|\mathbf{M}_{i,:}\|_2 \right] \tag{11}$$

Elastic Net can be preferable to ℓ_1 norm, in the case of highly correlated variables, and also to ℓ_2 norm since it inherits the possibility of finding a sparse solution [27].
Here λ_1 and λ_2 weight the two norms separately while $\alpha \in [0, 1]$ balances the contribution of the two terms. The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \left(\frac{1}{1 + \alpha\lambda_2} \right) \text{prox}_{\lambda_1 \|\cdot\|_1}(\mathbf{M}_{i,:}) \tag{12}$$

- `L0Penalty` - **ℓ_0 pseudo-norm**

$$\Phi|\Psi(\mathbf{M}) : \forall i \quad \|\mathbf{M}_{i,:}\|_0 \leq s \tag{13}$$

where $\|\mathbf{M}_{i,:}\|_0$ counts the number of non-zero elements in the row. The regularisation parameter $s \in \mathbb{N}$ impose the maximum number of non-null elements in $\mathbf{M}_{i,:}$, naturally leading to sparse results. The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \begin{cases} m_{ij}, \text{ if } m_{ij} \in \mathcal{S} \\ 0, \text{ otherwise} \end{cases} \tag{14}$$

where $\mathcal{S}$ is the set containing the first s biggest components of $\mathbf{M}_{i,:}$.

- `LInfPenalty` - **ℓ_∞ norm**

$$\Phi(\mathbf{M}) = \lambda \sum_j \|\mathbf{M}_{:,j}\|_\infty \tag{15}$$

where $\|\mathbf{M}_{:,j}\|_\infty$ returns the maximum element in the column. This regularisation term acts column-wise only on the coefficients and it is useful in presence of a redundant dictionary [24].

The effect of this regulariser is to discard the atoms that overall have a low impact in the reconstruction while emphasising the atoms that, even if only in few samples, contribute largely.
The proximal operator is

$$\text{prox}_{\Phi}(\mathbf{m}_j) = \mathbf{m}_j - \lambda \Pi_1(\mathbf{m}_j/\lambda) \tag{16}$$

The algorithm for the projection on the ℓ_1 ball is explained in [7].

- `GroupLassoPenalty` - $\boldsymbol{\ell_{1,2}}$ **norm**

$$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \sum_{g \in \mathcal{G}} \|\mathbf{M}_{i,g}\|_2 \tag{17}$$

where $\mathcal{G}$ is the set of the groups (i.e. the indices of the columns) defined by the user. For each row of the matrix $\mathbf{M}$ the penalty enforces all the values of a group to be selected or discarded together (i.e. all of them set to zero). The groups cannot be overlapping and they have to cover all the columns indices. Its proximal mapping is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:})_{\mathcal{G}} = \max(1 - \lambda/\|\mathbf{M}_{i,g}\|_2, 0)\ \mathbf{M}_{i,g} \quad \text{for all } g \in \mathcal{G} \tag{18}$$

- **Additional user-implemented penalties** As introduces before DALILA is flexible in the sense that it allows to use different penalties without changing the minimisation flow and it further allows the user to declare new non-considered penalties. More details are given in Appendix C.

Both for `DictionaryLearning` and `RepresentationLearning` the user can impose non-negativity constraints on the involved matrices. When this requirement is applied both on the dictionary and the coefficients it is called Non-negative Matrix Factorization [11]. The non-negativity condition can, moreover, be imposed only on the coefficients in order to obtain a more interpretable contribution of the dictionary elements to the reconstruction of the original signal [19]. The projection is performed by setting to zero all the negative elements in the considered matrix.

Furthermore, in the `DictionaryLearning` class the user can impose the normalization condition on the dictionary matrix, which is equivalent to set the euclidean norm of each row equal to 1.

$$\|\mathbf{D}_{i,:}\|_2 = 1 \quad \text{for all } i \in \{1, \ldots, k\} \tag{19}$$

Model selection

A critical aspect of these reconstruction techniques is constituted by the choice of the free parameters, which are the number of atoms k that define the dictionary and the regularisation values that weight the penalty terms. This choice depends on the dataset given as input $\mathbf{X}$ and it can varies depending on different factors, as the high level of noise in the measurements, the redundancy of the founded dictionary and the interpretability of the solution. Given the fact that there is an infinite set of possible values for each parameter and no theoretical formulation that guides to the best solution exists, the only feasible approach is to empirically solve a searching problem over the parameters space.

DALILA allows for a fine tuning of the free parameters of the model on the dataset by performing a grid search based on cross validation. The best combination of parameters is selected as the one that returns the best mean score over multiple iterations. As score we use BIC (Bayesian Information Criterion) [20], shown in Equation (20).

$$\text{BIC} = -\log(n) \cdot k - c \cdot \ell(\mathbf{X}, \mathbf{CD}) \tag{20}$$

where c is a positive constant. The highest value of the BIC corresponds to the best model in the search space. This procedure is available both for `DictionaryLearning` and `RepresentationLearning`.

The two procedures, `tune_parameters_DL` and `tune_parameters_RL`, allow the user to specify different search modalities. In `tune_parameters_DL` the user can choose among different configurations.

- tuning the number of atoms together with the dictionary penalty and after searching the regularisation parameter on the coefficients;
- fixing the number of atoms in the estimator and tuning the penalties together;
- fixing the penalties values and tuning the best number of atoms;
- tuning all the possible value together, number of atoms and regularisation parameters, analysing every possible combination in the grid.

4 Related work

As of today other libraries addressing similar tasks are available, SPAMS[1] (SPArse Modeling Software) [13] and the Decomposition modules of scikit-learn [5, 12].

SPAMS, implemented in C++, performs the decomposition tasks through dictionary learning, non negative matrix factorization and sparse PCA. It offers a good set of options, but, even if it is interfaceable with Python, it is not `scikit-learn` compatible. Therefore, it cannot be integrated in `scikit-learn` pipelines. Moreover, it is non trivial to customise or extend it.

The other main competitor, the decomposition module of `scikit-learn` library, implements dictionary learning and NMF but it only has few fixed penalty terms.

5 Conclusions and further work

In this work we introduced DALILA, a library for dictionary learning and representation learning. We presented its main features: the wide variety of penalties, the possibility to customise the library on specific problems, its compatibility with `scikit-learn` library, its high flexibility and its scalable architecture which allows to perform parallel parameter searching procedures.

The wide variety of penalties applicable on the matrices allow the user to solve a broad range of problems. Moreover, since scientific problems can introduce more specific and new needs, the possibility to customise and adapt the library is essential.

DALILA is fully compliant with one of the most complete machine learning Python libraries that is `scikit-learn`. This makes almost effortless its integration with the majority of machine learning Python pipelines.

The possibility to parallelise or distribute computationally heavy routines [6] greatly reduce the wall-clock time. Nevertheless our implementation is still basic and therefore the time performance are worse compared to the other presented libraries. In the future we plan to replace the use of `dask` with an hybrid parallelised system which will take advantage both of MPI tasks distribution and the computational acceleration given by GPUs.

Given DALILA flexibility and the existence of other Dictionary Learning related problems, we aim at extending it. The planned expansions are: 1) Discriminative Dictionary Learning [14], a variant of the dictionary learning problem which includes the classification task; 2)

[1] `http://spams-devel.gforge.inria.fr/doc/html/`

Shift Invariant Dictionary Learning [8] that allows the reconstruction of signals using atoms with smaller support than the original signal and 3) Total Variation penalty in combination with Lasso [19].

References

1 Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

2 Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. *Deciphering signatures of mutational processes operative in human cancer*. *Cell reports*, 3(1):246–259, 2013.

3 Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013. doi:10.1109/TPAMI.2013.50.

4 Jérôme Bolte, Shoham Sabach, and Marc Teboulle. *Proximal alternating linearized minimization for nonconvex and nonsmooth problems*. *Mathematical Programming*, 146(1-2):459–494, 2014.

5 Andrzej Cichocki and PHAN Anh-Huy. *Fast local algorithms for large scale nonnegative matrix and tensor factorizations*. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

6 Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL: http://dask.pydata.org.

7 John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. *Efficient projections onto the l 1-ball for learning in high dimensions*. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

8 Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y Ng. *Shift-invariance sparse coding for audio classification*. *arXiv preprint arXiv:1206.5241*, 2012.

9 Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

10 Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

11 Daniel D Lee and H Sebastian Seung. *Learning the parts of objects by non-negative matrix factorization*. *Nature*, 401(6755):788–791, 1999.

12 Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007. doi:10.1162/neco.2007.19.10.2756.

13 Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. *Online Learning for Matrix Factorization and Sparse Coding*. *Journal of Machine Learning Research*, 11:19–60, 2010. URL: http://dl.acm.org/citation.cfm?id=1756006.1756008.

14 Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis R Bach. *Supervised dictionary learning*. In *Advances in neural information processing systems*, pages 1033–1040, 2009.

15 Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

16 Neal Parikh, Stephen Boyd, et al. *Proximal algorithms*. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

17 Alain Rakotomamonjy. *Direct optimization of the dictionary learning problem*. *IEEE Transactions on Signal Processing*, 61(22):5495–5506, 2013.

18 Saiprasad Ravishankar and Yoram Bresler. *MR image reconstruction from highly undersampled k-space data by dictionary learning*. *IEEE transactions on medical imaging*, 30(5):1028–1041, 2011.

19 Saverio Salzo, Salvatore Masecchia, Alessandro Verri, and Annalisa Barla. *Alternating proximal regularized dictionary learning*. *Neural computation*, 26(12):2855–2895, 2014.

20 Gideon Schwarz et al. *Estimating the dimension of a model. The annals of statistics*, 6(2):461–464, 1978.

21 Robert Tibshirani. *Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

22 Andreĭ Nikolaevich Tikhonov, Vasiliĭ Iakovlevich Arsenin, and Fritz John. *Solutions of ill-posed problems*, volume 14. Winston Washington, DC, 1977.

23 Ivana Tosic and Pascal Frossard. *Dictionary learning. IEEE Signal Processing Magazine*, 28(2):27–38, 2011.

24 Joel A Tropp. *Just relax: Convex programming methods for identifying sparse signals in noise. IEEE transactions on information theory*, 52(3):1030–1051, 2006.

25 Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Information Theory*, 53(12):4655–4666, 2007. doi:10.1109/TIT.2007.909108.

26 Martin Vetterli, Jelena Kovacevic, and Vivek K Goyal. *Fourier and wavelet signal processing. Book site*, 2013.

27 Hui Zou and Trevor Hastie. *Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

A Appendix: example of usage and related output

In this appendix we want to offer some insights of how DALILA code works and the possible outcomes we can obtain. All the experiments are performed on the ORL database of faces [2]. This database is saved as a matrix within the library with 400 samples and 112×92 features.

In order to perform Dictionary Learning on this dataset we firstly need to flatten the matrices into arrays and then apply the fitting procedure. In this estimator we are using an arbitrary number of atoms that we choose randomly and we impose non-negativity on both the matrices since we are dealing with gray scale images that have values in the range of [0,255].

```
import numpy as np

from dalila.dictionary_learning import DictionaryLearning
from dalila.penalty import L1Penalty

dataset = np.load("/path/to/dalila/folder/dalila/databases/
    ORL_database.npy")
d1, d2, n = dataset.shape
data = np.empty((n, d1*d2))
for i in range(n):
data[i,:] = np.ravel(dataset[:,:,i])

estimator = DictionaryLearning(k=60, non_negativity="both")
estimator.fit(data, n_iter=1000)
C, D = estimator.decomposition()
```

Some of the results obtained with this code are depicted in Figure 2. Here the most used atoms in the reconstructions are showed. In Figure 3, instead, we show some reconstructions and their original signals to perform a qualitative comparison.

[2] The database is available for download at the link `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`

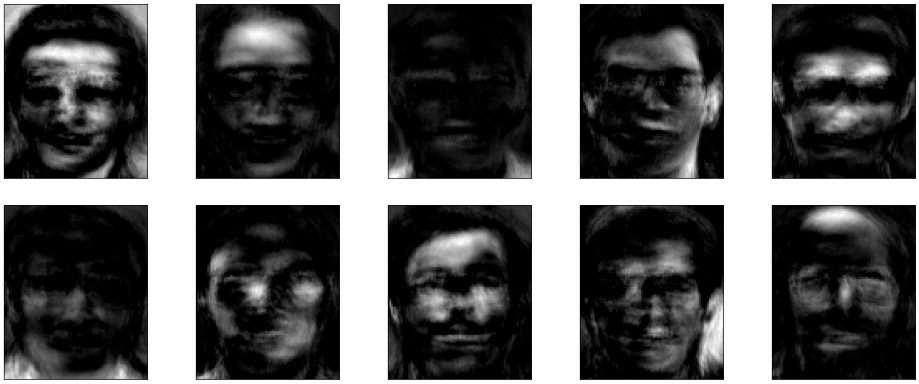


Figure 2 The ten more recurrent atoms for the reconstruction of the samples in ORL dataset.

In this example we tune only the right number of atoms we should use for this particular dataset. Given the dimensionality of the dataset we picked as range $k \in [5, 10, 15, 20, 30, 40, 50]$.

Moreover we did not use the basic BIC score but a normalised one since the values within the BIC computation, with this dataset, are not comparable in their magnitude (see `scoring_function`). The results are showed in Figure 4 where we can see that the best number of atoms is 15.

```
import matplotlib.pyplot as plt
import numpy as np
from dalila.parameters_research import tune_parameters_DL
from dalila.dictionary_learning import DictionaryLearning

dataset = np.load("/path/to/dalila/folder/dalila/databases/
    ORL_database.npy")
d1, d2, n = dataset.shape
data = np.empty((n, d1*d2))
for i in range(n):
data[i,:] = np.ravel(dataset[:,:,i])
```

```
def scoring_function(estimator, X, y=None):
C, D = estimator.decomposition()
r_error = (np.linalg.norm(estimator.X - C.dot(D))/
np.linalg.norm(estimator.X))
n = estimator.X.shape[0]
return -(2.3*np.array(r_error) + 0.001*estimator.k*np.log(n))

possible_ks = [5,10,15,20,30, 40,50]
estimator = DictionaryLearning(k=5, non_negativity="both")
gscv = tune_parameters_DL(data, estimator, analysis=2,
range_k=possible_ks,  fit_params={'n_iter':500},
scoring_function=scoring_function)
```

Figure 3 Examples of the reconstruction obtained with DL decomposition. In the first row the reconstructions are shown and beneath them their original picture.

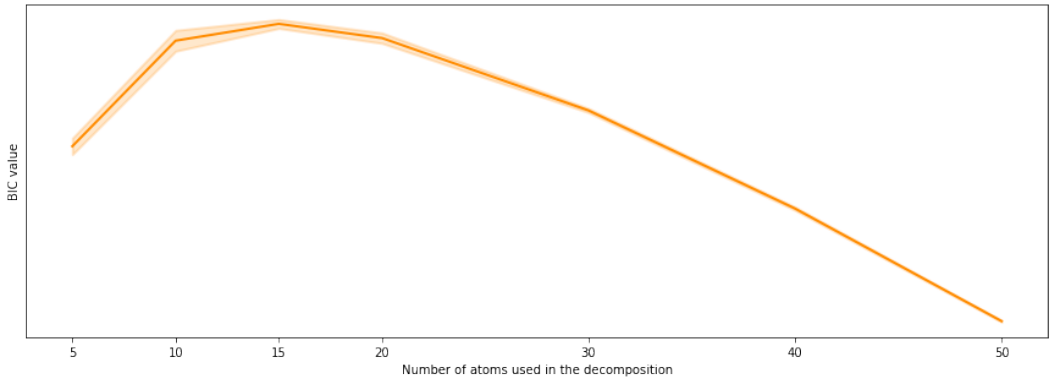


Figure 4 Curve of the `scoring_function` values (refined BIC) w.r.t. the number of atoms used for the decomposition. To an higher value corresponds a better model for the dataset. In this case the best model is the one with 15 atoms in the dictionary.

B Appendix: interchangeability of the penalties

With DALILA trying different penalties on the same dataset requires only few lines of code.

For example suppose we have the right number of atoms (with the dataset we use is 7) in which to decompose the dataset and that, some oracle, told us the perfect regularisation parameters for each penalty on that dataset. Then we may want to try different sparsifications on the coefficients to see which one better approximates the original signal. We tried `L1Penalty`, `L0Penalty` and `ElasticNetPenalty`.

```
import numpy as np

from dalila.dictionary_learning import DictionaryLearning
from dalila.dataset_generator import synthetic_data_non_negative
from dalila.penalty import L1Penalty, L0Penalty, ElasticNetPenalty

X, D, C = synthetic_data_non_negative()

estimator = DictionaryLearning(k=7, coeff_penalty=L1Penalty(0.01),
    non_negativity="both")
estimator.fit(X)
C_l1, D_l1 = estimator.decomposition()
error_l1 = estimator.reconstruction_error()

estimator = DictionaryLearning(k=7, coeff_penalty=L0Penalty(3),
    non_negativity="both")
estimator.fit(X)
C_l0, D_l0 = estimator.decomposition()
error_l0 = estimator.reconstruction_error()

estimator = DictionaryLearning(k=7, coeff_penalty=ElasticNetPenalty
    (0.01, 0.1, 0.7), non_negativity="both")
estimator.fit(X)
C_en, D_en = estimator.decomposition()
error_en = estimator.reconstruction_error()
```

After the execution of this piece of code the comparison between the results is straight-forward and you can notice that the effort is minimal.

C Appendix: addition of a new penalty

DALILA allows to easily introduce new customised penalties for the optimisation of dictionary learning or representation learning. We want to underline that, even if the implementation steps are easy, the function that computes the proximal mapping has to be correct and no theoretical inconsistencies should be present. The behaviour is otherwise unpredictable.

The first step is the import of the super-class Penalty that our new penalty has to extend. We also import other things that we need later.

```
from dalila.representation_learning import RepresentationLearning
from dalila.penalty import Penalty
import numpy as np
```

The implementation of the new class, besides the construction, has to expose the method `apply_prox_operator` that is the one called during the minimisation. In this method the prox operator is implemented.

```
class NewPenalty(Penalty):

def __init__(self, regularization_parameter):
self.regularization_parameter = regularization_parameter

# x is the matrix on which apply the prox
```

```
# gamma is the gradient descent step
def apply_prox_operator(self, x, gamma):
# if you are declaring a real penalty
# change the implementation and
# transform x according to your penalty
return x
```

Once we have declared the penalty, in this case a penalty that does nothing, we put it into the representation learning procedure.

```
fake_data = np.random.rand(50,50)
fake_dictionary = np.random.rand(5, 50)
estimator = RepresentationLearning(fake_dictionary, penalty=
    NewPenalty(5))
estimator.fit(fake_data)
```

It is also possible to use the parameter research procedures with the new penalty provided that we also overwrite the method `make_grid` since the searching function assumes it exists. We here show a basic example of how it can be implemented, one may want to vary the interval or the sampling procedure.

```
class NewPenalty(Penalty):

# ..as above..

def make_grid(self, low=0.001, high=1, number=10):
# possible regularization parameters to analyse
values = np.linspace(low, high, number)
l = []
# the list has to be composed of NewPenalty objects
for (i, v) in enumerate(values):
l.append(NewPenalty(v))
return l
```

Again we show that it is usable right away without further code.

```
from dalila.parameters_research import tune_parameters_RL
estimator = RepresentationLearning(fake_dictionary, penalty=
    NewPenalty(5))
gscv = tune_parameters_RL(fake_data, estimator, coeff_penalty_range
    =(0.1, 1, 3))
```

Faster Concurrent Range Queries with Contention Adapting Search Trees Using Immutable Data*

Kjell Winblad

Department of Information Technology, Uppsala University, Sweden
kjell.winblad@it.uu.se

Abstract

The need for scalable concurrent ordered set data structures with linearizable range query support is increasing due to the rise of multicore computers, data processing platforms and in-memory databases. This paper presents a new concurrent ordered set with linearizable range query support. The new data structure is based on the contention adapting search tree and an immutable data structure. Experimental results show that the new data structure is as much as three times faster compared to related data structures. The data structure scales well due to its ability to adapt the sizes of its immutable parts to the contention level and the sizes of the range queries.

1998 ACM Subject Classification D.2.8 Performance measures, E.1 Trees, H.2.4 Concurrency

Keywords and phrases linearizability, concurrent data structures, treap

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.7

1 Introduction

The use of concurrent ordered set data structures[1] with support for linearizable[2] range queries[3] is increasing as multicores are becoming more readily available and due to the rise of big scale data processing platforms and in-memory databases such as Google's F1 [23] and Yahoo's Flurry [1]. Both of these require set data structures with fast updates[4] to store incoming data while concurrently serving (typically large) linearizable range queries for analytics [3]. Although there are many concurrent set data structures (e.g. [22, 12, 16]) and ordered set data structures (e.g. [8, 13, 6]), there are only a few concurrent data structures with efficient linearizable range queries [5, 2, 17, 19, 7, 3].

This paper proposes a new concurrent ordered set data structure that internally makes use of an immutable data structure. The difference between an immutable data structure and its mutable counterpart is that the immutable data structure's update operations do not modify the given data structure instance in-place but instead return a new version, leaving the input instance intact. For many data structures, e.g. binary search trees, the operations of the immutable version are asymptotically as efficient as in its mutable counterpart [14]. As an example, the insert operation of an immutable balanced binary search tree only needs

* This work was supported by the Linnaeus centre of excellence UPMARC (Uppsala Programming for Multicore Architectures Research Center).

[1] A concurrent ordered set data structure represents a set of items that can be manipulated concurrently by several threads and where an item consists of an ordered key and optionally some additional data.

[2] A linearizable operation appears to happen instantly between the operation's invocation and return [10].

[3] A range query operation returns all items with keys within the given range (specified by two keys).

[4] An update operation is an insert operation or a remove operation where the former inserts an item (replacing an existing item if one with an equal key already exists) and the latter removes an item with the given key if such an item exists.

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 7; pp. 7:1–7:13

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to make a copy of the nodes on the path to the inserted node, which only consists of $\mathcal{O}(\log n)$ nodes, where n is the number of items that are stored in the search tree.

One can derive a concurrent ordered set data structure with linearizable range query support from a single mutable reference to an immutable data structure [9]. A lookup or a range query simply performs the operation in the referenced immutable data structure. An update operation repeatedly tries to update the reference using an atomic compare-and-swap operation [9] until the update succeeds. Unfortunately, this coarse-grained approach does not scale when concurrent updates are common due to the scalability bottleneck that exists in the updating of the shared reference. Instead, several data structures [5, 2] use immutable parts that can store a fixed number of items to shorten the time range queries need to spend reading shared mutable data. This fine-grained approach can be efficient when it is possible to fine-tune the size of the data structure's immutable parts to fit the sizes of the range queries (the number of items within the range) and the contention level. However, the fine-grained approach does not work well when the access pattern is unknown or differs in different parts of the data structure.

The *main contribution* of this paper is a new concurrent ordered set data structure with linearizable range query support that solves the problems with the coarse-grained and fine-grained approaches described above by dynamically changing the sizes of its immutable parts to fit the workload at hand. The new data structure is based on the contention adapting search tree (CA tree) [18, 19, 20] and an immutable data structure. Previous results [19] show that CA trees using mutable data structures provide good scalability in scenarios with short range queries. However, previous CA tree variants' scalability for large range queries is limited as their range queries lock out other threads from large portions of the data structure for a time period whose length is proportional to the number of items with keys in the given range. The new CA tree variant eliminates this problem by utilizing an immutable data structure. As is shown in this work, the new CA tree variant's ability to reduce the lock holding times does not only make its scalability substantially better compared to the old CA tree variants but also much better than the other recently proposed data structures with linearizable range query support.

This paper starts with a high-level description of CA trees (Section 2). The new CA tree variant and its implementation are described in Sections 3 and 4. Analytical and experimental comparisons with related data structures are given in Sections 5 and 6. The paper finishes with a conclusion (Section 7).

2 High-Level Description of Contention Adapting Search Trees

CA trees are structured as depicted in Figure 1. The items that are stored in a CA tree are located in sequential data structures under the base nodes (see Figure 1). To efficiently find a specific item in a CA tree, the search is directed by the keys in the routing nodes. All items stored under the left branch of a routing node have keys that are less than the key of the routing node and all items stored under the right branch have keys that are greater than or equal to the key in the routing node. The sequential data structures are protected from concurrent accesses by locks in the base nodes. A base node lock has a statistics counter which is incremented when a thread needs to wait to acquire the lock and decremented when no waiting is required. If the statistics counter in a base node reaches a certain threshold, the items stored in the base node are split between two new base nodes to reduce the contention; see Figure 2a and Figure 2b. In the reverse direction, if the statistics counter in a base node reaches the threshold for low contention adaption, the items in the base node and a neighbor

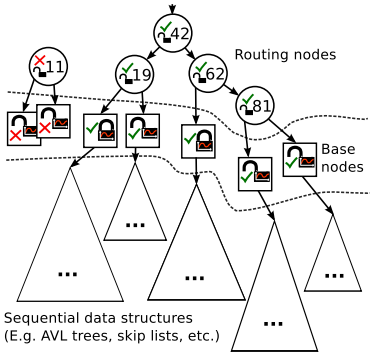
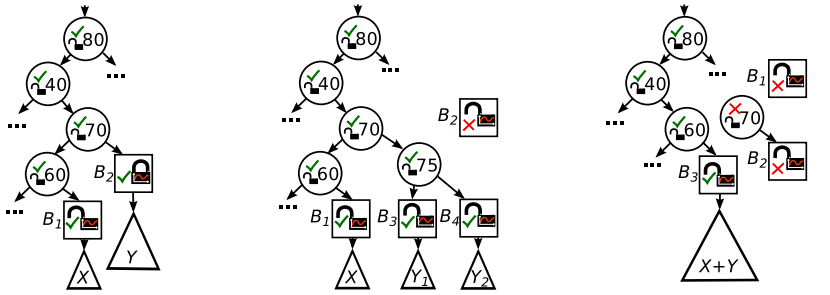


Figure 1 The structure of a CA tree. Numbers denote keys.



(a) Initial CA tree. **(b)** CA tree after a split. **(c)** CA tree after a join

Figure 2 Effect of the split and join operations on the CA tree of Figure 2a.

base node are joined into one new base node; see Figure 2a and Figure 2c. Base nodes also have a valid flag (depicted by ✓ and ✗) which is used to indicate if a base node is in the CA tree or if it has been removed. Operations that end up in an invalid (✗) base node need to retry the search until they end up in a valid (✓) base node. Routing nodes also have a valid flag and a lock that are only used rarely during the low contention optimizing join. Range queries are performed in CA trees by first finding and locking the base node containing the first key in the range and then traversing and locking subsequent base nodes until a base node containing a key which is equal to or greater than the largest key in the range is found.

An optimization that has been shown to greatly enhance performance of read operations (lookup and range query) is to let read operations optimistically attempt to do their operation without writing to shared memory [18, 19]. This can be done by using a *sequence lock* [11] as base node lock. A sequence lock has an operation to read a sequence number from the lock. If a thread gets the same even sequence number from two calls of this operation, then the sequence lock guarantees that the lock has not been acquired between the two calls. An optimistic attempt of a read operation first scans the sequence numbers and checks the valid flags of the base nodes that the operation needs to read data from, and then performs the operation, after which the sequence numbers from the locks have to be checked again to make sure that the sequence numbers match the previously read sequence numbers. If the optimistic attempt fails, the operation is done by acquiring the base node locks in read-mode (several read-mode lock holders can hold the lock at the same time).

The reader is referred to the earlier papers on CA trees [18, 19, 20] for a detailed description including pseudo-code and arguments that their operations provide linearizability, deadlock freedom, and livelock freedom.

3 CA Tree Optimization Enabled by Immutable Data Structures

By using a mutable reference to an immutable data structure as a CA tree's sequential data structure, it is straightforward to reduce the amount of time that read operations spend on reading shared mutable data. Assuming that the CA tree's sequential data structure component is implemented with a mutable reference to an immutable data structure, a lookup or range query operation only needs to copy the values of the references that are needed by the operation while traversing shared mutable data. The immutable data structures referenced by the copied references are then traversed after the base node locks have been unlocked (or after the second sequence number scan).

Especially for range queries, this optimization can give a large reduction of the amount of time which is spent on reading shared mutable data as the time that range queries need to spend on traversing the sequential data structures is at least linear in the number of items in the range. With the optimization, range queries may only need to traverse a few base nodes (one in the best case) while reading shared mutable data even when the number of items in the range is large.

It is straightforward to see that this optimization does not jeopardize correctness as the result of a read operation would be the same if the traversal of the sequential data structures happened instantly at the linearization point (due to the immutability of the data structures referenced by the copied references).

4 The Implementation of the Optimized CA Tree

To experimentally evaluate the optimization described in the previous section, a CA tree using a mutable reference to an immutable treap [21] as its sequential data structure has been implemented in Java. A treap is a self-balancing binary search tree with expected time complexity of $\mathcal{O}(\log n)$ for insert, remove and lookup and an expected time complexity of $\mathcal{O}(\log n + r)$ for range queries, where n is the number of items stored in the data structure and r is the number of items in the range. The treap also has efficient split and join operations [21] which is important for the CA tree's low and high contention adaptions [18]. To facilitate cache friendly range queries, the treap implementation stores all items in fat leaf nodes containing arrays that can store up to 64 items.

One heuristic, that CA trees use to reduce the time that future similar range queries need to spend on traversing base nodes, is to decrement the contention statistics counters in the locks of base nodes needed by a range query, if more than one base node is needed; cf. [19]. With the optimization described in the previous section in place, the portion of a range query that is spent on traversing shared mutable data is even more affected by the number of base nodes that the range query needs to access than without the optimization. The reason for this is that the optimization moves the traversal of the sequential data structures from within the period that is spent on reading shared mutable data to after this period. It therefore makes sense to decrement the contention statistics counters with a larger value, when the optimization is used, as the potential benefit for similar range queries is larger than without the optimization. Indeed, experiments show that changing the heuristic to decrement by the value 100 instead of the value one (which is used by the old CA tree implementations) gives significantly better performance in scenarios with large range queries. Except for the change of the value used to decrement the statistics counters in the described heuristic, the immutable treap version of the CA tree has the same constants and thresholds for low and high contention adaptions as described in the previous work [19].

5 Related Work

The CA tree is the only one of the previously proposed approaches for linearizable range queries [4, 5, 2, 17, 7, 3] that dynamically changes the synchronization granularity to optimize for the conflicting requirements of range queries of different sizes and single-key operations [19].

The *SnapTree* by Bronson *et al.* [4] has an efficient linearizable clone operation that returns a copy of the data structure from which a range query operation can easily be derived. *SnapTree*'s clone operation waits for active update operations and forces subsequent update operations to copy nodes lazily before node modifications so that the copy is not modified. The behavior of *SnapTree* resembles the behavior of the data structure implemented from a single mutable reference to an immutable data structure (that is discussed in Section 1) when range queries are common. The *SnapTree* can thus serve as an example of the coarse-grained approach for doing range queries.

The lock-free k-ary search tree is an unbalanced external search tree with up to k keys stored in every node [6]. Range queries in k-ary search trees are performed by doing a read scan and a validation scan of the immutable leaf nodes containing items in the range [5]. The range query operation needs to retry if the validation scan fails. The k-ary is an example of the fine-grained approach discussed in Section 1. Another example of this fine-grained approach based on software transactional memory is the Leaplist [2].

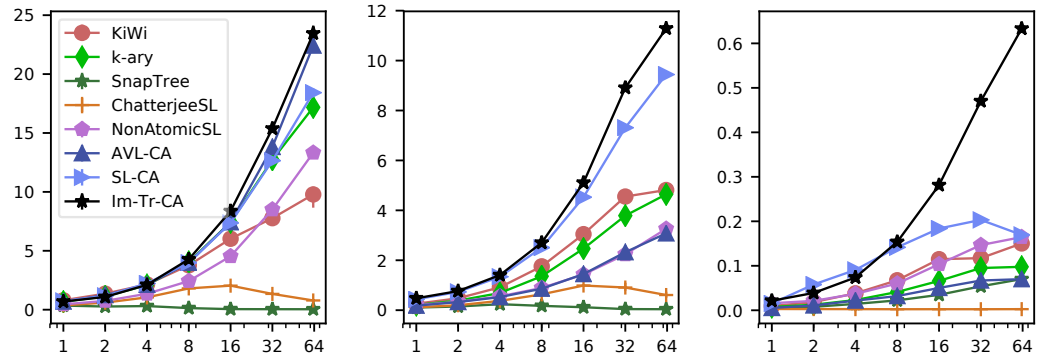
Chatterjee has proposed a general method for doing range queries in lock-free ordered set data structures [7] based on the work by Erez and Shahar [15]. Unfortunately, the scalability of Chatterjee's method suffers from the global sequential hot spot in the list of range-collector objects that all range queries have to write to in the worst case.

The *KiWi* data structure by Basin *et al.* [3] supports wait-free range queries and lookup operations as well as lock-free update operations. Update operations help range queries by storing multiple versions of inserted items when it is needed for the range queries. Similarly to Robertson's data structure [17], *KiWi*'s range queries atomically increment a global version counter which is used by update operations to decide if storing an additional version is necessary. *KiWi*'s global version number counter is bound to become a scalability bottleneck with a high enough level of parallelism. Similarly to the treap based CA tree, *KiWi* tries to improve cache locality by storing items in arrays that can store up to k items.

A fundamental difference between the other efficient methods for range queries in ordered set data structures and the optimized CA tree is the time spent by range queries reading shared mutable data and thus the time in which conflicts with update operations can happen. In the other methods [5, 2, 17, 19, 7, 3], this time is at least linear in the number of items inside the range while the optimized CA tree can do much better as is described in Section 3.

6 Evaluation

We now experimentally evaluate the optimized CA tree implementation using the immutable treap described in Section 4 (*Im-Tr-CA*). *Im-Tr-CA* will be compared to the recently proposed methods for doing linearizable range queries in ordered sets: *SnapTree* [4], k-ary [5], Chatterjee's method applied to a lock-free skiplist [7] (*ChatterjeeSL*), *KiWi* [3] and the two CA tree variants that use a mutable skiplist with fat nodes (*SL-CA*) and a mutable AVL tree (*AVL-CA*) as sequential data structures [19]. The lock-free skiplist, called ConcurrentSkipListMap, from the Java library that only supports range queries that are not linearizable (*NonAtomicSL*) is also included in the comparison. All data structures were provided by their respective authors and are implemented in Java. The maximum number of items in



(a) w:20% r:55% q:25%-10 **(b)** w:20% r:55% q:25%-1000 **(c)** w:20% r:55% q:25%-100000

Figure 3 Throughput (operations/μs) on the y-axis and thread count on the x-axis.

the nodes (k) is set to 64 for k-ary, *Im-Tr-CA* and *SL-CA* as this value has previously been shown to give good results [5]. *KiWi*'s constants are set as described in the *KiWi* paper [3].

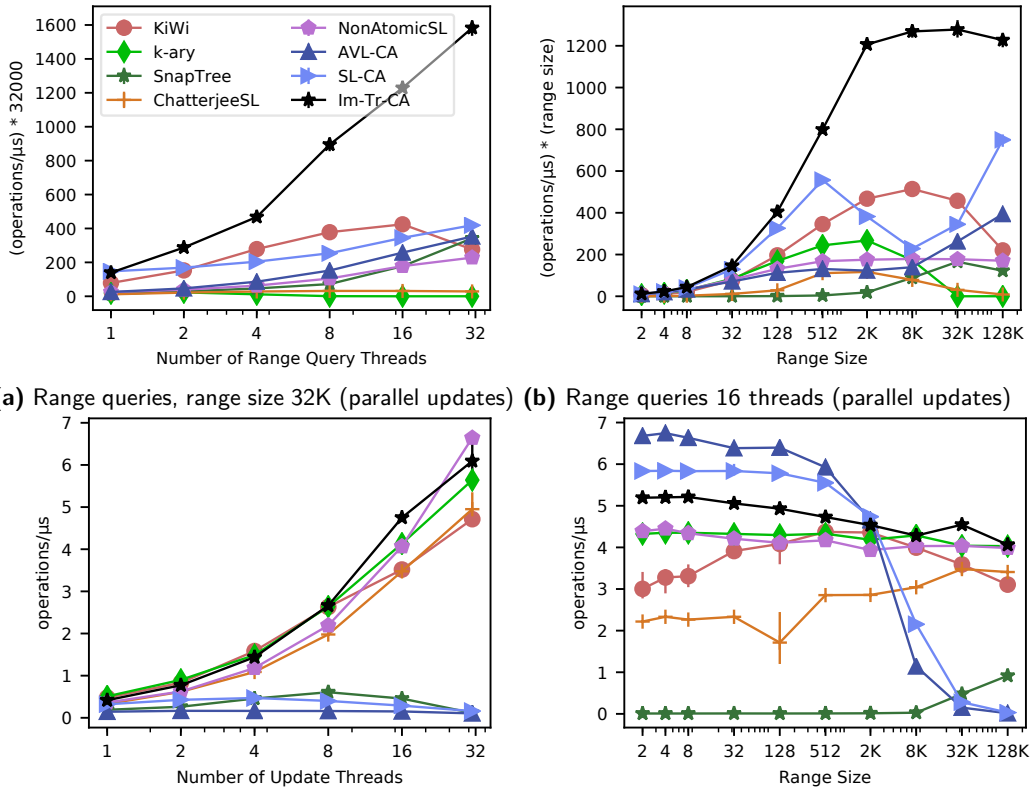
The benchmarks were run on a machine with four Intel(R) Xeon(R) E5-4650 CPUs (2.70GHz each with eight cores and hyperthreading, giving a total of 32 actual and 64 logical cores), turbo boost turned off, 128GB of RAM, running Linux 3.16.0-4-amd64 and Oracle JVM 1.8.0_131 (with the JVM flags -Xmx8g -Xms8g -XX:+UseCondCardMark -server -d64). Each data point comes from the average of three measurements runs of 10 seconds each that were preceded by 3 warm up runs, also of 10 seconds each. The purpose of the warm up runs is to give the just-in-time compiler enough time to compile the code. Error bars showing the minimum and maximum measurements are displayed when they are large enough to be seen.

The keys for the operations lookup, insert and remove as well as the starting keys for range queries are randomly generated from a range of size S. The data structure is pre-filled before the start of each benchmark run by performing $S/2$ random insert operations. In all experiments presented in the main part of this paper $S = 10^6$. The interested reader can find results for $S = 10^5$ and $S = 10^7$ in the appendix of this paper. Range queries calculate the sum of the items in the range and the number of items in the range. As a sanity check, the average number of items that are traversed per range query is calculated and checked against the expected value.

The Random Operations Benchmark. This benchmark measures throughput of a mix of operations performed by N threads. In all captions, benchmark scenarios are described by strings of the form w:A% r:B% q:C%-R, meaning that the benchmark performs $(A/2)$% insert, $(A/2)$% remove, B% lookup operations and C% range queries of maximum range size R. The range sizes are randomly set to values between 1 and R.

Figure 3 shows the results from three scenarios with increasing range sizes. In the scenario with small range queries of maximum size 10 (Figure 3a), the best performing data structures (the CA trees and k-ary) are almost indistinguishable. *ChatterjeeSL* and *KiWi* that have a global scalability bottleneck, as explained in the previous section, both scale worse in this scenario. *Im-Tr-CA* scales well in this scenario due to its ability to adapt the sizes of its immutable parts but does not get much benefit from its quick traversal of shared mutable data as conflicts between threads are rare for all data structures in this scenario.

Conflicts are still relatively rare in the scenario with range queries of maximum size 1000 (Figure 3b). The top performing data structures in this scenario (*Im-Tr-CA*, *SL-CA*, *KiWi* and k-ary) are those with cache locality friendly nodes that store several items in arrays.



(a) Range queries, range size 32K (parallel updates) **(b)** Range queries 16 threads (parallel updates)

(c) Updates (parallel range queries, range size 32K) **(d)** Updates 16 threads (parallel range queries)

Figure 4 Results for benchmark with separate threads doing updates and range queries.

However, *Im-Tr-CA* and *SL-CA*, that adapt their synchronization granularity to the scenario at hand, outperform *KiWi* and *k*-ary with a wide margin in this scenario.

The scenario that has large range queries of maximum size 100000 (Figure 3c) shows the distinguishing feature of *Im-Tr-CA* that outperforms all the other data structures with a large margin. Conflicts between range queries and update operations are very likely with these large range queries but the conflicts are significantly less costly in *Im-Tr-CA* due to its short critical sections, as is explained in Section 3.

Separate Threads for Range Queries and Updates. Most of the time is spent doing range queries when large range queries are used in the random operations benchmark. Thus, another benchmark is needed to measure the data structures' ability to handle large range queries concurrently with frequent update operations. To this end, we use a similar benchmark to the one developed by the *KiWi* authors. This benchmark is motivated by large scale applications that require quick updates of a data set while other threads do large linearizable range queries concurrently (for analytics) [3]. In this benchmark, half the threads do update operations (insert and remove with equal probability) while the other half do range queries with a range of fixed size. The throughput for updates is presented separately from the range query throughput so that one can study the performance of these operations separately. Note that in the graphs that show the range query throughput, the number of operations per μs multiplied by the range query size is shown on the y-axis to make the graphs more readable.

Table 1 Statistics for *Im-Tr-CA* in the scenarios displayed in Figure 4b and Figure 4d.

Range Size	2	4	8	32	128	512	2K	8K	32K	128K
# base nodes	2.5K	2.1K	1.7K	1.0K	590	390	310	310	390	430
$\frac{\text{\# traversed base nodes}}{\text{\# range queries}}$	1.0	1.0	1.0	1.0	1.1	1.2	1.6	3.5	13	56

Figure 4a and Figure 4c show the results of this benchmark with a range query size of 32K and with varying thread counts. In Figure 4b and Figure 4d, the thread count is fixed to 32 (16 updaters and 16 threads doing range queries) and the x-axis shows varying range query sizes. First of all, *Im-Tr-CA* with its short range query critical section is overall the fastest data structure in the scenarios. *KiWi* is the second most performant data structure in the scenarios with range query sizes larger than 2000. The bumpy performance of *CA-SL* in Figure 4b can be explained by the fact that the range queries acquire the base node locks in read-mode which enables concurrent range queries to bypass waiting update operations and take over the lock.

CA-SL thus provides good throughput for range queries with a range size of 512 because conflicts are rare and with a range size of 128K as "conflicts" with other range queries that have already acquired the relevant base nodes are common.

ChatterjeeSL's and *KiWi*'s update operations need to read the RangeCollector list (*ChatterjeeSL*) or the global version number (*KiWi*) which are updated by range queries. The more frequent updates of these global objects with smaller range queries can explain the slight partial upward trend that exists for *ChatterjeeSL* and *KiWi* in Figure 4d.

k-ary's range queries are starved by update operations in the scenarios with large range queries. The *SnapTree*'s operations are slow in most scenarios due to its coarse-grained approach for doing range queries, but the *SnapTree*'s performance is better in scenarios with larger and less frequent range queries.

Table 1 shows the average number of base nodes and the number of base nodes traversed per range query in *Im-Tr-CA* after the benchmark runs of the scenarios displayed in Figure 4b and Figure 4d. It is evident from the table data that *Im-Tr-CA*'s range queries spend a very short time traversing shared mutable data even for large range queries (e.g. approximately the time it takes to traverse 13 base nodes in the case with range queries of size 32K). After the base nodes have been traversed, the collected immutable data can be traversed without any need to care about other threads and without disturbing other threads.

7 Conclusion

A new CA tree variant that makes use of an immutable data structure has been presented. The advantage of the new CA tree variant over the CA tree variants that use mutable data structures as the sequential data structure component is that the new variant drastically reduces the time period in which conflicts between large range queries and other operations can happen. Compared to all other data structures with linearizable range query support, the CA trees have the advantage that they dynamically adapt the synchronization granularity to fit the workload at hand. The experimental comparison shows that the presented implementation's quick traversal of shared mutable data and cache friendly design makes the implementation outperform the best of the other data structures with a wide margin in scenarios with large range queries. Furthermore, the new CA tree variant also performs better or close to the best of the other data structures in scenarios with small range queries due to its ability to dynamically change its synchronization granularity. As future work, we plan to design and evaluate a lock-free CA tree variant. A lock-free CA tree variant could potentially give even better performance as it could avoid priority inversions and other lock related problems.

Acknowledgments. Vincent Gramoli gave me the idea of looking into immutable data structures in combination with the CA tree. Amelie Lind, Martin Viklund, Stephan Brandauer, Andreas Löscher, Elias Castegren and Konstantinos Sagonas helped me improve the language.

References

1 Flurry analytics. https://developer.yahoo.com/flurry/docs/analytics/. Accessed: 2017-07-26.

2 Hillel Avni, Nir Shavit, and Adi Suissa. Leaplist: lessons learned in designing tm-supported range queries. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 299–308. ACM, 2013. doi:10.1145/2484239.2484254.

3 Dmitry Basin, Edward Bortnikov, Anastasia Braginsky, Guy Golan-Gueta, Eshcar Hillel, Idit Keidar, and Moshe Sulamy. KiWi: A key-value map for scalable real-time analytics. In *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '17, pages 357–369, New York, NY, USA, 2017. ACM. doi:10.1145/3018743.3018761.

4 Nathan Grasso Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. A practical concurrent binary search tree. In R. Govindarajan, David A. Padua, and Mary W. Hall, editors, *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2010, Bangalore, India, January 9-14, 2010*, pages 257–268. ACM, 2010. doi:10.1145/1693453.1693488.

5 Trevor Brown and Hillel Avni. Range queries in non-blocking k-ary search trees. In Roberto Baldoni, Paola Flocchini, and Binoy Ravindran, editors, *Principles of Distributed Systems, 16th International Conference, OPODIS 2012, Rome, Italy, December 18-20, 2012. Proceedings*, volume 7702 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2012. doi:10.1007/978-3-642-35476-2_3.

6 Trevor Brown and Joanna Helga. Non-blocking k-ary search trees. In Antonio Fernández Anta, Giuseppe Lipari, and Matthieu Roy, editors, *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*, volume 7109 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2011. doi:10.1007/978-3-642-25873-2_15.

7 Bapi Chatterjee. Lock-free linearizable 1-dimensional range queries. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, pages 9:1–9:10, New York, NY, USA, 2017. ACM. doi:10.1145/3007748.3007771.

8 Keir Fraser. *Practical lock-freedom.* PhD thesis, University of Cambridge Computer Laboratory, 2004.

9 Maurice Herlihy. A methodology for implementing highly concurrent data structures. In David A. Padua, editor, *Proceedings of the Second ACM SIGPLAN Symposium on Princiles & Practice of Parallel Programming (PPOPP), Seattle, Washington, USA, March 14-16, 1990*, pages 197–206. ACM, 1990. doi:10.1145/99163.99185.

10 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

11 Christoph Lameter. Effective synchronization on Linux/NUMA systems. In *Proc. of the Gelato Federation Meeting*, 2005. URL: http://www.kde.ps.pl/mirrors/ftp.kernel.org/linux/kernel/people/christoph/gelato/gelato2005-paper.pdf.

12 Maged M. Michael. High performance dynamic lock-free hash tables and list-based sets. In *SPAA*, pages 73–82, 2002. doi:10.1145/564870.564881.

13 Aravind Natarajan and Neeraj Mittal. Fast concurrent lock-free binary search trees. In José E. Moreira and James R. Larus, editors, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '14, Orlando, FL, USA, February 15-19, 2014*, pages 317–328. ACM, 2014. doi:10.1145/2555243.2555256.

14 Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

15 Erez Petrank and Shahar Timnat. Lock-free data-structure iterators. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2013. doi:10.1007/978-3-642-41527-2_16.

16 Aleksandar Prokopec, Nathan Grasso Bronson, Phil Bagwell, and Martin Odersky. Concurrent tries with efficient non-blocking snapshots. In J. Ramanujam and P. Sadayappan, editors, *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2012, New Orleans, LA, USA, February 25-29, 2012*, pages 151–160. ACM, 2012. doi:10.1145/2145816.2145836.

17 Callum Robertson. Implementing contention-friendly range queries in non-blocking key-value stores. Bachelor thesis, The University of Sydney, 2014.

18 Konstantinos Sagonas and Kjell Winblad. Contention adapting search trees. In Daniel Grosu, Hai Jin, and George Papadopoulos, editors, *14th International Symposium on Parallel and Distributed Computing, ISPDC 2015, Limassol, Cyprus, June 29 - July 2, 2015*, pages 215–224. IEEE Computer Society, 2015. doi:10.1109/ISPDC.2015.32.

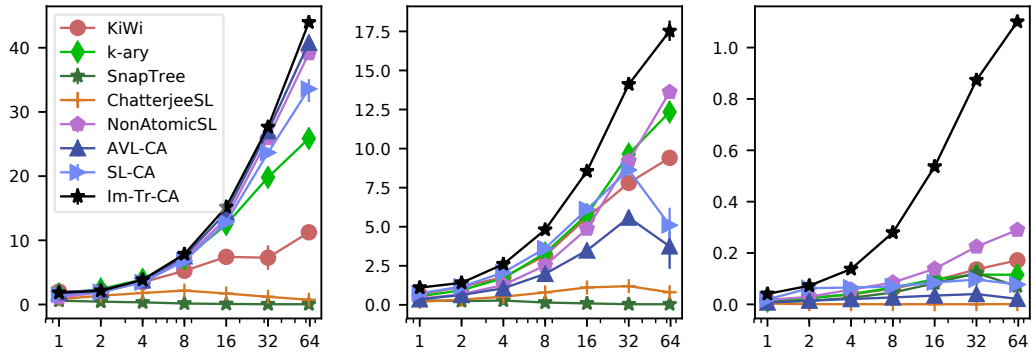
19 Konstantinos Sagonas and Kjell Winblad. Efficient support for range queries and range updates using contention adapting search trees. In Xipeng Shen, Frank Mueller, and James Tuck, editors, *Languages and Compilers for Parallel Computing - 28th International Workshop, LCPC 2015, Raleigh, NC, USA, September 9-11, 2015, Revised Selected Papers*, volume 9519 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2015. doi:10.1007/978-3-319-29778-1_3.

20 Konstantinos Sagonas and Kjell Winblad. A contention adapting approach to concurrent ordered sets. *Journal of Parallel and Distributed Computing*, Forthcoming.

21 Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996. doi:10.1007/BF01940876.

22 Ori Shalev and Nir Shavit. Split-ordered lists: Lock-free extensible hash tables. *J. ACM*, 53(3):379–405, 2006. doi:10.1145/1147954.1147958.

23 Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed SQL database that scales. *Proceedings of the VLDB Endowment*, 6(11):1068–1079, 2013. doi:10.14778/2536222.2536232.



(a) w:20% r:55% q:25%-10 **(b)** w:20% r:55% q:25%-1000 **(c)** w:20% r:55% q:25%-100000

Figure 5 Results with key range of size 10^5 which corresponds to a set size of approximately 5×10^4. Throughput (operations/μs) on the y-axis and thread count on the x-axis.

Table 2 Statistics for *Im-Tr-CA* in the scenarios displayed in Figure 6b and Figure 6d.

Range Size	2	4	8	32	128	512	2K	8K	32K	128K
# base nodes	890	720	570	330	190	120	97	130	90	96
$\frac{\#\text{ traversed base nodes}}{\#\text{ range queries}}$	1.0	1.0	1.1	1.1	1.2	1.6	3.0	11	27	98

Table 3 Statistics for *Im-Tr-CA* in the scenarios displayed in Figure 8b and Figure 8d.

Range Size	2	4	8	32	128	512	2K	8K	32K	128K
# base nodes	6.0K	5.3K	4.5K	2.9K	1.8K	1.2K	900	860	1.0K	1.1K

A Source Code

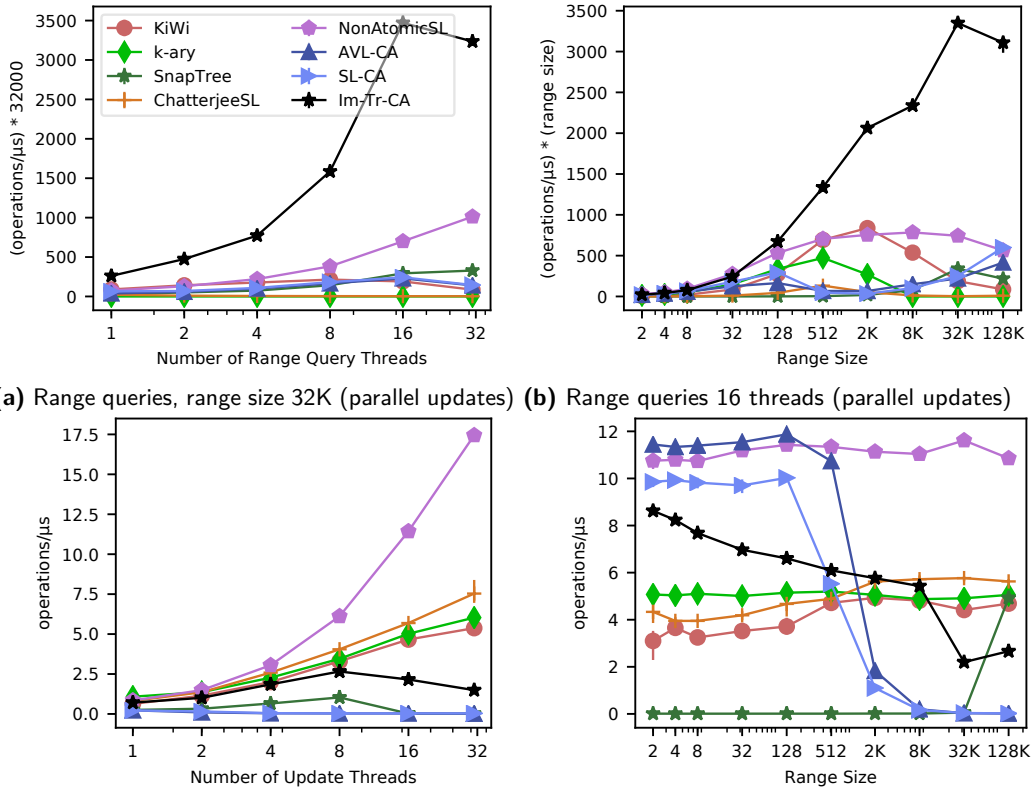
The source code for the CA tree implementations and the benchmarks can be found online (`https://www.it.uu.se/research/group/languages/software/im_tr_ca`).

B Results with Other Set Sizes

Results corresponding to the results in figures 3 and 4 but with smaller set sizes (the key range size $S = 10^5$) can be found in figures 5 and 6. The corresponding results for larger set sizes ($S = 10^7$) can be found in figures 7 and 8. The statistics corresponding to the statistics in Table 1 but with the smaller and larger set sizes can be found in tables 2 and 3.

In the cases with the smallest set size ($S = 10^5$), the ranges of size 32K and 128K span 32% and 100% of the set represented by the data structures; see Figure 6. *Im-Tr-CA*'s range queries lock out update operations from the portion of the set that is covered by the range query. Even though this only happens for a short period of time as Table 2 shows, it still has a negative effect on update operations as Figure 6d shows. This is compensated by *Im-Tr-CA*'s excellent performance for range queries in these scenarios as Figure 6b shows.

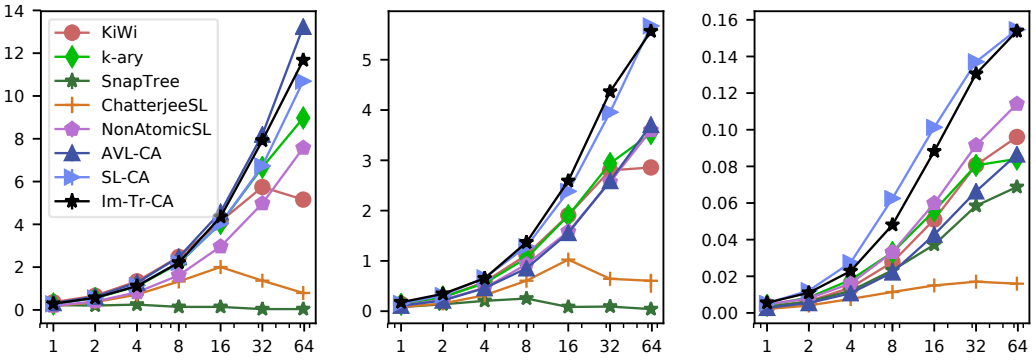
In the cases with the largest set size ($S = 10^7$), the ranges span a smaller part of the sets represented by the data structures which explains why many of the other data structures are closer to *Im-Tr-CA* in these scenarios; see figures 7 and 8.



(a) Range queries, range size 32K (parallel updates) **(b)** Range queries 16 threads (parallel updates)

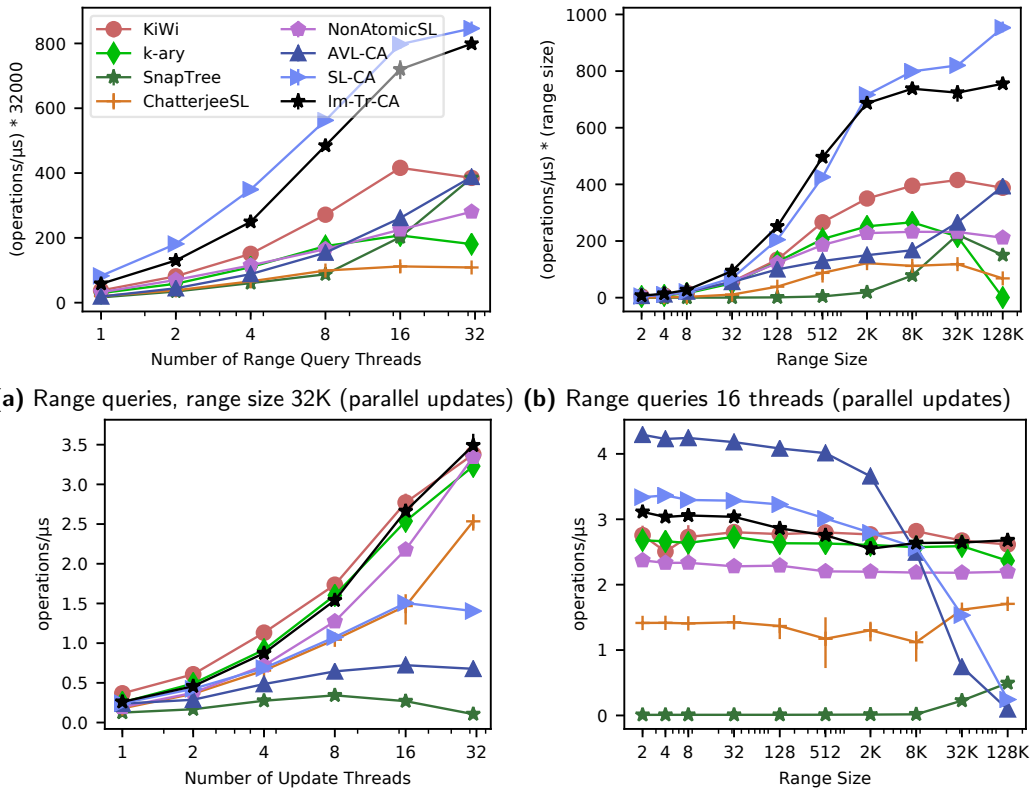
(c) Updates (parallel range queries, range size 32K) **(d)** Updates 16 threads (parallel range queries)

Figure 6 Results for benchmark with separate threads doing updates and range queries with key range of size 10^5 which corresponds to a set size of approximately 5×10^4.



(a) w:20% r:55% q:25%-10 **(b)** w:20% r:55% q:25%-1000 **(c)** w:20% r:55% q:25%-100000

Figure 7 Results with key range of size 10^7 which corresponds to a set size of approximately 5×10^6. Throughput (operations/μs) on the y-axis and thread count on the x-axis.



(a) Range queries, range size 32K (parallel updates) **(b)** Range queries 16 threads (parallel updates)

(c) Updates (parallel range queries, range size 32K) **(d)** Updates 16 threads (parallel range queries)

Figure 8 Results for benchmark with separate threads doing updates and range queries with key range of size 10^7 which corresponds to a set size of approximately 5×10^6.

Gesture Recognition and Classification using Intelligent Systems

Norah Alnaim[1] and Maysam Abbod[2]

1 **Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB8 3PH, United Kingdom**
Norah.alnaim@brunel.ac.uk

2 **Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB8 3PH, United Kingdom**
Maysam.abbod@brunel.ac.uk

Abstract

Gesture Recognition is defined as non-verbal human motions used as a method of communication in HCI interfaces. In a virtual reality system, gestures can be used to navigate, control, or interact with a computer. Having a person make gestures formed in specific ways to be detected by a device, like a camera, is the foundation of gesture recognition. Finger tracking is an interesting principle which deals with three primary parts of computer vision: segmentation of the finger, detection of finger parts, and tracking of the finger. Fingers are most commonly used in varying gesture recognition systems.

Finger gestures can be detected using any type of camera; keeping in mind that different cameras will yield different resolution qualities. 2-dimensional cameras exhibit the ability to detect most finger motions in a constant surface called 2-D. While the image processes, the system prepares to receive the whole image so that it may be tracked using image processing tools. Artificial intelligence releases many classifiers, each one with the ability to classify data, that rely on its configuration and capabilities. In this work, the aim is to develop a system for finger motion acquisition in 2-D using feature extraction algorithms such as Wavelets transform (WL) and Empirical Mode Decomposition (EMD) plus Artificial Neural Network (ANN) classifier.

WL is an image processing algorithm that performs signal analysis with one signal frequency differing at the end of time. EMD is an innovative technology used in both non-stationary and non-linear data. The primary function of this method is decomposing a signal into Intrinsic Mode Functions consistently through the domain. For classification, ANN is used which is defined as a system that processes information and has structure much like that of the biological nervous system. What is most unique is that this system inhibits an abstract but familiar structure as an information processing system.

In this work, three different finger motions are recorded using an iPhone 6s Plus camera. The gesture classification system is developed for three types of finger gesture recognition. WL and EMD algorithms are used to extract features which are fed to ANN for gesture classification. The classification results of training, validation, and testing mean square error using WL are 5.1312E-4/0.01245/0.0079 respectively, while the classification mean square error using EMD are 1.1035E-11/9.676E-09/2.5616E-9 respectively. Feature extraction execution time, in seconds, for Wavelet Transform is 131 and EMD is 7200. The classification accuracy for training, validation, and testing using WT are 0.9984/0.9909/0.9953 and using EMD are 1.0/1.0/1.0. The results of this experiment clearly identify EMD being a suitable method to extract features from the image but it is time consuming.

1998 ACM Subject Classification H.5.2 User Interfaces, I.2.6 Learning

Keywords and phrases Wavelets, Empirical Model Decomposition, Artificial Neural Network, Gesture Recognition, HCI

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.8

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 8; pp. 8:1–8:1

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICCSW17

KubeNow: A Cloud Agnostic Platform for Microservice-Oriented Applications*

Marco Capuccini[1], Anders Larsson[2], Salman Toor[3], and Ola Spjuth[4]

1 **Department of Information Technology, Uppsala University and Department of Pharmaceutical Biosciences, Uppsala University, Sweden**
marco.capuccini@it.uu.se; marco.capuccini@farmbio.uu.se
2 **Department of Cell and Molecular Biology, Uppsala University, Sweden**
anders.larsson@icm.uu.se
3 **Department of Information Technology, Uppsala University, Sweden**
salman.toor@it.uu.se
4 **Department of Pharmaceutical Biosciences, Uppsala University, Sweden**
ola.spjuth@farmbio.uu.se

Abstract

KubeNow is a platform for rapid and continuous deployment of microservice-based applications over cloud infrastructure. Within the field of software engineering, the microservice-based architecture is a methodology in which complex applications are divided into smaller, more narrow services. These services are independently deployable and compatible with each other like building blocks. These blocks can be combined in multiple ways, according to specific use cases. Microservices are designed around a few concepts: they offer a minimal and complete set of features, they are portable and platform independent, they are accessible through language agnostic APIs and they are encouraged to use standard data formats. These characteristics promote separation of concerns, isolation and interoperability, while coupling nicely with test-driven development. Among many others, some well-known companies that build their software around microservices are: Google, Amazon, PayPal Holdings Inc. and Netflix [11].

Cloud computing is a new technology trend that enables the allocation of virtual infrastructure on demand, giving place to a new business model where organizations can purchase resources with a pay-per-use pricing arrangement [8]. Microservices in cloud environments can help to build scalable and resilient applications, with the goal of maximing resource usage and reducing costs. At the time of writing, Docker and Kubernetes are the most broadly adopted container engine and container orchestration framework [10, 9]. Even though these software tools ease microcroservices operations considerably, their setup and configuration is still complex, tedious and time consuming. When allocating cloud resources on demand this becomes a critical issue, since applications need to be continuously deployed and scaled, possibly over different cloud providers, to minimize infrastructure costs. This new challenging way of provisioning infrastructure was the main motivation for the development of KubeNow.

KubeNow provides the means to rapidly deploy fully configured clusters, automating Docker and Kubernetes configuration, while providing a mechanism for the application layer setup. We designed KubeNow using the Infrastructure as Code (IaC) paradigm, meaning that the virtual resources and the provisioning process are defined as machine-readable language. A natural consequence of this choice is that KubeNow is immutable and repeatable over different cloud providers, being cloud agnostic in this sense. In addition, IaC enables infrastructure version control and collaborative development.

KubeNow has been adopted by the PhenoMeNal H2020 consortium as the platform used to launch on demand Cloud Research Environments (CRE) [6]. The PhenoMeNal CRE allows

* This work was supported by the PhenoMeNal H2020 consortium.

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 9; pp. 9:1–9:2

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for running reproducible large-scale medical metabolomics analysis. In addition, we are currently developing additional software layers for large-scale analysis on top of KubeNow including: Apache Spark [12], Pachyderm [5] and Slurm [7]. KubeNow supports Amazon Web Services [1], Google Compute Engine [2] and OpenStack [4]. The software is generally applicable and publicly available as open source on GitHub [3].

1998 ACM Subject Classification D.2.11 Software Architectures

Keywords and phrases Microservices, Cloud computing, Infrastructure as Code, Docker, Kubernetes

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.9

References

1 Amazon Web Services. `https://aws.amazon.com`. [Online; accessed 03-07-2017].
2 Google Compute Engine. `https://cloud.google.com/compute`. [Online; accessed 03-07-2017].
3 KubeNow repository. `https://github.com/kubenow/KubeNow`. [Online; accessed 03-07-2017].
4 OpenStack. `hhttps://www.openstack.org`. [Online; accessed 03-07-2017].
5 Pachyderm. `http://pachyderm.io/`. [Online; accessed 03-07-2017].
6 PhenoMeNal. `http://phenomenal-h2020.eu/home/`. [Online; accessed 03-07-2017].
7 Slurm. `https://slurm.schedmd.com/`. [Online; accessed 03-07-2017].
8 Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.
9 Matt Asay. Why Kubernetes is winning the container war. `http://www.infoworld.com/article/3118345/cloud-computing/why-kubernetes-is-winning-the-container-war.html`, sep 2016. [Online; accessed 03-07-2017].
10 Alan Shimel. Docker becomes de facto Linux standard. `http://www.networkworld.com/article/2226751/opensource-subnet/docker-becomes-de-facto-linux-standard.html`, 2016. [Online; accessed 03-07-2017].
11 C. L. Williams, J. C. Sica, R. T. Killen, and U. G. Balis. The growing need for microservices in bioinformatics. *J Pathol Inform*, 7:45, 2016.
12 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=1863103.1863113`.