Raghunath Nambiar
Meikel Poess (Eds.)

LNCS 14247

Performance Evaluation and Benchmarking

15th TPC Technology Conference, TPCTC 2023
Vancouver, BC, Canada, August 28 – September 1, 2023
Revised Selected Papers

Lecture Notes in Computer Science **14247**

The series Lecture Notes in Computer Science (LNCS), including its subseries Lecture Notes in Artificial Intelligence (LNAI) and Lecture Notes in Bioinformatics (LNBI), has established itself as a medium for the publication of new developments in computer science and information technology research, teaching, and education.

LNCS enjoys close cooperation with the computer science R & D community, the series counts many renowned academics among its volume editors and paper authors, and collaborates with prestigious societies. Its mission is to serve this international community by providing an invaluable service, mainly focused on the publication of conference and workshop proceedings and postproceedings. LNCS commenced publication in 1973.

Raghunath Nambiar · Meikel Poess
Editors

Performance Evaluation and Benchmarking

15th TPC Technology Conference, TPCTC 2023
Vancouver, BC, Canada, August 28 – September 1, 2023
Revised Selected Papers

Springer

Editors
Raghunath Nambiar
Advanced Micro Devices Inc.
Santa Clara, CA, USA

Meikel Poess
Oracle Corporation
Redwood City, CA, USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-031-68030-4 ISBN 978-3-031-68031-1 (eBook)
https://doi.org/10.1007/978-3-031-68031-1

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Founded in August 1988, the Transaction Processing Performance Council (TPC) is a non-profit organization that has wielded considerable influence over the computing industry's adoption of standardized benchmarks for over 35 years. These benchmarks serve as a means for vendors to showcase the performance competitiveness of their current offerings and to enhance and track the performance of products in development. Additionally, many purchasers rely on TPC benchmark results as benchmarks for evaluating and comparing new computing systems, whether for on-premises or public cloud deployment.

This volume presents the proceedings of the 15th TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2023), held concurrently with the 49th International Conference on Very Large Data Bases (VLDB 2023) in Vancouver, Canada, on August 28th, 2023. Over the years, TPCTC has evolved into a vital platform for the performance benchmarking community. Among the 17 papers submitted, seven were accepted, which brings this year's acceptance rate to 41%. Each paper was single-blindly peer reviewed by four peers. Additionally, the proceedings feature one paper stemming from panel discussions led by industry and research experts, as well as an invited paper from the chairman of the TPC's public relations committee, highlighting new initiatives within the TPC.

The success of this conference was a result of the dedication and collaborative efforts of numerous individuals. We extend our gratitude to the members of the TPC and the organizers of VLDB 2023 for their sponsorship, as well as to the members of the Program Committee and Publicity Committee for their invaluable support. Additionally, we would like to express our appreciation to the authors and participants whose contributions were fundamental to the triumph of this event.

Raghunath Nambiar
Meikel Poess

Redefining Performance Evaluation and Benchmarking in the Era of Artificial Intelligence

In 2017, the Transaction Processing Performance Council (TPC) revealed the establishment of a Working Group dedicated to crafting standardized benchmarks for both hardware and software platforms utilized in executing Artificial Intelligence (AI) workloads. Fast forward to 2021, the Transaction Processing Performance Council (TPC) announced the immediate availability of TPCx-AI, marking it as the inaugural industry-standard, vendor-neutral benchmark tailored for evaluating real-world AI and machine learning (ML) scenarios alongside data science use cases. TPCx-AI showcases versatility by utilizing a diverse dataset and was meticulously designed to be adaptable across a broad spectrum of scale factors.

AI has become increasingly significant in recent times. The central focus of TPCTC 2023 revolved around contributions that are primarily focused on performance methodologies within the vast realm of AI. AI encompasses a diverse spectrum mirroring the complexity and depth of human intellect it aims to emulate. Ranging from narrow AI, engineered to surpass human capability in specific tasks such as language translation, facial recognition, or chess playing, to the ambitious pursuit of general AI, capable of understanding, learning, and applying intelligence across various domains, the breadth of AI is vast. This diversity is evident not only in AI applications but also in the methodologies employed for their performance evaluation. Performance analysis of AI applications plays a pivotal role in ensuring these systems efficiently, effectively, and accurately fulfill their intended objectives. At its essence, performance analysis entails evaluating an AI system's accuracy, speed, reliability, and scalability, among other metrics. This is crucial as it directly impacts the AI's utility and applicability across diverse domains, ranging from healthcare and finance to autonomous vehicles and smart cities. By rigorously analyzing performance, developers can pinpoint and rectify weaknesses, thereby enhancing the AI's capacity to make accurate predictions efficiently, process data at requisite speeds, and operate reliably and accurately under varying conditions.

Furthermore, performance analysis is integral to optimizing resource utilization, a critical consideration given the substantial computational power and data storage that AI applications inherently rely on. Efficient resource utilization not only reduces operational costs but also minimizes environmental impact by lowering energy consumption and hardware demands. Through performance analysis, AI systems can be fine-tuned to strike a balance between performance and resource efficiency, ensuring they deliver the highest possible value to customers. This optimization is especially crucial as AI applications scale up and out, where minor inefficiencies can exponentially magnify costs.

Meikel Poess

TPCTC 2023 Organization

General Chairs

Raghunath Nambiar	AMD, USA
Meikel Poess	Oracle, USA

Program Committee

Ajay Dholakia	Lenovo, USA
Andrew Bond	Red Hat, USA
Anil Rajput	AMD, USA
Hans-Arno Jacobsen	University of Toronto, Canada
Harry Le	University of Houston, USA
John Poelman	IBM, USA
Karthik Kulkarni	Infobell IT Solutions, India
Klaus-Dieter Lange	Hewlett Packard Enterprise, USA
Michael Brey	Oracle, USA
Miro Hodak	AMD, USA
Nicholas Wakou	Dell, USA
Paul Cao	Hewlett Packard Enterprise, USA
Rodrigo D. Escobar	University of Texas at San Antonio, USA
Shahram Ghandeharizadeh	University of Southern California, USA
Tariq Magdon-Ismail	VMware, USA
Tilmann Rabl	Hasso Plattner Institute, Germany

Publicity Committee

Meikel Poess	Oracle, USA
Andrew Bond	Red Hat, USA
Paul Cao	HPE, USA
Gary Little	Nutanix, USA
Raghunath Nambiar	AMD, USA
Michael Majdalany	L&M Management Group, USA
Forrest Carman	Owen Media, USA
Andreas Hotea	Hotea Solutions, USA

About the TPC

Introduction to the TPC

The Transaction Processing Performance Council (TPC) is a non-profit organization with a primary objective of spearheading the development of industry standards tailored for data-centric workloads. Committed to promoting fairness and transparency in performance evaluation, the TPC diligently collects and disseminates vendor-neutral performance data across the industry. With a vision to foster collaboration and drive innovation, the TPC serves as a pivotal platform for industry stakeholders to benchmark and assess the efficiency of their technologies.

For further insights into the TPC's initiatives, standards, and resources, visit their official website at http://www.tpc.org/.

TPC Memberships

Full Members

Full Members of the Transaction Processing Performance Council (TPC) play a vital role in shaping the organization's activities. They are actively involved in various facets of the TPC's operations, including the formulation, refinement, and implementation of benchmark standards. These members contribute to the continuous development of benchmarks that accurately reflect the evolving landscape of data-centric workloads.

Furthermore, Full Members of the TPC actively participate in setting the strategic direction of the organization. Their insights, expertise, and perspectives help guide the TPC in its mission to promote fairness, transparency, and innovation in performance evaluation across the industry.

For those interested in becoming Full Members and contributing to the TPC's mission, the Full Member application can be accessed at http://tpc.org/information/about/join5.asp.

Associate Members

Certain organizations have the opportunity to join the Transaction Processing Performance Council (TPC) as Associate Members. While Associate Members have the privilege to attend TPC meetings, they do not possess the eligibility to vote or hold office within the organization. This membership category is accessible to a diverse range of entities, including non-profit organizations, educational institutions, market researchers, publishers, consultants, governments, and businesses that are not directly involved in the creation, marketing, or sale of computer products or services.

Associate Membership offers valuable networking and knowledge-sharing opportunities within the TPC community, enabling members to stay abreast of the latest developments and trends in performance evaluation and benchmarking. By fostering collaboration among a broad spectrum of stakeholders, Associate Membership contributes to the advancement of industry standards and practices.

For those interested in becoming Associate Members and engaging with the TPC community, the Associate Member application can be accessed at http://tpc.org/information/about/join5.asp.

Academic and Government Institutions

Academic and government institutions are extended a special invitation to become part of the Transaction Processing Performance Council (TPC) community. Recognizing the invaluable contributions of these entities to the advancement of technology and research, the TPC welcomes their participation in shaping industry standards and best practices.

By joining the TPC, academic institutions gain access to a collaborative platform where they can engage with industry experts, share knowledge, and contribute to the development of benchmark standards. Government institutions, on the other hand, have the opportunity to leverage the TPC's resources to stay informed about the latest advancements in performance evaluation and benchmarking.

This special invitation underscores the TPC's commitment to fostering collaboration and innovation across academia, government, and industry. For academic and government institutions interested in joining the TPC community, the invitation can be found at http://tpc.org/information/about/join5.asp.

Professional Affiliates

TPC Professional Affiliates are individuals appointed by the TPC, engaged in business activities that complement or contribute to fulfilling the TPC's mission. Affiliates cannot hold membership status or represent members or associate members. They must be involved in business activities aligned with the TPC's mission. The appointment of TPC affiliates is entirely at the discretion of the TPC.

Contact the TPC

TPC
2150 N 107th Street, Suite 205
Seattle, WA 98133
Voice: 415-561-6272
Fax: 415-561-6120
Email: info@tpc.org

How to Order TPC Materials

We are pleased to announce that all our materials are now available free of charge on our website. Whether you're seeking benchmark standards, research publications, or organizational information, you can access it easily and conveniently online.

Should you have any inquiries or require further assistance, please do not hesitate to reach out to our office directly. You can contact us via phone during regular business hours or send us an email at info@tpc.org. We are here to assist you and provide the information you need.

Staying Connected

The TPC provides a convenient option for users to stay updated on the latest benchmark results and changes through email notifications. By signing up for our email notification service, you will receive alerts whenever new benchmark results are published or when the primary metric of a published benchmark is modified. To subscribe to this service, simply visit http://tpc.org/information/about/mailinglist_signup5.asp and fill out the subscription form.

In addition to email notifications, the TPC maintains an active presence on social media platforms such as LinkedIn and Twitter. You can follow us on LinkedIn at https://www.linkedin.com/company/tpcbenchmarks/ to stay informed about our latest news, events, and updates. Similarly, you can connect with us on Twitter at https://twitter.com/TPCBenchmarks for real-time updates, announcements, and insights from the TPC community.

By staying connected with us through email notifications and social media channels, you can ensure that you are always up to date with the latest benchmark results, developments, and activities within the TPC ecosystem.

TPC 2023 Organization

Full Members

Actian
Alibaba
AMD
Boray Data
Cisco
Dell Technologies
Fujitsu
Hewlett Packard Enterprise
Hitachi
Huawei

IEIT Systems
Intel
Lenovo
Microsoft
Nutanix
Nvidia
Oracle
Red Hat
Transwarp
TTA
VMware

Associate Members

Tsinghua University
University of Coimbra, Portugal
China Academy of Information and Communications Technology
Imec

Professional Affiliates

Hotea Solutions
Infobell IT Solutions
Infosizing
SBIMS

Steering Committee

Michael Brey (Chair)	Oracle
Matthew Emmerton	IBM
Jamie Reding	Microsoft
Ken Rule	Intel
Nicholas Wakou	Dell EMC

Public Relations Committee

Meikel Poess (Chair)	Oracle
Paul Cao	HPE
Gary Little	Nutanix
Rodrigo Escobar	Intel
Nirmala Sundararajan	Dell Technologies

Technical Advisory Board

Jamie Reding (Chair)	Microsoft
Paul Cao	HPE
Gary Little	Nutanix
Mike Brey	Oracle
Ken Rule	Intel
Nicholas Wakou	Dell EMC

Technical Subcommittees and Chairs

Transaction Processing

- TPC-C: Jamie Reding (Microsoft)
- TPC-E: Jamie Reding (Microsoft)

Decision Support

- TPC-H: Meikel Poess (Oracle)
- TPC-DS: Meikel Poess (Oracle)
- TPC-DI: Meikel Poess (Oracle)

Virtualization

- TPCx-V Tariq Magdon-Ismail (VMware)
- TPCx-HCI Tariq Magdon-Ismail (VMware)

Big Data

- TPCx-HS: Tariq Magdon-Ismail (VMware)
- TPCx-BB: Rodrigo Escobar (Intel)

Internet of Things

- TPCx-IoT: Meikel Poess (Oracle)

Artificial Intelligence

- TPCx-AI: Hamesh Patel (Intel)

Open Source

- TPC-OSS: Andy Bond (VMware)

Common Benchmark Specifications

- TPC-Pricing: Jamie Reding (Microsoft)
- TPC-Energy: Paul Cao (HPE)

Contents

Multivariate Time Series Anomaly Detection: Fancy Algorithms and Flawed Evaluation Methodology

Mohamed El Amine Sehili and Zonghua Zhang(✉)

Huawei Technologies France, 8 Quai du Point du Jour, 92100 Boulogne-Billancourt, France
zonghua.zhang@imt-lille-douai.fr

Abstract. Multivariate Time Series (MVTS) anomaly detection is a long-standing and challenging research topic that has attracted tremendous research effort from both industry and academia in recent years. However, a careful study of the literature makes us realize that 1) the community is active but not as organized as other sibling machine learning communities such as Computer Vision (CV) and Natural Language Processing (NLP), and 2) most proposed solutions are evaluated using either inappropriate or highly flawed protocols, with an apparent lack of scientific foundation. So flawed is one very popular protocol, the so-called **point-adjust** protocol, that a random guess can be shown to systematically outperform *all* algorithms developed so far. In this paper, we review and evaluate a number of recent algorithms using more robust protocols and discuss how a normally good protocol may have weaknesses in the context of MVTS anomaly detection and how to mitigate them. We also share our concerns about benchmark datasets, experiment design and evaluation methodology we observe in many works. Furthermore, we propose a simple, yet challenging, baseline algorithm based on Principal Components Analysis (PCA) that surprisingly outperforms many recent deep learning based approaches on popular benchmark datasets. The main objective of this work is to stimulate more effort towards important aspects of the research such as data, experiment design, evaluation methodology and result interpretability, as opposed to putting the highest weight on the design of increasingly more complex and "fancier" algorithms (Code repository associated with this paper can be found at https://github.com/amsehili/MVTSEvalPaper).

Keywords: Multivariate time series · Anomaly detection · Evaluation protocols · point-adjust

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 1–17, 2024.
https://doi.org/10.1007/978-3-031-68031-1_1

1 Introduction

1.1 MVTS Anomaly Detection: a Hot Research Topic

Time series are a kind of data characterized by its ease of collection and storage as well as its wide range of applications (forecasting, classification, anomaly detection, etc.).

Anomaly detection in multivariate time series, particularly unsupervised one, is a very important topic for the modern, data-powered, industry. This explains the high interest in the subject in both industry and academia and the proliferation of approaches in recent years, especially deep learning (DL) based ones. While one should salute the agility and effectiveness of the community at leveraging advancements from other fields such as NLP and CV, and repurposing them for anomaly detection, one may also argue that evaluation methodology, benchmark datasets and objective algorithms' comparison are still open challenges in the field as of today [11,16].

Our point is straightforward: algorithm design and pipeline complexity have been granted much more effort than experiment design and methodology. Actually, over the last decade or so, works on MVTS anomaly detection have adapted and experimented with a few of the most recent and most influential ideas in deep learning such as Generative Adversarial Networks (GANs) [3,5], Transformers [17] and Graph Neural Networks (GNNs) [7,9,19]. At the same time, we notice that a big number of approaches are evaluated in terms of one very flawed protocol, the **point-adjust** protocol, making it impossible, as we will show in this paper, to know whether an algorithm is outputting random noise as prediction or doing something cleverer. This may sound overly pessimistic, but as formally proven by [11] (and also in this work), a random guess based approach outperforms all "sophisticated" algorithms on popular datasets when evaluated with this protocol.

In this work, we review the **point-adjust** protocol and show that by randomly selecting a small, fixed, number of points and tagging them as anomalous, while tagging the rest of points as normal, we can achieve very high scores with high probabilities. Our goal is to continue warning the community against this protocol and its derivatives and encourage it to drop it in favor of more reliable protocols. We also review the more objective **point-wise** protocol and show that it is not appropriate for all kinds of datasets and use-cases. We then introduce and discuss the **event-wise** protocol, a new *event-aware* protocol for MVTS anomaly detection.

Besides, we demonstrate that an approach as basic as Principal Components Analysis (PCA), with very simple pre-processing and post-processing blocks, can outperform many complex DL-based approaches on many datasets. Finally, we analyze three of the most recent approaches and show how algorithms developed by exclusively targeting high **point-adjust** scores fail to distinguish themselves from a random guess when challenged with other protocols. Finally, without claiming to be an authority in the field, we share what we believe are vectors of

improvement based on our experience working on the topic for many years at an industrial level.

2 Related Work

While the number of works on MVTS anomaly detection has been increasing over the years, very few studies about issues in benchmark datasets, algorithms and especially evaluation protocols and metrics have been proposed so far.

In their paper entitled "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress" [16], Wu and Keogh argue that many popular univariate and multivariate benchmark datasets for anomaly detection have serious flaws that make the claimed performance of many algorithms questionable. The authors demonstrate that using the so-called *one-liners* (i.e., very short code with a simplistic logic such as the difference between two consecutive points or the moving average), one can achieve state-of-the-art performance on the datasets considered in the study. They particularly pinpoint four flaws that the majority of the studied datasets suffer from: triviality, unrealistic anomaly density, mislabeled ground truth and run to failure bias (i.e., anomalies systematically located at the end of time series).

While we agree with most of these points and find the conclusions of the paper highly important, we believe that the point about the unrealistic anomaly density in datasets should not always be regarded as a flaw. Actually, as the authors point out, anomalies in data are rare by nature, and it is not easy to build a real-world benchmark dataset that contains 10% or 15% of anomalies for example. However, having many anomalies in the data, ideally with different distributions, is necessary to reliably evaluate algorithms and their generalization capability. Many of the frequently used MVTS benchmark datasets (e.g., SWaT [12], Wadi [2] and PSM [1]) are gathered from real-world industrial systems and contain a relatively high ratio of anomalies obtained by deliberately altering the normal functioning of the system. These datasets are good to evaluate unsupervised anomaly detection approaches because their training part is anomaly free[1].

Nevertheless, we believe that this point may become concerning when *training* data are provided with an "abnormally" high ratio of *labeled* anomalies, making it possible to use supervised approaches. This may be problematic because in the real-world, supervised algorithms would not have access to such big amounts of labeled anomalies for training.

In terms of evaluation protocols and metrics, recent works on MVTS anomaly detection have been marked by the use of the highly defective **point-adjust** protocol (discussed in Sect. 3.1), claiming very high scores. Kim et al. [11] propose the first thorough review of this protocol, exposing its flaws. They particularly introduce three basic methods that outperform all state-of-the-art algorithms evaluated using this protocol. These methods include: 1) drawing a random

[1] Assumption based on the description shared by datasets' publishers. In any case, *no anomaly labels* are provided for the training fold of these datasets, and almost all approaches using them are unsupervised.

anomaly score from a uniform distribution over $[0, 1]$, 2) using raw feature values as an anomaly score, and 3) predicting anomalies using an *untrained* neural network.

Based on the conclusions of [11], Garg et al. [8] propose the **composite** protocol in which the recall is calculated based on the number of detected anomalous *events* (as opposed to anomalous points, see Sect. 3.3), whereas the precision is calculated, as usual, at the point level. The F1 score is then computed using the *event-level* recall and *point-level* precision This protocol alleviates the effects of lengthy events on the **point-wise** protocol (introduced in Sect. 3.2) and allows for a more intuitive evaluation of algorithms. With such a protocol, we can, for example, know how many events an algorithm has detected. This is an important information that the **point-wise** protocol does not provide.

3 Evaluation Protocols for MVTS Anomaly Detection

3.1 Point-Adjust: A Non-protocol for Time Series Anomaly Detection

According to this protocol, if an algorithm correctly detects at least one anomalous point within a segment of many, contiguous, anomalous points, then all points within the segment are systematically considered detected and count as True Positive (TP). The rationale behind this is that one segment represents one anomalous event (which is a quite reasonable assumption) and thus, if the algorithm detects one point within it as anomalous, then: 1) the whole event should be declared as successfully detected, and 2) the algorithm should be *rewarded* with as many TP as there are points in the segment, as illustrated in Fig. 1.

Fig. 1. Illustration of the **point-adjust** evaluation protocol. A single anomalous point detected by the algorithm implies that all other points within the segment be counted as True Positive, even if the algorithm did not actually detect them.

While the fact of "successfully detecting one anomalous point within a segment is sufficient to consider the event detected" is straightforward, we find the second part of the reasoning problematic. First, it starts off by presenting the evaluation criterion as event-based: the most important is whether the event is detected once, regardless of the number of actually detected anomalous points within it. The reasoning then seamlessly switches to a point-wise evaluation, multiplying, *on its way there*, the number of TP by the *length* of reasoning would be to count just *one* TP for *one* detected event, like in protocols described in Sect. 3.3.

Second, this reasoning seems to rely on the tacit assumption that anomalous points within the same segment are *similar*, and therefore it is "acceptable" to detect just one or a few of them to consider all of them successfully detected. We believe that, if this assumption holds, then from an algorithm developer's perspective, the fact that the algorithm only detects a subset of presumably similar anomalous points should be an argument to investigate the behavior of the algorithm instead of rewarding it with points it did not detect. If the assumption does not hold, however, and anomalous points within the same segment may have different distributions, then considering that all the points were correctly detected based on the mere fact that one of them was detected by the algorithm sounds deliberately misleading.

As a result, using this protocol, an algorithm that has many false alarms but happens to detect one or a couple of anomalies within an anomalous segment would achieve a high score. Based on the distribution of anomalies in many frequently used benchmark datasets, a very efficient such an algorithm would be a one that *randomly* selects a small number of points and tag them as anomalous while considering the remaining points as non-anomalous. This section introduces, analyses and evaluates an algorithm of this kind.

Kim et al. [11] show that this protocol can be *hacked* using several procedures, including: sampling random anomaly scores from a uniform distribution, using raw feature values as an anomaly score, and predicting anomalies using an *untrained* neural network. They also provide a formal proof for the first case.

In this work, we propose a procedure to reach a **point-adjust** F1 score of choice, and calculate the probability of reaching it. The goal of the current part is two-fold: 1) show how manipulable the **point-adjust** protocol can be, and 2) lay the ground for discussing the weaknesses of the F1 score in general and how inappropriate it can be for some kinds of datasets and anomaly distributions. The latter will be further discussed in Sect. 3.2.

The F1 score is calculated in terms of precision P and recall R as follows:

$$\mathrm{F1} = \frac{2 \times \mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}} \quad \text{for} \quad \mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \quad \text{and} \quad \mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

where TP stands for True Positive, FP for False Positive, and FN for False Negative.

We define r, the contamination rate in evaluation data, that is, the ratio of anomalous points to the total number of points. If we randomly pick up α points from evaluation data and tag them as anomalous, then the probability of *not* hitting any anomalous point is $(1-r)^{\alpha}$. Thus, the probability of hitting one anomalous point at least is $1-(1-r)^{\alpha}$.

Based on this, if the anomalies make up one single segment, then applying the **point-adjust** procedure yields a perfect recall, $\mathrm{R}_{pa} = 1$ with the same probability:

$$p(\mathrm{R}_{pa} = 1 | r, \alpha) = 1 - (1-r)^{\alpha} \tag{1}$$

This is the probability of selecting at least one single point within the anomalous segment with α trials, which is also the probability of selecting *at most* $(\alpha - 1)$ points from outside the segment. In other words, this is the probability of having $(\alpha - 1)$ false alarms at most: $p(\text{FP} \leq \alpha - 1 | r, \alpha)$.

The **point-adjust** precision, P_{pa}, obtained from adjusted TP, TP_{pa}, is:

$$\text{P}_{pa} = \frac{\text{TP}_{pa}}{\text{TP}_{pa} + \text{FP}} \tag{2}$$

Let A be the length of the anomalous segment. When $\text{R}_{pa} = 1$, then $\text{TP}_{pa} = A$ and FP equals $(\alpha - 1)$ at most. Thus, the worst **point-adjust** precision is $\text{P}_{pa} = \frac{A}{A+(\alpha-1)}$. We can write the probability of having a **point-adjust** precision of at least $\frac{A}{A+(\alpha-1)}$, given r and α, as:

$$p(\text{P}_{pa} \geq \frac{A}{A + (\alpha - 1)} | r, \alpha) = 1 - (1 - r)^{\alpha} \tag{3}$$

Similarly, when $\text{R}_{pa} = 1$ we can write F1_{pa} as:

$$\text{F1}_{pa} = \frac{2 \times \text{P}_{pa} \times 1}{\text{P}_{pa} + 1} = \frac{2 \times \text{P}_{pa}}{\text{P}_{pa} + 1} \tag{4}$$

From Eq. 3 and Eq. 4 we have:

$$p(\text{F1}_{pa} \geq \frac{2 \times \frac{A}{A+(\alpha-1)}}{\frac{A}{A+(\alpha-1)} + 1} | r, \alpha) = 1 - (1 - r)^{\alpha}$$

which can be written as:

$$p(\text{F1}_{pa} \geq \frac{2A}{2A + \alpha - 1} | r, \alpha) = 1 - (1 - r)^{\alpha} \tag{5}$$

Equation 5 means that the larger α, the more confident we are about the *minimum* F1_{pa} we can achieve by tagging α random points as anomalous, but the lower is that *minimum* value itself. However, we can also see that larger values of A yield higher values for the *minimum* F1_{pa} value because $\lim_{A \to \infty} \left[\frac{2A}{2A+\alpha-1}\right] = 1$.

To illustrate this, we consider four hypothetical datasets with different sizes but the *same* contamination rate, $r = 0.1$, as shown in Fig. 2[2]. Each dataset contains one anomalous segment whose length is 10% of the total dataset length. For each dataset, we compute the probability of having a perfect **point-adjust** recall ($p(\text{R}_{pa} = 1)$) by randomly tagging 1% of the points as anomalous, and show the corresponding Cumulative Distribution Function (CDF) of the F1_{pa} score, that is, the probability of F1_{pa} being $\leq$ a given value.

As can be observed on Fig. 2, the larger A, the higher $p(\text{R}_{pa} = 1)$ and the less likely that F1_{pa} falls below the fairly high value of about 0.95. Actually, for

[2] These values are not arbitrary but are close to what we observe in popular benchmark datasets, especially for a $A = 500$.

$A = 50$, we have $p(\text{F1}_{pa} = 0) = 0.59$ and for $A = 500$, $p(\text{F1}_{pa} = 0) = 0.005$. For illustration purposes, these values are not shown on Fig. 2 but in a separate figure, Fig. 3. Figure 3 is similar to Fig. 2 (for $A = 50$ and $A = 500$) but shows the CDF of F1_{pa} starting from $\text{F1}_{pa} = 0$. We can see that with the **point-adjust** procedure, detecting one single anomaly within an anomalous segment may result in an unrealistically big jump in F1_{pa}.

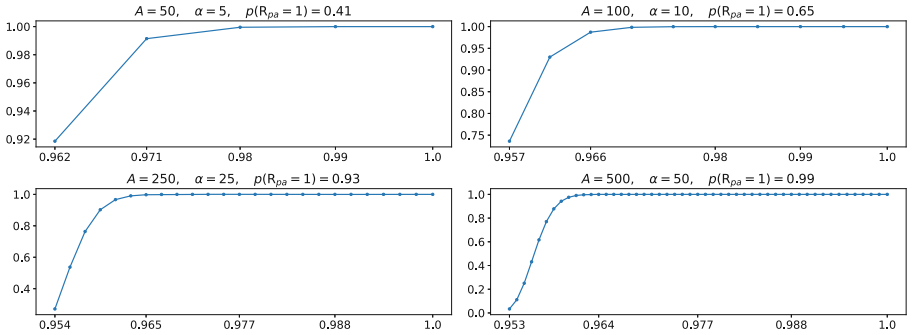


Fig. 2. Probability of achieving a perfect **point-adjust** recall ($p(\text{R}_{pa} = 1)$ for four different datasets that have the *same* contamination rate, $r = 0.1$, but different anomalous segment's lengths. For each dataset, we randomly select 1% of the points (α) and tag them as anomalous. Each subplot represents the CDF of the corresponding F1_{pa} score. That is, the x-axis represents threshold values and the y-axis is the probability of obtaining an F1_{pa} score $\leq$ the corresponding threshold value. Each value corresponds to $s \in [1, \alpha]$, the number of successes hitting the anomalous segment. We can see that for the same contamination rate across datasets, we are more confident about obtaining fairly high F1_{pa} scores as the length of the anomalous segment increases. For $A = 500$ for example, the probability of having $\text{F1}_{pa} \leq 0.953$ is close to 0.

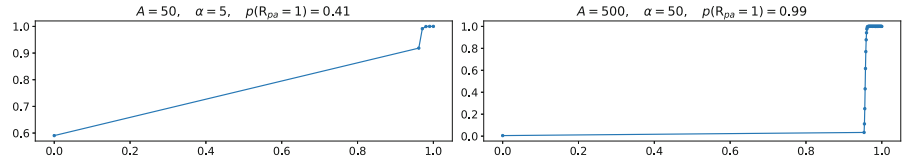


Fig. 3. Probability of achieving a perfect **point-adjust** recall ($p(\text{R}_{pa} = 1)$ for $A = 50$ and $A = 500$. Unlike Fig. 2, this figure shows the CDF starting from $\text{F1}_{pa} = 0$.

In Fig. 2 and 3, the probability of achieving a perfect **point-adjust** recall for $A = 50$ is 0.41 and that of having $\text{F1}_{pa} = 0$ is fairly high (0.61). These probabilities are based on $\alpha = 5$. It turns out that we can improve these scores and their probabilities by using *better* values for α while keeping other parameters unchanged (i.e., $A = 50$ and $r = 0.1$). Figure 4 shows the probability of

achieving a perfect **point-adjust** recall for different values of α, and the worst resulting P_{pa} and $F1_{pa}$ scores. These worst scores are reached if, out of α randomly selected points, one single point falls within the anomalous segment and $\alpha - 1$ points outside of it. For $\alpha = 26$, for example, $p(F1_{pa} \geq 0.8) \approx 0.935$.

In practice, using this procedure with $\alpha = 1000$ yielded an average $F1_{pa}$ score of about 0.95 for SWaT and Wadi datasets and 0.98 for PSM. These scores are higher than the ones obtained using elaborate DL-based pipelines.

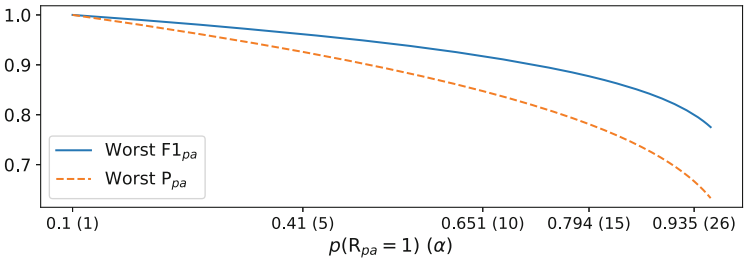


Fig. 4. Probability of achieving a perfect **point-adjust** recall ($p(R_{pa} = 1)$ for $A = 50$ and $r = 0.1$ for different values of α, as well the *worst* corresponding values of $F1_{pa}$ and P_{pa}. The *worst* values correspond to the case where one single point within the anomalous segment and $\alpha - 1$ points from outside the segment are respectively tagged as anomalous. Each x value corresponds to $p(R_{pa} = 1)$ given a value of α.

3.2 Point-Wise: A Good Protocol but Not for All Situations

This protocol consists of computing the precision, recall and F1 score based on the actual output of the algorithm, without any "adjustment". There is nothing special about this procedure, which is used in many other fields. One interesting feature of the F1 score is that it is generally a good default choice for unbalanced datasets. An unbalanced dataset contains a dominant class with the majority of points belonging to it[3]. However, in many cases, including anomaly detection, it is often desirable to know the number of false alarms raised by an algorithm or at least their rate, referred to as False Alarm Rate (FAR), and computed as the number of false alarms to the total number of normal points.

Consider for instance an anomaly detection algorithm used to detect upcoming hard disk failures in a data center. Detecting an upcoming failure means that the disk should be replaced. Using the FAR as one of the metrics to evaluate the algorithm helps estimate the expected number of unduly replaced disks and the resulting cost. This is not possible when using the precision or the F1 score.

[3] Machine learning students are usually advised to use F1 as a better alternative to the *accuracy* score for unbalanced datasets because using the latter would yield a high score for a trivial algorithm that predicts the dominant class for every input.

To further illustrate the limits of the F1 score, we consider many anomaly detectors that have all the same recall of 0.99, but a FAR that varies across detectors ($0.001 \leq \text{FAR} \leq 0.2$). We compute the F1 score of each of these detectors with three datasets that have the *same* number of normal points (10000) but a different number of anomalous points (5000, 1000 and 100 points respectively, meaning that the datasets have a different contamination rate). Figure 5 shows the F1 score of each of these detectors for the three datasets. As can be observed, for a given detector (one tick on the x-axis), the F1 scores vary considerably depending on the number of anomalous points in the dataset.

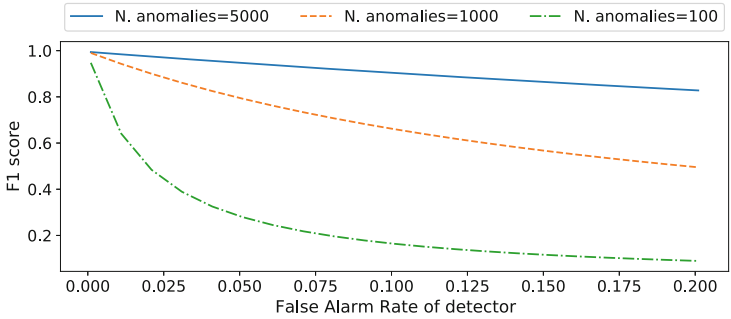


Fig. 5. F1 score of many anomaly detectors on three different datasets. All detectors have the same recall of 0.99, but a different False Alarm Rate (i.e., each detector has its own FAR). Each curve corresponds to a dataset that has 10000 normal points but a different number of anomalous points (the datasets have, therefore, a different contamination rate). We can see that, even if the recall and the FAR of a detector are the same for all datasets, its F1 score may be much higher for datasets with a high contamination rate than for datasets with a low contamination rate. This is particularly true for detectors with a high FAR.

This happens because the precision and, as a result, the F1 score, are also a function of TP: a higher TP results in a better precision and F1 score. In fact, as the FAR of a detector is the same across datasets, and as the three datasets have the same number of normal points, the expected FP is the same for the detector across datasets. However, as the number of anomalous points is not the same for all datasets, the TP of a detector is higher for datasets with more anomalies. As a result, the precision and F1 score are better for datasets with a higher contamination rate. Consequently, the precision and the F1 score can only yield a *coarse* estimation of false alarms and can hardly be used to accurately estimate their cost in terms of money, time, human effort, etc.

More related to time series, another case where F1 score computed in the **point-wise** fashion has limits is when anomalies make up lengthy segments as opposed to isolated outliers. We refer to these segments as anomalous *events*. In principle, if two algorithms have roughly the same FAR, one would prefer the

one that detects *more* events even if its **point-wise** recall is lower. The **point-wise** F1 score does not allow algorithm comparison from that perspective, and does not let us know how many events were ever detected by an algorithm.

The limits of the **point-wise** F1 score may be exacerbated for benchmark datasets that contain one or a couple of very long events compared to the rest of events. In such cases, an algorithm may be tuned to efficiently detect anomalies within the lengthy events, while probably (and seamlessly) being less efficient for shorter events[4]. This is for example the case for the SWaT dataset, in which the median event length is 450 points, but there is one 35K-point event that is, furthermore, the most *anomalous* and the easiest to detect.

3.3 Alternative Evaluation Protocols

The Composite Protocol. By design, the **point-wise** F1 score treats TP and FP alike (and also FN, but this point in not necessary to the current discussion). As a result, each TP is *celebrated* as one more detected anomaly and each FP is regarded as a false alarm. The problem is that, for benchmark datasets that contain anomalous events, the very first anomaly detected within an event is normally more informative than consecutively detected anomalies within the same event. Hence, the symmetry with which **point-wise** protocol considers TP and FP is no longer justified. Based on this reasoning, Garg et al. [8] propose to count just one TP for each detected event, regardless of the number of anomalous points detected within it. The recall is hence computed at the *event level*. However, to compute the precision, the TP and FP at the *point level* are used, just like with the **point-wise** protocol. The resulting F1 score, referred to as $F1_C$ in the following, is called **composite**. This protocol is much more appropriate than the **point-wise** protocol for datasets in which anomalies take the form of long events, and we believe it should be paid more attention in future works.

The Event-Wise Protocol. Motivated by the conclusions of [8], and in an attempt to: 1) partially disentangle the precision from TP and give more weight to false alarms, and 2) make the performance of algorithms easier and more intuitive to interpret[5], we propose the **event-wise** protocol.

The **event-wise** protocol computes TP at the event level, as in the **composite** protocol. However, FP is not counted at the point level, as with the **point-wise** and **composite** protocols, but is the number of anomalous *segments* found by the algorithm that do not overlap with *any* Ground Truth (GT) event. For clarity, we use "segment" to refer to any set of contiguous points that

[4] This situation has subtle links to the one of imbalanced datasets for which the F1 score is usually recommended in the first place.

[5] By interpretation we mean understandinghow many anomalous events the algorithm detects and how many false alarm it raises. This has nothing to do with model *interpretability* whose goal is to answer *why/how* an algorithm made a given decision.

the algorithm considers anomalous, and "event" to refer to a GT anomalous event. Based on this, we define the following **event-wise** metrics:

- TP_E: number of GT events that fully or partially overlap with one segment or more. The **event-wise** recall is calculated as $\mathrm{R}_E = \frac{\mathrm{TP}_E}{\#\ \mathrm{GT\ events}}$. This is the same recall used in the **composite** protocol.
- FN_E: number of GT events that do not overlap with any segment.
- FP_E: number of segments that do not overlap with any GT event. This includes segments of many points as well as isolated points. The **event-wise** precision can be computed as $\mathrm{P}_E = \frac{\mathrm{TP}_E}{\mathrm{TP}_E + \mathrm{FP}_E}$.

A similar protocol was proposed by [10] but has not been much used in consecutive works. This is likely due to one serious issue it has: an algorithm that predicts an alarm for every point (e.g., algorithm A3 in Fig. 6) would have prefect **event-wise** recall (R_E) and precision (P_E). To avoid this, we propose to involve the FAR at the point level in the computation of the **event-wise** precision. The precision at the event level is therefore computed as:

$$\mathrm{P}_E = \frac{\mathrm{TP}_E}{\mathrm{TP}_E + \mathrm{FP}_E} \times (1 - \mathrm{FAR}) \tag{6}$$

where $\mathrm{FAR} = \frac{\mathrm{FP}}{N}$ and N is the number of normal points in data. The **event-wise** F1 score, denoted $\mathrm{F1}_E$, is then computed in terms of R_E and P_E.

Figure 6 illustrates this protocol with the outputs of three algorithms. Note that algorithm A3 has $\mathrm{F1}_E = 0$ despite the fact that it has a *perfect* **event-wise** recall. This is the case because this algorithm has a FAR = 1. Thanks to the introduction of FAR in the computation of the precision, the proposed **event-wise** protocol is more sensitive to false alarms compared to other protocols, and is less dependent on TP or on the number of anomalous points in the dataset.

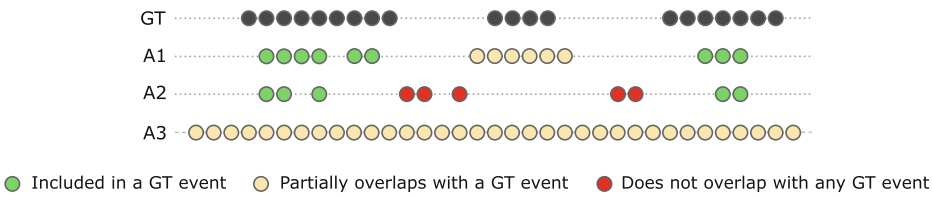


Fig. 6. llustration of the **event-wise** protocol with three GT events and the outputs of three algorithms. Algorithm A1 detects all events without false alarms (i.e., it has no segment that does not overlap with any event). Despite this, its precision is not perfect because its second segment is too long. Algorithm A2 achieves an **event-wise** recall of 2/3 and a low precision because its $\mathrm{FP}_E = 3$. Algorithm A3 detects all events but its **event-wise** precision, P_E, and its $\mathrm{F1}_E$ score equal 0 because its FAR is 1.

4 Benchmark Datasets, Experiment Design and Algorithms Comparison

Wu and Keogh [16] discuss flaws of a few datasets for univariate and multivariate time series used for anomaly detection, and how trivial solutions can achieve state-of-the-art performance on them. In this section, we discuss other issues related to a few popular MVTS datasets and certain practices in terms of evaluation that hinder objective algorithm comparison.

SWaT and Wadi are two very popular MVTS benchmark datasets for anomaly detection. They are fairly big in terms of number of points and have each many anomalous events introduced by deliberately manipulating parts of the system from which they are collected. However, these datasets are shared by their publishers in a raw format upon subscription, with incoherent labeling for the former and two data versions for the latter.

The SWaT dataset for example has one integrated label column as well as a separate file that contains the start and the end of anomalous events. Using events' start and end to reconstruct the labels does not result in the same content as in the integrated label column. Wadi has two versions from 2017 and 2019, with different sizes and different anomaly ratios. This situation has led to a disagreement about the characteristics of these datasets across papers.

Furthermore, as mentioned earlier, the SWaT dataset contains one anomalous event that is much longer than other events. This event contributes to the F1 score much more than other events and explains the relatively high scores of many algorithms using the **point-wise** protocol with this dataset compared to other datasets. Actually, by shortening this event to the median event length, [8] showed a significant drop in the **point-wise** F1 score for all algorithms. While this is an interesting finding, we do not advocate for deliberate event shrinking because it may introduce artifacts and subtle changes in the data, making the event *easier* to detect. We believe that this issue can be better dealt with using event-aware evaluation protocols like the ones introduced in Sect. 3.3

The use of different evaluation protocols across publications has also led to situations where authors evaluate their algorithm with the **point-adjust** protocol but report results of other approaches using the **point-wise** protocol. This is for example observed in [6], which evaluates the proposed algorithm using **point-adjust** and compares the results to at least one algorithm (GDN [7]) using the **point-wise** protocol. Other works, such as [15], use the **point-adjust** protocol without even mentioning it in the text. These practices, which may be observed in other works (and which we assume are not intentional), are very likely to be misleading for uninformed readers, including reviewers. Finally, in [13] the authors explicitly confirm that they do *not* use the **point-adjust** protocol, citing what is probably the main work highlighting the flaws of this protocol as of today, [11]. However, reported scores for the proposed algorithm, as well as for state-of-the-art algorithms considered for comparison, look as high as the usual **point-adjust** scores encountered in many studies with the same datasets.

5 Algorithms

In this part, we evaluate three DL-based algorithms for MVTS anomaly detection, as well as a baseline algorithm based on PCA. These DL-based approaches are selected because they are quite recent (2021), introduce very interesting ideas, and achieve a very good performance with the used protocols. Moreover, all of these approaches have an official open source implementation that we used in our experiments. The approaches are:

- **Anomaly Transformer** [17], uses Transformers to model time series dynamic patterns. Evaluated using the **point-adjust** protocol only in original work.
- **NCAD** [4], uses a 1D Convolutional Neural Network (CNN) to obtain feature representations of data. Also evaluated using the **point-adjust** protocol only in original work.
- **GDN** [7], uses GNNs to learn the relationship between time series to achieve a better forecasting performance. Evaluated using the **point-wise** protocol only in original work.

Our goal is to challenge these algorithms with different evaluation protocols and confront them with PCA. Table 1 summarizes the obtained results with **point-wise**, **composite** and **event-wise** protocols. It also reports the number of detected events (TP_E) and the number of predicted anomalous segments that do not overlap with any GT event (FP_E, see Fig. 6). Results are obtained using the SWaT (with labels constructed from attacks' start and end), Wadi (2017 version), and PSM datasets.

Table 1. Comparative results of AnomalyTransformer (AT), NCAD, GDN and PCA using the **point-wise** (F1), **composite** ($F1_C$) and **event-wise** ($F1_E$) protocols. All metrics are computed based on the detection threshold that yields the best **point-wise** performance. The SWaT, Wadi and PSM datasets have 35, 14 and 72 events and a contamination rate of 11.98%, 5.70% and 27.76% respectively.

	SWaT				Wadi				PSM			
	F1	$F1_C$	$F1_E$	TP_E/FP_E	F1	$F1_C$	$F1_E$	TP_E/FP_E	F1	$F1_C$	$F1_E$	TP_E/FP_E
AT	0.214	0.214	0.000	35/0	0.108	0.108	0.000	14/0	0.434	0.434	0.000	72/0
NCAD	0.217	0.217	0.002	35/694	0.114	0.115	0.003	14/1394	0.429	0.429	0.000	72/0
GDN	0.821	0.488	0.478	11/0	0.567	0.764	0.485	9/14	0.594	0.640	0.096	63/843
PCA	0.801	0.476	0.440	11/4	0.325	0.498	0.151	6/59	0.470	0.457	0.156	61/85

These results reveal a number of interesting findings, summarized as follows:

- Algorithms that were developed using **point-adjust** as the sole target fail to reach any score better than a random guess when evaluated with other protocols. This is the case of **AnomalyTransformer** and **NCAD**. For the

sake of completeness, we also evaluated these algorithms using the **point-adjust** protocol and achieved the same, very high, scores reported in original papers. It is worth mentioning that we also achieved essentially the same high scores using *untrained* versions of these models. We assume that many other approaches developed using the same setting would face the same problem[6].
- **GDN**, however, which was developed based on the more realistic **point-wise** protocol, shows more resilience when evaluated with other protocols.
- Datasets that have a very high contamination rate, such as PSM, yield **point-wise** F1 scores that can be misleading. **AnomalyTransformer** and **NCAD**, for example, achieve a **point-wise** F1 score around 0.43, which may look as a *fairly good* baseline score. In actual fact, based on the contamination rate of the dataset, this score is achievable by predicting all or most points as anomalous, and this is what these algorithms are doing. The **composite** score is comparable to the **point-wise** score in this case and does not bring any useful information. The **event-wise** score, however, severely penalizes such approaches thanks to the inclusion of FAR in score computation.
- Finally, we show that PCA, which is not considered as a particularly advanced approach for MVTS anomaly detection, achieves an honorable **point-wise** score compared to many recent DL-based approaches (e.g. USAD [3], MSCRED [18], OmniAnomaly [14] and DAGMM [20], see [11] for a summary of the performance of these approaches). Our conclusion here is that many works have been developed without establishing a simple but enough challenging baseline. Actually, the bar of 0.8 **point-wise** F1 score for the SWaT dataset had been a symbolic target for many years until recent GNN-based approaches, such GDN [7] and FuSAGNet [9], reached it. In our PCA-based pipeline, we use simple pre-processing and post-processing blocks (input scaling, clipping and score smoothing) that significantly improve the score.

Scores in Table 1 are obtained using the threshold that yields the best **point-wise** F1 score. To further understand the decisions of algorithms such as **AnomalyTransformer** and **NCAD**, we used the threshold that yields the best **point-adjust** score for these algorithms and looked at their outputs to better understand their behavior. As aforementioned, using the **point-adjust** protocol, we obtained scores comparable to those reported in original papers for these algorithms (actually, we achieved even higher scores). The first row in Fig. 7 shows all GT events of SWaT as well as the outputs of **AnomalyTransformer** that lead to a 0.97 $F1_{pa}$ score. By zooming in on smaller parts of the dataset (the three first events, then just the first one), we see that the algorithm predicts anomalies at an almost regular pace. More precisely, using the threshold that ensures the best $F1_{pa}$ score results in a total of 4097 points predicted as anomalous, of which 473 lie within an anomalous event and 3624 outside any anomalous event. Knowing that the whole test dataset is about 5 days long, has one point per second, and a total of 35 anomalous events, the algorithm, with such an output, helps detect

[6] Also check out this issue and related ones on AnomalyTransformer's official repository: https://github.com/thuml/Anomaly-Transformer/issues/34.

all 35 events while raising 3624 false alarms over 5 days. On average, it raises an alarm every 110 s, making it, in our opinion, barely useful for deployment. Such a behavior is generously rewarded by the **point-adjust** protocol.

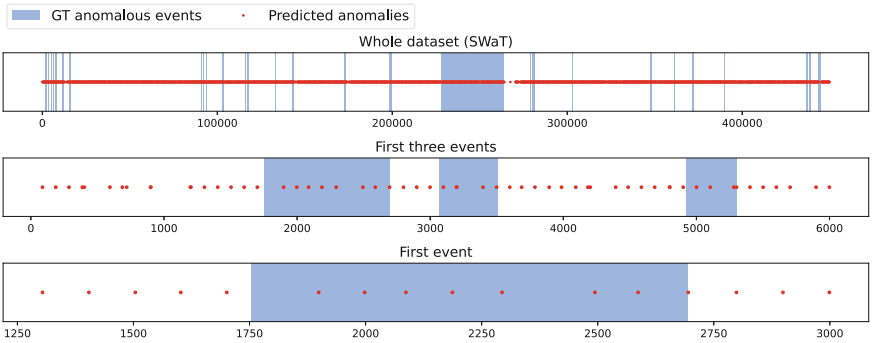


Fig. 7. Ground Truth anomalous events of the SWaT dataset and outputs of **AnomalyTransformer**. The second and third rows zoom in on the first three events and on the first event respectively. We can see that all the algorithm is doing is output an anomaly prediction at almost regular intervals. This ensures a 0.97 $F1_{pa}$ score. Note that on the first row (whole dataset), there are exactly 4097 red dots ($\leq 1\%$ of the total points) but they look many more because the size of the dots is intentionally big for visualization purposes.

6 Discussion: Towards Better Practices for MVTS Anomaly Detection

6.1 Evaluation Protocols and Metrics

The **point-adjust** protocol has had its *very illegitimate* hour of glory and created what Wu and Keogh [16] call "illusion of progress" in probably quite a unique way in machine learning. We believe that this protocol should no longer be used. Authors should resist the temptation of publishing high but misleading scores using it, and reviewers should be aware of its flaws and advise authors to report results using other protocols.

Evaluating algorithms for time series anomaly detection is not a trivial task. The **point-wise** protocol may be good as a straightforward way to compare and rank algorithms but other protocols and metrics, especially event-aware ones, should be used alongside this protocol when possible. For datasets that contain anomalous events/episodes, at least the number of detected events should be reported for each algorithm alongside **point-wise** metrics.

6.2 Datasets and Experiment Design

By sharing the raw versions of SWaT and Wadi datasets, the publishers apparently wanted to give researchers the freedom to experiment with the data the way they want. However, this has led to a divergence in the way these datasets are described and used in the literature. We believe that it would be better that the data be stored in a unique, accessible place using a format that anyone can start experimenting with quickly. Ideally, part of the test data would be provided without labels so that researchers upload the output of their algorithms to a third party server for evaluation and publication on a public leaderboard.

6.3 Algorithms

As mentioned in Sect. 1, recent approaches for MVTS anomaly detection have been marked by quite a few innovative ideas. However, based on the methodology issues discussed along this paper, our position on this is the following: *while many of the proposed approaches are conceptually very appealing, we believe that the merits of a majority of them are yet to be confirmed in the light of more appropriate evaluation protocols.*

Researchers should find a balance between the effort allocated to designing efficient algorithms and that allocated to running sound experiments. Based on our own experiments and on the results shown by [11] using an *untrained* model, it is reasonable to assume that studies using the **point-adjust** protocol achieved high scores in a relatively short time. Oftentimes in machine learning, an abnormally high score is achieved with little effort either because the problem at hand is trivial, due to a data leakage somewhere in the pipeline, or due to a bug in evaluation code. Admittedly, when the main cause is a widely used evaluation protocol, it can be much harder to uncover, hence the need to discuss evaluation protocols and their potential weaknesses and biases.

References

1. Abdulaal, A., Liu, Z., Lancewicki, T.: Practical approach to asynchronous multivariate time series anomaly detection and localization. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 2485–2494 (2021)
2. Ahmed, C.M., Palleti, V.R., Mathur, A.P.: Wadi: a water distribution testbed for research in the design of secure cyber physical systems. In: Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks, pp. 25–28 (2017)
3. Audibert, J., Michiardi, P., Guyard, F., Marti, S., Zuluaga, M.A.: USAD: unsupervised anomaly detection on multivariate time series. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 3395–3404 (2020)
4. Carmona, C.U., Aubet, F.X., Flunkert, V., Gasthaus, J.: Neural contextual anomaly detection for time series. arXiv preprint arXiv:2107.07702 (2021)

5. Chen, X., et al.: Daemon: unsupervised anomaly detection and interpretation for multivariate time series. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 2225–2230. IEEE (2021)
6. Chen, Z., Chen, D., Zhang, X., Yuan, Z., Cheng, X.: Learning graph structures with transformer for multivariate time series anomaly detection in IoT. IEEE Internet Things J. **9**, 9179–9189 (2021)
7. Deng, A., Hooi, B.: Graph neural network-based anomaly detection in multivariate time series. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 4027–4035 (2021)
8. Garg, A., Zhang, W., Samaran, J., Savitha, R., Foo, C.S.: An evaluation of anomaly detection and diagnosis in multivariate time series. IEEE Trans. Neural Netw. Learn. Syst. **33**(6), 2508–2517 (2021)
9. Han, S., Woo, S.S.: Learning sparse latent graph representations for anomaly detection in multivariate time series. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 2977–2986 (2022)
10. Hundman, K., Constantinou, V., Laporte, C., Colwell, I., Soderstrom, T.: Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 387–395 (2018)
11. Kim, S., Choi, K., Choi, H.S., Lee, B., Yoon, S.: Towards a rigorous evaluation of time-series anomaly detection. In: Proceedings of the AAAI Conference on Artificial Intelligence, no. 7, pp. 7194–7201 (2022)
12. Mathur, A.P., Tippenhauer, N.O.: Swat: a water treatment testbed for research and training on ICS security. In: 2016 International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWater), pp. 31–36. IEEE (2016)
13. Pan, J., Ji, W., Zhong, B., Wang, P., Wang, X., Chen, J.: Duma: dual mask for multivariate time series anomaly detection. IEEE Sensors J. (2022)
14. Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., Pei, D.: Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2828–2837 (2019)
15. Tuli, S., Casale, G., Jennings, N.R.: TranAD: deep transformer networks for anomaly detection in multivariate time series data. Proc. VLDB **15**(6), 1201–1214 (2022)
16. Wu, R., Keogh, E.: Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. IEEE Trans. Knowl. Data Eng. **35**, 2421–2429 (2021)
17. Xu, J., Wu, H., Wang, J., Long, M.: Anomaly transformer: time series anomaly detection with association discrepancy. arXiv preprint arXiv:2110.02642 (2021)
18. Zhang, C., et al.: A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 1409–1416 (2019)
19. Zhang, W., Zhang, C., Tsung, F.: Grelen: multivariate time series anomaly detection from the perspective of graph relational learning. In: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-2022, pp. 2390–2397 (2022)
20. Zong, B., et al.: Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In: International Conference on Learning Representations (2018)

A Comprehensive Study on Benchmarking Permissioned Blockchains

Jeeta Ann Chacko[1(✉)], Ruben Mayer[2], Alan Fekete[3], Vincent Gramoli[3], and Hans-Arno Jacobsen[4]

[1] Technical University of Munich, Munich, Germany
chacko@in.tum.de
[2] University of Bayreuth, Bayreuth, Germany
ruben.mayer@uni-bayreuth.de
[3] University of Sydney, Sydney, Australia
{alan.fekete,vincent.gramoli}@sydney.edu.au
[4] University of Toronto, Toronto, Canada
jacobsen@eecg.toronto.edu

Abstract. Blockchain benchmarking systems are actively discussed in the literature, focusing on increasing the number of blockchains that can be supported. However, the constant inception of new blockchains into the market and their vast implementation differences make it a massive engineering challenge. We provide a general discussion on the main aspects of benchmarking blockchains, highlighting the necessary contributions from the developers and users of blockchains and benchmarking systems. We identify problem statements across four benchmarking factors by investigating five popular permissioned blockchains. Further, we define a broad methodology to tackle these problems. We conduct a case study of five existing blockchain benchmarking systems for our evaluation and identify their limitations, clarifying the need for our methodology.

Keywords: permissioned blockchains · benchmarking systems

1 Introduction

Though blockchains were initially considered digital currency exchange systems, introducing smart contracts led to the classification of blockchains as decentralized transactional management systems that could support more use cases [64]. Later, the conception of permissioned blockchains that restricted the network access to authorized users and improved the overall performance made blockchains attractive for enterprise use cases. Currently, the most popular permissioned blockchain platforms, such as Fabric, Corda, Multichain, and Quorum, have around 30 to 70 enterprise partners using their systems for various use cases, such as banking, supply chain transparency, and digital asset management [20,42,54,59].

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 18–33, 2024.
https://doi.org/10.1007/978-3-031-68031-1_2

However, the plethora of blockchain systems currently available in the market creates uncertainty in the selection process. A recent survey shows that 26% of users switched from their initially chosen blockchain at a later stage of development and that performance is one of the top selection criteria for blockchains [62]. Though most blockchains report their individual performance data, the vast differences in implementation, system configuration, and workloads make a fair comparison challenging [33]. This highlights the demand for a comprehensive and impartial blockchain benchmarking approach. Currently, there are multiple benchmarking system implementations available for blockchains [11,28,33,40,57,65]. Each of them targets one or a specific set of blockchains to benchmark. The current focus in this research space is on increasing the number of blockchains supported by a benchmarking system. For example, Blockbench [28], the first benchmarking system for permissioned blockchains, supports four blockchains, while Diablo [33] and Gromit [57], the latest benchmarking systems, support seven blockchains. However, the rapid inception of new blockchains into the market makes this a massive engineering challenge.

Additionally, as is the case with most transaction processing systems, initially, the lines between the design and implementation of benchmarking systems are often blurred [4]. A well-implemented benchmarking system may still fail to consider all crucial aspects of benchmarking due to poor design [34]. For example, many existing benchmarking systems only support simple asset transfer scenarios [57,65], while in reality, blockchains are employed for numerous other use cases. Therefore, we identify the need for a thorough discussion on the different aspects of benchmarking blockchains, which will assist in implementing a comprehensive and extensible benchmarking system in the future.

One needs to understand the similarities and differences between the various blockchain platforms to identify the diverse factors of benchmarking accurately. A significant challenge in this direction is the insufficient scientific literature. Since many blockchains are commercialized, apart from research papers, we must also analyze technical documentation and blog posts from the respective blockchain developers to understand their systems thoroughly. Further, given the vast implementation distinctions among the different blockchain systems, discussions regarding blockchain benchmarking should not be limited to developers of benchmarking systems, but should also include developers of blockchain systems.

Our discussions address the problems regarding crucial benchmarking elements such as system configuration, parameter tuning, workloads, and metrics. We emphasize the importance of these issues by extensively analyzing five different permissioned blockchains. We then define the contributions required from the entire blockchain community to tackle them. We also conduct a case study of five existing benchmarking systems to identify their limitations and highlight the need for contributions. For example, we identify various system configuration settings that affect the performance of each of the multiple blockchains, while

current benchmarking studies only employ the default value for these settings. In detail, we provide the following contributions:

1. We formulate problem statements across four aspects of benchmarking based on five different permissioned blockchains (Fabric, Corda, Multichain, Quorum and Diem). This highlights the importance of these problems across different blockchain platforms.
2. We define a general methodology to tackle the problems that spans across developers and users of blockchains as well as benchmarking systems. This highlights the contributions required from each of them to improve the domain of blockchain benchmarking.
3. We provide a case study of five different blockchain benchmarking systems and the corresponding benchmarking studies to highlight the current limitations. This can help benchmarking system developers to extend their implementations to adhere to our methodology.

2 Permissioned Blockchains

In permissioned blockchains, access is restricted to a set of authorized users, making them suitable for many enterprise use cases that cannot support anonymity. They are peer-to-peer networks with access controls operating on a distributed ledger. Despite being in the same classification, the multiple permissioned blockchain systems currently available have vast differences in their implementation. This section briefly overviews the basic concepts and transaction flow of five popular permissioned blockchains, accentuating their similarities and differences.

2.1 Hyperledger Fabric

Hyperledger Fabric (a.k.a Fabric) is an open-sourced, permissioned blockchain system under the Linux foundation [2]. Fabric follows an execute-order-validate (EOV) model, one of its unique features. The main components of a Fabric network are peers, endorsers, and the ordering service. Only the endorsers store the smart contracts, and transactions are sent to the endorsers for execution based on an endorsement policy. Speculative transaction execution results in a read-write set of all the keys in the transaction which is then forwarded to the ordering service. The ordering service is a cluster of nodes that employs the Raft consensus protocol to decide on the order of the transactions. Upon consensus, a block of ordered transactions is broadcasted to all peers. Every peer validates the speculative results of every transaction in the block with the current world state. After successful validation, the world state is updated, and the block of transactions is committed to the ledger.

2.2 Corda

R3 Corda is an open-sourced permissioned blockchain mainly designed for financial use cases [7]. In Corda, data is only shared among the network participants

on a need-to-know basis. The nodes in a Corda network are authorized using an identity service. Further, a network map service is employed for node lookup, enabling point-to-point communication between nodes. An immutable object called a state describes any data known to the nodes at a specific point in time. Each node has a vault or database that stores all the state sequences it knows. Constraints to ensure that a state is valid are defined using smart contracts. A transaction defines the input and output for a state transformation. Further legal prose can be attached to a transaction to settle future disputes, which makes Corda appealing to financial use cases. Notaries are specific nodes assigned with the responsibility of ensuring that output states are unique successors of input states thereby preventing double spending. When a transaction proposal is created, only the entities related to it execute the smart contract to ensure its validity. Further, notaries check each input state object in a transaction to ensure that they have not been consumed earlier and prevent double-spending. Transactions are committed after the transaction-related entities and the notaries sign them.

2.3 Multichain

Multichain is a fork of Bitcoin and shares many of its features [35]. However, it is designed for a permissioned environment where nodes prove their identity using a handshaking protocol when connecting to other nodes. Each node defines the public address for which it has a private key, and other nodes can send challenge messages to be signed with this key. Unlike Bitcoin, only a few nodes are granted mining privileges, and there is a single validator per block. The validator is scheduled in a round-robin style with tunable parameters. Other participants then execute the individual transactions in a block in the defined global order.

2.4 Quorum

Quorum is a fork of the Go implementation of Ethereum, where the P2P layer was redesigned to allow only authorized nodes [50]. A privacy layer is implemented to support private and public transactions in a permissioned environment. Quorum uses transaction managers to handle encrypted data, including an enclave, which is a hardware security module, to hold private keys. Private transactions are sent to transaction managers for encryption after verifying the sender, and only the entities related to the transaction can receive the decrypted data. Different protocols, such as Raft, IBFT, and QBFT, are employed to attain consensus in the Quorum network. When consensus is reached, all the nodes in the network execute the public transactions in a block, while private transactions are only executed by the entities related to the transaction.

2.5 Diem

Diem (earlier known as Libra) is a permissioned blockchain introduced by Facebook (now Meta) [27]. The network consists of two types of nodes - full nodes

and validators. For every incoming transaction, validators check the signature, balance, and whether the transaction has been replayed, before sharing them with other validators. A BFT protocol (DiemBFT) is used to reach a consensus on the order of transactions. When a validator is elected as the leader, it proposes a block which is forwarded to the other validators for approval. Meanwhile, the transactions in the block are speculatively executed and also shared. Upon consensus, all the transactions of the proposed block are committed. Full nodes are employed to re-execute and store all transactions to provide evidence in the event of a history rewrite attempt. It ensures that validators cannot collude on transaction executions.

3 Benchmarking Guidelines

In this section, we focus on four important aspects to address when benchmarking permissioned blockchains. We consider examples from Fabric [2], Corda [37], Multichain [55], Quorum [60] (four of the most commonly used permissioned blockchains [62]) and Diem [27] when defining each problem statement. We then propose a general methodology for tackling each problem. The methodology targets blockchain developers, benchmarking system developers, as well as those conducting benchmarking studies on blockchains. We aim to bring to light the contributions required from each of them to the blockchain benchmarking space.

3.1 System Configuration

Problem Statement. There is a vast distinction in the system components that compose the different permissioned blockchain implementations. The choice, count, and distribution of the different components significantly affect the performance. For example, the Hyperledger Fabric network consists of validating peers, endorsers, orderers, and clients where the count and distribution of each of these components impact the performance [9,12,38]. Corda offers two configurations for its notary nodes: validating and non-validating. Deploying multiple validating notary clusters can aid load balancing, improving performance [19]. In the Quorum network, performance is influenced by the choice between full nodes with a privacy manager or light nodes [32] for process-intensive tasks as well as boot nodes [14] or static nodes [15] for different peer discovery strategies. Multichain has the concept of data streams, and nodes that subscribe to these streams ensure faster information retrieval [52]. Also, Diem has the concept of validator nodes and full nodes, the choice of which can introduce additional overhead depending on the use case [26]. Therefore, identifying the influential system components and designing the optimal setup is crucial to ensure the best performance for each blockchain implementation. Further, even though the system configuration of each blockchain needs to be individually optimized, the hardware requirements or the hardware cost must be uniform across all blockchains for a fair benchmarking approach [61].

Methodology

1. Blockchain system developers need to provide extensive documentation and experimental results to quantify the influence of system components on the performance for each blockchain. Identifying and documenting a priority-based list of the main components that significantly impact the performance will be highly beneficial. Multichain published a list of tips for performance optimization on their website [53] which includes ideal server specifications, and though they do not provide concrete suggestions, this highlights the need for such documentation from the developers.
2. Benchmarking system users must design an optimal system setup specific to each blockchain based on their documentation. This is a challenging yet crucial task. Individually optimizing the system setup ensures benchmarking the best performing setup of each blockchain. Further, all the blockchains benchmarked together must employ uniform hardware or be limited to uniform hardware costs to ensure fairness [61].
3. Benchmarking system developers must support easy integration and reconfiguration of all system components. Due to the large number and type of components involved, system setup is often complex for blockchains [71]. Benchmarking systems need to provide automation scripts or at least detailed documentation that supports the integration of influential system components apart from the default to ease the system setup process. Further, the optimal system setup varies with use cases, so easy reconfiguration should also be supported.

3.2 Parameter Tuning

Problem Statement. Parameter tuning is a significant factor to consider while benchmarking blockchains, and it is heavily discussed in the literature [9,47,70]. The literature mainly discusses generic parameters, such as block size, while system-specific parameters are largely ignored. However, both are equally important to ensure fair benchmarking. The number of transactions to include in a block is a well-known parameter that influences the performance of most blockchains [9,49], but Corda is an exception since the concept of blocks does not exist [18]. Further, there are system-specific parameters such as the set of cache-related parameters for GoQuorum [30,31] and Corda [17], the validator pool size, endorsement policy, and CouchDB parameters for Fabric [12], or the mining diversity and skip proof-of-work check [51] configurations in Multichain, all of which can be tuned for performance improvements. Also, Diem offers mempool [25] and consensus [24] configurations that are based on its unique implementation. Therefore, individually identifying and tuning the critical parameters for each blockchain is required to benchmark the ideal performance of every system. However, on the other hand, some parameters may impact the system's functionality and must be set equivalently to ensure a fair comparison. For example, Clique is byzantine fault tolerant with eventual finality, while Raft is only crash fault tolerant with immediate finality, and either can be chosen as the consensus protocol in Quorum [16]. If Fabric, which offers only the Raft consensus

protocol, is benchmarked with Quorum, then to ensure fairness, Quorum's consensus protocol needs to be set to Raft. Parameter tuning is often discussed in the literature on benchmarking transaction processing systems [4,34]. However, when considering blockchains, there is a more diversified set of parameters for tuning since the blockchain stack is comprised of numerous layers such as consensus models, access control protocols, database stores, smart contracts, and distributed ledgers.

Methodology

1. Blockchain developers should identify key parameters that influence the performance of their blockchain. They should ensure that all configuration parameters and quantitative evidence of their effect on performance are well documented. Workload-based analysis of these parameters should also be conducted and documented. Further, given the large number of parameters in blockchains, a prioritizing strategy would be beneficial. For example, Fabric has over 50 parameters, and a recent study quantitatively ranked the top parameters that significantly affect the performance [46].
2. Benchmarking system users must tune parameters based on the workload and system setup. Currently, benchmarking is often accomplished with the default parameter values or with a one-time tuning of limited parameters [28,33,57]. However, studies show that parameter tuning significantly depends on the workload and system setup [9,47,70]. Therefore, parameter tuning should be done dependent on the use case that is being benchmarked. Recently, autotuning of blockchains is also being discussed in the literature, which could ease this process [46].

3.3 Workloads and Use Cases

Problem Statement. The third important aspect to consider is the workload employed for benchmarking. Using existing workloads such as YCSB and TPC is a popular choice since these are well established in the community [28,44]. However, blockchains often target different use cases than traditional transaction processing systems. Therefore, reusing existing workloads is often unrealistic and leads to inaccurate assumptions about the performance of a system [33]. But traditional workloads are still useful to benchmark scenarios where enterprises port their existing applications to blockchain systems. Further, blockchain implementations are varied, and each is designed with a specific use case in mind. For example, Fabric cannot handle highly skewed workloads due to its optimistic concurrency control model [9], and Corda supports only point-to-point requests between entities involved in a transaction [18]. Also, Quorum and Corda are mainly popular for financial use cases while Fabric applications range across multiple domains such as supply chain management and healthcare [73]. Further, the system setup, parameters, and transaction definition also vary with the use case. Multichain recommends different performance optimization strategies based on the expected type of workload [53], and Diem defines three different types of transactions based on the client account type [72].

Methodology

1. Benchmarking system developers should focus on both traditional as well as blockchain-specific workloads. Porting traditional workloads such as TPC and YCSB to blockchain environments is a good practice as it corresponds to scenarios where existing enterprise applications are migrated to blockchain platforms. However, the focus should also be given to blockchain-specific workloads, such as supply chain and digital asset management scenarios, to capture realistic performance capabilities better. Apart from workload generation, converting or porting the workloads to support multiple blockchain implementations is an important and challenging engineering task.
2. Benchmarking system developers must also generate system-specific workloads. Such workloads that stress test distinctive blockchains based on their specific design are essential to highlight accurate performance expectations. For example, private transactions in Fabric and point-to-point requests in Corda would need specific workloads different from other generic broadcast transactions. Also, the targeted use cases of each blockchain implementation should be supported.
3. Benchmarking system users and blockchain developers should provide use case-based discussion of benchmarking results. Benchmarking results will quantitatively indicate the most or least performant blockchain. However, a specific blockchain's intended use case must be considered before reaching a viable conclusion. For example, it has been quantitatively shown that Fabric is more performant than Diem [76]. However, Diem supports a byzantine fault-tolerant consensus protocol, while Fabric uses a crash fault-tolerant consensus protocol, both of which are suitable for entirely different use cases. Therefore, evaluation results need to be explored extensively in relation to the blockchain implementation and envisioned use.

3.4 Performance Metrics

Problem Statement. The metrics used for benchmarking depend on the quality being benchmarked. Throughput and latency are the main client-visible metrics generally used in benchmarking blockchains when the focus is on performance; as well, some studies look at metrics reported from the blockchain platform, such as CPU usage or storage. However, there needs to be more clarity about how to define these metrics. Throughput is often defined as the number of transactions committed to a blockchain per second. However, for Fabric, failed transactions are also committed to the blockchain [2]. Latency is often described as the duration between transaction submission and final commit. However, submission time can be considered as the time the client submitted the transaction or the time the transaction entered the consensus protocol [28,45]. Further, depending on whether the blockchain supports immediate or probabilistic finality, the definition of commit time changes [57]. Also, latency is a distribution, and single summary values such as mean or 95-percentile can be quoted, depending on what matters most for the specific use case. Therefore, a uniform definition for

blockchain performance metrics is challenging. Further, system-specific metrics also need to be considered to provide a better understanding for the client. For example, apart from throughput and latency, Diem developers define a metric called *capacity* as "*the ability of the blockchain to store a large number of accounts*" [1].

Methodology

1. Blockchain developers must define generic as well as system-specific performance metrics. Generic metrics should either be uniformly defined for all blockchains along with the system-specific assumptions or be uniquely defined for each blockchain (or both). System-specific performance metrics must be clearly defined, and the necessity for these metrics must also be clarified.
2. Benchmarking system developers should support fine-grained result generation. Since the metric definition varies for each blockchain, publishing all variations of a metric in the results will be helpful for better understanding. In most cases, simple mathematical calculations can provide more fine-grained results. For example, the results from the Caliper benchmarking system display only the "*success throughput*" and not the "*commit throughput*" even though both can be derived from the available results [9].

4 Case Study

In this section, we analyze five different benchmarking systems that support permissioned blockchains [5,8,22,36,40] as well as the corresponding five benchmarking studies conducted using these systems [9,28,33,57,65]. Our discussion is mainly based on the benchmarking studies as this is representative of how the benchmarking system is used in practice. Table 1 summarizes the integrated blockchain systems, available workloads, and published performance metrics for each benchmarking system. We intend to identify the limitations of the current benchmarking systems through this case study which can help develop a more comprehensive system.

Scope. We observe that none of the benchmarking systems currently support all four of the most commonly used permissioned blockchains (Fabric, Corda, Multichain and Quorum). One of the main reasons for benchmarking is for clients to choose the appropriate blockchains based on their requirements. Therefore, a benchmarking system must support at least the most popular blockchain choices. However, the engineering challenge behind implementing such a comprehensive benchmarking system is immense. Alternatively, providing documentation that accurately details the exact procedure to integrate any new blockchain into an existing benchmarking system would be beneficial. Diablo, Gromit, and BCT-Mark provide short documentation or discussions on integrating new blockchains into their benchmarking system [23,57,65]. Caliper provides extensive documentation that details the steps required to implement a connector to integrate a

Table 1. Blockchain Benchmarking Systems

Benchmarking Systems	Supported blockchains (permissioned underlined)	Supported Workloads	Performance Metrics
Blockbench [28]	Ethereum [74], Fabric [2], Parity [58], Quorum [60]	YCSB, smallbank, etherId, doubler, wavesPresale, doNothing, analytics, IOHeavy, CPUHeavy [5]	success throughput, average latency
HyperledgerLab [9], Caliper [40]	Fabric, Ethereum, Besu [39]	simple asset transfer, smallbank, fabcar, synthetic generator, electronic health records, digital music management, e-voting, supply chain management [41,43]	commit throughput, success throughput, average latency
Diablo [33]	Algorand [29], Avalanche [63], Ethereum, Diem [3], Solana [68], Quorum, RedBelly [21,69]	exchange DApp, gaming DApp, webservice DApp, mobility service DApp, video sharing DApp [22]	throughput, average latency, proportion of committed transactions, peak transaction throughput, latency distribution over time
Gromit [57]	Ethereum, Algorand, BitShares [66], Diem, Fabric, Stellar [48], Avalanche	simple asset transfer [36]	peak transaction throughput, average latency
BCTMark [65]	Ethereum, Clique [13], Fabric	synthetic generator, history-based , sorting algorithms [8]	CPU usage, HDD usage, memory consumption, gas cost

new blockchain [75]. This includes the requirements of the connector, implementation, binding, and integration, as well as instructions on how to document the newly developed connector for future users. Despite the well-defined documentation, there has been little effort from the community to integrate more blockchains into Caliper.

System Configuration. The existing benchmarking systems support the evaluation of the different blockchains on scaling hardware configurations. Diablo, Gromit, and HyperledgerLab emulate geo-distribution. However, system configuration is not extensively evaluated in the corresponding benchmarking studies. The number of hardware nodes and, correspondingly, the number of peers in a system are scaled and evaluated but the peer configurations are kept constant. Currently, system configuration is considered independent from the benchmarking systems and is left to the client's responsibility. Providing automated testbed setups for the supported blockchains can ease the benchmarking effort with varying system configurations. The HyperledgerLab benchmarking system includes

such an automated testbed and therefore can evaluate the effect of endorser and database configurations, but it is limited to Fabric.

Parameter Tuning. In the studies we examined, system parameters are mostly kept with the default value used in whichever blockchain is being tested. Blockbench tunes the *difficulty* variable for Ethereum to limit miners from diverging [28]. HyperledgerLab evaluates the effect of system parameters such as *block size* and *endorsement policy* but is limited to Fabric [9]. Tuning the parameters of individual blockchains to ensure the fair comparison of the best performance of all the systems under test is a massive challenge due to the large number of parameters involved. Currently, we identified some of the prominent parameters for the different blockchains discussed in this paper by manually parsing through the multiple documentations and configuration files [15,17,18,26,30,32,38,51]. Blockchain developers must provide more intuitive documentation regarding the performance tuning of their specific blockchain implementation. Consequently, benchmarking systems could automate parameter tuning to ease the benchmarking process.

Workloads and Use Cases. Workloads are well investigated by the existing benchmarking systems, and the supported workloads for each are listed in Table 1. Diablo extracts the workload trace from five real centralized applications and designs corresponding decentralized applications (DApps) to provide a realistic blockchain-specific benchmarking scenario. Blockbench provides popular database benchmarking workloads such as YCSB and small bank, which provides a good understanding of the contrast between blockchains and databases. Blockbench also supports microbenchmarks such as IO-heavy and CPU-heavy, while HyperledgerLab provides synthetic workloads such as read-heavy, update-heavy, or skewed keys. All the existing benchmarking systems also support workloads at different transaction rates. Overall, the workloads supported by the existing benchmarking systems cover many practical and synthetic use cases, ensuring a comprehensive blockchain evaluation. There are also many other blockchain specific workloads available in the literature [9,10,56]. However, developing or extending a benchmarking system to include this extensive set of workloads would be advantageous. The evaluation results of the existing benchmarking systems are well explored and discussed in their corresponding papers. For example, Nasrulin et al. [57] highlight six different findings that summarize the performance of the compared blockchains. However, relating the evaluation results to the implementation specifics of the blockchains and the intended use cases would be helpful for a client trying to choose the ideal blockchain. Gramoli et al. [33] observe that Diem and Avalanche do not support challenging hardware configurations but also point out that such configurations may not be the intended use case for these blockchains. They also highlight that blockchains with eventual consistency scale better, providing a client who requires immediate consistency with realistic expectations. The intended use cases of a specific blockchain and

its implementation specifics, such as its consistency and fault tolerance models, need to be effectively explored while discussing benchmarking results.

Performance Metrics. The existing benchmarking studies evaluate a wide range of performance metrics. Gromit focuses on the peak transaction throughput, the maximum throughput supported by a system before it hangs. Diablo measures the average throughput and a throughput time series, including the peak throughput. BCTMark focuses more on system metrics such as CPU and memory usage. Blockbench measures the success throughput, while HyperledgerLab evaluates the committed throughput, including failed transactions. The importance and reasoning of each of the metrics are well-defined in the benchmarking studies. However, a single benchmarking system that provides a comprehensive set of all the different metrics would be valuable.

5 Related Work

The literature proposes various benchmarking systems and corresponding benchmarking studies for permissioned blockchains, which we have analyzed in our case study. Dinh et al. developed the first benchmarking system for permissioned blockchains with a precise definition for the different abstraction layers [28]. HyperledgerLab [9], which uses the Caliper benchmarking system [40], implemented an automated blockchain (Fabric) network deployment tool to simplify benchmarking experiments. Saingre et al. proposed a blockchain benchmarking system that adheres to the six criteria for a good benchmark [65,67]. Nasrulin et al. investigated the popular consensus protocols and benchmarked representative blockchain systems for each [57]. Gramoli et al. implemented realistic distributed applications to evaluate multiple blockchain systems' performance uniformly [33]. The existing publications focus on developing a benchmarking system, while our work highlights general benchmarking guidelines for the blockchain community, which includes both blockchain and benchmarking system developers. Benchmarks and benchmarking systems are well-established research areas in the database community [4,6,34,61]. However, despite the similarities, the implementation and application differences demand a separate discussion for benchmarking blockchains [64].

6 Conclusion

We analyzed five permissioned blockchains to define specific problem statements regarding four main aspects of benchmarking blockchains. We provide examples from each of the chosen platforms to clarify the problem statements. Further, we discuss a general methodology to tackle each problem statement, highlighting the need for contributions from the developers and users of blockchains and benchmarking systems. We then conducted a case study of five different permissioned blockchain benchmarking systems and the affiliated benchmarking studies

based on our problem statements. We emphasize the current limitations of these systems, which can help improve the state-of-the-art. Given the implementation differences between blockchains and the numerous components, configuration parameters, and metrics specific to each blockchain, one main conclusion from our work is the need for blockchain developers to actively engage in the benchmarking space. We urge blockchain developers to quantitatively identify and define system-specific factors such as the top parameters to tune, the ideal system setup for a fixed hardware configuration or cost, targeted use cases, and performance metrics that can ease the process of benchmarking blockchains.

Acknowledgement. This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008, and by the Bavarian Cooperative Research Program of the Free State of Bavaria - DIK-2002-0013//DIK0114/02.

References

1. Amsden, Z.: The libra blockchain (2020). https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf
2. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–15 (2018)
3. Bano, S., et al.: State machine replication in the libra blockchain (2020). https://developers.libra.org/docs/state-machine-replication-paper. Accessed 19 Dec 2020)
4. Bermbach, D., Wittern, E., Tai, S.: Cloud Service Benchmarking. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-55483-9
5. Blockbench (2020). https://github.com/ooibc88/blockbench. Accessed 14 Aug 2023
6. Brent, L., Fekete, A.: A versatile framework for painless benchmarking of database management systems. In: Chang, L., Gan, J., Cao, X. (eds.) ADC 2019. LNCS, vol. 11393, pp. 45–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12079-5_4
7. Brown, R.G., Carlyle, J., Grigg, I., Hearn, M.: Corda: an introduction. R3 CEV **1**(15), 14 (2016)
8. Btcmark (2020). https://gitlab.inria.fr/dsaingre/bctmark. Accessed 14 Aug 2023
9. Chacko, J.A., Mayer, R., Jacobsen, H.A.: Why do my blockchain transactions fail? a study of hyperledger fabric. In: Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS 2021, pp. 221–234. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3448016.3452823
10. Chacko, J.A., Mayer, R., Jacobsen, H.A.: How to optimize my blockchain? a multi-level recommendation approach. Proc. ACM Manag. Data **1**(1) (2023). https://doi.org/10.1145/3588704
11. Chainhammer (2020). https://github.com/drandreaskrueger/chainhammer. Accessed 14 Aug 2023
12. Chung, G., et al.: Performance tuning and scaling enterprise blockchain applications. arXiv preprint arXiv:1912.11456 (2019)
13. Clique proof-of-authority consensus protocol (2020). https://eips.ethereum.org/EIPS/eip-225. Accessed 14 Aug 2023
14. Configure bootnodes (2020). https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/bootnodes/. Accessed 14 Aug 2023

15. Configure static nodes (2020). https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/static-nodes/. Accessed 14 Aug 2023
16. Consensus protocols (2020). https://docs.goquorum.consensys.net/concepts/consensus. Accessed 14 Aug 2023
17. Corda configurations (2020). https://docs.r3.com/en/platform/corda/4.8/enterprise/node/setup/corda-configuration-fields.html. Accessed 14 Aug 2023
18. Corda key concepts (2020). https://docs.r3.com/en/platform/corda/4.10/community/key-concepts.html. Accessed 14 Aug 2023
19. Corda notaries (2020). https://docs.r3.com/en/platform/corda/4.10/community/key-concepts-notaries.html. Accessed 14 Aug 2023
20. Corda use case directory (2023). https://r3.com/products/use-case-directory-all/. Accessed 14 Aug 2023
21. Crain, T., Natoli, C., Gramoli, V.: Red belly: a secure, fair and scalable open blockchain. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 466–483. IEEE (2021)
22. Diablo blockchain benchmark suite (2020). https://diablobench.github.io/. Accessed 14 Aug 2023
23. Diablo blockchain benchmark suite (2020). https://diablobench.github.io/blockchain-howto. Accessed 14 Aug 2023
24. 20 Diem consensus configurations (2020). https://github.com/diem/diem/blob/latest/config/src/config/consensus_config.rs. Accessed 14 Aug 2023
25. Diem mempool configurations (2020). https://github.com/diem/diem/blob/latest/config/src/config/mempool_config.rs. Accessed 14 Aug 2023
26. Diem validator nodes (2020). https://developers.diem.com/docs/basics/basics-validator-nodes. Accessed 14 Aug 2023
27. Diem white paper (2020). https://developers.diem.com/docs/technical-papers/the-diem-blockchain-paper/. Accessed 14 Aug 2023
28. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: a framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1085–1100 (2017)
29. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, SOSP 2017, pp. 51–68. Association for Computing Machinery, New York (2017).https://doi.org/10.1145/3132747.3132757
30. Go ethereum (2020). https://geth.ethereum.org/docs/interface/command-line-options. Accessed 14 Aug 2023
31. Goquorum configuration file (2020). https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/use-configuration-file/. Accessed 14 Aug 2023
32. Goquorum qlight (2020). https://consensys.net/docs/goquorum/en/latest/concepts/qlight-node/. Accessed 14 Aug 2023
33. Gramoli, V., Guerraoui, R., Lebedev, A., Natoli, C., Voron, G.: Diablo: a benchmark suite for blockchains. In: 18th ACM European Conference on Computer Systems (EuroSys) (2023)
34. Gray, J.: Benchmark Handbook: for Database and Transaction Processing Systems. Morgan Kaufmann Publishers Inc., Burlington (1992)
35. Greenspan, G., et al.: Multichain private blockchain-white paper, pp. 57–60 (2015). http://www.multichain.com/download/MultiChain-White-Paper.pdf
36. Gromit blockchain benchmarking tool (2020). https://github.com/grimadas/gromit. Accessed 14 Aug 2023

37. Hearn, M., Brown, R.G.: Corda: A distributed ledger. Corda Technical White Paper **2016** (2016)
38. How fabric networks are structured (2020). https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html. Accessed 14 Aug 2023
39. Hyperledger besu (2020). https://www.hyperledger.org/use/besu. Accessed 14 Aug 2023
40. https://hyperledger.github.io/caliper/ (2020), [Online; accessed 14-August-2023]
41. Hyperledger caliper benchmarks (2020). https://github.com/hyperledger/caliper-benchmarks. Accessed 14 Aug 2023
42. Hyperledger foundation case studies (2023). https://www.hyperledger.org/learn/case-studies. Accessed 14 Aug 2023
43. Hyperledgerlab ii (2020). https://github.com/MSRG/HyperLedgerLab-2.0. Accessed 14 Aug 2023
44. Klenik, A., Kocsis, I.: Porting a benchmark with a classic workload to blockchain: TPC-c on hyperledger fabric. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, pp. 290–298 (2022)
45. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proceedings of the 25th USENIX Conference on Security Symposium, SEC 2016, pp. 279–296, USENIX Association, USA (2016)
46. Li, M., et al.: Auto-tuning with reinforcement learning for permissioned blockchain systems. Proc. VLDB Endow. **16**(5), 1000–1012 (2023). https://doi.org/10.14778/3579075.3579076
47. Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: a deep reinforcement learning approach. IEEE Trans. Ind. Inf. **15**(6), 3559–3570 (2019). https://doi.org/10.1109/TII.2019.2897805
48. Lokhava, M., et al.: Fast and secure global payments with stellar. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, pp. 80–96. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3341301.3359636
49. Mazzoni, M., Corradi, A., Di Nicola, V.: Performance evaluation of permissioned blockchains for financial applications: the consensys quorum case study. Blockchain: Res. Appl. **3**(1), 100026 (2022). https://doi.org/10.1016/j.bcra.2021.100026
50. Morgan, J.: Quorum Whitepaper. JP Morgan Chase, New York (2016)
51. Multichain configurations (2020). https://www.multichain.com/developers/blockchain-parameters/. Accessed 14 Aug 2023
52. Multichain data streams (2020). https://www.multichain.com/developers/data-streams/. Accessed 14 August 2023
53. Multichain performance optimization (2020). https://www.multichain.com/developers/performance-optimization/. Accessed 14 Aug 2023
54. Multichain product partners (2023). https://www.multichain.com/product-partners/. Accessed 14 Aug 2023
55. Multichain white paper (2020). https://www.multichain.com/download/MultiChain-White-Paper.pdf. Accessed 14 Aug 2023
56. Nasirifard, P., Mayer, R., Jacobsen, H.A.: Fabriccrdt: a conflict-free replicated datatypes approach to permissioned blockchains. In: Proceedings of the 20th International Middleware Conference, Middleware 2019, pp. 110–122. Association for Computing Machinery, New York (2019).https://doi.org/10.1145/3361525.3361540

57. Nasrulin, B., De Vos, M., Ishmaev, G., Pouwelse, J.: Gromit: benchmarking the performance and scalability of blockchain systems. In: 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), pp. 56–63. IEEE (2022)
58. Parity: Blockchain infrastructure for the decentralised web (2020). https://www.parity.io/. Accessed 14 Aug 2023
59. Quorum blockchain in action (2023). https://consensys.net/quorum/enterprise/. Accessed 14 Aug 2023
60. Quorum white paper (2020). https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf. Accessed 14 Aug 2023
61. Raasveldt, M., Holanda, P., Gubner, T., Mühleisen, H.: Fair benchmarking considered difficult: common pitfalls in database performance testing. In: Proceedings of the Workshop on Testing Database Systems, DBTest 2018. Association for Computing Machinery, New York (2018).https://doi.org/10.1145/3209950.3209955
62. Rauchs, M., Blandin, A., Bear, K., McKeon, S.B.: 2nd global enterprise blockchain benchmarking study. SSRN 3461765 (2019)
63. Rocket, T., Yin, M., Sekniqi, K., van Renesse, R., Sirer, E.G.: Scalable and probabilistic leaderless BFT consensus through metastability (2020)
64. Ruan, P., et al.: Blockchains vs. distributed databases: dichotomy and fusion. In: Proceedings of the 2021 International Conference on Management of Data, SIGMOD 2021, pp. 1504–1517. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3448016.3452789
65. Saingre, D., Ledoux, T., Menaud, J.M.: Bctmark: a framework for benchmarking blockchain technologies. In: 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), pp. 1–8. IEEE (2020)
66. Schuh, F., Larimer, D.: Bitshares 2.0: general overview (2017)
67. Smaalders, B.: Performance anti-patterns: want your apps to run faster? here's what not to do. Queue **4**(1) (2006). https://doi.org/10.1145/1117389.1117403
68. Solana: a new architecture for a high performance blockchain v0.8.13 (2020). https://solana.com/solana-whitepaper.pdf. Accessed 14 Aug 2023
69. Tennakoon, D., Gramoli, V.: Smart red belly blockchain: enhanced transaction management for decentralized applications. arXiv preprint arXiv:2207.05971 (2022)
70. Thakkar, P., Nathan, S., Viswanathan, B.: Performance benchmarking and optimizing hyperledger fabric blockchain platform. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 264–276 (2018). https://doi.org/10.1109/MASCOTS.2018.00034
71. Tran, N.K., Babar, M.A., Walters, A.: A framework for automating deployment and evaluation of blockchain networks. J. Netw. Comput. Appl. **206**, 103460 (2022). https://doi.org/10.1016/j.jnca.2022.103460
72. Types of transactions (2020). https://developers.diem.com/docs/transactions/txns-types/. Accessed 14 Aug 2023
73. Valenta, M., Sandner, P.G.: Comparison of ethereum, hyperledger fabric and corda (2017)
74. Wood, D.D.: Ethereum: a secure decentralised generalised transaction ledger (2014)
75. Writing connectors (2020). https://hyperledger.github.io/caliper/v0.5.0/writing-connectors/. Accessed 14 Aug 2023
76. Zhang, J., et al.: Performance analysis of the libra blockchain: an experimental study. In: 2019 2nd International Conference on Hot Information-Centric Networking (HotICN), pp. 77–83. IEEE (2019)

Benchmarking Generative AI Performance Requires a Holistic Approach

Ajay Dholakia[1(✉)], David Ellison[1], Miro Hodak[2], Debojyoti Dutta[3], and Carsten Binnig[4]

[1] Infrastructure Solutions Group, Morrisville, NC, USA
{adholakia.dellison,adholakia.dellison}@lenovo.com
[2] AMD, San Jose, CA, USA
miro.Hodak@amd.com
[3] Nutanix, , San Jose, CA, USA
debojyoti.dutta@nutanix.com
[4] TU, Darmstadt, Germany
carsten.binnig@cs.tu-darmstadt.de

Abstract. The recent focus in AI on Large Language Models (LLMs) has brought the topic of trustworthy AI to the forefront. Along with the excitement of human-level performance, the Generative AI systems enabled by LLMs have raised many concerns about factual accuracy, bias along various dimensions, authenticity and quality of generated output. Ultimately, these concerns directly affect the user's trust in the AI systems that they interact with. The AI research community has come up with a variety of metrics for perplexity, similarity, bias, and accuracy that attempt to provide an objective comparison between different AI systems. However, these are difficult concepts to encapsulate in metrics that are easy to compute. Furthermore, AI systems are advancing to multimodal foundation models that further make creating simple metrics a challenging task. This paper describes the recent trends in measuring the performance of foundation models like LLMs and multimodal models. The need for creating metrics and ultimately benchmarks that enable meaningful comparisons between different Generative AI system designs and implementations is getting stronger. The paper concludes with a discussion of future trends aimed at increasing trust in Generative AI systems.

Keywords: Artificial Intelligence · Benchmarks · Generative AI · Foundation Models · Trustworthy AI

1 Introduction

AI System benchmarks typically focus on throughput and accuracy, the so-called "speeds and feeds" metrics. The recent rise of AI systems across a broad spectrum of use cases has brought attention to new metrics that assess the efficacy of the end-user application enabled by AI. Standards and benchmarking organizations have begun to add performance per watt metrics and have either released the benchmarking requirements or are in the process of doing so [1–3]. In a recent paper [4], the authors argue for defining

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 34–43, 2024.
https://doi.org/10.1007/978-3-031-68031-1_3

benchmarks that target energy efficiency, explainability, bias and other such metrics. The ultimate objective of such initiatives is to increase trust in the use of AI systems.

The recent and meteoric rise of Generative AI in general, and Large Language Models (LLMs) in particular, is making it imperative that performance evaluation of AI systems address the needs of the end user [5]. It is not sufficient to say that the AI system under evaluation runs "fast" or uses "optimal energy". Rather, it is now becoming necessary to evaluate metrics such as context-aware factual accuracy, perplexity, toxicity, and bias. In addition, application-specific performance is also being evaluated, particularly when the LLM engine is used to carry out tasks otherwise attributed to human intelligence.

A host of LLMs are being designed and offered for use by teams in academia as well as industry [6–8]. In a paper included in this proceedings, recent LLMs are compared with each other and examined from a standard benchmarking perspective [9]. The need for optimizing the design of these LLMs for a more efficient energy usage, while maintaining target throughput, is discussed. It is argued that a “one size fits all” approach is not conducive to widespread adoption of LLMs across a spectrum of use-cases and usage profiles.

AI research community in academia as well as industry is actively working on developing metrics that go beyond just throughput and energy usage. The need for more comprehensive metrics has started to shift the focus of benchmarking from independent metrics to combined ones that encompass tasks, skills and usage scenarios [10, 11].

A larger trend of multimodal foundation model is also emerging, enabling use of training data beyond text to images and other formats in creating large pre-trained “foundation” models like Dall-E [5]. Furthermore, the use of multiple LLMs and foundation models in service of a single higher-level task has the potential of deploying AI systems in complex decision-making applications. We describe one such application of LLMs and generative AI to database management systems.

In summary, the adoption of Generative AI systems will require a holistic approach to benchmarking that measure not only how fast or efficient these systems are, but also how useful and trustworthy they are. This paper examines the emerging need for adopting such an approach, highlighting the challenges that need to be addressed and the expected benefits to be achieved.

This paper is organized as follows. Section 2 provides a summary of current benchmarking approaches for evaluating AI systems. Section 3 focuses on recent LLMs and summarizes their key features. Section 4 advocates the need for a holistic approach for benchmarking generative AI systems along multiple dimensions. Section 5 illustrates a specific area of application of generative AI, namely, database management systems (DBMS). Section 6 concludes the paper and outlines areas of ongoing work.

2 Benchmarking Approaches for AI Systems

The advent of generative AI made it clear that a holistic approach is needed for AI benchmarking. While performance remains important, other factors also come into play. For example, in generative AI it is unclear what the accuracy metric should be. This is straightforward in non-generative AI, such as image classification, but generative AI needs to rely on several proxy metrics instead. Similarly, bias and other non-desirable features are intrinsic part of generative AI and should not be treated separately.

A new evaluation need emerges in generative AI datasets. They can only be examined by automated tools, which creates an opportunity to "benchmark" individual datasets to compare both negative and positive characteristics of datasets.

Current understanding of benchmarking that narrowly focuses on performance needs to evolve. In AI, the state of the art, as embodied in MLPerf benchmarks [1], takes a piecemeal approach where only small parts of the AI workflow are evaluated, namely the compute intensive training and inference of deep learning models, which are benchmarked separately. TPCx-AI uses a somewhat larger aperture and includes data manipulation together with training and inference to create an end-to-end benchmark [2, 3, 12]. However, focusing on traditional machine learning makes this benchmark not representative of current AI trends.

We advocate for an approach that goes much further and considers performance as one of the characteristics of AI workloads alongside others. Currently, benchmarking fixes a workload ahead of time, but with AI having a multitude of options that affect performance, one can envision a system in which performance is measured along with metrics such as accuracy expressed in multiple definitions, level of bias, etc. This would enable end users to choose a particular model or a model/implementation infrastructure combination.

MODEL LEADERBOARD

Model	Macro F1 %	Throughput examples/second	Memory GiB	Fairness %	Robustness %	Dynascore
DeBERTa default params	73.62	7.61	4.66	94.16	61.10	60.38
RoBERTa default params	71.25	8.90	4.59	94.02	58.97	59.18
T5 default params	70.66	7.12	10.03	93.41	59.51	58.33
ALBERT default params	68.33	9.99	2.13	93.54	60.46	57.94
BERT default params	66.89	9.05	5.89	93.43	49.85	56.33
Roberta XLM	60.81	5.33	1.80	96.50	63.15	54.48
FastText default params	52.66	32.32	1.75	78.27	42.10	48.29
Majority Baseline	17.59	32.70	1.10	100.00	100.00	38.15

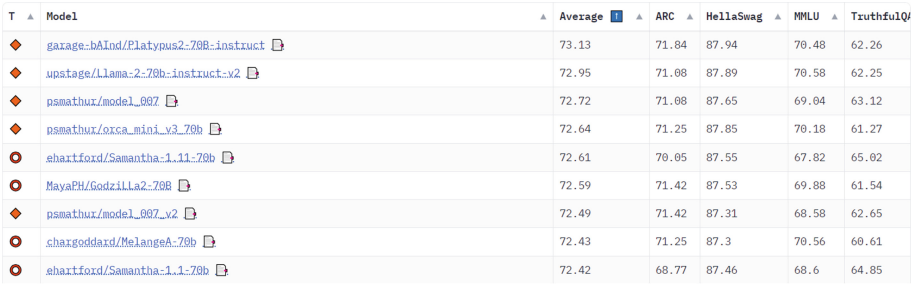
Fig. 1. Screenshot of Sentiment Analysis Challenge results for MLCommons DataPerf

There are several recent developments in that direction. One example is MLCommons DataPerf Work Group [13], whose purpose is to build leaderboards for data and data-centric algorithms by running challenges. The challenges run on DynaBench platform [14], which runs in a web browser and supports human-and-model-in-the-loop dataset creation. Annotators seek to create examples that a target model will misclassify, but that another person will not. The workgroup has launched several challenges and results for a sentiment analysis are shown in the screenshot in Fig. 1. Note that the results list several characteristics such as Fairness and Robustness alongside performance (Throughput). The challenge for DataPerf is adoption – leading AI companies have to date stayed away while most participants are from academia and startups.

Another example is Hugging Face, which publishes LLM Leaderboard evaluating LLM models using several metrics 15]. A screenshot is shown in Fig. 2, where for each model four key metrics are shown (higher score is a better score):

- AI2 Reasoning Challenge - a set of grade-school science questions.

- HellaSwag - a test of commonsense inference, which is easy for humans (~95%) but challenging for SOTA models.
- MMLU - a test to measure a text model's multitask accuracy. The test covers 57 tasks including elementary mathematics, US history, computer science, law, and more.
- TruthfulQA - a test to measure a model's propensity to reproduce falsehoods commonly found online.



T	Model	Average ↑	ARC	HellaSwag	MMLU	TruthfulQA
🔶	garage-bAInd/Platypus2-70B-instruct	73.13	71.84	87.94	70.48	62.26
🔶	upstage/Llama-2-70b-instruct-v2	72.95	71.08	87.89	70.58	62.25
🔶	psmathur/model_007	72.72	71.08	87.65	69.04	63.12
🔶	psmathur/orca_mini_v3_70b	72.64	71.25	87.85	70.18	61.27
⭕	ehartford/Samantha-1.11-70b	72.61	70.05	87.55	67.82	65.02
⭕	MayaPH/GodziLLa2-70B	72.59	71.42	87.53	69.88	61.54
🔶	psmathur/model_007_v2	72.49	71.42	87.31	68.58	62.65
⭕	chargoddard/MelangeA-70b	72.43	71.25	87.3	70.56	60.61
⭕	ehartford/Samantha-1.1-70b	72.42	68.77	87.46	68.6	64.85

Fig. 2. Screenshot of Hugging Face LLM Leaderboard

Overall, these efforts are very valuable, and it will be interesting to see if these approaches are more widely adopted.

3 LLMs Used for Benchmarking and Associated Responsible AI Concerns

The Large Language Models that are in current use have numerous advantages and disadvantages. In this section a few of the major LLMs options are considered as they apply to benchmarking. It is beyond the scope of this work to review all LLMs that currently exist. See [16] for a comprehensive review on the topic.

BERT, short for "Bidirectional Encoder Representations from Transformers" was for a very long time the industry standard for Natural Language Processing (NLP) [6]. At the time it was introduced, 340M (million) parameters were considered very large. However, it still fit within reasonable size constraints for the day, allowing it to be benchmarked in a similar manner to other AI benchmarks in computer vision or recommender systems as evidenced by early MLPerf benchmarks [17]. It is no longer considered the standard for NLP, and is arguably not even an LLM, but it bears mentioning because of its prominence in the history of LLMs and especially in the benchmarking community.

LLaMa varies from 7B (billion) to 65B parameters and hence is a LLM by our modern evaluation and is open-sourced [16]. The first version of LLaMa was not commercially licensed which limited its impact on industry, even though it had a large impact in the research community. LLaMA 2 changed that by being commercially licensed and therefore available for use, benchmarking, and reselling by major corporations.

GPTJ is critical in the benchmarking community because of its ease of use [16]. It is relatively small at 6B parameters and has a very small relative dataset to train on that

is only 800GB. This makes it ideal for benchmarking and indeed, MLPerf, the leading AI benchmark has released a round of benchmarks using this LLM [18].

The LLMs also raise some Responsible AI concerns, see [19] and for medically related issues see [20]. We will focus here Responsible AI concerns specifically with benchmarking.

Hallucinations are a major topic of discussion with LLMs and are broadly defined as responses with factual errors and posing potential risks for the users [5]. We need better benchmarks for measuring the degree and harmfulness of hallucinations. Unfortunately, this tends to be highly contextual and subjective and hard to measure. How harmful is it to manufacture a false reference in a legal case for example vs. a false reference in a medical report?

Context length is a concern, because many queries are context dependent, for example tied to a location or subject. "Who is the best lawyer?" needs to consider both the state and specialty you are talking about – such as a New York Divorce Attorney or a San Francisco Personal Injury Attorney. The Natural Questions NQ-Open dataset measures that 16.5% of information seeking questions are context dependent [21]. Controlling and measuring this is an important benchmarking consideration.

Once we start adding other modalities to LLMs such as image and video the benchmarking considerations start to grow exponentially. How do you standardize a LLM that generates video versus music? This area is largely unexplored at this moment. This is especially important in the medical field as diagnosis generally require both text (e.g., doctor's notes, medical records) and images (e.g., X-rays, MRI scans). In theory, because multimodality is adding data, it should boost benchmarking performance. Having images and text for a product description ideally is more informative than dealing with the data modalities separately for example. Does it make sense to judge single modality models in the same framework as multimodal models? As with many things in this space it remains an open question.

RLHF (Reinforcement Learning from Human Feedback) is not the best way to quantify the performance of a model. It just tells us that A is better than B. The question remains "How much better is A than B?" Without more continuously quantifiable metrics we will be held back in our benchmarking progress. We need to define a continuous variable and not a binary one.

We have discussed the models and what makes certain models usable and benchmarkable, starting from BERT, to commercially licensed models like LLaMa 2, to small open-source models like GPT-J. Then we have discussed four Responsible AI concerns with benchmarking LLMs – hallucinations, context length, other modalities, and quantifying performance with RLHF.

4 Holistic Approach for Benchmarking Generative AI System

Benchmarks for ML models have traditionally focused on performance metrics such as accuracy and power [1–3]. In recent times, it expanded into infrastructure components such as storage and it diversified into vertical specific benchmarks like medicine AI [22].

With the rapid growth in LLMs, we need holistic benchmarking, and explore a more multi-dimensional set of metrics to characterize both models and infrastructure. We also need to extend into datasets (data-centric AI):

- **Infrastructure**: Let us consider evaluating infrastructure for a moment. Instead of evaluating unidimensional metrics like time to complete training with restricted memory accelerator configuration, we need systems benchmarks including storage, memory and compute.
- **Models**: This is where the growth will happen. We will have specialized "standardized" LLM benchmarks. There has been a surge in LLM leaderboards and benchmarks. We need an industry standard that will help users pick the right models for the right job (e.g., summarization, coding).
- **Datasets**: Foundation models cannot be used with confidence unless customers understand the datasets they were trained on. The lack of this transparency and training data lineage makes it harder for enterprise legal teams to make decisions.

Examples of recent work in developing standardized benchmarks for LMs are frameworks such as HELM [10] and FLASK [11].

The HELM framework lays out the foundation for ongoing evaluation of LMs by identifying the scenarios where LMs can be used and the metrics that evaluate the LMs for these scenarios. The notion of LM applicability to multiple scenarios necessitates the need for multiple metrics. Considering this end-to-end approach puts the focus of evaluation on the target LM rather than a specific combination of LM and the underlying system for implementation. Definition and comparison of different options for LM implementation can then follow as the next step in the benchmarking effort to arrive at the most suitable design. The key evaluation components in HELM are the following:

- Scenario such as IMDB
- Adaptation such as prompting applied to a model such as GPT-3
- Metrics such as robustness

Each evaluation run selects a specific scenario, a specific model and adaptation process and one or more metrics. Multiple evaluation runs span the breadth of scenarios that an LM can be used for under standardized conditioned and measure relevant metrics to arrive at results that allow comparison of LMs for desired usage scenarios.

The FLASK framework focuses on skills rather than tasks that are carried out using LLMs. Their motivation is that benchmarking LLMs using specific metrics like accuracy does not provide sufficient guidance for selecting the best LLMs for a desired use case. Furthermore, overall scoring of model responses, either by humans or by automated model-based mechanisms also falls short of enabling meaningful comparison between different LLMs. The proposed protocol is based on four primary abilities and twelve fine-graine skills:

- Logical Thinking: Logical Correctness, Logical Robustness, Logical Efficiency
- Background Knowledge: Factuality, Commonsense Understanding
- Problem Handling: Comprehension, Insightfulness, Completeness, Metacognition
- Use Alignment: Conciseness, Readability, Harmlessness.

Evaluation using these skills allows task-agnostic scoring that, in turn, enables comparison of various open-source and proprietary LLMs. A total of 1740 evaluation instances were collected from various NLP datasets. A scoring scale between 1 and 5 is used in both human-based and model-based evaluations. These two mechanisms for evaluation are observed to be highly correlated.

The FLASK framework was used in [11] to compare the following LLMs:

- Vicuna 13B
- Alpaca 13B
- LLaMA2 Chat 70B
- GPT-3.5
- Bard
- Claude
- GPT-4.

The key findings from this comparison are summarized as follows:

- Open-source LLMs significantly underperformed proprietary LLMs for Logical Thinking and Background Knowledge abilities.
- Some skills like Logical Correctness and Logical Efficiency require larger model sizes to sufficiently acquire them.
- A subset of very difficult instances (so-called FLASK-HARD set) was created to evaluate LLMs in a challenging setting. Even the proprietary LLMs were seen to struggle with up to 50% performance degradation for some skills compared to the full FLASK evaluation set.

5 Applying LLMs to DBMSs

The integration of Large Language Models (LLMs) with Database Management Systems (DBMSs) is a rapidly evolving area of research that holds immense promise for transforming how we interact with and utilize databases. To ensure that the benchmarking of LLMs in this context is meaningful and effective, it is crucial to adopt a holistic approach to account for the diverse range of applications and functionalities that LLMs can bring to DBMSs. As such, not one benchmark but a variety of benchmarks are needed that cover the breadth of functions that LLMs can provide for DBMSs. Furthermore, a comprehensive set of benchmarks are needed that assess not only performance, but also various other critical aspects to provide a well-rounded evaluation.

Benchmarking LLMs in DBMSs requires a clear distinction between scenarios where LLMs are used to replace existing functionality and where they add new capabilities:

Existing Functionality: In cases where LLMs are used to replace existing functionality, such as using LLMs to enable better SQL query rewriting to improve the performance of SQL queries (e.g., by unnesting complex SQL statements), it is reasonable to start with established benchmarks like TPC-H or TPC-DS that primarily measure query performance. However, traditional benchmarks are inadequate in this context. We must introduce new metrics since LLM-based rewrites might not always return a correct SQL query. As such, new "correctness" criteria to evaluate the accuracy and reliability of LLM-generated queries need to be reported.

New Functionality: When LLMs are employed to support entirely new input modalities for databases, like text or images, novel benchmarks tailored to these specific use cases become imperative. These benchmarks should consider how effectively LLM-enhanced DBMSs handle these new features, ensuring that evaluation criteria align with the unique

demands of the added functionality. For example, there has been recent work that has shown that LLMs can be used to integrate other modalities then tables into databases [24] or integrate LLMs themselves as another data source in a DBMS and query them using SQL [23, 25].

In addition to performance metrics, benchmarking of LLM-enhanced DBMSs should encompass a spectrum of essential aspects:

- Robustness Analysis: Assess the robustness of LLM-enhanced DBMSs through negative test cases, highlighting the limitations of LLM models. For example, in the context of natural language interfaces for databases, it's crucial to identify which classes of queries are supported and which are not, shedding light on the system's boundaries.
- Systematic Evaluation of LLM Capabilities: Borrow from research beyond the database community and systematically analyze what LLMs can and cannot do. For instance, studies from other fields have examined an LLM's accuracy in arithmetic tasks. Incorporating such systematic tests into benchmarks can provide a more comprehensive view of LLM capabilities.
- Training overheads: Report on the overall training overheads. This includes measuring reporting on the initial training overhead against fine-tuning overhead, helping to understand the practicality of deploying these systems in real-world settings.
- Model Size vs. Accuracy: Examine the trade-offs between model size and accuracy of LLM-enhanced DBMSs.
- Energy Efficiency: Evaluate the energy efficiency of LLM-enhanced DBMSs, recognizing the growing concern for sustainable computing solutions.
- Latency: Measure the latency of LLM models, particularly if they are used in performance-critical DBMS tasks, ensuring that their response times align with practical requirements.

In summary, the benchmarking of LLMs for data management tasks, especially within LLM-enhanced DBMSs, requires a nuanced approach that addresses both the replacement and addition of functionality. Furthermore, comprehensive benchmarks should encompass not only performance but also robustness, systematic evaluation of capabilities, fine-tuning overhead, model size, energy efficiency, and latency. This holistic perspective will enable us to effectively harness the power of LLMs in database systems while providing a well-rounded evaluation framework for their integration.

6 Summary and Conclusions

This paper focused on articulating the need for developing holistic benchmarks for Generative AI systems. Benchmarking AI systems in general has largely focused on performance metrics like accuracy, throughput and, more recently, energy efficiency. The rapid rise of Generative AI systems has highlighted the need to go beyond such "speeds and feeds" metrics. Moreover, human-level performance of Generative AI systems in a vast array of tasks is enabling a shift towards metrics such as factuality, bias, toxicity and domain-specific completeness. Evaluation mechanism involving humans as well

as model-based techniques are being proposed. Multi-modal foundation models are expected to be deployed in a variety of complex decision-making scenarios.

In this paper, various ways to incorporate the holistic approach were described. In devising the benchmarks, it is necessary to include the models themselves, the datasets used for training as well as the infrastructure selected to implement and use the models. Within each of these categories, new metrics being proposed to capture the broader performance evaluation were presented.

All these trends motivate the need for adopting a holistic approach for evaluating and benchmarking modern AI systems. Early work in this area is laying the foundation for such a holistic approach. However, as discussed above, significantly more work is needed to develop, debate, agree on and select metrics and benchmarks that enable Generative AI systems to be evaluated in a comprehensive manner.

References

1. MLPerf. https://mlcommons.org/.
2. TPCx-AI. https://www.tpc.org/tpcx-ai/default5.asp.
3. Transaction Processing and Performance Council, "TPC Express Benchmark ™ AI – Full Disclosure Report. (2022)
4. Dholakia, A., Ellison, D., Hodak, M., Dutta, D.: Going beyond speeds and feeds: benchmarking considerations for trustworthy and responsible AI. In: 14th TPC Technology Conference (2022)
5. Bommasani, R., et.al.: On the opportunities and risks of foundation models (2022). arXiv preprint https://arxiv.org/pdf/2108.07258.pdf.
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019). arXiv preprint arXiv:1810:04805v2
7. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language Models are Unsupervised Multitask Learners (2019)
8. Touvron, H., et al.: LlaMA: Open and efficient foundation language models (2023). arXiv: 2302.13971v1.
9. Hodak, M., Van Buren, C., Jiang, X., Ellison, D., Dholakia, A.: Benchmarking large language models: opportunities and challenges. In: 15th TPC Technology Conference (2023)
10. Liang, P., et al.: Holistic Evaluation of Language Models (2022). arXiv:2211.09110. Accessed 28 Aug 2023
11. Ye, S., et al.: FLASK: Fine-grained Language Model Evaluation based on Alignment Skill Sets (2023).arXiv:2307.10928. Accessed 28 Aug 2023
12. Liu Olesiuk, Y., Hodak, M., Ellison, D., Dholakia, A.: More the merrier: comparative evaluation of TPCx-AI and MLPerf benchmarks for AI. In: Nambiar, R., Poess, M. (eds.) Performance Evaluation and Benchmarking, TPCTC 2022. Lecture Notes in Computer Science, vol 13860. Springer, Cham. https://doi.org/10.1007/978-3-031-29576-8_5
13. DataPerf Working Group. https://mlcommons.org/en/groups/research-dataperf/
14. Kiela, D., et al.: Dynabench: Rethinking Benchmarking in NLP (2021) arXiv:2104.14337. Accessed 28 Aug 2023
15. Beeching, E., et al.: Open LLM Leaderboard, Hugging Face (2023) https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.
16. Zhao, W.X., et al.: A Survey of Large Language Models (2023) https://arxiv.org/abs/2303.18223.
17. MLPerf Training. https://mlcommons.org/en/training-normal-07/

18. MLPerf Inference. https://mlcommons.org/en/inference-datacenter-31/
19. Lu, Q., et al.: Towards Responsible AI in the Era of ChatGPT: A Reference Architecture for Designing Foundation Model-based AI Systems (2023). https://arxiv.org/abs/2304.11090.
20. Harrer, S.: Attention is not all you need: the complicated case of ethically using large language models in healthcare and medicine (2023) https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10025985/.
21. Google. https://ai.google.com/research/NaturalQuestions
22. Karargyris, A., et al.: MedPerf: Open Benchmarking Platform for Medical Artificial Intelligence using Federated Evaluation (2021) arXiv:2110.01406, Accessed 28 Aug 2023
23. Saeed, M., De Cao, N., Papotti, P.: Querying Large Language Models with SQL (2023). CoRR abs/2304.00472
24. Urban, M., Binnig, C.: Hybrid database operations: learned operations for seamless querying of textual and tabular data. In: LWDA, 8–11 (2022)
25. Urban, M., Nguyen, D.D., Binnig, C.: OmniscientDB: a large language model-augmented DBMS that knows what other DBMSs do not know. In: aiDM '23: Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, pp. 1–7 (2023)

Graph Stores with Application-Level Query Result Caches

Hieu Nguyen[1(✉)], Jun Li[1], and Shahram Ghandeharizadeh[2]

[1] eBay, San Jose, CA, USA
{hieunguyen,junli5}@ebay.com
[2] USC, Los Angeles, CA, USA
shahram@usc.edu

Abstract. At eBay, our graph store is experiencing an exponential growth. Its workload consists of read-only queries and two types of read-write batch updates, streaming and scheduled batch loaders. Our objective is to enhance the latencies of the 95^{th} and 99^{th} percentile of our read-only queries, because their results are used for real time decision making. We achieve this by caching the final result of queries. With repeat queries, we look up their final results using the cache instead of processing them using the graph store and its transactional key-value store. Writes compute their impacted cache entries and delete them. The resulting graph store with application level query result caches provides strong consistency. We present performance numbers from our production workload, highlighting both the benefits and the overheads of using the query result cache.

1 Introduction

Graph data model is a preferred choice by multiple applications, ranging from product recommendation, fraud detection to information technology infrastructure management and knowledge graph. For these application, exploring relationships of entities via graph traversals in a graph model is easier and more intuitive than alternative data models such as SQL of a relational database. A graph model represents the entities and their relationships naturally via graph elements, including vertices and edges. At eBay, several important use cases rely on our transactional graph database service for their daily critical business operations.

One of our biggest use cases has been in production for several years. At the time of this writing, it consists of billions of vertices and edges with a peak read traffic in excess of 4000 graph traversals per second. The workload consists of read-only queries and two types of read-write batch updates: streaming and scheduled loaders. While the read-only queries are interactive and their latency impact real-time decision making, the read-write batch updates do not impact end-user and hence, favor throughput over latency.

Figure 1 shows our workload's read to write ratio for twelve days. Typically there are hundreds of reads for every write. At times, there are no writes at all,

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 44–58, 2024.
https://doi.org/10.1007/978-3-031-68031-1_4

see top of Fig. 1 labeled 100% reads. The motivation for our work is to improve the latencies of the 95^{th} and 99^{th} percentile of the read-only queries. Although the 50^{th} percentile latency is competitive (5–8 ms), the application suffers from unacceptably high 95^{th} and 99^{th} percentile latency (150–700 ms).

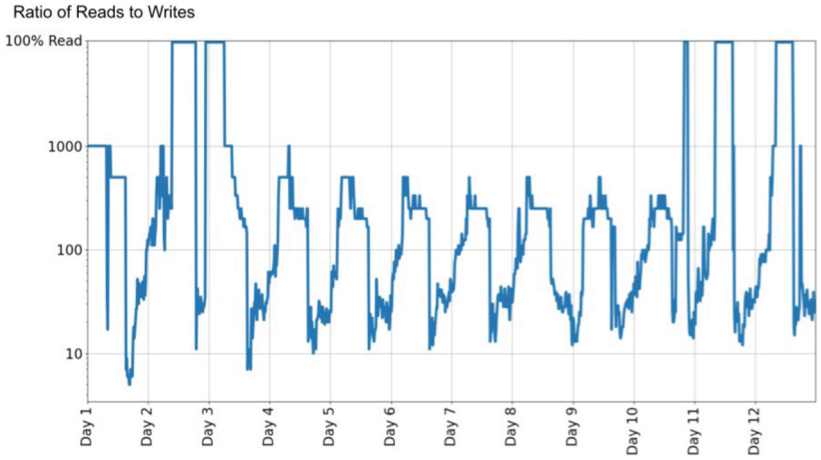


Fig. 1. Ratio of reads to writes for a twelve day period. The y-axis is log scale, highlighting the dominance of the read operations in our workload.

In this paper, we reduce the latency of select read-only queries by caching their final query results. We examine the latency of different query templates and pick a subset for caching. The approach is tailored for our specific application and is non-trivial because it keeps the cache consistent with the transactional data store in the presence of writes. With this approach, we could pick ten selective query templates to cache for our specific application and improve their latency significantly, 4–50×. It helped reduce the overall read latency (1.3× and 1.5× for the 95^{th} and 99^{th} percentile latency). Our approach pushes the overhead of caching to writes. Writes must find the impacted cache entries and delete them. This degrades the response time of read-write transactions by 5–10×. See Sect. 4 for details.

The rest of this paper is organized as follows. Section 2 presents a spectrum of caching techniques and the subset that constitutes the focus of this paper. We present eBay's graph store using a query result cache in Sect. 3. An evaluation of the system using mirrored traffic from a production system is presented in Sect. 4. Section 5 presents brief conclusions and future research directions.

2 Caching and Strong Consistency

Figure 2 shows a spectrum of caching techniques along three dimensions: Granularity of cache entries, Transparency, and Consistency. Granularity of cache

entries dictates how an application's read[1] processes the entries. In its simplest, the cache entry may correspond to the final result of a read. It changes the processing of an application read with a data store as follows. It requires the read to first lookup the cache for its final results. If its results are found (a cache *hit*) then the application proceeds to provide the results as its output. Otherwise (a cache *miss*), the read is processed using the data store to obtain its final results. Next, the read populates the cache with an entry that contains this final results and provide the results as its output. The two operations that constitute this last step may be performed concurrently. This last step populates the cache with entries.

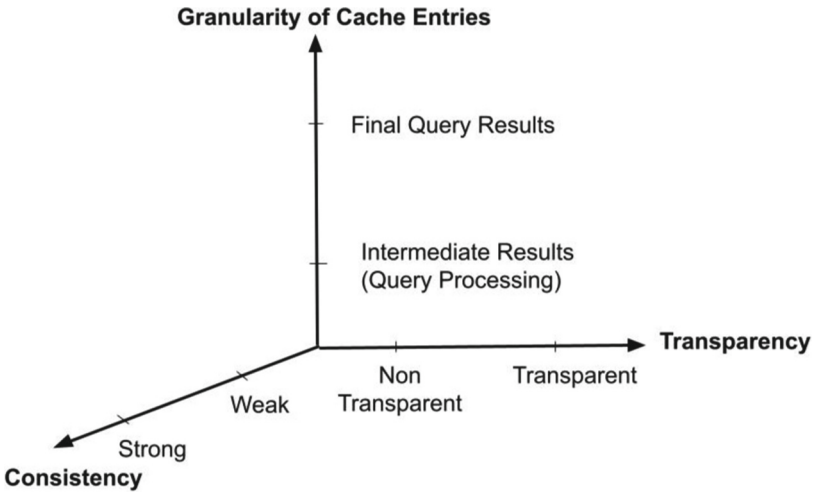


Fig. 2. A spectrum of caching techniques.

It is possible to have cache entries at the granularity of intermediate results required by a read. This requires the read to perform simple processing using one or more cache entries. An example is computing the intersection of two sets where each set is represented as a cache entry.

A caching solution may implement either a weak or a strong consistency technique. A form of weak consistency is eventual consistency [16]. It may be implemented by assigning a time-to-live (TTL) to each cache entry [7]. The cache deletes an entry once its TTL expires. A request that observes a miss for this entry queries the database to populate the cache with the latest value. During TTL, the cache may produce stale data. It becomes eventually consistent with the data store once the TTL of the stale cache entry expires.

With strong consistency, writes to the data store (inserts, updates, and deletes) must maintain the cache entries consistent with the database [4–6,9,10,15]. One may implement this using different write policies: write-around,

[1] A read may consist of one or more queries issued by an application as one transaction.

write-through, or write-behind. Figure 3 shows these write policies using a simple architecture that consists of an application node (AppNode) that implements the application logic using a transactional data store (Data Store) and a cache manager instance (CMI). Write-around and write-through apply AppNode's writes to the data store synchronously. Write-back[2] buffers the write and applies it to the data store asynchronously [5]. We describe these policies in turn.

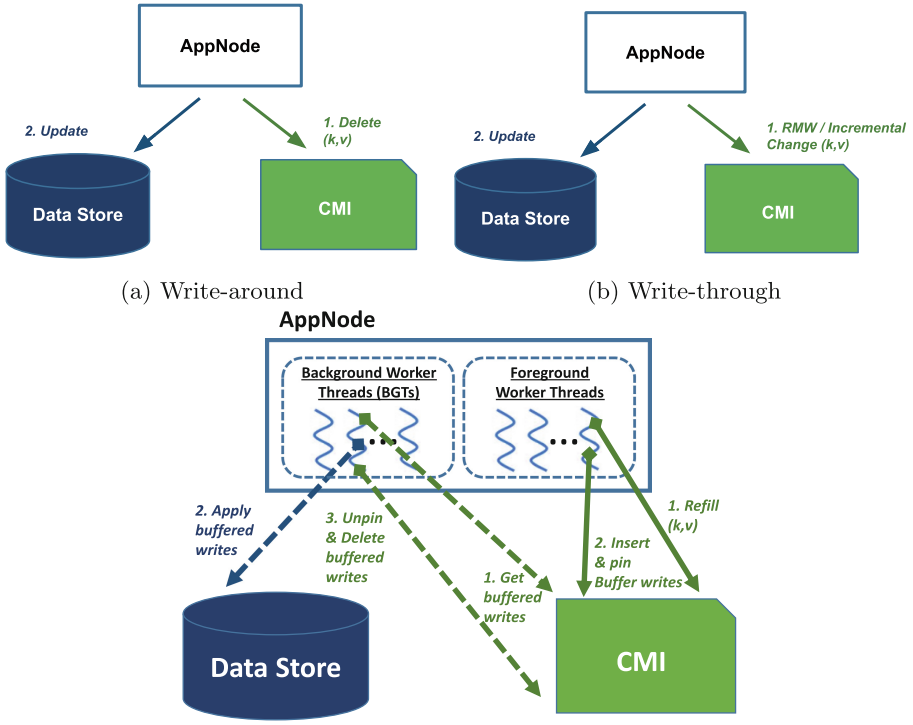


Fig. 3. Alternative write policies.

With write-around, Fig. 3a, the application identifies cache entries impacted by a write, deletes these entries from the cache, and then applies the write to the data store. A subsequent cache lookup for a deleted cache entry observes a cache miss, computes the latest results using the data store, and populates the cache with the most up-to-date results.

With write-through, Fig. 3b, a write updates its impacted cache entries using incremental update (e.g., append or increment) or read-modify-write. This pre-

[2] Also known as write-behind.

vents a cache miss for a subsequent read of this entry, minimizing the number of cache misses observed by an application. Similarly, write-back, Fig. 3c, updates the impacted key-value pairs by a write. However, it stores one or more replicas of its changes (termed *buffered write*) in the CMI instead of applying it to the data store. The write is then acknowledged to the user as successful. Background threads, BGTs, apply the buffered write to the data store asynchronously [5].

Finally, transparency dictates how much effort in the form of application specific software is required from a developer to implement different granularity of cache entries with varying degrees of consistency [17]. At one extreme, the cache may be non-transparent, requiring an application developer to identify the cache entries, categorize them, and provide application software for each category [1,5,7,9,13,14]. At the other extreme, the cache may be transparent with parameter settings that enable the database designer to fine-tune its configuration for a deployment [15]. And, one may have hybrids that provide transparency for some cache entries and require the developer to support others [8]. In general, transparent caches are tightly coupled with a data store and its data model. This explains why we cannot use one of the existing solutions such as [8,15,17] because they assume the relational data model[3] and SQL.

This paper describes a non-tranparent cache with entries at the granularity of final query results. It implements write-around with strong consistency.

2.1 Write-Around and Strong Consistency

Write-around may incur race conditions that produce stale cache entries [6,9]. This section describes a race condition and how we prevent it by using the transactional data store as the cache.

Figure 4 shows a system consisting of two separate components: A cache manager and a transactional data store. A race condition between two concurrent processes, a write (S1) and a read (S2), results in a stale cache entry [9]. It assumes the transactional data store uses multi-version concurrency control or snap-shot isolation to process reads of a data item that overlap with its writes. S1 deletes the impacted cache entries correctly, causing S2 to observe a cache miss for these entries in Step[4] S2.1. However, S1's write overlaps S2's read and the transactional data store processes the read using version V1 while S1 produces version V2. S2 proceeds to populate the cache with its obsolete value in S2.4. This cache entry is inconsistent with the database state. It violates the durability property of S1 because a subsequent read of this cache entry will observe a obsolete version even though S1 commits successfully.

[3] We use a graph data model.

[4] A miss in Step 2.1 causes the process S2 to initiate a transaction to read the data store, perform its processing to compute final results, and populate the cache with the results.

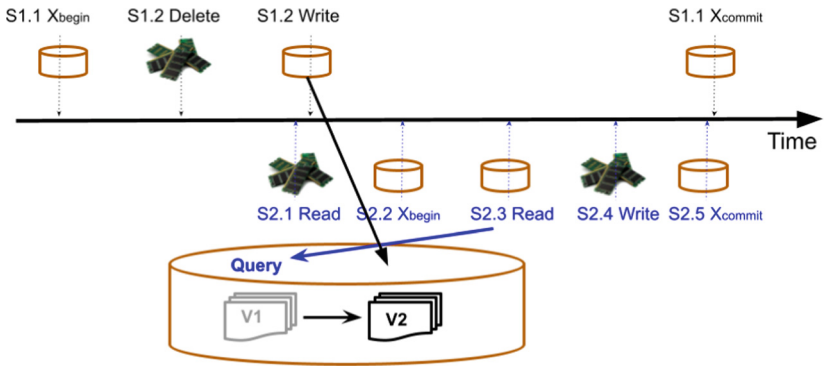


Fig. 4. Stale cache entries due to a race condition.

When the cache manager is a component separate from the transactional data store, one may extend the cache manager with the IQ framework [9] to prevent the race condition from inserting stale entries in the cache. An alternative approach is to use the transactional data store as the cache and require Process 2 to perform Step 2.1 as a part of the data store transaction. By executing Steps 2.1 and 2.4 as one transaction, S2 becomes a read-write transaction. Its write conflicts with the write of S1, causing either S1 or S2 to abort. This prevents stale cache entry. When S2 catches a transaction abort exception, it may check the read of Step S2.3 to see if it returned a value. S2 may return the valid value as its output without writing it to the cache. Hence, S2 may continue to benefit from multi-version concurrency control and snap-shot isolation techniques without producing a stale cache entry. We implement this approach in Sect. 3.

3 eBay's Graph Store and Query Result Cache

Figure 5 shows the three-tier architecture of our graph service. It consists of K Client instances, N Application (App) Service instances, and a transactional key-value store, FoundationDB [18] (FDB), as its storage backend. Both the Client and App Service instances are stateless. The App Service implements the core functionality of our service using JanusGraph [12], an open-source graph processing engine. It serves user requests issued by the Client instances. A load balancer strives to distribute these requests evenly across the N App Service instances.

FDB is a scalable distributed key-value store that provides strong consistency (the highest consistency guarantee is strict serializability [2]). It is an important choice since a graph traversal is executed as one FDB transaction that provides atomicity. This eliminates inconsistencies such as stale index entries, half-edges, ghost vertices [11] and others raised by the use of storage manager that provides a weaker consistency technique such as eventual [16].

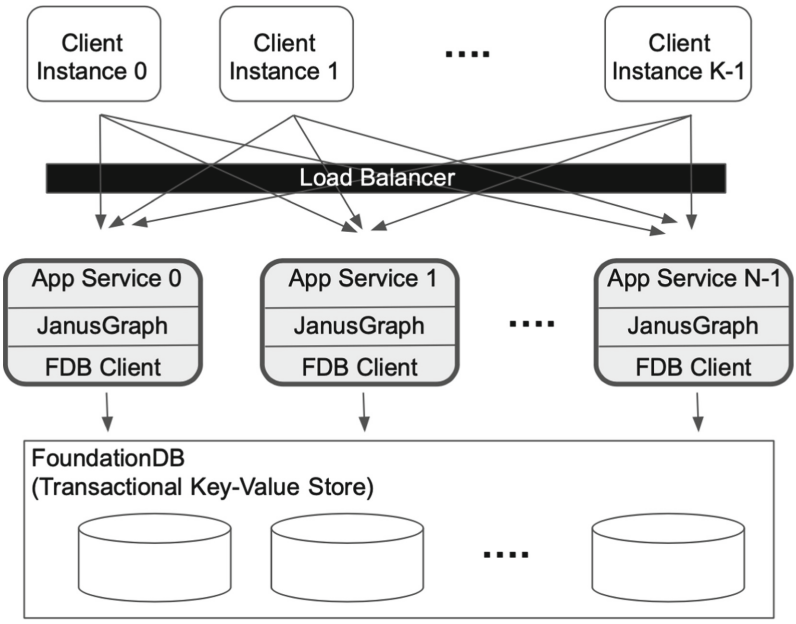


Fig. 5. eBay's graph store three-tier architecture.

Our target application is modeled as a bipartite graph consisting of two sets of vertices: accounts and attribute-values, following the labeled property graph modeling. An edge is bi-directional. It connects an account to an attribute-value and vice versa. The attribute values are typed, e.g., string, etc. Each type is represented as a vertex label with a total of 13 labels, e.g. "name", "address", "ip_address", etc.

As of this writing, the graph database consists of more than 16 billion vertices and approximately 26 billion edges. Figure 6 shows the frequency of the 13 vertex labels across unique account ids. Where there are multiple unique values for an attribute (Attr 13), others have duplicate[5] values.

Our schema enforces unique attribute values, using edges to capture a many-to-one relationship between an account and an attribute-value. The attribute values are indexed using JanusGraph's composite indices. These indices expedite processing of the exact match queries that look up the value of an attribute, enabling the system to fetch all account vertices with that attribute value in order to initiate edge traversals.

The distribution of the number of incoming edges into a vertex follows the power law with a long tail. More than 70% of the vertices have 1 incoming edge. Less than 117 nodes have more than 1 million incoming edges. The maximum number of edges for a node is approximately 4.3 million. Figure 7 shows the

[5] If each account had a unique value for each vertex label then, the sum shown on the y-axis should have exceeded 1300%. It is 4× lower.

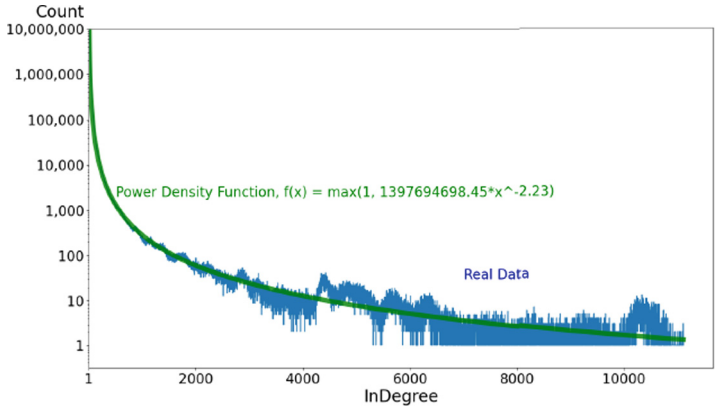


Fig. 6. Percentage of unique values for each of the 13 vertex labels. The length of the bar for each label shows its percentage of unique values. The y-axis is the cumulative sum of the percentages with the labels sorted in ascending order of their percentage of unique values.

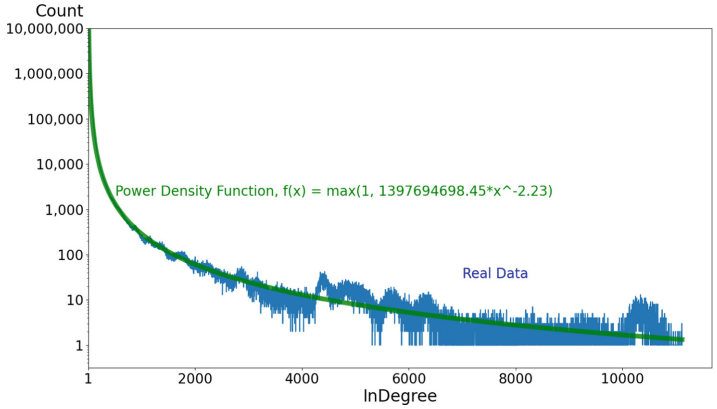


Fig. 7. Distribution of the number of incoming edges. This figure does not show vertices with only one incoming edge because there are several millions of them.

distribution of the incoming edges and a power distribution function that models the data.

While the distribution of the outgoing edges continues to follow the power law, it is less skewed than the incoming edges. Hence, a power density function is not as good a fit as the incoming edges. See Fig. 8. More than 16% of the

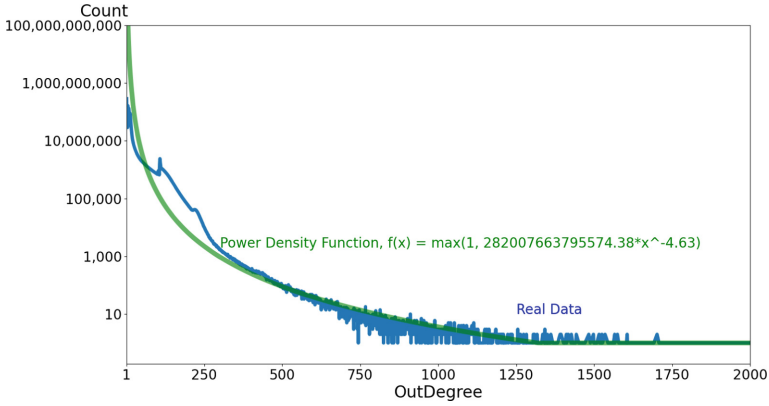


Fig. 8. Distribution of the number of outgoing edges. The x-axis ticks are cut early because it stretches to 13,319 with Count = 1. The shown portion highlights a poor fit between a power density function and the real data.

vertices have one outgoing edge. Less than 0.01% of the vertices have more than 250 outgoing edges. The maximum number of outgoing edges for a vertex is 13,319.

Our workload consists of three types of requests: Reads from the clients, periodic writes by batch loader instances, and continuous writes by streaming loader instances. A write operation is a transaction that may read some data, perform some processing, and update/delete/insert vertices and edges. The read operations access data in a skewed manner. 80% of read operations access 50% of the total data.

3.1 Caching Query Results

We extend the App Service to use the transactional data store as a cache, see Fig. 5. This extension includes a hash table of query templates whose instances are recognized by the caching framework. The results of these query instances may be found in the cache. When a query is submitted, the App Service extracts its template by removing its parameter values. It probes the hash table with the query template. If the template is found then it constructs the key for the query instance. This key is a concatenation of a 1 byte unique identifier for the identified query template, and the values of the parameters of the query instance. Next, the App Service looks up the cache with this key. If the cache returns a value (a cache *hit*) then it is used as the result of the query. Otherwise (a cache *miss*), the App Service processes the query instance using JanusGraph to obtain a result set, i.e., a value, and inserts this (key, value) pair in the cache.

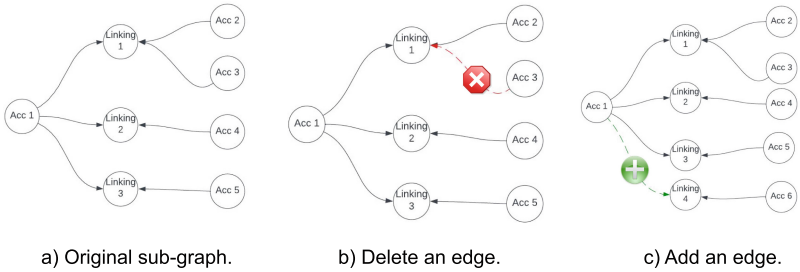


Fig. 9. Computing impacted keys in the presence of edge insertions and deletions.

A query instance traverses a fixed number of edges starting with a vertex. In the presence of edge insertions and deletions, we compute the impacted query instances whose results may have changed. To illustrate, Fig. 9a shows an original sub-graph. A transaction may delete an edge from "Acc 3" to "Linking 1", see Fig. 9b. Alternatively, a transaction may insert an edge from "Acc 1" to "Linking 4", see Fig. 9c. In both cases, we identify the vertex with the incoming new edge, i.e., a linking vertex, and use the traversal pattern of a query template to identify its impacted instances. We construct their keys and delete them.

4 An Evaluation

We evaluate the impact of caching using the workload of our production system. Our evaluation is done on a test deployment that mirrors the hardware of our production system, see Fig. 10. Initially, the test database is nearly identical to our application's production database. The software on the test system is experimental and may cause the test database to diverge from the production system. We fork the read, write, and read-write transactions of the production system to the test system as follows. When a client request is issued to a production service instance, the service instance places the request in a queue and proceeds to process the request. A thread removes a queued request and forwards it to one of the service instances of the test system for processing.

Request queues and background threads minimize the overhead on the production system. However, they introduce undesirable race conditions that result in a higher number of aborted transactions with the test deployment when compared with the production system. Below, we describe an undesirable race condition that causes the majority of aborts and our solution for it.

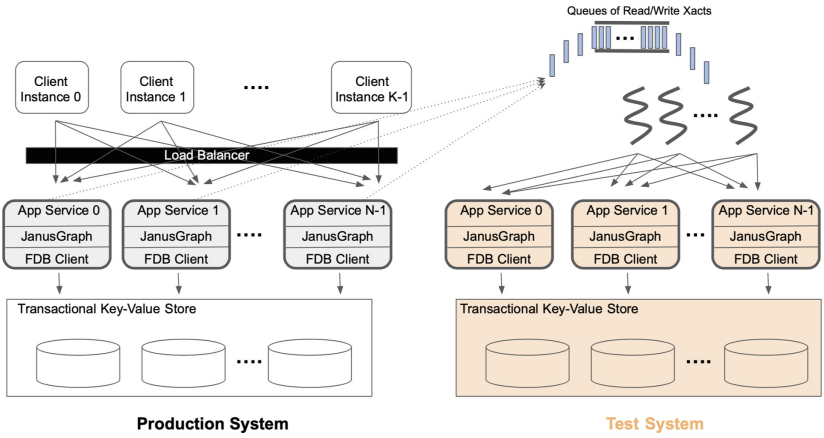


Fig. 10. Evaluation using mirrored traffic from a production system.

A background thread may send a sequence of transactions issued by one client to different service instances of the test deployment for concurrent processing. However, these transactions were processed sequentially by the production system because a client does not issue a transaction until its pending transaction completes execution. Concurrent processing of a client's transactions is a consequence of using the queue. This concurrent processing may result in conflicts. To elaborate, FoundationDB uses optimistic concurrency control, where conflicts of concurrent transactions are resolved at commit time. Read-only transactions commit always. Writes or read-write transactions may conflict with one another. FDB commits one and aborts the other conflicting transactions. While multiple conflicting writes or read-writes processed sequentially do not conflict, their concurrent processing may result in conflicts and a high number of transaction aborts.

We minimized the number of undesirable race conditions by requiring a production service instance to transmit the transactions issued by a client to one service instance in the test deployment. We do this mapping using a hash function applied to the id of the client that issues a transaction, directing all its transactions to one deployment service instance.

4.1 Results

Figure 11 shows the 50^{th}%, 95^{th}%, and 99^{th}% of the response time of one of the cached query templates using the test system with the cache enabled. We disable the cache at timestamp 52, causing the response time to increase. These results highlight the benefits of using the cache to reduce response time. In this figure, the cache reduces the 50^{th}%, 95^{th}% and 99^{th}% response time approximately 50×, 1.5× and 2×, respectively.

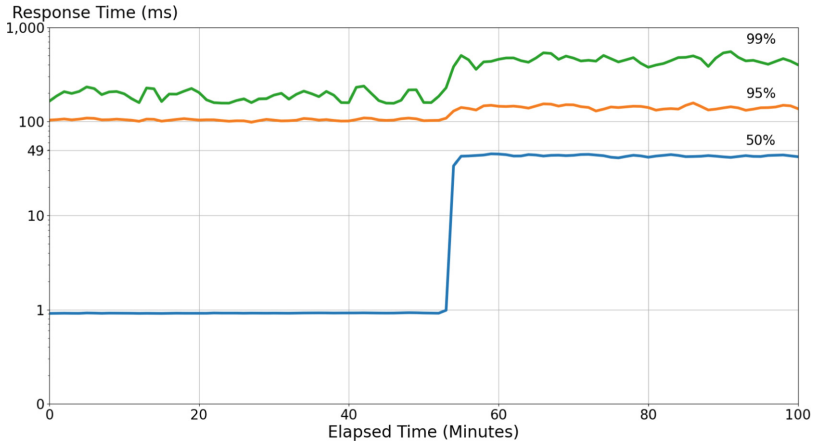


Fig. 11. Query response time of a cached query template. Cache is enabled at time 0 and disabled at time 52 min, causing a significant increase in the query response time.

In the presence of updates to the graph store, the system identifies the impacted cached keys and delete them. Figure 12 shows the number of impacted keys before and after the batch loader instances start their writes. It shows the $99^{th}\%$ of the number of keys impacted by a write is more than 10K. A write deletes these impacted keys, incurring a response time higher than its execution without a cache. Figure 13 shows the response time of a write operation before and after the batch loader instances start. It shows a significant increase in processing a batch write. Their degraded response time does not impact our users' experiences because users are not waiting for these writes to complete.

5 Conclusions and Future Research Directions

Application-level query result caches significantly speedup queries that observe a cache hit. In addition, they reduce the amount of processing required by the graph store and its transactional storage manager, enhancing the overall system latency. This is important for read-only real-time workloads, which are a critical component of many applications.

We are evaluating other variants of caching techniques in the spectrum of Fig. 2, such as a transparent cache using intermediate query results for query processing while providing strong consistency. This approach does not require application specific software. Hence, it may be used by many applications. Instead

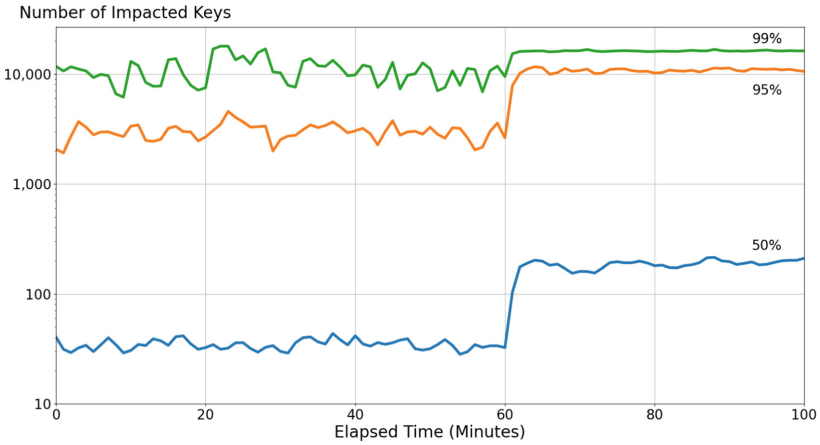


Fig. 12. Number of impacted keys.

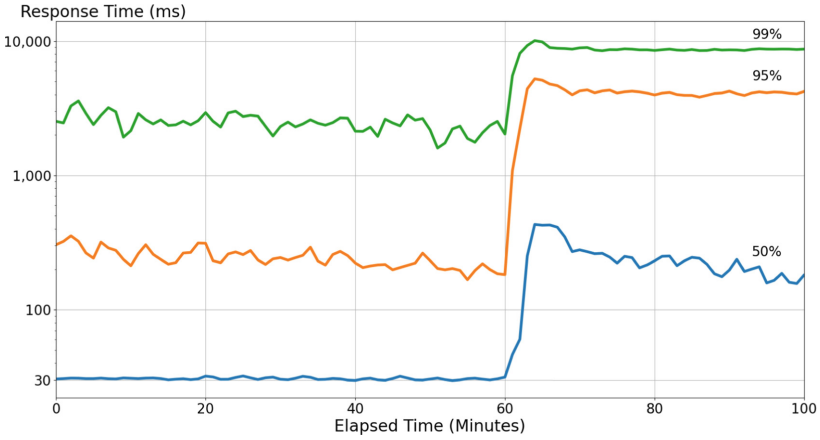


Fig. 13. Write operations are slowed down because they incur the overhead of finding and deleting the impacted cached keys.

of invalidation with a write-around cache, we may use other cache coherence techniques such as write-through and write-back [5,13]. We will also explore alternative architectures [3,4] such as having the in-memory cache layer as a part of the transactional data store.

Acknowledgments. We thank eBay's Hongjiang Zhang for gathering some of the presented statistics, USC's Ghazal Rafiei for plotting graphs, USC's Romtin Toranji for computing the probability distribution functions, and Meikel Poess for his constructive comments on an earlier draft of this paper.

References

1. Alabdulkarim, Y., Almaymoni, M., Cao, Z., Ghandeharizadeh, S., Nguyen, H., Song, L.: A comparison of flashcache with IQ-twemcached. In: 2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW), pp. 20–26 (2016)
2. Alabdulkarim, Y., Almaymoni, M., Ghandeharizadeh, S.: Polygraph: a plug-n-play framework to quantify application anomalies. IEEE Trans. Knowl. Data Eng. **33**(3), 1140–1155 (2021)
3. Ghandeharizadeh, S., Bernstein, P.A., Borthakur, D., Huang, H., Menon, J., Puri, S.: Disaggregated database management systems. In: Performance Evaluation and Benchmarking: 14th TPC Technology Conference, TPCTC 2022, Sydney, NSW, Australia, 5 September 2022, Revised Selected Papers, pp. 33-48. Springer-Verlag, Heidelberg (2023). https://doi.org/10.1007/978-3-031-29576-8_3
4. Ghandeharizadeh, S., Huang, H., Nguyen, H.: Nova: diffused database processing using clouds of components [vision paper]. In: Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D. (eds.) BDAS 2019. CCIS, vol. 1018, pp. 3–14. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19093-4_1
5. Ghandeharizadeh, S., Nguyen, H.: Design, implementation, and evaluation of write-back policy with cache augmented data stores. Proc. VLDB Endow. **12**(8), 836–849 (2019)
6. Ghandeharizadeh, S., Yap, J.: Gumball: a race condition prevention technique for cache augmented SQL database management systems. In: Proceedings of the 2nd ACM SIGMOD Workshop on Databases and Social Networks, DBSocial 2012, pp. 1–6. Association for Computing Machinery, New York (2012)
7. Ghandeharizadeh, S., Yap, J.: Cache augmented database management systems. In: Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks, DBSocial 2013, pp. 31–36. Association for Computing Machinery, New York (2013)
8. Ghandeharizadeh, S., Yap, J.: SQL query to trigger translation: a novel transparent consistency technique for cache augmented SQL systems. In: 2017 28th International Workshop on Database and Expert Systems Applications (DEXA), pp. 37–41 (2017)
9. Ghandeharizadeh, S., Yap, J., Nguyen, H.: Strong consistency in cache augmented SQL systems. In: Proceedings of the 15th International Middleware Conference, Middleware 2014, pp. 181–192. Association for Computing Machinery, New York (2014)
10. Huang, H., Ghandeharizadeh, S.: Nova-LSM: a distributed, component-based LSM-tree Key-value store. In: ACM SIGMOD (2021)
11. JanusGraph. Eventually-Consistent Storage Backends (2023). https://docs.janusgraph.org/advanced-topics/eventual-consistency/
12. JanusGraph (2023). https://janusgraph.org/
13. Nguyen, H.: Asynchronous writes in cache augmented data stores. Ph.D. dissertation, USC (2018)
14. Nishtala, R., et al.: Scaling memcache at facebook. In: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2013), Lombard, IL, April 2013, pp. 385–398. USENIX Association (2013)
15. Ports, D.R.K., Clements, A.T., Zhang, I., Madden, S., Liskov, B.: Transactional consistency and automatic management in an application data cache. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI 2010, pp. 279–292. USENIX Association (2010)

16. Vogels, W.: Eventually consistent. Commun. ACM **52**(1), 40–44 (2009)
17. Yap, J.: Transparent consistency in cache augmented database management systems. Ph.D. dissertation, USC (2014)
18. Zhou, J., et al.: FoundationDB: a distributed key value store. SIGMOD Rec. **51**(1), 24–31 (2022)

Chaosity: Understanding Contemporary NUMA-Architectures

Hamish Nicholson[1,2(✉)], Andreea Nica[1], Aunn Raza[1], Viktor Sanca[1], and Anastasia Ailamaki[1,2]

[1] EPFL, Data-Intensive Applications and Systems Lab, Lausanne, Switzerland
{hamish.nicholson,andreea.nica,aunn.raza,
viktor.sanca,anastasia.ailamaki}@epfl.ch
[2] Google, Sunnyvale, USA

Abstract. Modern hardware is increasingly complex, requiring increasing effort to understand in order to carefully engineer systems for optimal performance and effective utilization. Moreover, established design principles and assumptions are not portable to modern hardware because: 1) Non-Uniform Memory Access (NUMA) architectures are becoming increasingly complex and diverse across CPU vendors; Chiplet-based architecture provides hierarchical NUMA instead of flat-NUMA topology, while heterogeneous compute cores (e.g., Apple Silicon) and on-chip accelerators (e.g., Intel sapphire rapids) are also normalized in materializing the vision for workload- and requirement-specific compute scheduling. 2) Increasing IO bandwidth (e.g., arrays of NVMe drives approaching memory bandwidth) is a double-edged sword; having high-bandwidth IO can interfere with the concurrent memory access bandwidth as the IO target is also memory; hence IO itself consumes memory bandwidth. 3) Interference modeling is becoming more complex in modern hierarchical NUMA and on-chip heterogeneous architectures due to topology obliviousness. Therefore, systems designs need to be hardware topology-aware, which requires understanding the bottlenecks and data flow characteristics, and then adapting scheduling over the given hardware topology.

Modern hardware promises performance by providing powerful and complex yet non-intuitive computing models which require tuning specifically for target hardware or risk under-utilizing the hardware. Therefore, system designers need to understand, carefully engineer, and adapt to the target hardware to avoid unnecessarily hitting bottlenecks in the hardware topology. In this paper, we propose the Chaosity framework, which enables system designers to systematically analyze, benchmark, and understand complex system topologies, their bandwidth characteristics, and interference of effects of data access paths, including memory and PCIe-based IO. Chaosity aims to provide critical insights into system designs and workload schedulers for modern NUMA hierarchies.

Keywords: NUMA · Data Access · IO · NVMe · Throughput · Interference

H. Nicholson and A. Ailamaki—Work done entirely at EPFL.

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 59–76, 2024.
https://doi.org/10.1007/978-3-031-68031-1_5

1 Introduction

Post Moore's law era, hardware designs have tended to scale horizontally, scaling compute via partitioning and packing more computational units in a single chip. In addition to multi-core processors, NUMA (non-uniform memory access) sockets add another layer of compute partitioning, creating an almost distributed yet coherent and shared-everything system in a single scale-up server. Initially, NUMA hardware had non-uniformity, but all CPUs were treated as homogeneous processors, while accelerators were treated as co-processors for each NUMA node. Essentially, NUMA nodes were arranged as siblings in the hardware stack, having their own memory hierarchy, including PCIe-attached storage and a set of co-processors optionally.

Advancements in the hardware landscape challenge traditional system designs to avail the performance and efficiency offering from the modern hardware and maintain performance standards [11]. The fundamental changes are:

1. Hierarchical NUMA in chiplet-based architectures, even in single-socket machines in mainstream CPUs offered by AMD [12] and Intel [13].
2. IO-bandwidth competing with memory bandwidth. Moreover, co-processors and other sibling PCIe-attached devices may compete in bandwidth utilization by directly consuming from the devices (e.g., GPU reading from NVMe without involving CPUs), compared to previously strict uni-directional storage hierarchy [15].
3. Heterogeneous compute-on-chip is becoming the norm rather than a niche. For example, Apple Silicon [8] and Intel [4] consumer-grade chips have different types of compute units within the same chip, specialized for a range of workloads, and Intel Sapphire Rapids offer on-chip, off-core accelerators. This introduces additional heterogeneity in compute scheduling and data routing [13].

Consequentially, tuning systems with traditional but usurped design principles in mind, including but not limited to NUMA-aware partitioning and caching across memory hierarchy, may result not only in a lack of speedup but also in performance regressions. Modern hardware promises increased performance and scalability when tuned to the expected software design. This is because, previously, hardware across vendors and generations was mostly homogeneous in design principles. For example, a multi-socket, multi-core machine was assumed to have three layers of caches for each processor, the first two levels private to each core and a shared last-level cache, and a high-speed coherent link between all NUMA nodes. In general, each newer processor generation added new features but was mostly transparent to the user regarding software design, therefore, was compatible with existing NUMA-aware systems. However, with modern hardware, the hardware topology is not homogeneous across vendors: AMD EPYC has a heterogeneous chiplet-based architecture [11], while Intel Sapphire Rapids [13] offers on-chip accelerators and even high-bandwidth memory in certain models. Further, depending on the specific hardware, the NUMA topology may be hierarchical in the case of chiplets, creating a tree of NUMA nodes.

This results in the fundamental invalidation of software design principles and assumptions. For example, the used-to-be flat NUMA-aware partitioning would suffer from interference in remote accesses in a hierarchical NUMA topology. Such trends require hardware-software co-design, but it requires either software designed especially for specific hardware only, which is hardly the case, or adaptive across hardware. In any case, the first step in designing a system that could efficiently utilize the powerful features of modern hardware is *understanding the hardware*. Make no mistake: understanding hardware does not mean studying it at the silicon level, but understanding the hardware topology and capabilities, the data-flow paths, and associated characteristics. For example, understanding the bandwidth difference between memory and disk is critical in designing caching policies [14,15]. In another case, an algorithm design would differ based on the availability of coherent versus non-coherent interconnect between a CPU and an accelerator (e.g., GPUs).

Therefore, we propose Chaosity, a framework for systematically understanding hardware topology, bandwidth characteristics for memory and PCIe-based IO, and modeling interference between non-partitioned memory operations. This is a first step towards automatic benchmarking and bootstrapping critical and actionable insights required for a systems engineer to understand and for an adaptive system to tune itself for specific hardware. The rest of the paper is organized as follows:

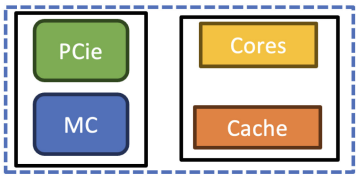
- Section 2 highlights the increasing heterogeneity in the hardware landscape, which motivates the importance and impact of systematic analysis,
- Section 3 focuses on the initial design of an automated non-uniformity benchmark,
- Section 4 and Sect. 5 presents experimental results that demonstrate the need for a chaos-aware heterogeneous platform on two such architecture configurations,
- Section 6 discusses the takeaways, implications and provides future directions for this work
- Section 7 concludes the vision of Chaosity.

2 Motivation: Rising Entropy in the Hardware Landscape

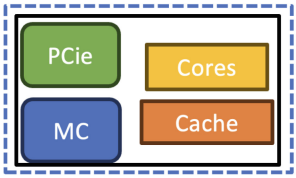
The advancements in computer architecture change the system landscape and the opportunities for hardware-software co-design. Figure 1 shows the evolution of recent mainstream CPU topologies. Besides the well-known transition from single-core to multi-core architectures, the chip shrinkage has allowed integrating components from a single Northbridge/memory controller hub (Fig. 1a) into a single chip die (Fig. 1b), alleviating the bottleneck of data transmission and introducing NUMA with on-socket memory controllers.

2.1 Hierarchical NUMA

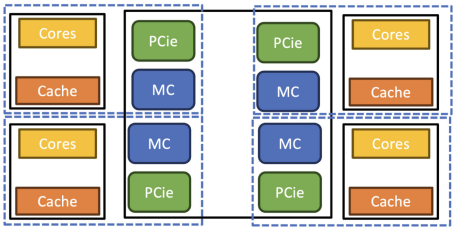
The continuation of chip downsizing has led to the post-Moore law era, leading to challenges in CPU scalability where vendors are increasingly adopting chiplet



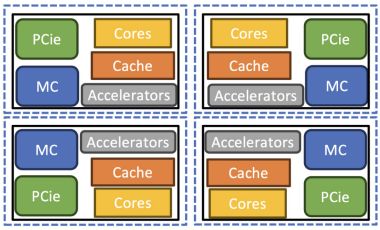
(a) Memory controller and IO on north bridge



(b) Integrated memory controller style



(c) AMD EPYC Milan style



(d) Intel Sapphire Rapids style

Fig. 1. Evolution of recent CPU topologies. Solid black boxes represent chips/chiplets, and dashed blue boxes represent NUMA regions. (Color figure online)

(multi-chip module) designs (Fig. 1c) [12,13]. This changes the traditional monolithic CPU design when non-uniform memory access (NUMA) resulted from multi-socket CPU servers. Chiplets introduce additional NUMA regions even inside a single socket in a hierarchical fashion, increasing the complexity of main memory access paths. Though chiplet-based CPUs are not the first CPUs to expose multiple NUMA regions within a single socket, they are more widely adopted than previous commercial offerings, such as Intel's Xeon Phi Knights Landing. [19,23]). AMD has been using chiplet designs since the EPYC Naples generation [12], while Intel recently moved to chiplets for many of their server CPUs in the Intel Sapphire Rapids [4] generation. The chiplet designs from Intel and AMD allow exposing the hierarchical NUMA regions to the operating system, e.g., 8 NUMA nodes in a 2-socket server.

2.2 On-Chip Heterogeneity

The complexities do not end at hierarchical NUMA. To add to memory and data access path complexity, even the compute units may not be uniform. Specialized accelerators and non-uniformity introduce differences in throughput and memory access (Fig. 1d).

The benefit of having specialized or heterogeneous cores on a single chip is clear to all hardware vendors, and thereby, in the last few years, we have seen consumer-grade CPUs like Apple M1/M2 silicon and 12th generation Intel Core desktop processors packaging performance and efficiency cores in a multi-core chip, scaling the non-uniformity axis to heterogeneous compute units. Further,

besides the disparity between different cores, both consumer- and server-grade hardware introduced specialized on-chip off-core accelerator components optimized for specialized or specific workloads, such as neural processing. To list a few, Apple Silicon has an accompanying neural engine for machine learning (ML) workloads. Intel Sapphire Rapids CPU comes with specialized tile registers and matrix multiplication intrinsic (AMX) for ML, data encryption and compression accelerators (QAT), and data streaming accelerator (DSA). Recently, AMD also announced MI300 APUs (accelerated processing units) composed of modular chiplets. The AMD MI300 APU will offer a combination of CPU (Zen4) or GPU (CDNA3) chiplets and on-chip high-bandwidth memory [22]. Further, Nvidia Grace Hopper Superchips tightly integrate an ARM-based CPU and an NVIDIA GPU chip with a fast inter-chip NVLINK interconnect [17].

2.3 Data Highways: Interconnect

Interconnects play a significant role in data access and movement across complex topologies as careful use of the limited available bandwidth is crucial for efficiency [6,15,21]. PCIe interconnects in complex topologies (Fig. 2) represent a shared resource between CPUs, accelerators, and IO devices.

An added complexity ensues with the addition of per-hierarchical-NUMA PCIe controllers and, for example, fast NVMe drives, which are in aggregate on par with the available main-memory bandwidth, interfering and contending with the even more complex memory access path. IO predominantly uses direct memory access (DMA) to transfer data between devices and main memory. To perform a DMA transfer to read data from an IO device, the CPU submits a request to the device and then the device's DMA engine is responsible for transferring the data directly to main memory.[1] The CPU can then read the data from memory using load instructions. The processing for writing data to an IO device inverts the order of operations. Still, there is no uniform design approach for interconnects, as Apple Silicon has a unified memory architecture for their CPU, GPU, and neural engines. Overall, the location of the IO device and the access path complexity requires careful coordination and placement.

2.4 Systems with Complex Data Access

Traditionally, the hardware topologies were mostly homogeneous and standard across the vendors, and therefore, the underlying hardware performance was more predictable and understandable by the on-paper specifications. The system designers would have to consider only a few metrics, including but not limited to memory bandwidth, CPU interconnect bandwidth, and PCIe or device bandwidth. Most scalable applications were designed with NUMA-aware partitioning and cache-aware algorithms, catering to both CPU caches when reading/writing

[1] Intel Xeon CPUs feature Data Direct IO (DDIO), which transparently allows PCIe devices to read/write directly to last-level caches initially bypassing DRAM. [1].

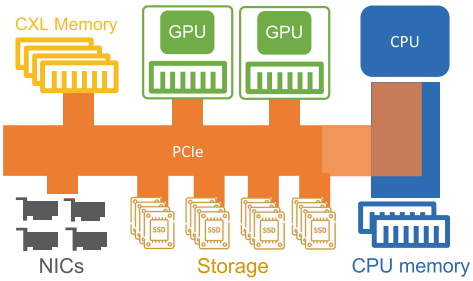


Fig. 2. Interconnects contribute to complex heterogeneous topologies

from memory to CPU and buffer pools when reading/writing from disk to memory. However, with ever-increasing heterogeneous and complex hardware, mere NUMA-partitioning and cache-aware algorithms do not fully utilize the underlying hardware but may result in performance regressions in some cases. For instance, Nicholson et al. [14,15] concluded that with high-bandwidth storage, simple frequency- or recency-based caching is sub-optimal and requires proportional caching to utilize the high-bandwidth storage fully. Moreover, not only do the system designs have to cater to fully utilizing the underlying hardware capabilities, but they also need to account for the interference domain. For instance, Raza et al. [20,21] partitioned latency-sensitive and bandwidth-intensive workloads across NUMA-boundaries to alleviate interference in the memory hierarchy but was limited by the interconnect; however, with chiplet architectures, the system could have partitioned the interfering workloads across chiplet boundaries.

Putting everything together, increasingly high hardware complexity opens up tuning opportunities to profit from and avoid performance regressions. Such opportunities may be trivial and intuitive: partitioning workload across NUMA boundaries, minimizing data movements, etc., or, non-intuitive based on the underlying hardware characteristics: de-prioritizing data locality in favor of partitioning workload based on interference, or staging/buffering IO in system memory for granular IO from PCIe-attached devices like GPUs [15]. Still, a given workload will have a combination of memory, computational, and data movement requirements that might have a desirable particular hardware constellation. This motivates a systematic study of the complexity and the chaos ensuing, not only for a given platform and workload but for any future change in platform or workload.

3 Chaosity Framework Understand Thy Hardware

Characterizing non-uniformity and interference is highly challenging. While benchmarks traditionally target a specific and often limited set of parameters, previously described complex hardware and data movement interactions instead exacerbate the need for a holistic benchmark, such as memory-IO interference or CPU-GPU interference with shared memory. Still, the design space of interference micro-benchmarks is vast, motivating for a framework that composes

micro-benchmarks that best represent a workload to analyze the complex effects of the underlying system and hardware.

To this extent, as good examples of characterizing system-crucial characteristics, we employ fio [3] and STREAM [10]. fio benchmarks persistent disk performance (IO), and STREAM measures sustained memory bandwidth. This way, we aim to analyze and understand the interference between the two subsystems. Our design allows easy incorporation of existing workloads or ones written from scratch. For example, rather than measuring memory-IO interference only, CPU cache interference would allow for finer-granularity benchmark experiments.

Benchmark Categorization and Selection. The first step in profiling any hardware is the categorization of benchmarks, that is, the target metric or characteristics to be measured. The benchmark category defines the benchmarks that will be executed across independent and shared configurations. Each benchmark category measures the specific target property or metric of the hardware under test. Currently, benchmarking categories include memory and PCIe-based storage bandwidth.

The second step is defining which benchmark to use, either a standard off-the-shelf benchmark or a customized benchmark. The invariant in one or more benchmark selections is the target metric. Chaosity provides a default standard benchmark for each category. However, one can add or replace the benchmark with another standard or custom benchmark. For instance, by default, Chaosity utilizes STREAM [10] for profiling memory bandwidth and fio [3] for profiling storage bandwidth.

Component and Topology Discovery. Chaosity begins profiling hardware under test by first discovering the available components/devices and their topology and memory model. For example, detecting the number of available cores, the number of hyper-threads per physical core, NUMA nodes, sockets, and connected devices, including but not limited to NVMe storage and GPU devices. Chaosity leverages *hwloc* and Linux *numactl* utilities to discover the hardware topology [5]; Further, Chaosity queries the underlying hardware properties, such as detecting if the devices have a unified memory and if the shared memory is coherent across devices, CPU cache-line size, etc. Chaosity also detects the availability of pre-defined specialized accelerators, such as on-chip accelerators in Intel Sapphire Rapids.

Discovering hardware components, their topology, and associated properties is critical in profiling and benchmarking as it defines the interaction between different components and the expected behaviors.

Executing Independent Benchmarks. After benchmark categorization, definition, and hardware discovery, Chaosity begins the profiling defined by each benchmark category. The benchmark utilizes the topology information and starts by profiling the minimum unit of each type of compute and, from thereon, profiles the combination of components in the hierarchy. Chaosity needs to profile

hierarchy for all combinations to detect any non-uniformity, asymmetry, or sometimes, even hardware defects or misconfigurations. In the following, we detail the memory and storage bandwidth benchmarking.

Benchmarking Memory Bandwidth. Analyzing memory bandwidth starts with the default unit of compute, that is, single-thread in CPU, and then proceeds by analyzing bandwidth of single-NUMA, single-socket, and then all CPUs. Then, the system profiles remote interactions, which in the case of memory, means accessing remote memory for each unit, starting from bandwidth for accessing the memory of different NUMA nodes within the same socket and then similarly for remote sockets.

Benchmarking Storage Bandwidth. Analyzing storage bandwidth proceeds similarly to analyzing memory bandwidths. However, it adds an additional basic unit, the number of drives attached locally to each NUMA node, to analyze the bandwidth scalability across combinations of the connected drives within the same and other NUMA nodes.

Memory-Storage Interference Modeling. A shared memory subsystem introduces competition in data accesses and hence, causes interference. This interference occurs at all levels of the memory hierarchy, including competing cache lines, load requests, and memory bandwidth itself. For now, we target and model memory bandwidth interference, which arises when accessing both CPU memory and storage or remote memory over PCIe. The bandwidth interference arises as memory access goes through the same memory controller in most processor architectures. Hence, a memory controller can only process a certain amount of data simultaneously, prioritizing one over another.

Chaosity profiles and models the interference by scheduling independent memory and storage bandwidth benchmarks concurrently. It profiles the interference by collocating and isolating the compute unit and the read drive set using topology information. In doing so, it models the interference when both accesses are issued from the same controller or are routed through a different controller. Ideally, when co-scheduled, the total bandwidth (memory + storage) should be equal to the max of either; however, when scheduled across NUMA boundaries, should not interfere with each other as the PCIe root complex should be directly accessible from the requesting memory controller. However, it is hardly the ideal case due to the hidden complexities of hardware design, and therefore, it is crucial to profile and understand the bandwidth degradation in all cases, guiding system designers to account for and schedule workloads accordingly.

4 Heterogeneous Compute Units Apple M1 Pro Silicon

In this section, we use Chaosity to test Apple M1 Pro silicon, explicitly targeting the unified memory bandwidth across heterogeneous CPU cores and GPU and analyzing the maximum memory bandwidth when executed in isolation and the

degradation due to interference when all compute units compete for memory bandwidth.

Hardware. Apple Macbook Pro 2021 running macOS 12.5.1 with a M1 Pro processor (Model identifier: MacBookPro18.3, Model number: Z15G002BDSM/A), having 10 CPU cores (eight performance and two efficiency cores), 16 GPU cores, and 16 neural-engine cores, with a total of 32GB LPDDR5 memory and a 512 GB NVMe SSD.

Benchmark. On the CPU, memory bandwidth tests utilize the STREAM Triad benchmark [10]. We report the average bandwidth over 100 iterations for STREAM, not including the first iteration. We compile STREAM with Clang 13.0.1 with the `-O2 -fopenmp -DSTREAM_ARRAY_SIZE=80000000` compiler flags. In this configuration, each array element is an 8-byte double. While OpenMP is used to control the number of STREAM benchmark threads, it cannot bind threads to cores as on Linux. This is because macOS has no underlying API to pin threads. To run STREAM on efficiency cores, we use the `taskpolicy` system utility to launch STREAM with the `PRIO_DARWIN_BG` scheduling priority [2]. On the GPU, memory bandwidth tests use a variation of the `bw_benchmark` from [9]; this benchmark performs a multiple add on 3 input [8192 × 8192] matrices of 32 bit floats and storing the output in another matrix of the same type, using a total of 1 GiB of memory. For NVMe bandwidth tests, we use fio [3] to perform sequential reads. Our fio configuration for macOS uses the posixaio engine, a 1MB blocksize, `O_DIRECT`, and is time based to run for 30 s.

4.1 Interference in Unified Memory

Apple Silicon has a unified memory across all types of compute units, including performance and efficiency CPU cores, GPU cores, and neural-engine cores. Unified memory offers coherent access across heterogeneous consumers, that is, compute units or networks and other devices in some cases. Apple silicon is different from traditional CPU-GPU unified memory (like Nvidia's Unified Memory) in the sense that all types of compute cores are at the same level, and the last-level-cache is shared across all heterogeneous cores, which in our view, simplifies the cache coherency implementation in hardware. However, there is no free lunch. In the general case, not all compute devices will be running data-intensive operations. Still, for high-performance or analytical data processing tasks, all devices will execute data-intensive tasks and, thereby, require the maximum possible bandwidth to underlying unified memory.

Table 1 shows the experimental results when running memory benchmarks on Apple M1 silicon in different configurations. For standalone CPU baselines, a single performance core can consume a maximum memory bandwidth of 75 GB/s, while a single efficiency core can only achieve maximum memory bandwidth of 11.5 GB/s. Whereas, utilizing all eight performance cores only, we get 128 GB/s while using both efficiency cores only, we get 15 GB/s, and utilizing all CPU

Table 1. Memory Bandwidth Analysis of Apple M1 Pro

Scheduling mode	Compute units	Memory bandwidths (GB/S)
CPU-only	1 Efficiency core	11.7
	2 Efficiency cores	14.8
	1 Performance core	75.2
	8 Performance cores	128.7
	All cores (8P + 2E)	138.3
GPU-only	All 16 GPU cores	176.3
CPU-GPU	8 P-CPU (w/ 16 GPU)	59.8
	16 GPU (w/ 8 P-CPU)	118.9
	10 CPU (w/ 16 GPU)	60.8
	16 GPU (w/ 10 CPU)	115.0

cores, that is, ten cores (eight performance and two efficiency), we get a maximum memory bandwidth of 138 GB/s. In the case of GPU, when benchmarking in GPU-only mode, the benchmark achieves 176 GB/s, and to the best of our knowledge, there is no way of scheduling and affinitizing workload on the neural engine; hence, it is not included in the scope of this study.

From the baselines described above, which do not have any conflicting or interfering workload, it is clear that GPU-cores have access to 27% and 37% more memory bandwidth compared to all CPU cores and all of the performance CPU cores.

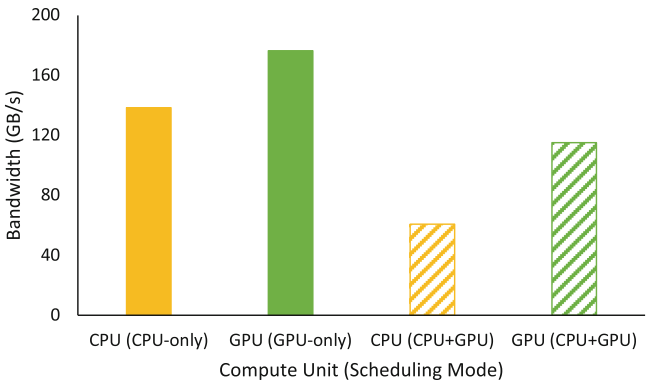


Fig. 3. Memory-storage bandwidth interference in Apple M1 Pro

Unified memory shares the memory access across all compute devices, hence sharing the bandwidth accordingly. We study bandwidth interference and priority by running parallel independent benchmarks on both CPU and GPU. As the

benchmarks are executed independently the bandwidth interference arises purely from resources competing for accessing memory, but not from accessing objects in memory shared by the benchmarks. Figure 3 plots the bandwidth degradation when both efficiency and performance CPU cores are used concurrently with the GPU. Table 1 in addition shows results for just the performance CPU cores.

We observe that GPU still gets priority over CPU in bandwidth allocation. The GPU memory bandwidth drops 35% and 32% when running using only CPU performance and all CPU cores, respectively. Whereas CPU memory bandwidth degrades by 54% when using only performance cores and 56% when using all cores. In both cases, the sum of CPU and GPU memory bandwidths are nearly equal to the bandwidth the GPU observes when running independently. This shows that to leverage the full memory bandwidth, the GPU must be used, but with the trade-off that less bandwidth will be available for the CPU cores.

5 Chiplet-Based Server AMD EPYC

In this section, we use Chaosity to test and analyze the AMD EPYC (Milan) server processors having a chiplet-based architecture. AMD EPYC is a representative of modern hardware which has hierarchical NUMA, that is, chiplets. Additionally, it provides enough PCIe 4.0 lanes to saturate more than half of the memory bandwidth for each chiplet.

Hardware. All experiments were conducted on a server with a 2×24-core AMD EPYC 7413 processor, having two threads per core, totaling 96 threads and 256 GB of DRAM. Each CPU socket has 16 Corsair MP600 Pro NVMe drives, each using 4 PCIe 4.0 lanes. The manufacturer-specified maximum read bandwidth of each drive is 7 GB/s [7]. At measurement time, two drives failed; therefore, NUMA nodes 0 and 4 have 3 NVMe drives each, while all other NUMA nodes have four drives each. AMD EPYC has 128 PCIe 4.0 lanes per socket, theoretically having a total bandwidth of 256 GiB/s, whereas the main CPU can independently achieve an aggregate bandwidth of 128 GiB/s per socket. In a two-socket configuration, 48 lanes of PCIe are used on each chip for the inter-socket interconnect. The remaining lanes are available for other PCIe devices.

Our server is running Ubuntu 20.04 with Linux Kernel 5.4.

Benchmark. Memory bandwidth tests utilize the STREAM Triad benchmark [10]. STREAM is compiled using GCC 9.4 with the `-O2 -fopenmp -DSTREAM_ARRAY_SIZE=100000000 -mcmodel=medium` compiler flags. In this configuration, each array element is an 8-byte double. Numactl is used to set the NUMA nodes STREAM will execute on as well as to bind the memory used by STREAM to specific, possibly different, NUMA nodes. For NVMe bandwidth tests, we use fio [3] 3.32. Our default fio benchmark is a sequential access benchmark that uses the io_uring engine, a 1MB blocksize, `O_DIRECT`, and is time-based to run for 30 s. **Memory Bandwidth.** Figure 4 shows experimental results of measuring memory bandwidth grouped by the CPU cores and

target memory nodes. Using numactl, STREAM is CPU bound to the NUMA node on the x-axis and memory bound to the NUMA node on the y-axis. Each number represents an independent run; hence, no interference or contention in accessing local or remote memory. The memory bandwidth within the socket is similar while accessing the memory of a remote socket is 33% slower, regardless of whichever chiplet in the socket itself.

5.1 Hierarchical NUMA

In the following analysis, we employ Chaosity to analyze memory and storage bandwidth in AMD EPYC Milan architecture and model the interference between the two.

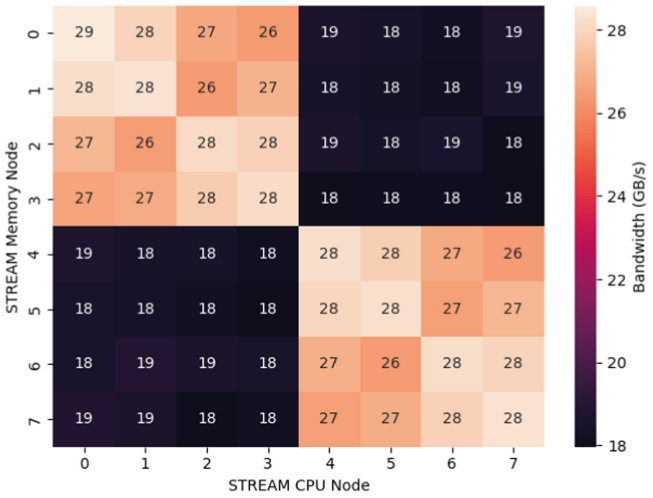


Fig. 4. Memory bandwidth in AMD EPYC – Each NUMA node accesses the memory of target NUMA node with STREAM TRIAD benchmark

Figure 5 shows the results of analyzing the maximum storage bandwidth when all cores of the NUMA node read from all drives of the target node. The interesting thing to observe here is that the average maximum bandwidth substantially degrades when all the experiments where NUMA nodes of the first socket access the NVMe drives on the second socket. However, this is not the case for the opposite: NUMA nodes of the second socket accessing NVMe drives connected to the first socket.

To further elaborate on this behavior, Fig. 6 shows the maximum storage bandwidth achieved by each NUMA node for all combinations of NVMe drives. The throughput degradation observed in Fig. 5 is shown when the first socket accesses all drives from individual NUMA nodes of the second socket. However, it is compensated when read from drives of multiple NUMA nodes of the second

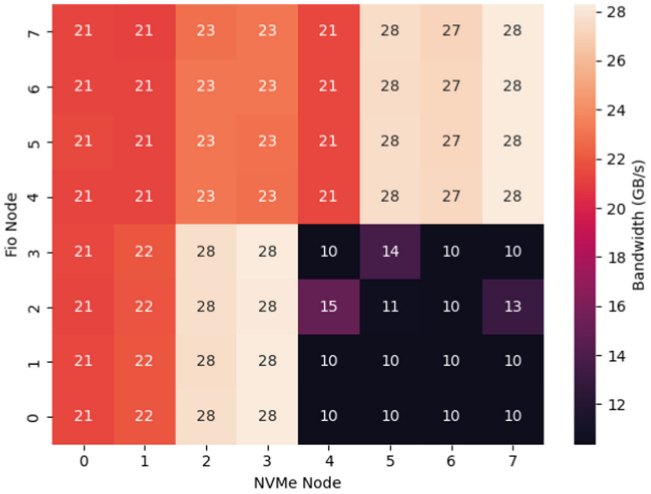


Fig. 5. Storage bandwidth (GB/s) in AMD EPYC – All cores of NUMA node accesses all NVMe drives in target NUMA node

socket. Secondly, for both sockets, when accessing from NVMe drives of the first and last NUMA node of the remote socket, the throughput degrades (fio-node 4,5,6,7 accessing data from [0,3] and fio-node 0,1,2,3 accessing data from [4,7]). To the best of our knowledge, the reason for this behavior is unknown and may be a fault in hardware or software configuration. However, this is one of the main benefits of Chaosity: targeting and identifying such unexpected and anomalous behaviors. Without Chaosity, one would have deployed a fully functional system and then spent time analyzing system performance regression while not knowing that the actual issues are not in the system design but in the hardware or hardware configuration.

5.2 Bandwidth Interference – Interconnect & Memory

PCIe data transfers also consume memory bandwidth when reading or writing from/to CPU memory. One such case is when reading or writing data from/to NVMe drives. This causes interference and, counter-intuitively, consumes the memory bandwidth, limiting the processors' data processing performance. In what follows, we analyze the interaction of PCIe bandwidth with memory bandwidth and provide insights for data-intensive processing.

Figure 7 plots the interaction between IO and memory bandwidth on a single NUMA node. Data is read from the NVMe drives using fio, while simultaneously, STREAM is run on the same NUMA node. As the number of NVMe drives read from increases, more memory bandwidth is used for IO, and we observe lower memory bandwidth consumed by STREAM. This exemplifies the point that IO bandwidth consumes memory bandwidth. This is increasingly relevant as storage bandwidths increase, resulting in memory bandwidth competition.

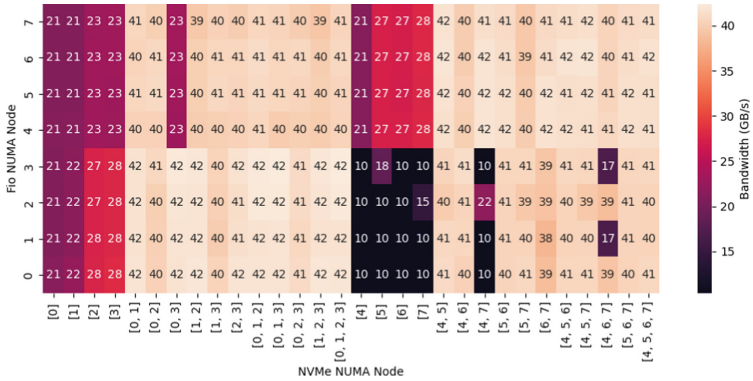


Fig. 6. Storage bandwidth (GB/s) in AMD EPYC – All cores of NUMA node accesses specified NVMe drives in target NUMA node

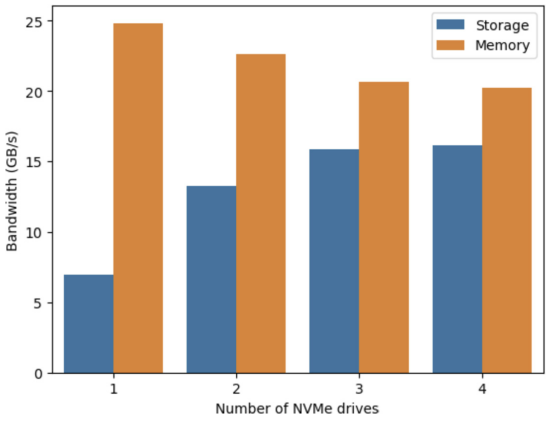


Fig. 7. Interference in read bandwidth between storage and memory

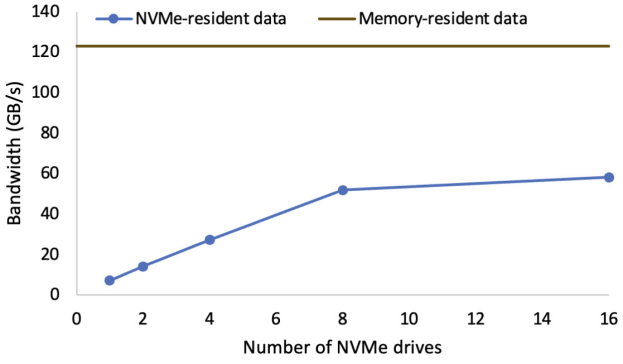


Fig. 8. CPU and IO compete for memory bandwidth within the same workload.

Figure 8 demonstrates competition for memory bandwidth between IO and a simple compute kernel in a single program. The compute kernel performs a summation over an array of integers, using one socket of the server. The black line shows the processing throughput when the array is entirely in memory. The blue line shows the throughput when the array is striped across an increasing number of NVMe drives on the socket. When the array is NVMe resident, it is asynchronously transferred in 2 MiB chunks into memory using io_uring, as the chunks arrive in memory they are consumed by the compute kernel so that the compute and NVMe transfers are overlapped. This experiment differs from simply using fio, as fio only transfers data from storage to memory and does not also load the transferred data again from memory to the CPU.

Operating on storage resident data consumes twice the memory bandwidth than operating on in-memory data. We observe that for the baseline memory-resident data, the processing throughput is 120 GiB/s, near the single socket memory bandwidth. When reading from storage, the number of drives, and hence the storage bandwidth, is the initial bottleneck. Whereas, with eight or more drives, the available storage bandwidth does not improve the processing throughput; as the transfers from storage consume memory bandwidth to write to memory and the CPU utilizes the remaining memory bandwidth to read the data from memory. At this point, the bottleneck has shifted to memory bandwidth.

6 Discussion

In Sects. 4 and 5, we presented experimental results which demonstrate some of the complexities of the modern hardware landscape that data-intensive systems developers must account for. Architectures like the Apple M1 can only fully utilize the memory bandwidth by using the GPU but at the cost of interfering with the CPU's memory bandwidth. The EPYC Milan architecture allows for tremendous PCIe bandwidth, which can be used for NVMe storage, but the PCIe/storage bandwidth cannot be fully utilized if the CPU also needs to transfer the data between its caches and memory as memory bandwidth becomes the bottleneck. Our results are only for two specific topologies. However, servers can be configured in many different ways, for example, with PCIe network interface cards (NIC) and accelerators. This enables additional data transfer paths such as storage to accelerator and storage to NIC transfers, which bypasses the main memory but still consume IO bandwidth [16,18].

The complex topology of modern servers, both due to varying CPU architectures and possible server configurations, severely increases the cognitive load on designers of data-intensive systems. System designers strive to maximize hardware utilization in order to minimize the cost and energy use of their systems. We expect two common approaches to achieve good utilization in the era of diverse hardware. First, organizations that both develop software and deploy on hardware they manage, such as large cloud companies, may evaluate multiple types of servers and settle on one or a small number to deploy and optimize for.

Second, systems that need to achieve portable performance can adapt to the topology at run-time. Still, both approaches necessitate that system designers have a deep understanding of hardware.

Our long-term goal is to assist and enable system designers to understand hardware better. Through topology and interference-aware benchmarking, Chaosity enables the exploration of the limits of the hardware in more realistic scenarios than individual system-wide single-metric benchmarks. Chaosity can be a complimentary tool to software system-specific benchmarks, as it aims to reveal the characteristics of the hardware rather than the performance of a, potentially untuned, software system on new hardware. This can be especially helpful because discovering performance bottlenecks due to interference at the hardware level is time-consuming to discover through profiling alone. Chaosity synchronizes the hardware expectations and reality given the current configuration and may also detect misconfigurations and defects early rather than wasting time in debugging/profiling a full software system.

Future Directions. We envision Chaosity to be integrated with automatic topology discovery and adaptive components in a system as an input provider, thereby assisting systems in adapting to increasingly complex underlying hardware. We aim to include support for more types of benchmarks in Chaosity, including but not limited to benchmarking latency profiles, on-chip, off-chip accelerators and devices connected via Compute Express Link (CXL). Further, we also plan to add a shared and private profiling database to compare different hardware characteristics across different hardware types, vendors, and generations. We will encourage vendors and third parties to publish and compare results against standard and non-trivial configurations.

7 Conclusion

Modern hardware requires co-optimizing hardware and software. However, modern servers are becoming more diverse and heterogeneous. This complexity is a result of both CPUs that are scaling silicon horizontally and may also contain heterogeneous compute, as well as the increasing use of high-bandwidth IO devices and accelerators attached via an interconnect like PCIe. The diversity of server topologies will only continue to grow as novel CPUs come to market and new interconnects such as CXL enable new types of devices. Collectively, this poses new challenges for efficient and high-performance system designs.

This paper proposes an initial vision for the Chaosity framework, which assists system designers and developers in understanding the target hardware topology and associated performance characteristics. Further, hardware or software configurations are prone to misconfigurations, given the complex hardware topologies and systems designs. Chaosity will assist in detecting such problems in the early stages of hardware or software deployments by providing insights into expected hardware performance.

Acknowledgements. We would like to thank the reviewers for their valuable feedback. This work was partially funded by SNSF project "Efficient Real-time Analytics on General-Purpose GPUs" subside no. 200021_178894/1.

References

1. Alian, M., Yuan, Y., Zhang, J., Wang, R., Jung, M., Kim, N.S.: Data direct I/O characterization for future I/O system exploration. In: 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 160–169 (2020). https://doi.org/10.1109/ISPASS48437.2020.00031
2. Apple Inc.: macOS Taskpolicy Man Page. https://github.com/apple-oss-distributions/system_cmds/blob/559f661c5687f7828307cb3b1026a45f849243c6/taskpolicy.tproj/taskpolicy.8
3. Axboe, J.: Flexible I/O Tester (2022). https://github.com/axboe/fio
4. Biswas, A.: Sapphire rapids. In: 2021 IEEE Hot Chips 33 Symposium (HCS), pp. 1–22 (2021). https://doi.org/10.1109/HCS52781.2021.9566865
5. Broquedis, F., et al.: hwloc: a generic framework for managing hardware affinities in HPC applications. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pp. 180–186 (2010). https://doi.org/10.1109/PDP.2010.67
6. Chrysogelos, P.: Efficient analytical query processing on CPU-GPU hardware platforms. Ph.D. thesis, EPFL, Lausanne (2022). https://doi.org/10.5075/epfl-thesis-8068. http://infoscience.epfl.ch/record/296204
7. Corsair MP600 PRO 2TB M.2 NVMe PCIe Gen. 4 ×4 SSD. https://www.corsair.com/us/en/p/data-storage/cssd-f2000gbmp600pro/mp600-pro-2tb-m-2-nvme-pcie-gen-4-x4-ssd-cssd-f2000gbmp600pro
8. Kenyon, C., Capano, C.: Apple silicon performance in scientific computing. In: IEEE High Performance Extreme Computing Conference, HPEC 2022, Waltham, MA, USA, 19–23 September 2022, pp. 1–10. IEEE (2022). https://doi.org/10.1109/HPEC55821.2022.9926315
9. Liu, T.: Tf-metal-experiments (2021). https://github.com/tlkh/tf-metal-experiments
10. McCalpin, J.D.: Memory bandwidth and machine balance in current high performance computers. In: IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 19–25 (1995)
11. Naffziger, S., et al.: Pioneering chiplet technology and design for the AMD Epyc^{TM}and Ryzen^{TM}processor families: industrial product. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). pp. 57–70. IEEE (2021). https://doi.org/10.1109/ISCA52012.2021.00014
12. Naffziger, S., Lepak, K., Paraschou, M., Subramony, M.: AMD chiplet architecture for high-performance server and desktop products. In: 2020 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 44–45. IEEE (2020). https://doi.org/10.1109/ISSCC19947.2020.9063103
13. Nassif, N., et al.: Sapphire rapids: the next-generation Intel Xeon scalable processor. In: 2022 IEEE International Solid- State Circuits Conference (ISSCC), vol. 65, pp. 44–46 (2022). https://doi.org/10.1109/ISSCC42614.2022.9731107
14. Nicholson, H., Chrysogelos, P., Ailamaki, A.: Hpcache: memory-efficient OLAP through proportional caching. In: Blanas, S., May, N. (eds.) International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022, pp. 7:1–7:9. ACM (2022). https://doi.org/10.1145/3533737.3535100

15. Nicholson, H., Raza, A., Chrysogelos, P., Ailamaki, A.: Hetcache: synergising NVMe storage and GPU acceleration for memory-efficient analytics. In: 13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, 8–11 January 2023 (2023). www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p84-nicholson.pdf
16. NVIDIA: GPUDirect storage: a direct path between storage and gpu memory (2019). https://developer.nvidia.com/blog/gpudirect-storage/
17. NVIDIA: NVIDIA grace hopper superchip architecture whitepaper (2023). https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper
18. NVMe standard: NVM express RDMA transport specification (2022). https://nvmexpress.org/wp-content/uploads/NVM-Express-RDMA-Transport-Specification-1.0b-2022.10.04-Ratified.pdf
19. Ramos, S., Hoefler, T.: Capability models for manycore memory systems: a case-study with Xeon Phi KNL. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 297–306 (2017). https://doi.org/10.1109/IPDPS.2017.30
20. Raza, A., Chrysogelos, P., Anadiotis, A.G., Ailamaki, A.: Adaptive HTAP through elastic resource scheduling. In: Maier, D., Pottinger, R., Doan, A., Tan, W., Alawini, A., Ngo, H.Q. (eds.) Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Online Conference, Portland, OR, USA, 14–19 June 2020, pp. 2043–2054. ACM (2020). https://doi.org/10.1145/3318464.3389783
21. Raza, A., Chrysogelos, P., Sioulas, P., Indjic, V., Anadiotis, A.G., Ailamaki, A.: GPU-accelerated data management under the test of time. In: 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, 12–15 January 2020, Online Proceedings (2020). www.cidrdb.org. http://cidrdb.org/cidr2020/papers/p18-raza-cidr20.pdf
22. Su, L.: AMD CES 2023 keynote (2023). http://www.ces.tech/sessions-events/keynotes.aspx
23. Williams, S., Ionkov, L., Lang, M.: Numa distance for heterogeneous memory. In: Proceedings of the Workshop on Memory Centric Programming for HPC, MCHPC 2017, pp. 30–34. Association for Computing Machinery, New York (2017). https://doi.org/10.1145/3145617.3145620

Benchmarking Large Language Models: Opportunities and Challenges

Miro Hodak[1(✉)], David Ellison[2], Chris Van Buren[2], Xiaotong Jiang[2], and Ajay Dholakia[2]

[1] AMD, Data Center Solutions Group, Austin, TX, USA
Miro.Hodak@amd.com
[2] Lenovo, Infrastructure Solutions Group, Morrisville, NC, USA
{dellison,cvanburen,jiangxt,adholakia}@lenovo.com

Abstract. With exponentially growing popularity of Large Language Models (LLMs) and LLM-based applications like ChatGPT and Bard, the Artificial Intelligence (AI) community of developers and users are in need of representative benchmarks to enable careful comparison across a variety of use cases. The set of metrics has grown beyond accuracy and throughput to include energy efficiency, bias, trust and sustainability. This paper aims to provide an overview of popular LLMs from a benchmarking perspective. Key LLMs are described, and the associated datasets are characterized. A detailed discussion of benchmarking metrics covering training and inference stages is provided and challenges in evaluating these metrics are highlighted. A review of recent performance and benchmark submissions is included, and emerging trends are summarized. The paper lays the foundation for developing new benchmarks to allow informed comparison of different AI systems based on combinations of models, datasets, and metrics.

Keywords: Artificial Intelligence · Inference · Training · MLPerf · TPCx-AI · Deep Learning · Performance · Large Language Models

1 Introduction

Currently, Large Language Models (LLMs) are the hottest trend in Artificial Intelligence (AI) development and in datacenter workloads in general. Enabled by the development of transformers, they have replaced vision workloads as the cutting edge of AI. While LLMs have been popular in the AI community at least since the release of BERT in 2018, it was public release of ChatGPT in November 2022 that captured public attention and became a watershed moment in LLMs. ChatGPT demonstrated that it can answer questions, write text and computer programs, pass exams, and much more. It became the fastest service to reach 100 million users [1].

The success of ChatGPT has catalyzed LLM development and deployment. It has led to multiple companies rushing to deploy their own LLM models. For example, in February 2023, Google announced its own rival chatbot, Bard [2].

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 77–89, 2024.
https://doi.org/10.1007/978-3-031-68031-1_6

At the same time, it has become evident that LLM training and deployment is very costly. While GPT-3-based ChatGPT remains free, the large size of the models combined with the need for large throughput - i.e., generating a large amount of text quickly - means that deployments usually require a large number of accelerators. This makes benchmarking indispensable for evaluating different modes of deployments. Yet, LLM benchmarking lags far behind deployments.

Beyond the rapid public adoption, other reasons benchmarking has lagged are a lack of standardization and high compute requirements. So far, the gap has been filled with many ad hoc efforts with chip makers claiming leadership in workloads best suited to their products and individuals and enthusiasts publishing their own results. Given the importance of this area, it is imperative that standardization and benchmarking quickly catch up. On the standardization side, there is a need for identifying the most representative LLMs along with datasets and with accuracy criteria – unlike in vision workloads, accuracy scoring in LLMs is less obvious and several competing metrics have been developed. Once the models and datasets are identified, a set of benchmarking rules can be defined to arrive at a fair LLM evaluation.

The current state of the field is that only a single standards organization has brought LLM benchmarks: MLCommons. This organization, which publishes MLPerf benchmarks has published its first round of LLM benchmarking within its MLPerf Training suite as of writing of this paper (June 2023) with MLPerf Inference slated to include LLM in the next version scheduled for release in September 2023. Other organizations have yet to announce their plans for LLM inclusions.

This paper is organized as follows: Sect. 2 describes difficulties in LLM benchmarking, Sects. 3 and 4 give an overview of LLM models and datasets, respectively. Sects. 5 and 6 give benchmarking considerations for LLM training and inference, respectively. Sect. 7 discusses metrics for evaluating LLM performance while Sect. 8 reviews current state of LLM benchmarking in established benchmarking suites. Finally, Sect. 9 give a Summary and Conclusions.

2 Difficulties in LLM Benchmarking

Running LLM workloads is challenging because it requires large computational resources. LLM training is usually done on many – tens, hundreds, or even thousands – of GPUs. Therefore, realistic LLM training benchmarking requires an accelerated scale out cluster, which means that an organization needs to put in a substantial investment into its benchmarking compute resources. Beyond raw computational power, a storage system needs to be in place – a commonly used C4 dataset is about 750GB in size – capable of enough throughput for the compute cluster. A common pitfall in LLM performance published so far is measuring throughput only – throughput can often be increased at the cost of worsened convergence and thus does not reflect the actual HW performance. A realistic benchmark needs to take convergence into account, but because the full convergence is too costly it can only be estimated.

LLM inference requires less computational resources, but even then, scale out or at least scale up may still be required owing to the large size of LLM models. These can be pruned and quantized to decrease the size, but that needs to be balanced with

accuracy. Beyond the size, realistic LLM inference should generate a reasonable number of words per second – at least a few per second – meaning that just making an inference benchmark to run is not sufficient to evaluate its deployment performance.

These and other considerations are discussed in more detail in the following sections.

3 LLM Models

Table 1 shows a quick overview of most common LLMs, which are also discussed in more details below. These are built on transformer architecture [3].

Table 1. A selection of noteworthy base LLMs.

Model	Creator	Availability	Parameters	Training Data	Architecture
BERT	Google	Apache 2.0 (commercial)	110M (base), 340M (large)	3.3B words	Bidirectional (not generative) transformer
GPT-3	OpenAI	Proprietary (API only)	175B	300B tokens	Auto-regressive (generative) transformer
LLaMa	Meta	CC BY-NC-SA 4.0 (non-commercial)	7B, 13B, 33B, and 65B	1T tokens (7B, 13B models), 1.4T tokens (33B, 65B models)	Auto-regressive (generative) transformer
LLaMa 2	Meta	Llama 2 community license (commercial)	7B, 13B, 34B, 70B	2T tokens (7B, 13B, 34B, 70B models)	Auto-regressive (generative) transformer
MPT-7B	MosaicML	Apache 2.0 (commercial)	7B	1T tokens	Auto-regressive (generative) transformer
PaLM 2	Google	Proprietary (API only)	Four sizes, count unknown	unknown	Auto-regressive (generative) transformer
BLOOM	BigScience	Big Science RAIL License (commercial)	176B	366B tokens	Auto-regressive (generative) transformer
GPT-J	EleutherAI	Apache 2.0 (commercial)	6B	402B	Auto-regressive (generative) transformer

3.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is one of the most widely adopted transformer self-supervised language models. A comparatively simple model to modern standards it has 340M parameters and is not truly a generative model as it is technically an encoder-only bidirectional transformer. However, it serves as the base of so many subsequent models it bears mentioning here. BERT is a masked language model in that it takes the sample dataset and "masks" 15% of all the tokens which it then tries to predict [4]. This was the breakthrough that allowed it to be used on unsupervised datasets such as English Wikipedia (2,500M words) and the BooksCorpus (800M words).

3.2 GPT-3

Generative Pre-trained Transformers (GPT)-3 is a decoder only unidirectional autoregressive model that has 175 billion parameters. This model by OpenAI is the model that powers ChatGPT. This model was trained on 499 billion tokens consisting of CommonCrawl (570 GB), WebText, English Wikipedia, and two books corpora (Books1 and Books2) [5]. A previous restriction on many models has been a relatively small context such as only examining a sentence at a time. However, GPT-3 uses 2048-token-long context which is long enough to demonstrate strong zero-shot and few-shot learning on many tasks.

3.3 PaLM

Pathways Language Model (PaLM) is a recent breakthrough from Google that is 540 billion parameters. It gets its name from the novel method in which it is trained, the Pathways system, which enables efficient multi-node training of the model. PaLM has a modified encoder-decoder transformer architecture. It has achieved state-of-the-art results on 28 out of the 29 most widely used English Natural Language Processing (NLP) tasks that "span question-answering tasks (open-domain closed-book variant), cloze and sentence-completion tasks, Winograd-style tasks, in-context reading comprehension tasks, common-sense reasoning tasks, SuperGLUE tasks, and natural language inference tasks" [6]. In addition to this, PaLM is notably achieved state-of-the-art performance on arithmetic and commonsense reasoning task which has traditionally been especially difficult for LLMs.

3.4 LLaMA

Large Language Model Meta AI (LLaMA) was released by Meta AI in February of 2023. This model has a variety of model sizes from 7 billion to 65 billion parameters, with the largest models rivaling PaLM [7]. It draws from an enormous training dataset including webpages (CommonCrawl), open-source repositories (GitHub), Wikipedia in 20 languages, books (Project Gutenberg), scientific papers (ArXiv), and questions and answers (Stack Exchange) [6]. LLaMA 2 was released by Meta AI in July of 2023, LLaMA 2 has a variety of model sizes from 7 billion to 70 billion parameters. LLaMA

2 used public data source, but it didn't specify the source name [30]. Architecturally different from GPT-3, LLaMA/LLaMA 2 uses a SwiGLU activation function, rotary positional embeddings, and root-mean-squared layer normalization.

3.5 GPT-J

Generative Pre-trained Transformer-J (GPT-J) or GPT-J-6B, is a 6 billion parameter model developed by EleutherAI. As the name suggests, it is a GPT-3 like model with a few adjustments. The three main differences are that it uses dense attention, Rotary Position Embeddings, and that the attention and feedforward network were computed in parallel during training [8]. The model is trained on a published dataset that is 800GB consisting of 22 smaller, high-quality datasets [9].

3.6 BLOOM

BigScience Large Open-science Open-access Multilingual Language Model (BLOOM) was created by over 1000 AI researchers to provide an open-source LLM. Unlike a lot of the other LLMs that are only trained on English, BLOOM was trained on the ROOTS corpus which comprises of hundreds of datasets in 46 different natural languages and 13 programming languages [10]. The architecture is modified from Megatron-LM GPT2 and is a decoder-only architecture with ALiBi positional encodings, GeLU activation functions, and layer normalization applied to word embeddings layer.

4 LLM Datasets

Preparing datasets for pre-training the LLMs is a multi-step process. Each of the LLMs described in this paper were pre-trained using datasets prepared uniquely for this purpose. That said, there are also significant commonalities. In this section, we describe these similarities and differences and distill key considerations from a benchmarking perspective.

In general, LLMs are trained in two phases: pre-training and fine-tuning. The pre-training is typically unsupervised and uses a large dataset. The fine-tuning initially was supervised, using a labeled dataset for specific target tasks. This step has now been replaced by few-shot training, using a small number of example prompts. A special case is zero-shot where no examples are used, and the model is directly used with new data. The change from target-specific fine-tuning to few-shot training has been possible by progressively increasing the scale of the LLM by a factor of ten to a thousand and more. The datasets used for fine-tuning are now largely part of the evaluation and benchmarking stage and span a number of different target tasks.

We focus on the pre-training datasets and describe their evolution over the past several years. The initial GPT [11] used BookCorpus [12]. GPT-2 [13] used WebText created from the Common Crawl web scrape. GPT-3 [14] started with CommonCrawl [15], performed deduplication and added WebText, Books1, Books2, and Wikipedia. GPT-4 [16] has used data from public sources as well as data licensed from third-party sources.

A team from Meta has recently released LLaMA [6]. C4 [15] dataset was used, along with CommonCrawl. The key difference in its training dataset is that it was created from only publicly available data sources. This contrasts with OpenAI's GPT models that use proprietary data licensed from 3rd parties along with publicly available data.

A number of models and chatbots have been released by fine-tuning the LLaMA model: Alpaca, Vicuna, and Koala. Most recently, MosaicML team released the MPT-7B model [17] trained on dataset curated by combining data from various public sources.

Table 2 shows a comparison of data sources used by popular LLMs.

Table 2. Training datasets used by various models.

Model	Training tokens	Main data sources for pre-training	License
GPT	0.04T	BookCorpus	Non-commercial
GPT-2	0.4T	WebText, Common Crawl	Proprietary
GPT-3	0.3T	Common Crawl, Books1, Books2, Wikipedia	Proprietary
LLaMA	1T	C4, Common Crawl	Non-commercial (CC BY-NC-SA 4.0)
MPT-7B	1T	Text and code	Commercial (Apache 2.0)

The question of license needs to also be addressed. Most of the LLMs are released under a non-commercial, research use only license. One exception is MosaicML's MPT-7B, which is released under a commercial license.

One key lesson from these modifications to datasets is that limiting the dataset to public-domain accessible sources only does not impact the performance as long as the size of the dataset is large enough. This is good news from a benchmarking perspective. A curated dataset based on public sources can be the basis for use in a benchmarking suite.

5 LLM Training: Benchmarking Considerations

LLM training requires very large computational resources and takes a very long time. As a result, it is very costly. For example, Meta has reported that their LLaMA 65B model training took 21 days on 2,048 NVIDIA A100 GPUs [18]. This is about 1 million GPU hours, which would cost about $2.4 million on AWS [19]. Similarly, Hugging Face has reported that training their Bloom LLM took more than two-and-a-half months using about 500 GPUs. OpenAI's GPT-3 training cost was estimated at $4 million. This makes realistic benchmarking very difficult because such a high cost cannot be justified for benchmarking work.

Duration of LLM training is determined by the number of tokens processed. For each model, there is an optimal number of tokens needed for training, using more does not improve the results due to overfitting. In general, models with more parameters need

more tokens. For example, the GPT-J 6B model needs about 134 billion tokens, while it is about 3.5 trillion tokens for GPT-3 175B. In fact, some of the largest LLMs are limited by insufficient number of quality data to train them.

Optimum number of tokens is determined by testing the model against a chosen accuracy metric. In practice, multiple checkpoints are saved during training, which are later evaluated for performance and best performing ones are chosen for further fine-tuning. This training process is very different from training of convolutional neural nets (CNNs) used for computer vision. There, training iterates multiple times – expressed in number of epochs - over the entire dataset until the loss function is minimized. In LLMs, repeated iterations cause overfitting and degrade performance and thus the training happens within a single epoch on a subset of the dataset.

In general, training time depends on three factors: (i) Number of tokens, (ii) Number of model parameters, and (iii) Number of GPUs used for training.

The training process described above generates an all-purpose model, such models are called foundation models. In practice, fine-tuning is usually applied before deployment. Fine-tuning adapts the model to achieve better performance in a specific domain.

6 LLM Inference: Benchmarking Considerations

Given the extremely high cost of LLM training, one might expect inference to be less costly and easier to run, but that is not necessarily the case. A popular LLM inference service can have thousands of users and needs to have sufficient throughput to generate output quickly. That implies, at a minimum, multi-accelerator deployment, or even scale-out to multiple servers. Thus, a realistic LLM inference benchmark needs to be scalable.

A necessary input for inference is a pre-trained model. This has been straightforward, but with LLMs' extremely high training cost, availability of high-quality checkpoints for models such as GPT-3 175B has been an issue. Beyond the cost, additional issue is that these checkpoints are seen as a competitive advantage for companies that spent large amount of resources creating them. This creates additional barrier for performance evaluation of cutting edge LLMs.

Another issue is model size. For example, GPT-3 175B checkpoints need about 700 GB for storing parameters and about an equal amount for activations. These need to be stored in memory, which means that, at minimum, 16 80GB GPUs are needed to execute such models. To decrease memory requirements and improve compute throughput, pretrained models are usually pruned and quantized. The former refers to removing layers of neural networks, while the latter means converting weights to a lower numerical precision. Both may result in decreased performance and thus the result needs to be carefully tested to ensure that the performance loss is acceptable. Sometimes, retraining may be needed to recover lost accuracy, but this is impractical for LLMs. In general, this process is complicated, involves using of multiple software tools, and is not captured in current benchmarking tools. Nevertheless, it is an essential part of AI workflow and benchmark creators need expertise in these techniques to create relevant benchmarks.

In LLM inference, pruning and quantization are critical because they decrease size and computational requirements of the models. For pruning, one notable study found

that LLMs can be pruned to at least 50% sparsity in one-shot, without any retraining, at minimal loss of accuracy [20]. Similarly, powerful quantizing techniques have been developed, with one of them delivering up to 1.56x speedup and 2x memory reduction [21]. These techniques are critical, because they enable running inference on CPUs and ASIC chips without the need for powerful GPUs.

7 LLM Performance Evaluation

Evaluating the performance of LLMs is an active area of research and discussion. The variety and number of possible evaluations is quite large because of the numerous domains in which LLMs are being used. The range of target tasks include language understanding, question answering systems, text summarization, machine translation and knowledge testing across a variety of disciplines and subjects. In this section, we summarize key metrics used in evaluating the performance of LLMs across the breadth of tasks.

The straightforward evaluation is done by humans examining the output of the LLMs. These are subjective evaluation of quality of the output and can include factors such as coherence, correctness and contextual relevance. Such evaluations can be averaged by employing many human evaluators, making it a time-consuming and expensive exercise. These considerations have resulted in a number of different metrics specific to target tasks that can be programmatically evaluated.

The intrinsic quality of a language model is measured by its perplexity, the normalized inverse probability of the test set. As an inverse, a lower value of perplexity is better because it indicates a higher corresponding probability value (Table 3).

For text summarization tasks, the metric used is ROUGE (Recall-Oriented Understudy for Gisting Evaluation). It is a measure of similarity between human-generated, so-called golden annotated summary and the model-generated summary. Evaluation of ROUGE metrics includes 1-g (ROUGE-1), bi-gram (ROUGE-2) and L-gram (ROUGE-L) based F1 scores, calculated as a harmonic mean of the respective precision and recall values.

Evaluation of machine translation is done using the BLEU (Bilingual Evaluation Understudy) and measures the similarity between a candidate translation generated by the model and a reference translation. BLEU score values are in the range from 0 to 1, and higher values indicate better performance.

LLMs are also measured for knowledge tasks and a common benchmark is the MMLU (Massive Multitask Language Understanding) [22], which includes 57 tasks ranging from elementary mathematics, US history, computer science, law and more.

There are also considerations for measuring diversity in the sense of uniqueness and variety of model output, bias along various dimensions, toxicity, and accuracy in terms of factual correctness. These are all difficult metrics but will play an important role in increasing trust in AI systems [23].

In summary, many of the metrics described in this section are commonly used in academic and industry publications of new LLMs. These are very good candidates for inclusion in benchmarks for LLMs.

Table 3. Definitions of metrics for LLM benchmarking

Term	Definition
Accuracy	The correctness of objective model output
Bias	The systematic prejudice of model output, including stereotypes of certain groups or performance disparities between groups [24]
Coherence	The collective quality of the entire model output; the combined degree to which each part of the model output, given the rest of the output, contributes to the output's quality [25]
Contextual relevance	The degree to which model output is on topic in response to model input
Cost	The explicit cost (e.g., paying for cloud compute) or implicit cost (e.g., the opportunity cost of using owned computing resources) associated with acting on (i.e., training or inferencing) the model
Diversity	The variety of model outputs from varied inputs
Energy Efficiency	The energy consumption required by acting on the model, typically measured per query or per token for inference and in total energy to reach a certain checkpoint (number of tokens ingested) for training
Sustainability	The ability of a model to operate with minimal environmental impact
Throughput	The speed of generation during inference, typically measured in tokens per second
Toxicity	The tendency of a model to output "a rude, disrespectful, or unreasonable comment that is likely to make you leave a discussion" [26]
Trust	A concept comprised of several measurable metrics – interpretability, reproducibility, and replicability – as well as qualities of a model's creation – transparent knowledge of data and model provenance [27]

8 Current Benchmarking Efforts

8.1 MLPerf Training V3.0 LLM Benchmark Design

MLPerf Training is the first major benchmarking suite that includes an LLM model as of v3.0 released on June 27th, 2023 [28]. MLPerf's model is based on GPT-3 175B model trained on C4 dataset, which is about 350 GB in size and contains 174B tokens. To address the long runtimes, only a small portion of training is executed. Training starts from an initial checkpoint that has been trained on 12.5B tokens and continues to train on 1.3B tokens to the quality target of 2.69 log perplexity. Thus, the benchmark captures about 0.4% of the full GPT-3 training. Even that still requires substantial resources with reference implementation requiring a minimum of 64 accelerators mainly due to the model's large memory size. To ensure that model is on its way to convergence an accuracy evaluation is performed using about 5% of the total validation set.

8.2 MLPerf Training V3.0 LLM Benchmark Results

MLPerf v3.0 results, which include LLM for the first time, are shown in Table 4. The results show that only a few submitters submitted LLM scores; out of 12 submitters in the Closed division and Available categories, only two submitted results: (i) Nvidia (on their own and also in collaboration with CoreWeave), (ii) Intel-Habana Labs. This indicates that, as designed, this benchmark exceeds resources that most submitters have for their MLPerf work. Indeed, the smallest submission is on 256 Gaudi2 accelerators. The smallest GPU submission is on 768 GPUs, which is 96 Nvidia DGX H100 servers. On the top end is a 3,584 GPU submission using 448 Nvidia DGX servers.

The results confirm that LLM training is highly scalable as shown in Fig. 1. On Nvidia servers, going from 768 to 3584 GPUs, 4.67x more, the training time decreases by 4.17x, about 90% efficiency. Because MLPerf timings include accuracy evaluation, this underestimates the actual scaling efficiency. Intel's Gaudi results show even better scaling, of about 95% but over less accelerators – increasing from 256 to 384.

Table 4. LLM results in MLPerf Training v3.0

Submitter	CPU Qty	Accelerator Type	Accelerator Qty	Benchmark Results (min)
NVIDIA	128	NVIDIA H100-SXM5-80GB	512	64.264
NVIDIA + CoreWeave	192	NVIDIA H100-SXM5-80GB	768	45.606
NVIDIA	192	NVIDIA H100-SXM5-80GB	768	44.816
NVIDIA + CoreWeave	384	NVIDIA H100-SXM5-80GB	1536	23.611
NVIDIA + CoreWeave	896	NVIDIA H100-SXM5-80GB	3584	10.940
Intel-HabanaLabs	64	Habana Gaudi2	256	442.578
Intel-HabanaLabs	96	Habana Gaudi2	384	311.945

8.3 MLPerf Inference

MLPerf inference has not released an LLM model yet but given that it is usually synchronized with MLPerf training, an LLM inclusion is expected soon. Indeed, its reference code repository contains two LLM implementations, one based on GPT-3 175B corresponding to the MLPerf Training LLM model, while the other is GPT-J 6B, which is not included in the training. This likely reflects the fact that the 175B model is too resource intensive and the 6B model makes it easy to run on smaller computational resources. Because upcoming version of MLPerf Inference is still some time away (September

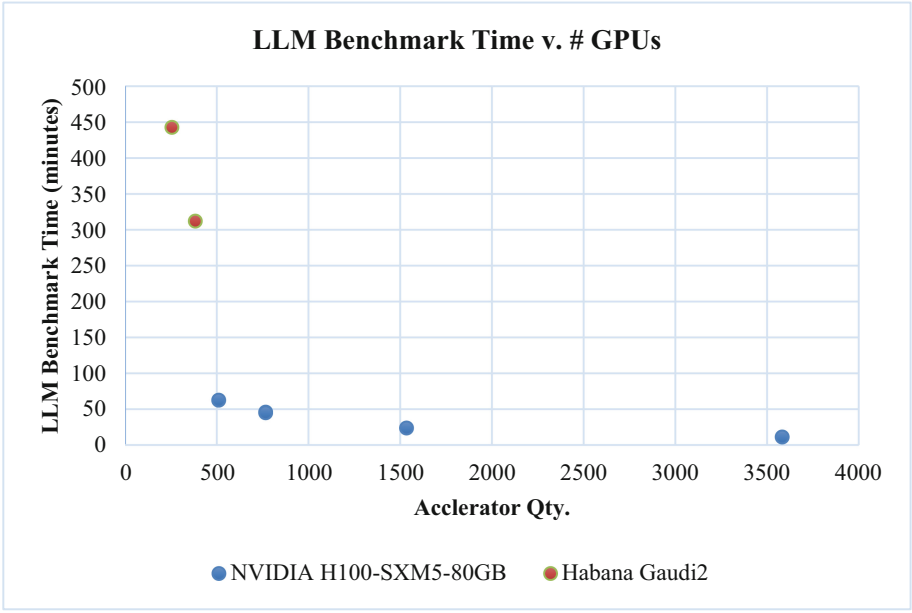


Fig. 1. Runtime vs number of accelerators for LLM results in MLPerf Training v3.0

2023, while this paper is being prepared in June 2023), it is currently unknown whether both models will ultimately be included in the next iteration of the MLPerf Inference suite, but based on publicly available information it looks like a strong possibility.

8.4 Other AI Benchmarking Suites

TPCx-AI is an AI benchmarking suite developed by TPC consortium. Some of the authors of this paper have recently reviewed this suite and compared it to MLPerf [29]. TPCx-AI focuses on the low-end AI, while LLM is on the opposite end of the AI workload spectrum. Current TPCx-AI release does not include LLM nor are there any public indications that it will be included soon.

SpecML is an AI benchmarking effort from another well-established benchmarking consortium, SPEC. Despite being announced some time ago, it has not been released yet and there is no indication that an LLM is being worked on.

9 Summary and Conclusions

This paper has discussed challenges and opportunities in LLM benchmarking as well as reviewed current state of LLM benchmarking in main AI benchmarking suites.

The opportunity part of the equation is quite clear, with exponential growth in LLM model sizes and current rush to deploy these in practice, there is a clear need for representative LLM benchmarks. There are many models to choose from along with many datasets that can be used to train these models. Additionally, there is a wealth of hardware

to test on – from high-end GPUs to midsize accelerators, datacenter CPUs all the way down to edge and embedded systems. The amount of reliable LLM benchmarking data is currently very limited meaning that many organizations have to make purchasing and deployment decisions without reliable data.

On the other side, LLM benchmarking also face multiple challenges described in this paper. One of them is the sheer scale of applications meaning that a single model will only address a limited scope of deployments and thus multiple LLM benchmarks are needed to better map out the deployment modes.

Another challenge is a high computational cost of LLM training and inference. The only existing LLM standard benchmark – GPT-3 175B included in MLPerf training – covers only 0.4% of LLM training and yet all the submissions use hundreds of accelerators, which is outside of capabilities of benchmarking setups. The high cost is an inevitable part of LLMs, but using smaller models, such as GPT-J 6B, included in the upcoming MLPerf Inference, can be a way to decrease the computational cost while being a relevant representation of LLM workloads. Ultimately, the practice will show whether the current trend of ever-increasing model sizes will continue or whether practical considerations for deployment will lead to more easily usable model while preserving most of the performance of the cutting edge LLM models.

An important aspect of LLMs is measuring their performance: Our paper discusses some of the most popular options for doing so. While performance testing for, say, image classification, is straightforward, the right metric for generative AI is less obvious. Most likely a combination of several metrics is the best way forward to evaluate usefulness of the results while minimizing bias and other undesirable effects.

In summary, LLM is an extremely active field and benchmarking is currently lagging behind. However, there are efforts under way to improve the current state indicating that the situation is changing. This work surveyed current state of the field of LLM benchmarking pointing out the unique challenges and opportunities for this workload.

References

1. Reuters. Accessed 29 June 2023. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/
2. Google. Accessed 29 June 2023. https://blog.google/technology/ai/bard-google-ai-search-updates/
3. Vaswani, A., et al.: Attention is all you need. Adv. Neural. Inf. Process. Syst. **30**, 6000–6010 (2017)
4. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805
5. Brown, T., et al.: Language Models are Few-Shot Learners. arXiv:2005.14165
6. Narang, S., Chowdhery, A.: Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance. https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html
7. Touvron, H., et al.: LLaMA: Open and efficient foundation language models. arXiv: 2302.13971v1 (2023)
8. Cerebras. Accessed 29 June 2023. https://www.cerebras.net/blog/cerebras-makes-it-easy-to-harness-the-predictive-power-of-gpt-j

9. Gao, L., et al.: The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv: 2101.00027
10. Le Scao, T., et al.: BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. arXiv:2211.05100
11. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018)
12. Zhu, Y.: Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: Proceedings of the IEEE International Conference on Computer Vision, pp.19–27 (2015)
13. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)
14. Brown, T.B., et al.: Langage models are few-shot learners, NeurIPS (2020)
15. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(1), 5485–5551 (2020)
16. OpenAI, GPT-4 Technical Report. arXiv:2303.08774v3 (2023)
17. MosaicML, Introducing MPT-7B: A new standard for open-source, commercially usable LLMs, May 2023, accessed June 29, 2023, https://www.mosaicml.com/blog/mpt-7b
18. Pursuing groundbreaking scale and accelerating research using Meta's Research SuperCluster. Accessed 29 June 2023. https://ai.facebook.com/blog/supercomputer-meta-research-supercluster-2023/
19. ChatGPT and generative AI are booming, but the costs can be extraordinary. Accessed 29 June 2023. https://www.cnbc.com/2023/03/13/chatgpt-and-generative-ai-are-booming-but-at-a-very-expensive-price.html#:~:text=Analysts%20and%20technologists%20estimate%20that,could%20cost%20over%20%244%20million
20. Frantar, E. Alistarh, D.: SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. arXiv:2301.00774 (2023)
21. Xiao, G., Lin. J., Seznec, M., Wu, H., Demouth, J., Han, S.: SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438 (2023)
22. Hendrycks, D., et al.: Measuring massive multitask language understanding, ICLR (2021)
23. Dholakia, A., Ellison, D., Hodak, M., Dutta, D.: Benchmarking considerations for trustworthy and responsible AI (Panel). In: Nambiar, R., Poess, M., (eds) Performance Evaluation and Benchmarking. TPCTC 2022. Lecture Notes in Computer Science, vol 13860. Springer, Cham. https://doi.org/10.1007/978-3-031-29576-8_8
24. Bias and Toxicity in Large Language Models. Accessed 18 July 2023. https://www.cs.princeton.edu/courses/archive/fall22/cos597G/lectures/lec14.pdf
25. Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv:2303.16634
26. Perspective. Accessed 18 July 2023. https://support.perspectiveapi.com/s/about-the-api-faqs?language=en_US
27. Dholakia, A., Ellison, D., Hodak, M., Dutta, D.: Benchmarking Considerations for Trustworthy and Responsible AI (Panel). In: Nambiar, R., Poess, M. (eds) Performance Evaluation and Benchmarking. TPCTC 2022. Lecture Notes in Computer Science, vol 13860. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-29576-8_8
28. MLCommons. Accessed 29 2023. https://mlcommons.org/en/training-normal-30/
29. Liu Olesiuk, Y., Hodak, M., Ellison, D., Dholakia, A.: More the merrier: comparative evaluation of TPCx-AI and MLPerf benchmarks for AI. In: Nambiar, R., Poess, M. (eds) Performance Evaluation and Benchmarking. TPCTC 2022. Lecture Notes in Computer Science, vol 13860. Springer, Cham. https://doi.org/10.1007/978-3-031-29576-8_5
30. Touvron, H., et al.: Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv: 2307.09288 (2023)

The Linked Data Benchmark Council (LDBC): Driving Competition and Collaboration in the Graph Data Management Space

Gábor Szárnyas[1(✉)], Brad Bebee[2], Altan Birler[3], Alin Deutsch[4,5], George Fletcher[6], Henry A. Gabb[7], Denise Gosnell[2], Alastair Green[8], Zhihui Guo[9], Keith W. Hare[8], Jan Hidders[10], Alexandru Iosup[11], Atanas Kiryakov[12], Tomas Kovatchev[12], Xinsheng Li[13], Leonid Libkin[14], Heng Lin[9], Xiaojian Luo[15], Arnau Prat-Pérez[16], David Püroja[1], Shipeng Qi[9], Oskar van Rest[17], Benjamin A. Steer[18], Dávid Szakállas[19], Bing Tong[20], Jack Waudby[21], Mingxi Wu[5], Bin Yang[13], Wenyuan Yu[15], Chen Zhang[20], Jason Zhang[13], Yan Zhou[20], and Peter Boncz[1]

[1] CWI, Amsterdam, The Netherlands
gabor.szarnyas@ldbcouncil.org
[2] Amazon Web Services, Seattle, USA
[3] Technische Universität München, Munich, Germany
[4] UC San Diego, La Jolla, USA
[5] TigerGraph, Redwood City, USA
[6] TU Eindhoven, Eindhoven, The Netherlands
[7] Intel Corporation, Austin, USA
[8] JCC Consulting, Granville, USA
[9] Ant Group, Hangzhou, China
[10] Birkbeck, University of London, London, UK
[11] VU Amsterdam, Amsterdam, The Netherlands
[12] Ontotext AD, Sofia, Bulgaria
[13] Ultipa, San Ramon, USA
[14] University of Edinburgh; RelationalAI, Eindhoven, The Netherlands
[15] Alibaba Damo Academy, Hangzhou, China
[16] Barcelona, Spain
[17] Oracle, Austin, USA
[18] Pometry Ltd., London, UK
[19] Budapest, Hungary
[20] CreateLink, Mountain View, USA
[21] School of Computing, Newcastle University, Newcastle upon Tyne, UK

Abstract. Graph data management is instrumental for several use cases such as recommendation, root cause analysis, financial fraud detection, and enterprise knowledge representation. Efficiently supporting these use cases yields a number of unique requirements, including the need for a concise query language and graph-aware query optimization techniques. The goal of the Linked Data Benchmark Council (LDBC) is to design a set of standard benchmarks that capture representative categories of

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 90–106, 2024.
https://doi.org/10.1007/978-3-031-68031-1_7

graph data management problems, making the performance of systems comparable and facilitating competition among vendors. LDBC also conducts research on graph schemas and graph query languages. This paper introduces the LDBC organization and its work over the last decade.

1 Introduction

The Graph Data Management Space. The category of data management software with *graph features* has grown steadily in the last 15 years [36]. This includes graph databases [10], relational DBMSs with graph extensions [44], graph analytics libraries [26], and graph streaming systems [12]. While graph data management systems are already popular for several use cases, such as financial fraud detection, recommendation, and data integration [35], they did not yet reach mass adoption. We believe the two key obstacles to this are: (1) the lack of standardized query languages and APIs [35], (2) limited and/or unpredictable performance in systems [36]. LDBC makes significant efforts to address these problems.

Query Languages. The adoption of graph processing systems, particularly those supporting the *property graph data model*, is considerably hindered by the lack of a standard query language [35]. Currently, systems use several different query languages, including Cypher, Gremlin, GSQL, PGQL, and DQL, which causes (potential) customers concern over lack of portability. Starting in 2017, a concentrated effort was launched to create standard query languages. The SQL/PGQ (Property Graph Queries) extension was released as part of SQL:2023 and the standalone GQL (Graph Query Language) is scheduled to be released in 2024. Both of these languages have been influenced by LDBC's G-CORE design language and LDBC has been involved in their design via its liaison with ISO.

Performance Challenges. Graph processing problems, including graph pattern matching [32], graph traversal (navigation) [4], and graph mining [13], have irregular memory access patterns and provide little spatial locality or opportunities for data reuse [17]. Contemporary CPUs are ill-suited to handle these workloads, leading to performance problems [37]. Moreover, while there were attempts to harness modern hardware such as GPUs [38] and FPGAs [11], these only proved beneficial for narrow domains and did not generalize to a wider set of use cases.

The Importance of Benchmarks. To expedite the speed of progress in graph data management systems, a group of industry and academic organizations founded the Linked Data Benchmark Council (LDBC). LDBC is an independent benchmarking organization, which defines standard benchmarks to make

A. Prat-Pérez—Work done while at UPC Barcelona and Sparsity.
D. Szakállas—Independent author.

graph query performance measurable and thus facilitate competition between vendors. In this sense, LDBC aims to fulfill a role similar to the Transaction Processing Performance Council (TPC), which defined a number of influential benchmarks. LDBC uses TPC's design and auditing processes as inspiration for its operations.

LDBC Benchmarks. LDBC has six main benchmark workloads covering different aspects of graph processing with different transactional characteristics, set of CRUD operations, and data distributions. With the exception of Graphalytics, a leaderboard-style benchmark, all benchmarks define stringent *auditing processes* for ensuring that implementations are faithful to the specification and the derived results are reproducible. As of August 2023, LDBC published 45 audited results.

Paper Structure. This paper is structured as follows. Section 2 gives an overview of the LDBC organization, including its history and structure. Section 3 presents LDBC's benchmarks, Sect. 4 describes the benchmark creation and auditing processes, and Sect. 5 summarizes our benchmark design experiences. Section 6 introduces LDBC's working groups and Sect. 7 outlines future directions.

2 The LDBC Organization

2.1 History of the Organization

Research Project (2012–2015). LDBC started as a European Union-funded research project by the same name[1] with the participation of 4 academic and 4 industry partners [3,14]. The project was coordinated by Josep Larriba Pey from Universitat Politècnica de Catalunya and Peter Boncz (CWI & VU Amsterdam). The consortium designed the Social Network Benchmark suite (SNB, Sect. 3.2), releasing its first workload, SNB Interactive [19], and the Semantic Publishing Benchmark [28] (SPB, Sect. 3.3). The non-profit company "Linked Data Benchmark Council" was established and registered in the UK.[2]

Sustained Research Efforts (2016–2018). After the EU project concluded, research efforts continued with the participation of industry partners and resulted in G-CORE [5], a declarative language designed to formulate composable graph queries. The Graphalytics benchmark (Sect. 3.4) [25] was released, and a draft version of the SNB Business Intelligence workload was published [40].

Expansion and Auditing Ramp-Up (2019–2022). LDBC's membership increased from 7 organizations in 2019 to 22 organizations in 2022 (Sect. 2.2). While in the previous phase LDBC was economically supported by CWI and Sparsity, one of the organizational improvements realized by Alastair Green was

[1] FP7-ICT grant ID 317548, https://cordis.europa.eu/project/id/317548.
[2] https://find-and-update.company-information.service.gov.uk/company/08716467.

to get LDBC a bank account, to start collecting membership fees.[3] New working groups were established to research property graph schemas and query language semantics (Sect. 6). The SNB Business Intelligence workload was completed [41] (Sect. 3.2). A new Task Force was set up to design the FinBench [24] (Sect. 3.5). LDBC's benchmark adoption process (Sect. 4.1) and auditing processes crystallized (Sect. 4.2) with multiple audits occurring per year. The term "LDBC benchmark result" was trademarked (Sect. 4.3).

Restructuring and New Benchmarks (2023–). In early 2023, the organization was restructured to simplify governance (Sect. 2.2). The benchmark Task Forces released the initial version of the FinBench, updated the SNB Interactive workload (Sect. 3.2), and organized a Graphalytics competition (Sect. 3.4).

2.2 Organizational Structure and Operations

Historical Structure (2013–2022). Until 2023, LDBC member organizations could appoint a director to the *Board of Directors*, which at its peak consisted of 20+ members, an unusual and unwieldy structure for a small non-profit company.

Current Structure (2023–). To simplify its governance, LDBC was restructured in 2023, resulting in new articles of association [29] and updated byelaws [30]. The new structure (Fig. 1) has *Voting Members*, who contribute via the *Members Policy Council*[4] and *Associate Members*, who pay no fees (and have no vote) but contribute to the day-to-day work of LDBC. There is a new, smaller *Board of Directors* (3–5 members), who are also part of the Members Policy Council.

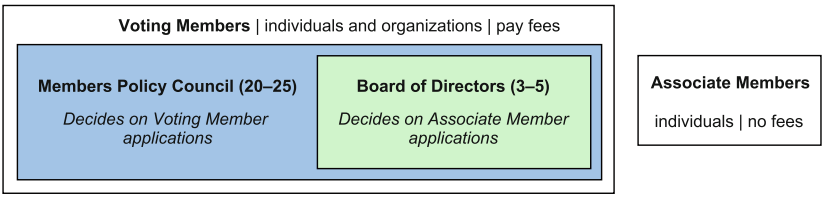


Fig. 1. Organizational structure of the LDBC from May 2023.

Membership. As of August 2023, LDBC has 24 member organizations, including database, hardware, and cloud computing vendors, and academic institutes. There are 3 sponsor companies[5], 18 member companies[6], and 3 non-commercial

[3] This seemingly trivial matter posed a practical hurdle for an organization with many directors (one per member at the time) located in different parts of the world.

[4] The *Members Policy Council* is called the *Members Council* in official documents.

[5] Ant Group, Beijing Volcano Engine Technology Co., and Oracle Labs.

[6] Amazon, Alibaba Damo Academy, ArangoDB, Beijing Haizhi Xingtu Company, CreateLink, Fabarta, Intel, JCC Consulting, Katana Graph, Memgraph, Neo4j, Ontotext, Pometry, RelationalAI, Sparsity Technologies, TigerGraph, Ultipa, and vesoft.

institutions[7]. LDBC has 3 individual voting members[8] and 60+ associate members.

Intellectual Property Rights. The intellectual property rights policies of LDBC cover assets created by members while participating in LDBC activities, including software components, written specifications, discussion proposals, academic papers, etc.. [30]. Software contributions are licensed under a licence substantively identical to the Apache Software Licence 2.0; copyright for documentary or pictorial contributions are licensed to LDBC with a right to sublicence. Typically LDBC publishes contributions either to its members or to external standards groups like ISO, or generally to the public under the Creative Commons CC-BY licence. Participants in activities of LDBC are also required to comply with the LDBC Patent Rules, which include disclosing patents that may be infringed by the implementation of an LDBC benchmark specification, or by an external standard that incorporates contributions from LDBC.

Teams. LDBC's members form teams that work on specific aspects of graph processing. *Task forces* design, implement, and maintain benchmarks (Sect. 3). *Working groups* conduct research on graph query languages and graph schema (Sect. 6). The establishment of these teams is initiated by the creation of a *work charter* and is voted on by the Members Policy Council.

Finances. LDBC's sources of revenue are its membership and auditing fees. As of 2023, membership costs 1,100 GBP for non-commercial institutes, 2,200 GBP for commercial companies, and 8,800 GBP for sponsor members. A fee of 2,000 GBP is applied to audits commissioned by non-sponsor members. Individual associate membership is free. LDBC uses its funds to pay for cloud compute, storage, and collaboration services in addition to usual company overheads.

2.3 Liaison with ISO on Standard Query Languages (GQL, SQL/PGQ)

In 2017, the international standards committee responsible for the SQL database language standard (ISO/IEC/JTC 1/SC 32/WG 3 "Database Languages")[9], established a Category C Liaison relationship with LDBC. This liaison allows WG 3 to share its working documents, draft specifications, and draft digital artifacts with LDBC participants. This access gives LDBC members early visibility into standards development efforts.

Peter Boncz, Alastair Green, and Jan Hidders are LDBC's liaison participants on the official ISO roster for WG 3, and can participate in any WG 3 meeting. This liaison relationship has helped to make the SC 32/WG 3 work more visible to organizations participating in LDBC and give the LDBC work more visibility in the Database Language standards process.

[7] FORTH; Birkbeck, University of London; and Zhejiang Lab.
[8] Peter Boncz, Alexandru Iosup, and Gábor Szárnyas; all co-authors of this paper.
[9] https://www.iso.org/organization/6720817.html.

WG 3 has been working on two projects that are of interest to LDBC members:

- ISO/IEC 9075-16 Information technology—Database languages SQL—Part 16: Property Graph Queries (SQL/PGQ)
- ISO/IEC 39075 Information technology—Database languages—GQL

SQL/PGQ is completed and was published by ISO at the end of May 2023. GQL is currently undergoing a Draft International Standard (DIS) ballot that started on 2023-05-23 and ends on 2023-08-15. WG 3 aims to resolve any issues identified during the DIS ballot and to have the GQL standard ready for publication in early 2024.

Within the database language standards committees, there is ongoing work to expand the Graph Pattern Matching (GPM) language [18] in areas such as cheapest path queries. This GPM work will be integrated into the next editions of both SQL/PGQ and GQL.

The LDBC Extended Schema (LEX) working group [21] (Sect. 6.4) aims to propose expanded schema capabilities in a future edition of the GQL standard.

2.4 Technical User Community (TUC) Meetings

Table 1. LDBC Technical User Community meetings between 2018 and 2023.

#	Year	Date	Location	Format	Program
16	2023	June 23–24	Seattle, WA	hybrid	31 talks
15	2022	June 17–18	Philadelphia, PA	hybrid	26 talks
14	2021	August 16	Copenhagen, Denmark	hybrid	19 talks
13	2020	June 30–July 1	*online*	online	4 sessions
12	2019	June 5	Amsterdam, the Netherlands	in-person	13 talks
11	2018	June 8	Austin, TX	in-person	11 talks

Since 2012, LDBC organizes Technical User Community (TUC) meetings.[10] These are 1–2 day informal workshops, where LDBC's leaders, task forces, and working groups report on their progress. Additionally, member companies give updates of their products, researchers in the graph space discuss their latest results, and users of graph data management systems present their use cases. The meetings provide an opportunity for members to contribute to LDBC's *choke points* (Sect. 3.1), and to influence the future direction of LDBC. In recent years, the TUC meetings have been steadily gaining popularity (Table 1).

[10] https://ldbcouncil.org/tags/tuc-meeting/.

3 Benchmarks

Table 2. Key characteristics of LDBC benchmarks. *Scale:* size of the largest data set, *GP lang.:* is the use of general-purpose programming languages allowed for implementations? *Req. isol.:* required isolation level (SI: snapshot isolation, RC: read committed). Legend: ⊗ yes, ○ no, ⊘ optional; ⊛ the benchmark is under design and audits are not yet possible; (i) larger sizes can be generated using the Graphalytics graph generator, but are not part of the standard benchmark.

Benchmark	Year	Workload	Scale	Min. scale	#Queries	#Audits	Inserts	Deletes	GP lang.	ACID test	Req. isol.
SNB BI	2022	analytical	30,000	30	20	4	⊗	⊗	○	⊘	SI
SNB Interactive v1	2015	transactional	1,000	30	21	24	⊗	○	⊗	⊗	RC
SNB Interactive v2	⊛	transactional	30,000	30	21	⊛	⊗	⊗	⊗	⊗	SI
SPB	2015	transactional	5	1	12	17	⊗	⊗	○	○	RC
FinBench	2023	transactional	10	0.1	40	0	⊗	⊗	⊗	⊗	RC
Graphalytics	2016	algorithms	320(i)	–	6	–	○	○	⊗	–	–

In this section, we describe LDBC's benchmarks. We first present LDBC's common benchmark terminology (Sect. 3.1). We then describe the workloads of the Social Network Benchmark suite (SNB,Section 3.2), followed by the Semantic Publishing Benchmark (SPB, Sect. 3.3), the FinBench (Sect. 3.5), and Graphalytics (Sect. 3.4). The benchmarks are summarized in Table 2. Systems that implement at least two LDBC benchmarks are shown in Table 3.

3.1 Benchmark Terminology

Choke Points. LDBC's benchmark design process uses *choke points* [15], i.e. well-chosen technical difficulties that are challenging for the present generation of data processing systems and whose optimization likely results in significant overall performance improvements. The choke points are identified by expert data systems architects and also subject to feedback received at the Technical User Community meetings (Sect. 2.4). LDBC workloads are designed to cover the set of choke points triggered by a given workload category.

Table 3. Systems with implementations for 2+ LDBC benchmarks. Legend: ⊗ full, ⊘ partial, ○ no implementation; (a) audited implementation; (s) the implementation was used for a standard-establishing audit.

Benchmark	CreateLink Galaxybase	OntotextGraphDB	Graphscope-Flex	Neo4j	PostgreSQL	Sparsity Sparksee	Microsoft SQL Server	TigerGraph	Ant Group TuGraph	Umbra	OpenLink Virtuoso
SNB BI	○	○	○	⊗	⊗	⊘	○	⊗(a)(s)	○	⊗(s)	⊘
SNB Interactive v1	⊗(a)	⊗(a)	⊗(a)	⊗	⊗	⊗(s)	⊘	⊗	⊗(a)	⊗	⊗(s)
SNB Interactive v2	○	○	○	⊗	⊗	○	⊗	○	⊘	⊗	○
SPB	○	⊗(a)	○	○	○	○	○	○	○	○	⊗(a)
FinBench	⊗(s)	○	○	○	○	○	○	○	⊗(s)	○	○
Graphalytics	⊗	○	⊗	⊗	○	○	○	○	⊗	⊘	○

Auditing. Similarly to TPC's *Enterprise Class benchmarks* [34], most of LDBC's benchmarks must undergo an auditing process conducted by a certified auditor before they can be published as official results. The auditors check compliance with the specification, run the benchmark independently and present their findings in a *full disclosure report* (FDR). The FDR documents the benchmark setup and the derived results in detail, typically spanning over 20–50 pages. Additionally, audited benchmark results are accompanied by a *supplementary package*, which includes the benchmark implementation and the binary of the system-under-test (SUT), ensuring that the results are reproducible.

Scale Factors. LDBC's benchmark suites include data generators that produce synthetic data sets of increasing sizes. Each data set is characterized by its *scale factor* (SF) which corresponds to the data set's disk usage when serialized in CSV (comma-separated values) format, measured in GiB.

ACID Compliance. Several of LDBC's benchmarks require the SUT to comply with ACID properties. An important aspect of this is *durability*: the SUT must be able to recover from a crash or power outage without losing any committed data.[11] For the SNB workloads and FinBench, the *isolation* properties are tested with an ACID test suite.

3.2 Social Network Benchmark (SNB) Suite

The Social Network Benchmark suite pioneered a number of techniques used in LDBC benchmarks: choke point-driven design [15], scalable correlated dynamic graph generation [33,43], and parameter curation for stable query runtimes [22]. The detailed specification of the SNB workloads is available at [6].

SNB Data Generators. The first version of the SNB data generator was implemented in Hadoop and only supported insert operations [33]. In 2020, it was ported to Spark for improved scalability,[12] and was extended with support for producing deep (cascading) delete operations [43]. To the best of our knowledge, its ability to generate a scalable graph where structure and values correlate, with flashmob-style spikes and deep delete operations are features unique to the SNB data generator [16].

SNB Interactive Workload v1

The SNB Interactive v1 workload was published in 2015 [19]. It is a transactional benchmark that targets OLTP systems with graph features (e.g. path-finding). The workload consists of three types of operations: 14 complex read queries, 7 short read queries, and 8 inserts. The workload has a balanced mix of operations with approximately 8% complex reads, 72% short reads, and 20% inserts.

[11] Unlike TPC, LDBC does not require systems to tolerate hardware failures.
[12] https://github.com/ldbc/ldbc_snb_datagen_spark.

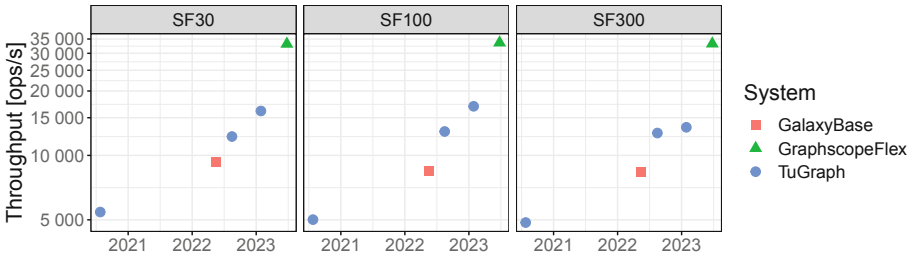


Fig. 2. Throughput of the audited results over time for the SNB Interactive v1 workload for scale factors 30, 100, 300 for implementations using general-purpose programming languages (C++ and Java), obtained between 2021 and 2023.

SNB Interactive v1 has been influential in the graph data management space: as of June 2023, 24 audited results were published using this workload.[13] Fig. 2 shows the top 15 audited results published between 2021 and 2023, which demonstrate a performance increase of more than 6× over this period.

SNB Business Intelligence Workload

The SNB Business Intelligence (BI) workload [41] captures an OLAP scenario with heavy-hitting analytical queries that touch on large portions of the graph (e.g. aggregating all Messages created within a 100-day period or exploring the neighbourhoods of the Persons living in China). The queries include multi-source *cheapest path-finding* (also known as *weighted shortest paths*), cyclic graph patterns, and have correlated and anti-correlated variants. The workload uses an improved data generator which produces fully dynamic graphs with insert and delete operations [43], and scales to SF30,000. The BI workload targets both DBMSs and data analytical systems such as Spark. To this end, updates can be applied in two modes: in *concurrent read–write mode*, reads and writes are executed concurrently, while in *disjoint read–write mode*, alternating blocks of reads and writes (accounting for one day's worth of updates) are executed. The BI workload was officially approved in November 2022. Since then, it has accumulated 4 audited results, including one for SF10,000.[14]

SNB Interactive Workload v2

In 2022, the SNB task force initiated the renewal of the Interactive v1 workload. The Interactive v2 workload adopted several features from the BI workload: support larger scale factors and delete operations, and coverage of the *cheapest path-finding* algorithm. Moreover, the updated workload introduces a new temporal parameter curation algorithm to ensure stable runtimes for path queries

[13] https://ldbcouncil.org/benchmarks/snb-interactive/.
[14] https://ldbcouncil.org/benchmarks/snb-bi/.

evaluated on a fully dynamic graph, and features a complete refactoring of the driver along with several usability improvements. The balance of the read and insert queries are similar to Interactive v1 workload with only 0.2% delete operations added – motivated by the fact that deletes are rare compared to inserts in most real-life systems [2]. As of June 2023, the workload is in draft phase with four complete implementations already available (Table 3).

3.3 Semantic Publishing Benchmark (SPB)

The Semantic Publishing Benchmark (SPB) targets RDF database engines and is inspired by the BBC's Dynamic Semantic Publishing approach [28]. The scenario behind the benchmark considers a media that deals with a large volume of streaming content, namely articles and other "creative works". This content is enriched with metadata that describes it and links it to reference knowledge – taxonomies and databases that include relevant concepts, entities, and factual information. This metadata allows publishers to efficiently retrieve relevant content and increase engagement.

The main interactions with the repository are (i) updates, which add new creative works or alter the repository, and (ii) aggregation queries, which retrieve content. The engine should handle updates executed concurrently with a massive amount of aggregation queries. It must be able to deal with graph pattern matching, reasoning, geospatial constraints, and full-text search. The SPB includes ontologies, a data generator, query patterns, as well as test and benchmark drivers. The detailed specification of SPB version 2.0 is available at [27].

3.4 Graphalytics

The LDBC Graphalytics benchmark targets systems that evaluate graph algorithms on static graphs. While the scope of Graphalytics is similar to the influential GAP Benchmark Suite [9], Graphalytics has slightly different algorithms, more data sets (38 vs. 5), and enforces determinism in its algorithms.

Algorithms. The selection of graph algorithms for Graphalytics was motivated by input received during the TUC meetings (Sect. 2.4) and a literature survey of over 100 academic papers [23]. The benchmark includes the following six algorithms: breadth-first search (BFS), community detection using label propagation (CDLP), local clustering coefficient (LCC), PageRank (PR), single-source shortest paths (SSSP), and weakly connected components (WCC). Algorithms were adjusted if necessary such that they are *deterministic:* BFS returns the *distance* (level) for each node in the traversal (instead of picking one of the parent nodes), while the tie-breaking strategy employed in the CDLP algorithm always picks the smallest label (instead of picking randomly).

Data Set. Graphalytics uses untyped and unattributed graphs. The data set of the benchmark includes both directed and undirected graphs. Some graphs have edge weights, which are used exclusively by the SSSP algorithm.

Graphalytics Competition. Presently, it is not possible to commission audits for Graphalytics. Instead, LDBC organizes competitions with leaderboards, in the style of the Top500[15] and Graph500[16] competitions of the high-performance computing community. Solutions compete for both performance (execution time) and scalability. Implementers are also required to report system prices following the TPC Pricing Specification [42].

3.5 FinBench

The FinBench (short for "Financial Benchmark") is a benchmark, new in 2023, specifically designed to evaluate the performance of graph database systems in financial scenarios, such as anti-fraud and risk control, by employing financial data patterns and query patterns. This collective effort led by Ant Group adopts its rich experience in financial services, making it more applicable to graph users in the financial industry than previous LDBC benchmarks. Sharing some framework similarities with SNB, it distinguishes itself through differences in datasets and workloads.

Data Pattern. Financial graphs are distinguished from the social network in SNB by having hub vertices with higher degrees and allowing edge multiplicity. Financial graphs are asymmetric directed graphs causing more unbalanced load and less spatial locality. The maximum degrees of hub vertices are in the magnitude of thousands in the social network generated by the SNB data generator, while the degree of hub vertices in the financial graphs generated by the FinBench data generator[17] may scale up to millions in large data scales. The higher degree of hub vertices poses new challenges to the performance of systems. The edge multiplicity means multiple edges of the same type can exist between the same source vertex and destination vertex. This requires support in the system storage and provides optimization opportunities for filtering during traversal.

Transaction Workload. The *transaction workload* is the first workload included in the initial version of FinBench targeting OLTP data management systems, as the SNB Interactive workload [19] does. The key features of transaction workload include read-write query, time-window filtering, and recursive path filtering. The read-write query is a new query type, which wraps a complex query in a transaction. The write query to execute is determined by the result of the complex read. Time-window filtering is a pattern when financial businesses focus on the data, especially the edges, in a specific time range. Recursive path filtering is a pattern used by financial businesses to find fund traces in the graph. A fund trace may match the filter that the timestamp increases and the amount decreases of the edges sourcing from the origin vertex hop-by-hop. These typical features bring new challenges to the performance of systems. The transaction workload has 12 complex read, 6 simple read, 19 write, and 3 read-write queries.

[15] https://www.top500.org/.
[16] https://graph500.org/.
[17] https://github.com/ldbc/ldbc_finbench_datagen.

The initial version of FinBench was collaboratively developed by nine leading graph system vendors. It underwent cross-validation on three systems: TuGraph, Galaxybase, and UltipaGraph. For more detailed information, please refer to the repositories on GitHub: FinBench specification[18], FinBench driver[19], reference implementation of the FinBench transaction workload[20], FinBench ACID suite[21].

4 Benchmark Processes

4.1 Defining New LDBC Benchmarks

LDBC has a strict process for proposing new benchmarks. Based on our experience, the initial benchmark completion (phases 1 to 3) takes at least 2 years, followed by adoption and maintenance (phases 4 and 5), which may span 5+ years.

Phase 1: Benchmark Proposal. The benchmark proposer shall create a draft proposal. The benchmark must be motivated by real-world use cases and target a category of data processing systems that tackle some aspect of graph processing. The designers must reason why the benchmark is significantly different from existing LDBC benchmarks by identifying new performance challenges and formulating their choke points, providing data with unique characteristics (e.g. distribution, frequency/type of updates), etc.. The benchmark draft shall be presented the benchmark to the Members Policy Council to gather feedback.

Phase 2: Collaboration Setup. The proposer shall gather agreements from 2+ member companies who are willing to contribute to the benchmark specification and create reference implementations. They shall create a work charter for the benchmark task force (e.g. [24,39]), which includes the list the members interested in working on the benchmark. This shall be presented to the Members Policy Council, which votes *on the establishment of a new benchmark task force*.

Phase 3: Detailed Benchmark Design. The task force shall create the detailed benchmark specification, implement the data generator and the benchmark driver. The creation of at least 2+ complete reference implementations is required. The task force shall ensure that the specification contains the description of the data set, queries, and workload as well as the detailed auditing guidelines. They shall publish the specification in an open repository and release the software components as open-source. The task force shall conduct 2+ *standard-establishing audits*. These include the complete execution of the benchmark and the creation of FDR that detail their outcomes. The task force shall submit all resulting documents to the Members Policy Council, which votes *on the acceptance of the benchmark*.

[18] https://github.com/ldbc/ldbc_finbench_docs.
[19] https://github.com/ldbc/ldbc_finbench_driver.
[20] https://github.com/ldbc/ldbc_finbench_transaction_impls.
[21] https://github.com/ldbc/ldbc_finbench_acid.

Phase 4: Adoption and Auditing. The task force shall help adoption attempts by working closely with the benchmark's early users. They should create training material and exam questions for auditor exams, then train and certify auditors. Certified auditors can fulfill incoming audit requests, initially with close collaboration with the benchmark task force.

Phase 5: Maintenance and Renewal. The task force shall maintain the benchmark and optionally assist in further adoption and auditing attempts. If the benchmark remains popular, the task force should consider renewing it after 5–10 years to ensure its continued relevance.

4.2 Auditing Process

Phase 1: Preparation (1–2 Weeks). The Test Sponsor shall be an LDBC member company. If the Test Sponsor is not an LDBC sponsor member company, it shall pay LDBC an auditing fee of 2,000 GBP (as of 2023). The Test Sponsor shall create an initial version of the supplementary package of the benchmark and send it to an LDBC-certified Auditor for a preliminary review. The Test Sponsor shall establish the costs of its system setup using the TPC Pricing Specification [42].

Phase 2: Audit Setup (3–6 Weeks). The Test Sponsor and the Auditor shall negotiate the timeline and pricing of the audit, and sign a contract. The Test Sponsor shall hand over the supplementary package to the Auditor. Continuous communication in the form of emails, online meetings, DMs, etc.. between the Auditor and the Test Sponsor is recommended for status updates and clarifications.

Phase 3: Auditing (3–10 Weeks). For the SNB workloads, an audit consists of the following steps: (1) set up system, (2) run cross-validation, (3) perform code review, (4) run ACID isolation tests, (5) perform recovery tests, (6) conduct benchmark runs on the scale factors requested by the Test Sponsor, (7) write the FDR and the executive summary. The Test Sponsor then reviews the FDR and executive summary documents. If everything is in order, the Auditor, the Test Sponsor, and the leader of the task force sign the FDR.

Phase 4: Dissemination of Results (1–2 Weeks). The audit results are announced on the LDBC website, on mailing lists and on social media.

Timespan. The overall time required for an audit is between 8 and 20 weeks.

4.3 Trademark

To prevent misuse of the benchmarks, the term "LDBC benchmark result" is trademarked and is only allowed to be used for results that were achieved by an LDBC-certified Auditor in an official audit. That said, LDBC encourages use (including derived use) of its benchmarks provided that users comply with the *fair use policies*, described in LDBC's Byelaws [30].

5 Benchmarking Lessons Learnt

This section captures our key lessons learnt with designing and maintaining benchmarks, and conducting audits with them.

High Development Costs. We found that creating domain-specific application-level benchmarks is a big undertaking. Realistic benchmarks are bound to be complex as they require a (somewhat) realistic scalable data generator and a high-performance driver with reference implementations. To make matters more complicated, the graph domain has highly skewed and correlated data sets, causing queries to be sensitive to parameter selection [22]. Path-finding queries on fully dynamic graphs are particularly susceptible to this, necessitating expensive parameter generation steps [41]. Creating reference implementations is also labour-intensive due to the lack of a standard query language – fortunately, this is expected to improve with the introduction of SQL/PGQ and GQL.

Data Set Availability is Important. We found that users prefer downloading pre-generated data sets from an official repository instead of generating them using the data generators. To help adoption, it is best to make the data sets available in multiple serialization formats (different layouts, datetime formats, etc..) for all scale factors. However, this requires tens of terabytes of storage, leading to high storage costs. Moreover, we transferring large data sets stored at public cloud providers can be prohibitively expensive due to high egress fees (i.e. fees paid for transferring data out of the cloud). To work around this problem, we use storage services which do not have egress fees. For long-term archiving, we store our data in the SURF Data Repository,[22] which is operated by the Dutch national HPC support center, and offers tape-based storage. For short-term data distribution, we use Cloudflare's R2 service.

Shift to the Cloud. We observed a shift to the cloud for database management [1]: approximately half of LDBC's graph vendors have a cloud offering and some have cloud-native systems with no on-premise solutions. The use of the cloud is also popular for running audits (and is encouraged by LDBC for easier reproducibility): 35 out of 49 audits conducted so far were executed in the cloud.

Finding Problems Beyond Peformance. While the main goal of a benchmark implementation is to measure performance and to identify bottlenecks, implementing a full workload often leads to the discovery of other issues. Namely, we have found several issues such as insufficient query language features, correctness bugs, concurrency issues on different CPU architectures, crashes on large data sets, durability errors, parameter handling errors, issues with datetime and string handling, and deadlocks caused by concurrent transactions. The availability of public benchmark data sets made these errors easy to reproduce and reason about, leading to significant improvements in the systems-under-test.

[22] https://github.com/ldbc/data-sets-surf-repository.

6 Working Groups

LDBC's working groups conduct research on areas related to graph query languages, including formalization of (sub)languages and exploring possibilities for defining graph schemas.

6.1 Graph Query Languages Working Group

The *Graph Query Languages* working group created the G-CORE design language [5], which treats paths as first-class citizens and supports the composability of graph queries. While the working group ceased to exist after the publication of G-CORE, LDBC has a liaison with ISO (Sect. 2.3) that facilitates continued collaboration on query language standards.

6.2 Formal Semantics Working Group (FSWG)

The *Formal Semantics Working Group* (FSWG) gives formal treatment to standard graph query languages to prevent ambiguous interpretations. To this end, it formalized the Graph Pattern Matching (GPM) language of GQL and SQL/PGQ [18]. The group also produced a formal summary of the GQL language [31] and created a pattern calculus for property graphs [20] that serves as a theoretical basis of GPM.

6.3 Property Graph Schema Working Group (PGSWG)

Initial graph database systems were schemaless, which hindered their adoption in several enterprise domains. The *Property Graph Schema Working Group* (PGSWG) investigates the problem of defining schemas for the property graph data model [8]. The group also identified ways to define keys in property graphs [7].

6.4 LDBC Extended GQL Schema (LEX)

The *LDBC Extended GQL Schema* (LEX) working group was established in 2022 with an initial membership of 10 organizations and approximately 20 individuals [21]. The group aims to propose the addition of a future extended schema definition language to the GQL standard, which allows for more elaborate (and therefore more restrictive) constraints on the permitted values of GQL property graphs than can be imposed by graph types as defined in the GQL DIS. These additional constraints aim to establish parity with the constraints available in SQL schema, to incorporate features described in PG-Schema [8] and to support performant processing of incremental transactional updates of a graph database.

7 Conclusion and Future Outlook

In this paper, we summarized LDBC's history, organization structure, community management; as well as its benchmarks, working groups, and processes. At the time of writing (June 2023), LDBC has a healthy benchmark ecosystem, which is actively maintained and renewed. Our benchmarks are used by a number of database vendors for both internal benchmarking as well as impartial comparisons via audits, which are now performed routinely.

In the future, we plan to further improve our benchmarks, including support for larger scale factors (up to SF100,000 for SNB). We also plan to investigate the impact of incorporating long-running transactions in our transactional workloads. Finally, we are interested in creating benchmarks for new areas of graph data management such as streaming and machine learning on graphs.

References

1. Abadi, D., et al.: The Seattle report on database research. ACM Commun. (2022)
2. Almuhimedi, H., et al.: Tweets are forever: a large-scale quantitative analysis of deleted tweets. In: CSCW (2013)
3. Angles, R., et al.: The linked data benchmark council: a graph and RDF industry benchmarking effort. SIGMOD Rec. (2014)
4. Angles, R., et al.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**, 1–40 (2017)
5. Angles, R., et al. G-CORE: a core for future graph query languages. In: SIGMOD (2018)
6. R. Angles et al. The LDBC Social Network Benchmark. CoRR arxiv:2001.02299 (2020)
7. Angles, R., et al.: PG-Keys: keys for property graphs. In: SIGMOD (2021)
8. Angles, R., et al.: PG-schema: schemas for property graphs. In: SIGMOD. ACM (2023)
9. Beamer, S., et al.: The GAP benchmark suite. CoRR arxiv:1508.03619 (2015)
10. Besta, M. et al.: Demystifying graph databases: analysis and taxonomy of data organization, system designs, and graph queries. CoRR arxiv:1910.09017 (2019)
11. Besta, M., et al.: Graph processing on FPGAs: taxonomy, survey, challenges. CoRR arxiv:1903.06697 (2019)
12. Besta, M., et al.: Practice of streaming and dynamic graphs: concepts, models, systems, and parallelism. CoRR arxiv:1912.12740 (2019)
13. Besta, M., et al.: GraphMineSuite: enabling high-performance and programmable graph mining algorithms with set algebra. Proc. VLDB Endow. (2021)
14. Boncz, P.A., et al.: The linked data benchmark council project. Datenbank-Spektrum (2013)
15. Boncz, P.A., et al.: TPC-H analyzed: hidden messages and lessons learned from an influential benchmark. In: TPCTC (2013)
16. Bonifati, A., et al.: Graph generators: state of the art and open challenges. ACM Comput. Surv. **53**, 1–30 (2020)
17. Checconi, F., et al.: Breaking the speed and scalability barriers for graph exploration on distributed-memory machines. In: SC (2012)

18. Deutsch, A., et al.: Graph pattern matching in GQL and SQL/PGQ. In: SIGMOD (2022)
19. Erling, O., et al.: The LDBC social network benchmark: interactive workload. In: SIGMOD (2015)
20. Francis, N., et al.: GPC: a pattern calculus for property graphs. In: PODS (2023)
21. Green, A.: LDBC extended GQL schema (LEX) work charter. Work Charter WC-2022-02, LDBC (2022). https://doi.org/10.54285/ldbc.VSBC2149
22. Gubichev, A., Boncz, P.A.: Parameter curation for benchmark queries. In: TPCTC (2014)
23. Guo, Y., et al.: How well do graph-processing platforms perform? an empirical performance evaluation and analysis (2014)
24. Guo, Z.: Work Charter for FinBench v1.0. Technical report, LDBC (2022)
25. Iosup, A., et al.: LDBC graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. In: PVLDB (2016)
26. Kalavri, V., et al.: High-level programming abstractions for distributed graph processing. IEEE Trans. Knowl. Data Eng. **30**, 305–324 (2018)
27. Kotsev, V. et al.: LDBC Semantic Publishing Benchmark (SPB) - v2.0. Benchmark specification, LDBC (2014). https://ldbcouncil.org/benchmarks/spb/ldbc-spb-v2.0-specification.pdf
28. Kotsev, V., et al.: Benchmarking RDF query engines: the LDBC semantic publishing benchmark. In: BLINK at ISWC (2016)
29. LDBC. Articles of Association (2023). https://ldbcouncil.org/docs/LDBC.Articles.of.Association.ADOPTED.2023-03-30.pdf
30. LDBC. Byelaws of the Linked Data Benchmark Council v1.4 (2023). https://ldbcouncil.org/docs/LDBC.Byelaws.1.4.ADOPTED.2023-05-02.pdf
31. Libkin, L., et al.: A researcher's digest of GQL (invited talk). In: ICDT (2023)
32. Mhedhbi, A., et al.: LSQB: a large-scale subgraph query benchmark. In: GRADES-NDA at SIGMOD (2021)
33. Pham, M., et al.: S3G2: a scalable structure-correlated social graph generator. In: TPCTC (2012)
34. Poess, M.: New initiatives in the TPC. In: TPCTC (2022)
35. Sahu, S., et al.: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. VLDB J. (2020)
36. Sakr, S., et al.: The future is big graphs: a community view on graph processing systems. ACM Commun. (2021)
37. Shao, B., et al.: Trinity graph engine and its applications. IEEE Data Eng. Bull. **40**, 18–29 (2017)
38. Shi, X., et al.: Graph processing on GPUs: a survey. ACM Comput. Surv. (2018)
39. Szárnyas, G.: LDBC Social Network Benchmark task force work charter. Technical report, LDBC (2023)
40. x Szárnyas, X., et al.: An early look at the LDBC Social Network Benchmark's Business Intelligence workload. In: GRADES-NDA (2018)
41. Szárnyas, G., et al.: The LDBC social network benchmark: business intelligence workload. PVLDB (2022)
42. TPC. TPC Pricing Specification, revision 2.8.0 (2022). http://www.tpc.org/tpc_documents_current_versions/pdf/pricing_v2.8.0.pdf
43. Waudby, J., et al.: Supporting dynamic graphs and temporal entity deletions in the LDBC Social Network Benchmark's data generator. In: GRADES-NDA (2020)
44. Zhao, K., Yu, J.X.: Graph processing in RDBMSs. IEEE Data Eng. Bull. (2017)

The LDBC Social Network Benchmark Interactive Workload v2: A Transactional Graph Query Benchmark with Deep Delete Operations

David Püroja[1(✉)], Jack Waudby[2], Peter Boncz[1], and Gábor Szárnyas[1]

[1] CWI, Amsterdam, The Netherlands
{david.puroja,gabor.szarnyas}@ldbcouncil.org, boncz@cwi.nl
[2] Newcastle University, School of Computing, Newcastle upon Tyne, England
j.waudby2@newcastle.ac.uk

Abstract. The LDBC Social Network Benchmark's Interactive workload captures an OLTP scenario operating on a correlated social network graph. It consists of complex graph queries executed concurrently with a stream of updates operation. Since its initial release in 2015, the Interactive workload has become the de facto industry standard for benchmarking transactional graph data management systems. As graph systems have matured and the community's understanding of graph processing features has evolved, we initiated the renewal of this benchmark. This paper describes the draft Interactive v2 workload with several new features: delete operations, a cheapest path-finding query, support for larger data sets, and a novel temporal parameter curation algorithm that ensures stable runtimes for path queries.

1 Introduction

LDBC. The Linked Data Benchmark Council (LDBC)[1] is a non-profit organization dedicated to designing benchmarks for graph data management [20,21]. LDBC has strong industrial participation in the form of 21 companies, including database, hardware, and cloud vendors. Its membership also includes 3 non-commercial institutions and 60+ individual members. LDBC acts as an independent authority for benchmarks and oversees the use of its benchmarks through a stringent auditing process. Thanks to this, audited LDBC benchmark results allow quantitative and objective comparison of different technological solutions, which is expected to stimulate progress through competition. Next, we describe the two main workloads of the LDBC SNB suite.

SNB Interactive v1 Workload. The LDBC Social Network Benchmark (SNB) Interactive v1 workload was published in 2015 [7]. It is a transactional benchmark that targets OLTP data management systems with graph features

[1] https://ldbcouncil.org.

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 107–123, 2024.
https://doi.org/10.1007/978-3-031-68031-1_8

(e.g. path-finding). SNB Interactive has been influential in the graph data management space: as of August 2023, 24 audited results were published using this workload.[2]

SNB Business Intelligence Workload. The LDBC SNB Business Intelligence (BI) workload was released in 2022 [24]. This workload uses an improved data generator, which introduces support for delete operations [28] and scale factors up to SF30,000. The workload captures an OLAP scenario with heavy-hitting analytical queries that touch on large portions of the graph (e.g. Messages created within a 100-day period or Persons living in China) and applies daily batches of updates. It targets both DBMSs and data analytical systems such as Spark.

Motivation. As of 2023, more than 8 years passed since the SNB Interactive v1 workload's release. Therefore, we decided to renew it to ensure its continued relevance. The new version's key novel features are improved scalability, coverage of cheapest path-finding, and inclusion of delete operations. The first two new features were part of the natural evolution of the benchmark. The decision to support delete operations was motivated by a number of factors. On the technic side, running the workload's complex queries efficiently while applying delete operations assumes mature transaction support that is now expected by users of graph(-capable) DBMSs. Deletes also make certain graph algorithms, such as cheapest paths, more difficult to compute incrementally [19], thus limiting the effects of caching and incremental view maintenance. On the business side, supporting deletes is necessitated by law in several jurisdictions, exemplified by the EU's General Data Protection Regulation (GDPR) [22].

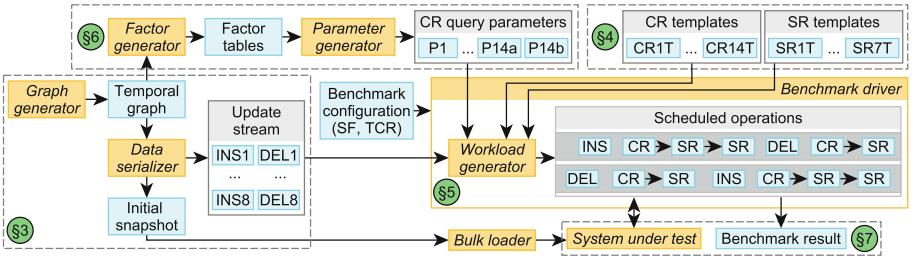


Fig. 1. Components and workflow of the Interactive v2 workload. The corresponding sections are shown in green circles (§). Legend: *Software component* | Data artifact

Contributions and Paper Structure. This paper presents the updated SNB Interactive v2 workload following the workflow shown in Fig. 1. Interestingly, while all three new features (scalability, cheapest path-finding, and delete

[2] https://ldbcouncil.org/benchmarks/snb-interactive/.

operations) were already supported in the BI workload, adopting them into the highly transactional, concurrent SNB Interactive v2 workload presented several complex technical challenges. We document the challenges and our key design principles in Sect. 2. We present the SNB data set in Sect. 3, the benchmark's operations in Sect. 4, and the workload in Sect. 5. In Sect. 6, we introduce the parameter generator's novel extensions that were necessitated by delete operations. In Sect. 7, we discuss how the benchmark is used in practice. We discuss LDBC's other transactional benchmark, the FinBench, in Sect. 8. Section 9 summarizes our contributions and outlines future directions.

2 Design Principles

During LDBC's benchmark design process, we follow the Benchmark Handbook [9], which prescribes four criteria for domain-specific benchmarks: (1) *relevance*,(2) *portability*,(3) *scalability*,(4) *simplicity*.(5) In the following, we discuss the SNB Interactive v2 workload's approach for complying with these criteria.

2.1 Relevance: Choke Point-Based Design Process and Domain

§

Choke Points. To ensure *relevance*, LDBC's benchmark design process uses *choke points* [4], i.e. technical difficulties that are known to be challenging for the present generation of DBMSs. Choke points are identified by expert data systems architects and are also influenced by the input from users of graph data management systems who contribute their use cases at LDBC's Technical User Community meetings[3]. The initial choke points of SNB Interactive were based on the influential TPC-H benchmark [25] benchmark and were later extended with choke points that target graph-specific features such as cardinality estimation for paths and the execution of path-finding queries. LDBC workloads are designed using an iterative process to ensure full coverage of the choke points required for a given workload category.

Social Network Domain. The LDBC SNB uses the *social network* domain because its concepts (Person, Forum, Message, etc..) are well-understood. Moreover, the social network domain makes it easy to reason about some of the interesting phenomena captured in the choke points. For example, the power law distribution and correlations (Sect. 3.2) observable in real-life (social) networks trigger the challenges for cardinality estimation.

2.2 Portability: Implementation Rules

Software component

The SNB Interactive v2 workload guarantees *portability* by taking an agnostic stance on implementation details.

[3] https://ldbcouncil.org/tags/tuc-meeting/.

Data Model. Implementations are allowed to use any data model, including the property graph, RDF, and relational models. They are also free to choose their input format for bulk loading (e.g. CSV, N-Triples).
Implementation Language. Implementations may use declarative query languages (SQL, Cypher [8], GQL, SQL/PGQ [6], etc..) or general-purpose imperative programming languages (C++, Java, etc..). However, results in these two categories are ranked on separate leaderboards as the latter systems have a significant advantage due to their use of hand-coded highly-optimized query plans.
Setup. There are no restrictions on the operating system, hardware architecture, or number of machines used (both single-node and distributed setups are allowed).

2.3 Scalability: Scalable Data Generator and Driver

Data artifact

Improving *scalability* was a key goal during the design of SNB Interactive v2. While we could leverage the improved data generator of the BI workload [24], scaling the *workload execution* posed additional challenges. By its nature, simulating a transactional database workload requires highly concurrent execution of the operations.[4] This requires the operations in the workload to be partitioned, which is a major challenge as most of the Persons in the social network belong to a single connected component that does not lend itself to any naïve partitioning strategy. Moreover, update operations often have long dependency chains that need to be tracked, e.g. a friendship can only be deleted if it already exists, the creation of a friendship requires both Persons to exist, etc.. Therefore, simulating a transactional graph processing scenario is not possible using on-the-fly workload generation techniques commonly employed in database benchmarks. Instead, SNB Interactive v2 requires extensive offline data, update stream (Sect. 3.3), and parameter generation steps (Sect. 6) prior to the benchmark.

2.4 Simplicity: Stable Query Runtimes, Single Output Metric

> The benchmark must measure the peak performance of systems when performing typical operations within the target problem domain.

Stable Runtimes. To make the benchmark results easy to interpret, it is desirable that instances of a given query type have similar expected runtimes (referred to as *stable runtimes*). Ensuring this for graph workloads is non-trivial due to the highly skewed distribution exhibited in real-world networks [16]. For example, in a social network, a few Person nodes have a very large number of edges while others only have a few connections. This has a significant impact on runtimes: if query parameters are selected using uniform random sampling, query runtimes will be unstable, often exhibiting a multimodal distribution that spreads

[4] Most audited Interactive v1 implementations use 48 read and 32/64 write threads.

across many orders of magnitude and has several outliers [12,24]. SNB Interactive v2 employs a sophisticated *parameter curation* process to select input parameters that ensure stable runtimes (Sect. 6).

Guaranteed Executability. Stable runtimes also necessitate that operations are *executable* at their scheduled start time. For example, if an operation targets entities that do not yet exist or were already deleted, the operation becomes trivial or results in a runtime exception, compromising stable runtimes. Therefore, our workload generator ensures that its operations are always executable.

Single Metric. The result of an SNB Interactive benchmark run is characterized by a single metric, *throughput* (operations/second), which captures the system-under-test's end-to-end performance on a transactional graph workload.

3 Data Sets

The LDBC SNB workloads include a scalable distributed data generator based on Spark.[5] Here, we give an overview of the data sets used in the benchmark.

3.1 Graph Schema

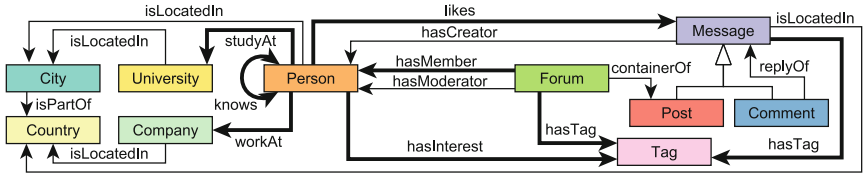


Fig. 2. A subset of the LDBC SNB graph schema visualized using a UML-like notation. Thick lines denote many-to-many relationships.

The graph schema of LDBC SNB has 14 node types connected by 20 edge types. The data set consists of a Person–knows–Person (friendship) graph and a number of Message threads within Forums. The root of a Message thread is a Post and the rest of the thread consists of Comments. All Messages are connected to Persons by creatorship and likes edges. A simplified schema is shown in Fig. 2.

3.2 Distribution and Correlations

The data set contains two types of graph-shaped data structures. First, the Message threads form trees and constitute the majority of the data. Second, the Person–knows–Person subgraph is a network with many-to-many relationships

[5] https://github.com/ldbc/ldbc_snb_datagen_spark.

whose distribution is modelled after Facebook [26] with the social graph exhibiting the small-world phenomenon [27] characterized by a small diameter.

A unique feature not observed in other data generators is that the attribute distributions are skewed and correlate both within an entity (e.g. people living in France have predominantly French names). Moreover, the graph has structural correlations: following the *homophily principle* [15], people are more likely to be friends if they studied at the same University at the same time, live in close proximity, and/or have the same interests. These correlations are exploited by the workload to stress choke points for querying correlated data (see Sect. 6.4).

3.3 Graph Generation Stages

Temporal Graph. The data generator first produces a *temporal graph*, which contains all entities that exist at some point in the simulated social network's 3-year time period, i.e. between Jan 1, 2010 and Dec 31, 2012. During this time, entities are inserted and deleted in the network, and the timestamps of these events are captured according to their time of occurrence in the *simulation time*. The insertion and deletion of entities follow realistic time intervals and conform to the semantics of the social network. Namely: (1) The deletion dates of Persons are based on the statistics collected from the collapse of a real-world social network [13]. When a Person is deleted, the content they created is also deleted [29]. (2) The network contains infrequent flashmob events such as spikes in insertions of Messages for a given Tag [7]. Deep delete operations and flashmob events are unique to the LDBC SNB data generator: according to a recent survey [5], these features are not supported by any other (graph) data generator.

Initial Snapshot and Update Stream. As the second step in the data generation, the *data serializer* splits the temporal graph into two parts by setting a *cutoff date* at 97% of the simulation time (Nov 29, 2012). The entities created before the cutoff date form the *initial snapshot*, while the entity creations and deletions occurring after the cutoff date form the *update stream*.

3.4 Scale Factors

Table 1. SNB Interactive v2 data sets. *k:* thousand, *M:* million, *B:* billion.

Scale Factor (SF)	10	30	100	300	1,000	3,000	10,000	30,000
#nodes	27M	78M	255M	738M	2.4B	7.2B	23B	82.76B
#edges	170M	506M	1.7B	5.1B	17B	51.9B	173B	340.5B
#Person nodes	68k	170k	473k	1.2M	3.4M	9M	26M	77M
#knows edges	1.8M	5.5M	19M	55.7M	187M	559M	1.9B	6.8B
#insert operations	44.6M	127M	399M	1.1B	3.3B	8.9B	27B	76.7B
#delete operations	353k	1M	3.3M	9.3M	28.9M	79.7M	245M	721.8M

The data generator produces dynamic social network graphs in different sizes, characterized by *scale factors* (SF) which correspond to the data set's disk usage when serialized in CSV (comma-separated values) format, measured in GiB. The data generator used for Interactive v1 only supports data sets up to SF1,000. To improve scalability, Interactive v2 uses the new Spark-based generator, which was optimized extensively,[6] allowing it to scale up to SF30,000. Table 1 shows the main statistics of the data sets.

4 Operations

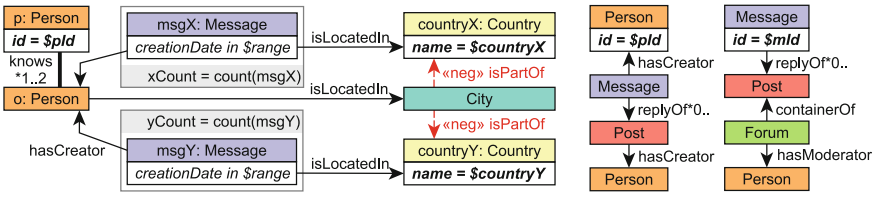
The LDBC SNB Interactive v2 workload uses four types of operations. There are 14 complex (CR) and 7 short read queries (SR). Update operations include 8 inserts (INS) and, newly introduced in the Interactive v2 workload, 8 deletes (DEL). The workload mix consists of approximately 8% CR, 72% SR, 20% INS, and 0.2% DELoperations. In this section, we describe the four operation types using examples. We also give ranges on how long operations are expected to take in state-of-the-art systems (Sect. 7.2).

4.1 Complex Read Queries (CR)

Complex read queries CR1–CR12 discover a given Person's social environment (one- to three-hop neighbourhoods) and retrieve related content (Forums, Messages, etc..). Queries CR13 and CR14 perform path-finding between pairs of Persons. The runtimes of complex read queries are typically between 1 and 500 ms, making them feasible to compute interactively, in line with the workload's name.

CR3. For a given Person, find their *friends* and *friends of friends*, who created Messages in both Country $countryX and $countryY within a given time period. Only consider Persons that are foreign to both of those Countries. Return the number of their Messages per Country, xCount and yCount (Fig. 3a).

CR13. Return the length of the (unweighted) shortest path between two Persons.



(a) Complex read query CR3. (b) SR2. (c) SR6.

Fig. 3. Graph patterns of complex and short read queries.

Cheapest Path-Finding. While we strived to keep the changes to the queries minimal, we replaced CR14 due to two reasons. First, we found the original query

[6] For details on the optimization steps, see https://ldbcouncil.org/tags/datagen/.

in Interactive v1 to be ill-suited to the workload as it required the enumeration of *all shortest paths* between two Persons, which can be prohibitively expensive on large scale factors. Second, we introduced a new choke point, CP-7.6 *Cheapest path-finding,*[7] a key computational kernel and a language opportunity for GQL [6]. Therefore, we changed `CR14` to use *cheapest paths* instead of *all shortest paths*.

`CR14 (new).` Given two Persons, find *any cheapest path* in the interaction subgraph. This graph contains edges from the Person–knows–Person graph where the endpoint Persons have exchanged at least one Message (i.e. one Person created a direct Comment to a Message of the other Person). The weights of knows edges are integers defined as $\max(\text{round}(40 - \sqrt{numInteractions}), 1)$.

4.2 Short Read Queries (`SR`)

Short read queries perform local neighbourhood lookups on Persons and Messages. Most short read queries can be evaluated in 0.1 to 75 ms.

`SR2.` Given a start Person, retrieve their last 10 Messages. For each Message, return it with the root Post in its thread, and the author of that Post (Fig. 3b).

`SR6.` Given a Message, retrieve its container Forum (directly for Posts, via the root Post for Comments) and the Person that moderates that Forum (Fig. 3c).

4.3 Insert Operations (`INS`)

Insert operations add new entities from the update stream to the graph. A typical insert operation takes between 0.1 and 100 ms.

`INS5.` Insert a hasMember edge between a Person and a Forum. The executability of this operation depends on the existence of its two endpoint nodes.

`INS6.` Insert a Post node. This operation's executability depends on two nodes: both the Person creating the Post and the Forum containing it must exist. When the Post is inserted, they are linked to it via hasCreator and containerOf edges.

4.4 Delete Operations (`DEL`)

The Interactive v2 workload uses *deep cascading delete operations*. Cascading deletes capture the behaviour of real social networks where users expect their content to be removed once they delete their accounts. The technical reasons for requiring cascading delete operations are two-fold: (1) **Preventing dangling edges.** To maintain the integrity of the graph, it is required there are no dangling edges thus nodes must be always deleted with all their edges. To prevent dangling edges, most graph DBMSs support the automatic deletion of edges attached to a given node, e.g. through Cypher's `DETACH DELETE` clause [11]. To achieve the same effect, RDBMSs can make use of `FOREIGN KEY` constraints with

[7] The term *shortest paths* refers to the problem of finding *unweighted shortest paths*, which can be solved with the BFS algorithm. We use *cheapest paths* to refer to the *weighted shortest paths* problem which can be solved using e.g. Dijkstra's algorithm.

the `ON DELETE CASCADE` clause. (2) **Testing triggered deletions.** Node deletions can trigger the deletion of other nodes (Fig. 4). For example, according to the SNB schema's constraints, the deletion of a Post implies the deletion of all its descendant Comments along with their edges. Such deletions may be implemented using triggers, constraints (RDBMSs may again harness `FOREIGN KEY`s), or by formulating (potentially recursive) subqueries that determine which other nodes need to be deleted with `DELETE ... USING` clause.

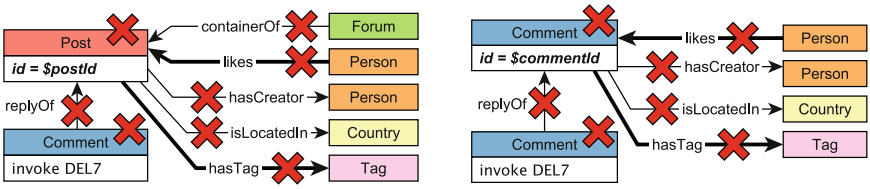
Choke Points. The coverage of delete features is ensured by three new choke points: CP-9.3 *Delete node* (stressed by 4 operations), CP-9.4 *Delete edge* (8 operations), and CP-9.5 *Delete recursively* (4 operations). In the following, we present two delete operations, both of which cover all new choke points.

`DEL6.` Remove a Post with all its edges and child Comments via `DEL7` (Fig. 4a).

`DEL7.` Remove a Comment with all its edges and child Comments, which are deleted recursively by invoking `DEL7` (Fig. 4b).

5 Workload Scheduling and Benchmark Driver

In this section, we explain how operations are scheduled in the SNB Interactive workload, how the driver operates, and how the final *throughput* metric is determined. In all cases, we assume that the system-under-test has been populated with the *initial snapshot* using a *bulk loader* before the driver runs the operations.



(a) Delete operation `DEL6`: Remove Post. (b) Delete operation `DEL7`: Remove Comment.

Fig. 4. Cascading delete operations in Interactive v2. Symbol denotes deletion.

5.1 Scheduling Operations

TCR (total compression ratio). The scheduling follows the *simulation time* of the temporal social network graph. The user-provided *total compression ratio* (TCR) value controls the speed at which the simulation is replayed. For example, a TCR value of 0.02 means that the simulation is replayed 50× faster, i.e. for every 20 milliseconds in wall clock time, 1 s passes in the simulation time.

Update Operations. The driver replays the update operations starting from the cutoff date (Sect. 3.3), Nov 29, 2012. The operations are scheduled according to the distance of their start time from this date, adjusted by the TCR. They are then used to set the cadence of the schedule for the complex reads and, in turn, the short read queries, as we will explain momentarily.
Complex Read Queries. The *complex read queries* differ significantly in their expected runtimes as they touch on different amounts of data. As each query instance contributes equally to the output metric,[8] we balance them such that each query type is expected to take the same amount of time to execute. For example, `CR14 (new)` is expected to be more difficult than `CR13`, therefore it is scheduled less frequently. Frequencies vary based on the SF as the relative difficulties of queries change with the data size (e.g. three-hop neighbourhood queries grow faster on larger SFs than one-hop ones).
Short Read Queries. Short read queries are triggered by complex read queries and other short read queries, and use their output as their input. For example, both `CR3` and `CR14` trigger `SR2`, which also triggers itself. This mimics the real-life scenario of a user retrieving more information about Person profiles based on the result of the earlier queries. The mapping between complex and short read queries is given in the specification [2, Chapter 5].

5.2 Driver

Driver Modes. The SNB driver has two key modes of operation. In *cross-validation mode*, it tests an implementation against the output of another implementation. To ensure deterministic results, operations in this mode are executed sequentially with no overlap between queries and updates. In *benchmark mode* the driver performs a benchmark run where queries and updates are issued concurrently from multiple threads. The run starts with a 30-minute warm-up period, followed by a 2-hour *measurement window*. This mode does not perform validation as query results may differ (slightly) due to concurrent updates.
Dependency Tracking. To ensure that updates are executable, concurrent threads must be synchronized so that an operation is only executed when its dependencies exist in the network (e.g. two Persons can only become friends if both of them already exist). This is achieved via maintaining a global clock in the driver and performing *dependency tracking* for the updates [7]: each update operation has a timestamp denoting the creation time of the last operation it depends on. The data generator calculates these timestamp during generation and ensures that there is a minimum time separation, T_{safe}, between dependent entities to reduce synchronization overhead in the driver when executing operations. The driver then only needs to check every T_{safe} time whether a given update operation can be executed. By default, T_{safe} is set to 10 s in the simulation time.
Latency Requirements. The workload simulates a highly transactional scenario where operations are subject to (soft) latency requirements. To incorporate

[8] Unlike in TPC-H [25] and SNB BI [24], which use *geometric mean* in their metrics.

this in the workload, it prescribes the *95% on-time requirement*: for a benchmark run to be successful, 95% of the operations must start *on-time*, i.e. within 1 s of their scheduled start time. Benchmark runs where the system-under-test falls behind too much from the schedule are considered invalid.

Throughput. The throughput of a run is the total number of operations (`CR`, `SR`, `INS`, `DEL`) executed per second. A lower TCR value implies a higher throughput.

Individual Execution Times. To facilitate deeper analyis, the benchmark driver also collects all individual query execution times. Based on these, the benchmark reports must include statics for each operation type (min, max, mean, P_{50}, P_{90}, P_{95}, and P_{99} of the execution times).

Driver Implementation in v2. The Interactive v2 is implemented in Java 17. It consists of 26,500 lines of code for the core project and an additional 18,000 lines of test code. The new version contains several patches including bug fixes, usability improvements, and performance optimizations.[9]

6 Parameter Curation

To prevent caching query results, the SNB Interactive v2 driver instantiates the parameterized complex read (`CR`) query templates with different *substitution parameters* (a.k.a. parameter bindings). However, as explained in Sect. 2.4, the naïve approach (using a uniform random sampling of parameters and ignoring updates) leads to unstable runtimes, which compromise both the benchmark's understandability and reproducibility. To ensure stable runtimes, LDBC invented *parameter curation* techniques, which select parameters that produce query runtimes with a unimodal (preferably Gaussian) distribution [12,24].

6.1 Building Blocks for Parameter Curation

Temporal Bucketing. To ensure that operations are always executable, i.e. they avoid targeting nodes that are yet to be inserted or ones that are already deleted, the parameter curation process in Interactive v2 employs *temporal bucketing*. Namely, we create a parameter bucket for *each day in the simulation time of the update streams*, i.e. each day in the simulation time has its own distinct set of parameters. This is a novel feature in Interactive v2 – previous SNB benchmarks lacked this feature and only selected parameters from the *initial snapshot*.

Factor Tables. As shown in Fig. 1, the parameter generation is a two-step process. The *factor generator* produces *factor tables*, which contain data cube-like summary statistics [10] of the temporal graph such as the number of `Messages` for friends. The factor generator is executed in a distributed setup using Spark as this computation includes expensive joins over large tables, e.g. $\texttt{knows}(\texttt{person}, \texttt{friend}) \bowtie \texttt{hasCreator}(\texttt{person}, \texttt{comment})$.

[9] github.com/ldbc/ldbc_snb_interactive_driver/releases/tag/v2.0.0-RC2.

6.2 Parameter Curation for Relational Queries

For relational queries (without path-finding), we based our parameter generation on two techniques.

(1) Selecting windows. To select the parameters that are expected to yield similar runtimes, we look for windows with the smallest variance for a given value using SQL window functions. The parameters are first sorted and grouped together based on their difference in frequency. Groups that are smaller than a given minimum threshold are discarded to select a group of parameters large enough to generate a sufficient amount of parameters. From the latter, we select the group with the smallest standard deviation.

(2) Selecting distributions. For queries where we want to select parameters that are correlated or anti-correlated, we use factor tables encoding possible combinations (e.g. `countryPairsNumFriends` for `CR3`): we select values near a high percentile for the correlated and a low percentile for the anti-correlated case.

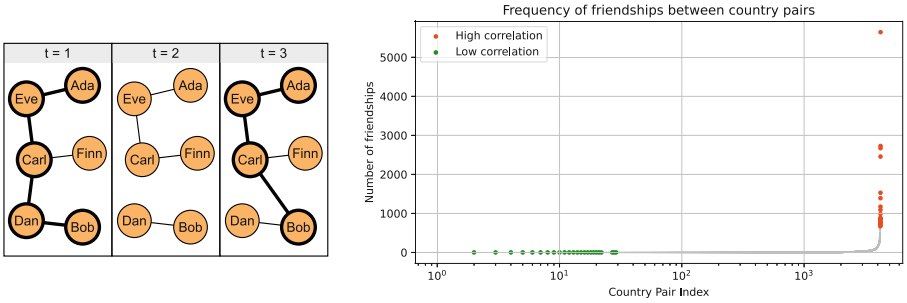
Generating the parameters. The parameter candidates discovered by the previous approaches are stored in temporary tables. The parameter generation step uses these tables to select parameters for each day in the update stream.

6.3 Parameter Curation for Path-Finding Queries

The Effect of Deletes. A key distinguishing feature of graph data management systems is their first-class support for path queries [1]. We demonstrate why ensuring stable query runtimes for path queries is particularly challenging through the example of Fig. 5a, where we query for the (unweighted) shortest path between *Ada* and *Bob* over a dynamic graph. Initially, at $t = 1$, the length of the shortest path is 4 hops. Then, the edge between *Carl* and *Dan* is deleted, making *Ada* and *Bob* unreachable from each other at $t = 2$. Finally, a new edge is inserted between *Carl* and *Bob*, yielding a shortest path of length 3 at $t = 3$. This illustrates how a given input parameter (a pair of Persons) can oscillate between being reachable and being in disjoint connected components over a short period. To ensure stable query runtimes for path queries in the presence of inserts and deletes, Interactive v2 introduces a novel *path curation* algorithm, which produces pairs of Person nodes whose shortest path length from each other is guaranteed to be exactly k hops at any point during a given day.

Graph Construction. The parameter curation algorithm builds two variants of the Person–knows–Person subgraph for each day based on the *temporal graph*: graph G_1 has the inserts applied until the beginning of the day and the deletes applied until the end of the day, while G_2 has the deletes applied until the beginning of the day and the inserts applied until the end of the day. For a given pair of Person nodes, their shortest path length in G_1 is an upper bound k_{upper} on their shortest path length at any point in the day – when the inserts during the day are gradually applied, the shortest path length can only become shorter. Conversely, G_2 gives a lower bound k_{lower} for the shortest path – the deletes can only make the shortest path length become longer.

Parameter Selection. The bounds provided by G_1 and G_2 guarantee for the shortest path length k that $k_{\text{lower}} \leq k \leq k_{\text{upper}}$ will hold at any point during the day. We can ensure that k will stay constant during the day by selecting Person pairs where $k_{\text{lower}} = k_{\text{upper}}$ holds. To this end, we select pairs who are exactly 4 hops apart in both G_1 and G_2, hence they will be always 4 hops apart during the given day. Unreachable pairs of nodes can be generated by calculating the connected components of G_2 and selecting nodes from disjoint components. The path curation for both the reachable and the unreachable cases is implemented using the NetworKit graph algorithm library [23].



(a) Shortest path (denoted with thick lines) between *Ada* and *Bob* in the presence of updates.

(b) Pairs of Countries in the `countryPairsNumFriends` factor table representing the number of friendships between both Countries.

Fig. 5. Example graph and distribution for path curation.

6.4 Query Variants

The new workload introduces variants for three queries: `CR3`, `CR13`, and `CR14`.

`CR3: Correlated vs. anti-correlated Countries.` We introduce variants for `CR3` (Fig. 5a): variant `CR3(a)` starts from Countries that have a high correlation in the friendship network, while variant `CR3(b)` starts from Countries that have a low correlation of friendships between. To generate these inputs, we use the `countryPairsNumFriends` factor table visualized in Fig. 5b and select values at percentile 1.00 for variant `(a)` and percentile 0.01 for variant `(b)`.

`CR13 and CR14: Reachable vs. unreachable Persons.` Path queries are expected to have different runtimes if there is a path vs. when there is no path. While the performance characteristics vary highly between systems, in principle, the "no path" case should be simpler in the SNB graph, where one of the nodes is always in a small connected component. To distinguish between these cases, we have two variants for the two path queries `CR13` and `CR14`. For variants `(a)`we select Person pairs which *do not have a path*, and for variants `(b)`we select pairs which *have* a path of length 4.

6.5 Parameter Generator Implementation

The parameter generator is implemented in Python using NetworKit [23] and SQL queries executed by DuckDB [18]. Based on our experiments in [17, Figure 4.3], the new parameter generator is scalable. Even with the significant extra work performed for temporal bucketing, it outperforms the old parameter generator by more than 100× on SF1,000, and finishes in less than 1.5 h on SF10,000.

7 Using the SNB Interactive v2 Workload

In Sects. 3 to 6, we presented the components that make up the SNB Interactive v2 benchmark (Fig. 1): its data sets, operations, driver, and parameter generator. We continue by describing how the benchmark is used, including its current implementations and considerations for auditing implementations.

7.1 Implementations

The portability of Interactive v2 is demonstrated by having four complete initial implementations[10] based on the Neo4j graph DBMS; and the Microsoft SQL Server, PostgreSQL, and Umbra RDBMSs. The Neo4j implementation uses the Cypher query language [8]. SQL Server uses the Transact-SQL language with the graph extension.[11] PostgreSQL and Umbra both use SQL's PostgreSQL dialect. All of these implementations passed cross-validation against each other.

7.2 Auditing

LDBC's benchmarks come with stringent auditing guidelines to ensure that they are implemented correctly and the results derived during benchmark runs are reproducible. Audits are carried out by independent auditors who are certified by the benchmark task force. The auditor ensures that an implementation is compliant with the LDBC specification by performing a thorough code review, running ACID tests, and executing the benchmark. The results of audits are published as *full disclosure reports* and systems are ranked on the LDBC website according to their throughput.[12] In the following, we highlight important aspects of the SNB auditing guidelines such as rules for precomputation and ACID tests. For the detailed auditing guidelines, we refer the reader to the SNB specification [2, Chapter 7].

Precomputation. The auditing guidelines permit the use of precomputed auxiliary data structures (views, indexes, views, etc..) provided that they are always kept up-to-date upon update operations. A frequent use of precomputations is

[10] https://github.com/ldbc/ldbc_snb_interactive_impls.

[11] https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview?view=sql-server-ver16.

[12] https://ldbcouncil.org/benchmarks/snb-interactive/.

the creation of a rootPost edge for each Message, which points to the root Post of the Message's thread. Implementers may decide to store information redundantly, e.g. by adding a property to the Forum–hasMember–Person edge that contains the number of Posts in the Forum, for improved locality during query execution.

ACID tests. To ensure thorough testing of transactional guarantees, SNB Interactive v2 uses a separate ACID suite [29], which tests for 10 transactional anomalies. While Interactive v1 only requires systems to guarantee the *read committed* isolation level, the inclusion of delete operations necessitates *snapshot isolation* to ensure queries read a consistent database state. To illustrate this consider a graph with four nodes n_1, n_2, n_3, n_4, and three edges $n_1 \to n_2 \to n_3 \to n_4$. Assume transaction T_a begins traversing from n_1, reading n_1, n_2, and n_3. Then, T_b deletes n_2 and commits. Then, T_c inserts n_5, connecting n_3 and n_4, $(n_3 \to n_5 \to n_4)$, and commits. T_a then reads n_5 and (incorrectly) concludes that n_4 is reachable from n_1 – when in fact at no point in time was this a valid database state. The ACID tests also include a *durability test:* during a benchmark run, the system-under-test is shut down abruptly and restarted afterward. The system is expected to guarantee durability, which is verified by the auditor who checks whether the last update operations issued by the driver are reflected in the database's state after recovery.

Full Disclosure Report (FDR). Audited benchmark results must be accompanied by a full disclosure report (FDR). The FDR documents the benchmark setup for reproducibility and contains the detailed results of the benchmark run (including statistics of the individual query runtimes).

8 Related Work: LDBC FinBench

The LDBC FinBench (Financial Benchmark) targets distributed scale-out transactional graph database management systems. It is set in the financial domain and uses concepts such as Account, Loan, and transfer. Its data distribution follows the characteristics of the financial domain, where a hub vertex (e.g. a large e-commerce vendor) may have billions of edges. To make queries tractable on such vertices, the workload employs *truncation*, i.e. a traversal only uses a truncated set of edges, e.g. the 5,000 most recent edges. A requirement directly derived from the financial domain is having *strict latency requirements* for some queries, e.g. 99% of a given query's executions have to finish in 100 ms. The workload also includes path-finding queries that can be expressed as *regular queries with memory* [14].

9 Conclusion

Summary. In this paper, we summarized the LDBC SNB Interactive v1 workload and explained its shortcoming to motivate its renewal. We then presented the draft version of the Interactive v2 workload, which is expected to be very close to the final version of the workload. The new workload uses a data generator producing deep cascading delete operations, includes a completely reworked

driver and workload scheduler, and a scalable parameter generator. We compared the workload against other benchmarks and highlighted its key novel features that allow the incorporation of delete operations while keeping important guarantees such as stable query runtimes. While the benchmark was substantially reworked, we made an effort to keep the user-facing changes minimal and only replaced a single read query, `CR14`. We believe users with an existing v1 implementation can adopt the new version with reasonable development cost and extend their experiments to use larger scale factors in a matter of days. Therefore, we expect users to quickly migrate to the new version upon its release.

Future Work. As the next step in the Interactive v2 workload's development process, the SNB task force will finalize the workload, conduct *standard-establishing audits* on two reference implementations and submit the workload for acceptance by the LDBC Members Policy Council. Audits are expected to commence in 2024. The task force will keep maintaining the workload in the coming years. In future versions of SNB Interactive, we plan to incorporate long-running transactions, schema constraints [3], and regular path queries [14].

Acknowledgements. We would like to thank our collaborators who contributed with feedback and implementations to the SNB Interactive v2 workload: Altan Birler, Arvind Shyamsundar, Benjamin A. Steer, and Dávid Szakállas. Jack Waudby was supported by the Engineering and Physical Sciences Research Council, Centre for Doctoral Training in Cloud Computing for Big Data [grant number EP/L015358/1].

References

1. Angles, R., et al.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1–68:40 (2017)
2. Angles, R., et al.: The LDBC Social Network Benchmark. CoRR, abs/2001.02299 (2020). http://arxiv.org/abs/2001.02299
3. Angles, R., et al. PG-Keys: Keys for property graphs. In: SIGMOD (2021)
4. Boncz, P.A., et al.: TPC-H analyzed: hidden messages and lessons learned from an influential benchmark. In: TPCTC (2013)
5. Bonifati, A., et al.: Graph generators: state of the art and open challenges. ACM Comput. Surv. **53**(2), 36:1–36:30 (2020)
6. Deutsch, A., et al.: Graph pattern matching in GQL and SQL/PGQ. In: SIGMOD (2022)
7. Erling, O., et al.: The LDBC social network benchmark: interactive workload. In: SIGMOD (2015)
8. Francis, N., et al.: Cypher: an evolving query language for property graphs. In: SIGMOD, pp. 1433–1445. ACM (2018)
9. Gray, J. (ed).: The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, 2nd edition (1993)
10. Gray, J., et al.: Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub totals. Data Min. Knowl. Discov. **1**(1), 29–53 (1997)
11. Green, A., et al.: Updating graph databases with Cypher. In: PVLDB (2019)
12. Gubichev, A., Boncz, P.A.: Parameter curation for benchmark queries. In: TPCTC (2014)

13. Lőrincz, L., Koltai, J., Győr, A.F., Takács, K.: Collapse of an online social network: burning social capital to create it? Soc. Networks **57**, 43–53 (2019)
14. Libkin, L., Vrgoc, D.: Regular path queries on graphs with data. In: ICDT (2012)
15. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: homophily in social networks. Ann. Rev. Sociol. **9**(1), 415–444 (2001)
16. Newman, M.E.J.: Power laws, Pareto distributions and Zipf's law. Contemp. Phys. **46**(5), 323–351 (2005)
17. Püroja, D.: LDBC Social Network Benchmark Interactive v2. Master's thesis, Universiteit van Amsterdam (2023). https://ldbcouncil.org/docs/papers/msc-thesis-david-puroja-snb-interactive-v2-2023.pdf
18. Raasveldt, M., Mühleisen, H.: DuckDB: an embeddable analytical database. In: SIGMOD (2019)
19. Roditty, L., Zwick, U.: On dynamic shortest paths problems. In: ESA, (2004)
20. Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., Özsu, M.T.: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. The VLDB J. **29**(2), 595–618 (2019). https://doi.org/10.1007/s00778-019-00548-x
21. Sakr, S., et al.: The Future is Big Graphs: A Community View on Graph Processing Systems. ACM, Commun (2021)
22. Shastri, S., et al.: Understanding and benchmarking the impact of GDPR on database systems. VLDB **13**(7), 1064–1077 (2020)
23. Staudt, C.L., Sazonovs, A., Meyerhenke, H.: NetworKit: a tool suite for large-scale complex network analysis. Netw. Sci. **4**(4), 508–530 (2016)
24. Szárnyas, G., et al.: The LDBC Social Network Benchmark: Business Intelligence workload. In: PVLDB (2022)
25. TPC. TPC Benchmark H, revision 2.18.0. pages 1–138, (2017). http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf
26. Ugander, J., Karrer, B., Backstrom, L., Marlow, C.: The anatomy of the Facebook social graph. CoRR abs/1111.4503 (2011). http://arxiv.org/abs/1111.4503
27. Watts, D.J., Strogatz, S.H.: Collective dynamics of "small-world" networks. Nature **393**, 440–442 (1998)
28. Waudby, J., et al.: Supporting dynamic graphs and temporal entity deletions in the LDBC Social Network Benchmark's data generator. In: GRADES-NDA (2020)
29. Waudby, J., et al.: Towards testing ACID compliance in the LDBC Social Network Benchmark. In: TPCTC (2020)

A Cloud-Native Adoption of Classical DBMS Performance Benchmarks and Tools

Patrick K. Erdelt(✉)

Berliner Hochschule fuer Technik (BHT), Luxemburger Strasse 10, 13353 Berlin, Germany
patrick.erdelt@bht-berlin.de

Abstract. Classical DBMS benchmarks cover a variety of use cases, for example: microbatch in-line insertion and highly concurrent row-level access (YCSB), batch offline loading into a data warehouse and concurrently running complex analytical queries (TPC-H) and business transactions (TPC-C). These use cases are still relevant in the cloud era, where we build data pipelines of microservices. In this paper we adopt the above benchmarks and four popular tools to the cloud-native pattern. On the one hand, this helps in assessing the performance of data pipelines that have a DBMS at their core. On the other hand, it makes benchmarking a scalable, elastic and observable process that can be automated. In a series of experiments, we (1) inspect Kubernetes jobs and benchmarking tools and whether they are suitable for combination, (2) monitor resource consumption of all components, i.e., also the drivers, (3) inspect scaling behaviour and look for peak performance points. We show that tools and workloads respond differently to scale-out and that the cloud-native pattern is fruitful for benchmarking.

Keywords: Database Management Systems · Performance Evaluation · YCSB · HammerDB · Benchbase · TPC-H · Benchmarking · Virtualization · Docker · Cloud-based Systems · Kubernetes · Microservices · Tools · Amazon Web Services

1 Introduction

There is an increasing significance of data pipelines in the cloud. The cloud-native pattern supports scaling out components and deploying them independently. This helps in combining, completing, scaling and observing components, that may be written in a mix of languages and by different teams of engineers. The modular nature also helps in speeding up development cycles. Testing in the cloud refers to not only having the system-under-test (SUT) in the cloud, but also the other components of the benchmarking process. In cloud-native benchmarking, the components are organized as microservices in containers and orchestrated for example by Kubernetes (K8s) [7,8,15,16,20] or Cloud Foundry

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 124–142, 2024.
https://doi.org/10.1007/978-3-031-68031-1_9

[28]. One main advantage of this pattern is to have reproducible and close-to-real-world testbeds. This also means relief in the tedious and complicated process of setup and it means easy repetition for statistical confidence and to isolate dependencies. Moreover it minimizes inbound and outbound traffic. Another advantage is flexibility. Last but not least this allows elastic scale-out of the components that represent parallel clients in a cost sensitive manner. In this paper, we regard the process of running the benchmarks YCSB, TPC-H and TPC-C as in Fig. 1. We pursue the question, if we can benefit from the cloud-native pattern in terms of making the process of benchmarking scalable, elastic and observable. We are also interested in finding out, if established tools fit in and if horizontal scaling (scale-out) and vertical scaling (scale-up) yield similar results.

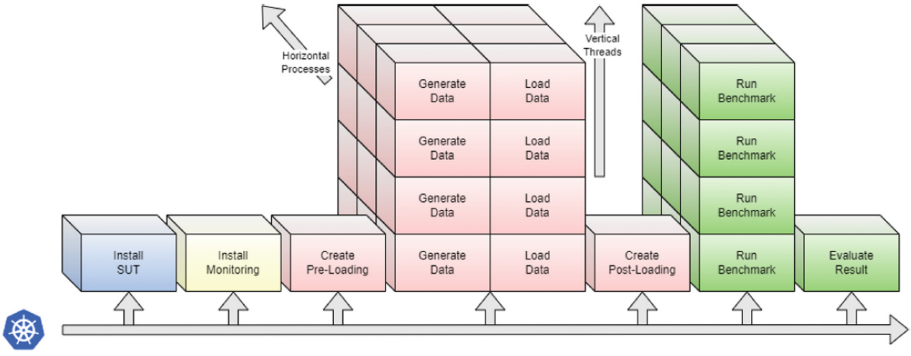


Fig. 1. Components of a Benchmarking Experiment

1.1 Contribution

We motivate research questions to design and evaluate a cloud-native implementation of classical DBMS performance benchmarks. We adopt YCSB, TPC-H and TPC-C (HammerDB and Benchbase version) to a cloud-native architecture. We therefore containerize benchmarking tools, deploy them as microservices to build data processing pipelines and orchestrate components with Kubernetes. This involves (synthetic) data generation, ingestion and processing. We focus on scaling properties, i.e., parallel loading and execution of workloads. We address throughput, latency and resources and report some rough numbers. We inspect scale-up vs. scale-out behaviour and scaling of drivers beyond single machines.

1.2 Related Work

Cloud-native systems in general are an active area of research. This also covers seeing those systems as objects of benchmarking. This in particular involves attributes that are specific to this setting like scalability, availability and fault

tolerance for example in chaos testing. In our approach however the subject of benchmarking (the framework) is cloud-native. This simulates a situation where the components that generate the workload are scaled-out (instead of a monolithic driver). This perspective has come more and more into focus in recent years. However, to our knowledge, no research has been done to compare scale-out to scale-up, about the influence of the drivers or about implementing benchmarks of the TPC.

In [7,8] we introduce *Bexhoma*, a Python package that helps to deploy components of a benchmarking experiment, that is the SUT, monitoring, an executor and an evaluator, to Kubernetes (K8s) and that serves as an orchestrator. In [8], we test 14 DBMS at 7 clouds to show this approach is broadly applicable. We only use standard K8s objects like services, deployments, jobs and labels. Python is more flexible and interactive than a declarative language like YAML. We use our Python package *DBMSBenchmarker* [9] to inspect details of the power test of TPC-H and to connect it to monitoring. We also show a data profiling benchmark on TPC-DS can be used to detect changes in the nodes of the cluster. In the experiments, execution comes from a single driver only and loading is not handled by drivers that push the data, but the DBMS pulls data from disk.

In [15] the authors introduce *Theodolite*, a framework that uses the Kubernetes operator pattern in order to produce reproducible and verifiable benchmarking results. The authors provide some implemetational details: It uses K8s, Prometheus, Apache Kafka and Helm on the technical side. The basic K8s objects are extended using customer resource definitions (CRD) for benchmarks and executions. This means the definitions and exections of benchmarks are given als YAML files for versioning and sharing, and the benchmarker can interact with these resources using the standard command line tool `kubectl`. In [14,16] the authors use this framework to benchmark and compare the scalability of the distributed stream processing engines Kafka and Flink in different workloads and use cases. They use Grafana as monitoring dashboard and Jupyter notebooks for evaluation. The experiments are conducted in a private cloud with fixed nodes. In [17] the authors stress the framework can be used to evaluate scalability (demand vs. capacity). The approach is shown to be fruitful for stream processing engines, not DBMS.

In [20] the authors introduce *Frisbee*. They use Telegraf, Prometheus and Grafana for monitoring and alerting. Frisbee has a declarative YAML-file-based language and CRD. The authors can define dependencies, events, actions and alerts and complex workflows as a DAG incorporating sequential and parallel executions. The authors run experiments with Redis and YCSB in two clouds of fixed shape, mainly in a cluster of four nodes. The demonstrations cover load tests (cascading mixed workloads), failover tests (network partition) and saturation tests (adding injectors).

In [19] the authors present *Kobe*. It consists of three subsystems: a deployment subsystem (K8s), a logging subsystem (EFK logging stack) and a network subsystem (Istio). Kobe has declarative YAML specifications and can handle sequential and parallel workloads. It aims at benchmarking federated query pro-

cessors. The authors report Kobe is extensible and can already be used with two database systems, Virtuoso and Strabon, and two federators, FedX and Semagrow. Two benchmarks are implemented, LUBM and FedBench. The concept deals with federated query processors, not DBMS. At the state of publication no experiments are included.

In [28] the authors present *ISABEL*. The authors aim at performing the complete workflow with a single command to simplify the process and achieve Benchmarking-as-a-Service in the end. In experiments they inspect MySQL using the benchmark tool Sysbench on three VMs. The framework is based on Cloud Foundry (not Kubernetes).

In a series of publications, the authors of [11,24–27] present the framework *Mowgli* and extensions. They aim at establishing Benchmarking-as-a-service. The framework is used to inspect cloud-typical properties of DBMS like scalability, elasticity and availability. On the technical side, Java is used for the components, Python for configuration templates, Cloudiator as COT and to distribute the DBMS in the cloud, InfluxDB for monitoring and Docker compose for managing the multi-container application. Experiments are run at AWS and Google Cloud. The authors for example benchmark Apache Cassandra and Couchbase with YCSB.

The authors of [30] presented a *scalable Benchmark-as-a-Service platform*. Although technically very different (they benchmark the auction system RUBiS on VMs at Grid'5000), the platform is organized in the same spirit in similar components: several load injectors, a supervision (monitoring) tool, a SUT and a service running a (web site) benchmark.

1.3 Motivation

TPC-C, TPC-H and YCSB are classical benchmarks, that are still widely used to assess DBMS performance and when comparing DBMS [29]. The driver is an important component of a benchmark and should be watched closely. Even details of the Java configuation of a driver may affect measured latencies [10]. Recently, a TPC-C result has been published that uses 400 hosts for the driver [34]. Industry also *"noticed that the TPC-C tool itself was becoming a bottleneck in terms of the CPU and the memory it consumed"* and thus sharded the benchmark tool OLTPBench, so that *"partial benchmarks were run concurrently from multiple nodes"* [23]. It is clear, that when we have a very powerful SUT, the driver becomes a bottleneck and we have to add power to it. At some point scale-up is not an option anymore. Moreover we want to have a testbed, that represents a real-world setting, and scaled-out drivers represent distributed sources and users of data.

There is an increasing popularity of the cloud-native pattern in software engineering. For example, it is desirable to elastically scale a single (web) service or function of an e-commerce application to meet business needs without having to scale all components or the entire application. The pattern promises to enable scaling (as well as design and development) of a specific workload processing

component independently of others. Investigating a complex network of components, each with different scales, is very complicated. We are only interested in components that communicate directly with a DBMS and we want to focus on the drivers. To control effects and avoid mutual dependencies, we choose a fixed-size, single-host DBMS as the SUT. PostgreSQL is a general-purpose DBMS that can cover all the benchmarks considered. Furthermore, we can easily reach an area where the DBMS is performing well, as well as reach a peak performance point and then overload the system.

Although the definiton of cloud-native just refers to technologies that *empower organizations to build and run scalable applications in public, private, and hybrid clouds* [2], broadly accepted best practices cover decomposition of (monolithic) applications into small (micro-)services that are run in containers. Key features of this approach are scalability, elasticity, observability, flexibility and automation. Containerized applications in a cloud environment can be managed for example by the open source platform Kubernetes[1] (K8s). K8s is a project of the Cloud Native Computing Foundation (CNCF) and the major cloud providers offer it as a managed service. As K8s evolves to become *the operating system of the cloud*, we think it is a good candidate for backing the experiments.

These considerations lead to the following research questions:

RQ1: Is Kubernetes suitable for powering such a setup?
RQ2: Can we make use of key benefits of a cloud-native architecture?
RQ3: Is scale-out possible for the given benchmark tools?
RQ4: How does performance for scale-out and scale-up compare?

2 Solution Concept

We use Bexhoma[2] [7,8] for **automation** and to run the experiments as sequences of loading and benchmarking with increasing stress. This also allows us to get a clean copy of the SUT for each run. For hardware monitoring we deploy a daemonset of cAdvisors[3] to have a metrics exposer running on each node. We also deploy an instance of Prometheus[4] per SUT, that collects metrics every 10 seconds. It aggregates the metrics per object (driver and SUT) by summing over all cores and machines. We in particular collect metrics about CPU, RAM (active and cached), network and disk access. We also compute CPU time (in CPU seconds), that is the total time all involved CPUs are processing. As a heuristic, we assume CPU time is constant for a given workload, no matter how many parallel clients are processing. Metrics are collected over time and get linked to the corresponding driver and workload. The **observability** paradigm states we should combine metrics, logging and tracing to understand the state of

[1] https://github.com/kubernetes/kubernetes.
[2] https://github.com/Beuth-Erdelt/Benchmark-Experiment-Host-Manager.
[3] https://github.com/google/cadvisor.
[4] https://prometheus.io/.

a complex system as fully as possible [18]. Additionally to monitoring, we read outputs of all containers and fetch information about performance metrics, errors and deadlocks from there using regex. The benchmark tools will be presented in more detail in the following sections. By the **flexibility** of the pattern, we use the same procedure for all tools: In general, we containerize the operating system process of a benchmark tool by packing executables and a bash script into a Docker image. The script receives basic parameters from environment variables, configures and runs the tool and adds output about the runtime of the container. Moreover the bash script communicates with a Redis server for synchronization of jobs. K8s has the concept of jobs for workloads, that terminate after finishing (contrary to permanently running servers). Jobs can be used in various ways [31]. What comes closest to our use case is a job that keeps starting pods (i.e., containers attached to eachother) up to a maximum number of parallel pods. Pods are restarted if they fail. The job runs until a predefined number of pods has finished successfully (i.e., exit 0). We define the job span time τ_{span} as the span from the start of the first pod to the end of the last pod. This corresponds to the measurement of parallel executions as suggested by the TPC for TPC-H [33]. It takes into account that pods do not start perfectly at the same time. Moreover we log the start and end time for each pod p to have the pod run time $\tau_{run}(p)$, the time the pod p needs to process it's portion of the data. In an ideal situation, $\tau_{span} = \tau_{run}(p)$ for all pods p of the same job (K8s starts all pods at the same time and workload distributes equally). Kubernetes does not guarantee start times of pods and we in fact observe time intervals of up to minutes between pods of the same job. We thus use Redis to number the pods of a job (so each pod knowns which portion of data it is supposed to generate) and to optionally synchronize the pods of a job by enforcing a common start time (every pod of a job waits for the last to be ready). This helps to get meaningful results about **scalability**, since pods supposed to be parallel really run in parallel. The cluster is hosted at AWS as an instance of EKS[5] and defined in a YAML file. We in particular predefine node groups at creation. The components SUT, drivers (loaders, benchmarkers), monitoring and evaluation can be guided to only use corresponding node groups. We do not want the SUT to be installed on a node dedicated to hosting drivers. We use the AWS command `eksctl` for scaling of the node groups before the experiment starts and to reduce the groups to size 0 after the experiments have finished. Depending on the type and size of an experiment, we need more or less resources. This approach helps to benefit from **elasticity**.

3 Experiments

In order to evaluate the scaling behaviour, we limit our experiments to mid-size, so that scale-out and scale-up can be compared. We take PostgreSQL 15.0 as the DBMS in the system-under-test. As a general purpose DBMS, we expect it to cope well with all use cases. We slightly adjust configuration (worker and

[5] https://aws.amazon.com/eks/.

wal settings) and deploy it as a Docker container. The database is stored inside the Docker container space on a local SSD. We use JDBC driver version 42.5.0. We mark the peak of performance in plots by a dashed line at the elbow of throughput (no significant increase in performance, but only in costs).

3.1 YCSB: Threads vs Processes

YCSB is a transactional stress test for DBMS [4] and has been widely used since its introduction in 2010. YCSB runs a configurable mixture of read and write queries and distinguishes between a loading and a transaction phase. YCSB provides a tool written in Java [3]. As it is conceptually a key/value benchmark, it is particularly suitable for NoSQL DBMS, but it also provides an interface to JDBC. The tool does not create the schema. It has parameters for vertical scaling, in particular `threadcount` and `target` ops (operations per second). YCSB generates random keys and values, but in a deterministic way, so parallel execution of the same container would generate the same data again. However the provided tool has a corresponding parameter to split workloads into parts, so it is well prepared for horizontal scaling, too. In the original paper [4], the authors load a 120 GB database into several systems on 6 servers with 8 GB RAM each. The driver runs 500 threads on an 8 core. The authors observe the driver is mostly idle and waits for the DBMS, so in this situation, the driver clearly is not a bottleneck. In our setup, we use the following machines: SUT and drivers one `c5d.9xlarge` (36 vCPUs, 72 GiB RAM) each, and one `r5dn.xlarge` (4 vCPUs, 32 GiB RAM) for monitoring and evaluation. The scaling is set to 30M rows at loading (= 30 Gb) and 30M operations at execution. We run a series of experiments, where we increase target ops and compare scale-out and scale-up and monitor resource consumption. Each run operates on a clean copy.

Loading Phase. Generation and loading is the same process (*in-line load*). We load data and compare splitting into 8 pods (4 threads each) to not-split (32 threads). The split is made into equal parts (rows, targets, operations), so we expect each part to bring about $\frac{1}{8}$ of performance of the unsplit variant. The plots in Fig. 2 show the comparison split / not-split. Peak performance is at about 32,000 ops/s. At the splitted setup, we aggregate throughput by summation, mean latency by averaging and maximum and percentiles of latencies by

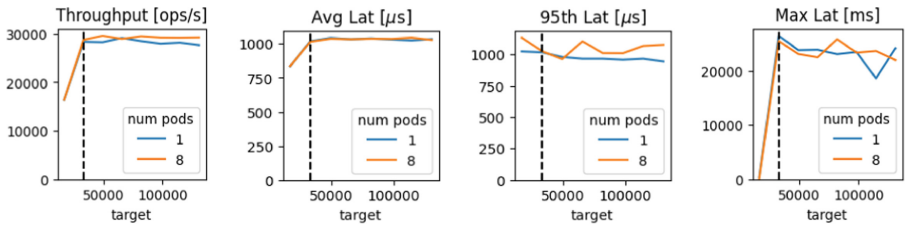
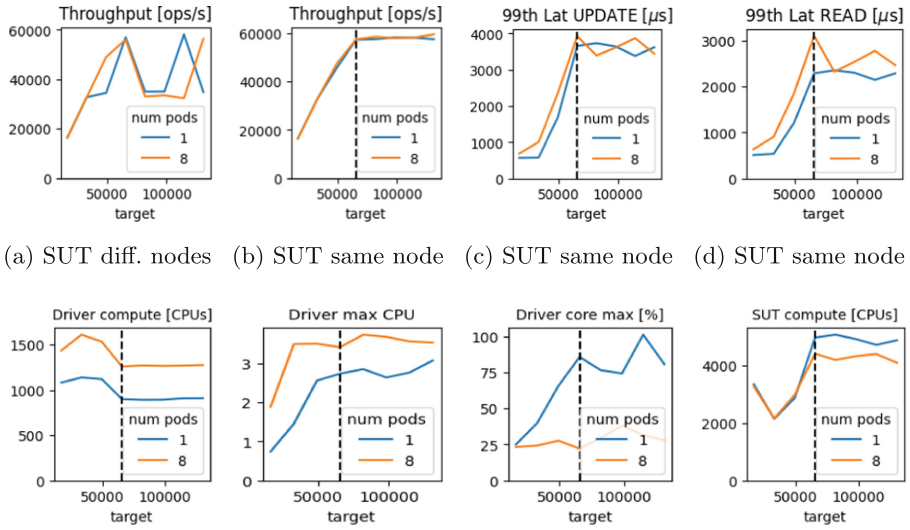


Fig. 2. YCSB Loading - PostgreSQL for 32 threads at driver

maximum. This is a conservative assumption, as mathematically the maximum of 95th percentiles not necessarily yields a 95th percentile again. We observe throughput is almost the same for split (processes) and not-split (threads). Also latencies are very similar.



(a) SUT diff. nodes (b) SUT same node (c) SUT same node (d) SUT same node

(e) SUT same node (f) SUT same node (g) SUT same node (h) SUT same node

Fig. 3. YCSB Workload A - PostgreSQL for 32 threads at driver

Execution Phase The plots in Fig. 3 show the results for workload A (50% READ/UPDATE). Peak performance is at about 60,000 ops/s. It is particularly interesting in Fig. 3 a), that throughput varies strongly, if we do not fix the host of the SUT to be the same node in each run. If we fix the node as in the other plots of Fig. 3, split and not-split show similar throughput and latencies. However the driver is more involved in the split situation: CPU cores are less utilized and this costs more compute in the end. Another interesting aspect is that the SUT is stressed more in the not-split scenario. For workload A, splitting the driver does not affect the performance. This does not hold for all workloads. In Fig. 4 we show some results for workload C (100% READ). Here, the not-split setting seems to be superior. It shows higher throughput and lower latency. Peak performance is at about 115,000 ops/s.

Evaluation YCSB splits very well. Scale-up and scale-out behave quite similarly when loading the data. The effect of scale-out on performance in the execution phase however may depend on the workload. It shifts compute from the SUT to the driver (Fig. 5).

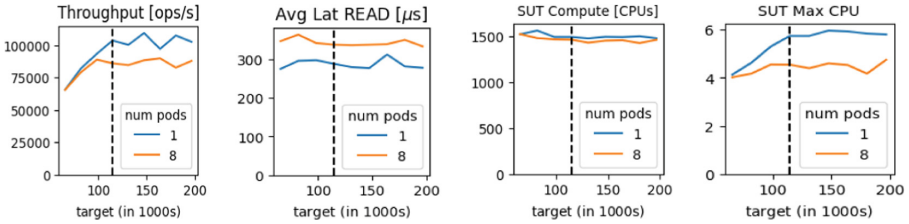


Fig. 4. YCSB Workload C - PostgreSQL for 32 threads at driver

component	RAM [Gb]	cache [Gb]	max vCPU	max core	CPUs	driver threads
SUT loading	40	68	2.5	15%	2000	32
driver loading 1	0.3	0.3	2	10%	1250	32
driver loading 8	2.5	2.5	1.5	10%	1250	8 x 4
Workload A						
SUT execution 1	50	68	16	50%	5000	32
SUT execution 8	50	68	14	30%	4400	8 x 4
driver execution 1	0.2	0.2	3	85%	900	32
driver execution 8	1.4	1.4	3.5	22%	1250	8 x 4
Workload C						
SUT execution 1	40	68	6	30%	1500	32
SUT execution 8	40	68	4	25%	1500	8 x 4
driver execution 1	0.2	0.2	3	98%	1000	32
driver execution 8	1.4	1.4	3.5	25%	1200	8 x 4

Fig. 5. YCSB component resources for 30 Gb at peak performance

Processes do cost more resources than threads in terms of RAM and compute. It is absolutely crucial to fix the host of the SUT to be the same node in each benchmark run for comparability.

3.2 HammerDB's TPC-C

TPC-C is a transaction processing benchmark [32]. It runs a mix of 5 concurrent transactions on 9 tables representing an E-Commerce situation. TPC-C is actively supported since 1992. The TPC also supports a (simplified) version, TPROC-C, a transaction processing benchmark derived from TPC-C, and an implementation, HammerDB [13]. HammerDB is written in Tcl due to its good multithreading properties and currently supports 11 DBMS [12]. A command-line version was introduced in version 3.0 and a Docker image in version 4.5. TPC-C expects a number of warehouses and a number of virtual users interacting with the warehouses. HammerDB recommends to try $200-500$ warehouses per server CPU socket and $4-5$ warehouses per vuser. The vusers are similar to threads, so that helps vertical scalability. In our setup, we again use the following machines: SUT and driver one `c5d.9xlarge` (36 vCPUs, 72 GiB RAM) each, and one `r5dn.xlarge` (4 vCPUs, 32 GiB RAM) for monitoring and evaluation.

Loading Phase In the generation phase schema, indexes and procedures are generated and data is loaded. HammerDB does not support horizontal scaling for loading: Schema creating is fixed and the same data is generated by each instance of loaders. We fix the number of warehouses to 320 in the following. We can see an ascending performance of up to 1400 warehouses per hour in Fig. 6, but also some quick saturation. There is no improvement of the ingestion speed although more resources are used. When looking at more details of the SUT in Fig. 7, as an example for 10 loader threads, we see peaks of 800 Mb/s at network, 100% at utilization of CPU cores and 200 Mb/s at disk, but these peaks are reached only occasionally. Compute usage is increasing constantly. It is not obvious, where the saturation comes from. As a test, we also exchange the server to a more powerful one, but this gives a similar picture. This means performance is limited by something we do not see here, e.g. by settings of the VM like swap or hugepages or by the PostgreSQL configuration.

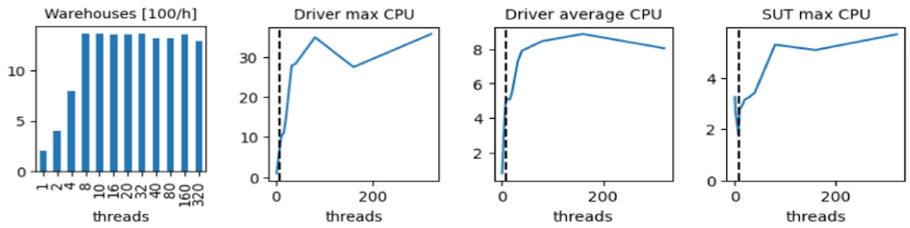


Fig. 6. HammerDB TPROC-C ingestion of 320 warehouses, peak at 8 threads

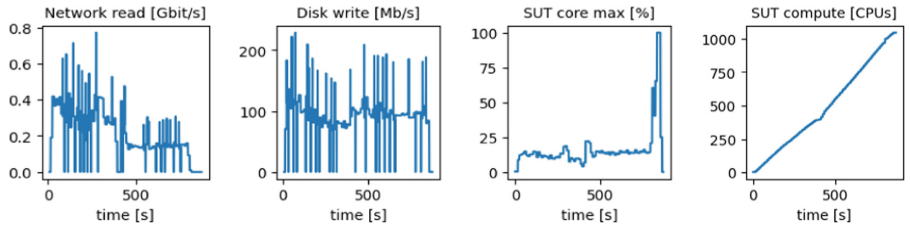


Fig. 7. HammerDB TPROC-C - 10 loader threads for 320 warehouses

Execution Phase In the benchmarking phase the transactions are run randomly at a fixed ratio on existing data. The tool uses procedures in DBMS for execution. NOPM (New Orders per minute) is a performance metric independent of any particular database implementation. It is computed by counting created orders inside the database, i.e., server-sided. We increase the number of threads

from 1 to 40 to query the same database. NOPM goes up to 40,000 for 10 threads and then drops dramatically in Fig. 8. This corresponds to an explosion of CPU utilization. Apparently this puts the configured system to a limit.

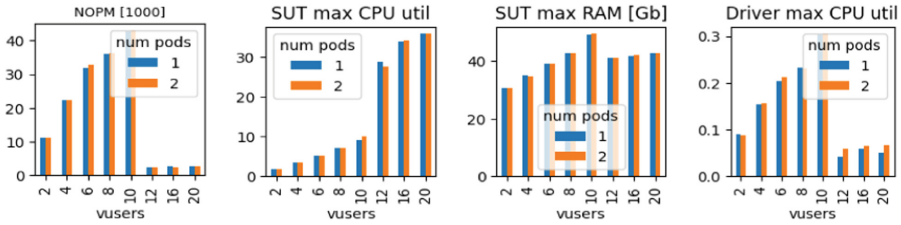


Fig. 8. HammerDB TPROC-C execution at 320 warehouses

Evaluation We expect the loading phase to benefit from scale-out. Although peak performance is reached fairly quickly here, it is noticeable that the driver is computationally more challenged than the SUT. The execution of the benchmark is not to be expected to put the driver host to a limit easily, since it uses only low resources., about 100 Mb of RAM and 0.3 of CPU.

component	RAM [Gb]	cache [Gb]	max vCPU	max core	CPUs	driver threads
SUT loading	30	60	2	100%	1000	8
driver loading	0.09	0.09	8	100%	4500	8
SUT execution 1	50	65	9	90%	10000	10
SUT execution 2	50	65	10	90%	10000	2 x 5
driver execution 1	0.09	0.09	0.3	5%	280	10
driver execution 2	0.12	0.12	0.3	5%	280	2 x 5

Fig. 9. HammerDB component resources for 320 warehouses

3.3 Benchbase's TPC-C

Benchbase is a Java-based collection of 17 mostly transactional benchmarks. It is the successor of OLTP-bench [5] and is available as a Docker image. Benchbase connects via JDBC and can be used for any DBMS that offers such an interface. In the following we focus on the implementation of TPC-C. We set isolation level to READ COMMITED (HammerDB's default) and we use the same machines.

Loading Phase As with HammerDB, each loading instance tries to install the schema and the same data. The tool automatically uses as many threads for loading as the driver has cores (but maximum number of warehouses). We run TPC-C with 48 warehouses on the same hardware as before. Loading takes ca. 220 seconds, that is we can load about 800 warehouses per hour, with 36 threads fixed by the tool.

Execution Phase The tool has parameters for vertical scaling, in particular `terminals` (threads) and `rate` (target operations per second). Business logic is implemented on client side. We assume this puts more stress to the driver than HammerDB does. As the workload is running transactions randomly (and independently), parallel execution of several drivers is possible. We compare the results for cascading terminals running in a single driver (not-split) to terminals split into 2, 4 and 8 drivers running simultaneously. Each of the four runs operates on a clean copy. In Fig. 10 we see a very mixed result. In terms of throughput and latency, the not-split and splitting into up to 4 process behave quite similarly. Splitting into 8 processes shows significantly lower performance, but also fewer deadlocks and compute consumption.

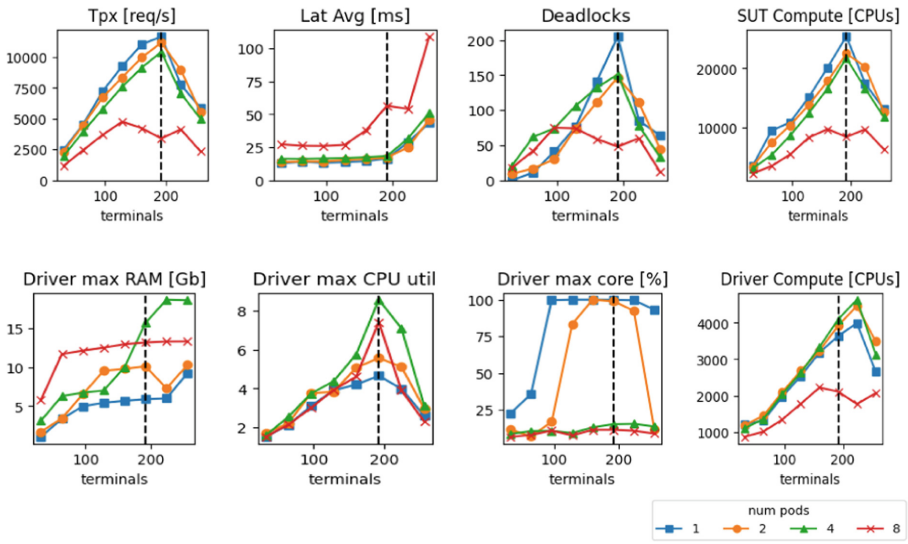


Fig. 10. Benchbase TPROC-C execution at 48 warehouses, target 16384 req/s

Evaluation Loading with Benchbase is more intensive computationally than with HammerDB, Fig. 11, and the tool would definitely benefit from scale-out here. At execution phase, efficiency clearly drops with increasing scale-out. We assume the handling of connection pools dominates this setting. It clearly uses more resources than HammerDB on the client side and more RAM than YCSB.

component	RAM [Gb]	cache [Gb]	max vCPU	max core	CPUs	driver threads
SUT loading	7	12	28	80%	1400	36
driver loading	0.8	0.8	35	96%	4500	36
SUT execution 1	53	65	30	128%	25500	192
SUT execution 2	50	65	28	96%	22500	2 x 96
SUT execution 4	47	65	26	94%	21500	4 x 48
SUT execution 8	28	49	18	67%	8500	8 x 24
driver execution 1	6	6	4.8	100%	3700	192
driver execution 2	10	10	5.5	99%	4000	2 x 96
driver execution 4	16	16	8.5	15%	4600	4 x 48
driver execution 8	13	13	7.5	11%	1800	8 x 24

Fig. 11. Benchbase component resources for 48 warehouses at peak performance

3.4 TPC-H: Throughput of Loading and Execution

TPC-H is a Decision Support Benchmark [33]. It sets up 8 normalized tables and runs 22 complex analytical SQL queries. TPC-H is actively supported since 1999. The benchmark has a scaling factor SF, that corresponds to the size of data in GB. The loading phase simulates filling a data warehouse in a batch load. The execution phase consists of a power test and a throughput test. Typically, benchmarking focuses on the power test: The database is loaded and we are interested in how fast the DBMS can process the workload of the 22 queries. TPC-H is known to include some choke points [1,6], that make it notoriously hard for the optimizer to find the best execution plan. As it involes no scaling, we skip the power test and instead focus on the throughput test.

Loading Phase The TPC provides a tool for generating data [21,22] in parts of almost the same size each in a single process per part. This supports horizontal scalability. The schema is not generated by the tool and data loading and driving the benchmark must be done externally. We use the following machines: SUT `c5d.9xlarge` (36 vCPUs, 72 GiB RAM), up to 10 driver nodes `r5dn.xlarge` (4 vCPUs, 32 GiB RAM) and another one for monitoring and evaluation. We test the common *load from stored records* approach with an increasing number of parallel pods. We therefore couple a generator as an initcontainer with an injector container in a pod. Ingestion starts when generation has been completed. The generator contains TPC's `dbgen` and stores data in the distributed file system EFS. Moreover it transforms the data (removes the last character in each line). The injector uses `psql` to send `\copy` commands to the DBMS to load data that is local to the injector. In Fig. 12 we see ingestion rate depends on the size and plateaus at around $150-200$ Gb/h. The loader takes up to 1 Gb of RAM per SF, no matter how many nodes are involved. The RAM usage of the generator, on the other hand, scales linearly with the number of pods, regardless of the SF.

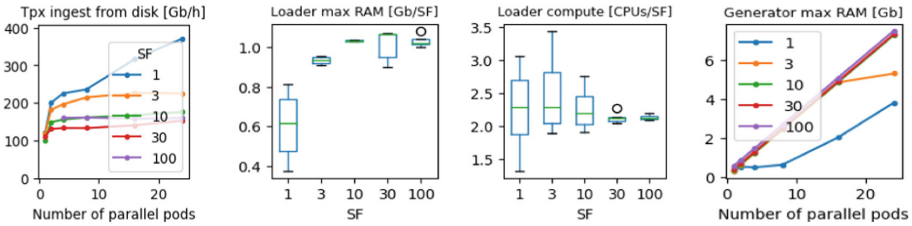


Fig. 12. TPC-H - PostgreSQL generate and load from disk

Execution Phase We use `x1e.16xlarge` (64 vCPU, 1952 GiB RAM) for the SUT now. After ingestion has finished and to prepare for the execution phase, we run a sequence of scripts: 1. apply unique indexes (PK) and non-unique indexes (on FK columns) at about 200 Gb/h, 2. apply FK-constraints at about 130 Gb/h, 3. let the DBMS analyze the tables to update statistics for the query optimizer at about 200 Gb/h. In the throughput test the driver runs several streams at the same time. We use `DBMSBenchmarker` as driver. It has a parameter for vertical scaling. This spans a pool of subprocesses per query to avoid Python's GIL. The subprocesses connect to the DBMS in parallel to simulate parallel clients. Here however we establish a single connection from Q1 to Q22 and run a single client per container. Specifications of TPC-H require for example at least 4 parallel streams for SF 30. They also require reordering and reparametrization in a predefined manner for each stream. We however run the queries with the same parameter and in the same natural ordering, since this should reduce resource consumption. We run a cascading sequence of parallel streams on the same database. For evaluation we use the official TPC-H metric TPX@Size(n, SF). In Fig. 13 we show some results. Throughput is in fact close to linear. Naively we can expect the memory consumption is equal to the size of the database, maybe scaled by the number of streams. We however see that the maximum is 46 times the size at 10 streams for SF=30, i.e., ca. 1400 Gb. Also remarkable is the total compute time scales almost perfectly being around $100 \cdot n \cdot SF$ each time. The only clear difference is at SF=30. Throughput is much lower there, too. We assume this comes from cache misses because of the size.

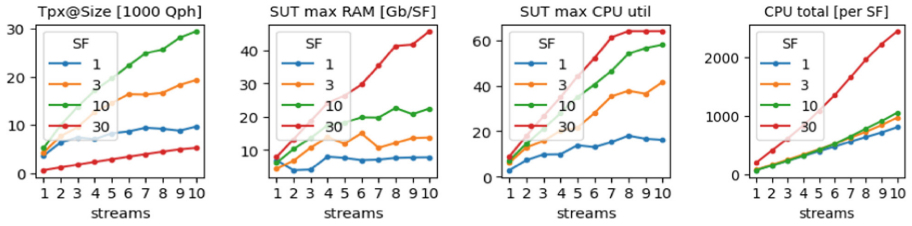


Fig. 13. TPC-H - PostgreSQL throughput test

Evaluation Loading benefits from scaling. Even for PostgreSQL, that is not optimized for such a setting, we see an improvement at throughput of up to 100%. The loading driver's total RAM usage depends on SF, not on n, while for the data generation component it is the other way around.

component	RAM [Gb]	cache [Gb]	max vCPU	CPUs	pods	SF
SUT loading	3	66	2.9	2300	4	100
driver generate | load	$0.3n$ | SF	$1.7n$ | SF	n | 0.5	75 SF | 2.5 SF	n	SF
SUT execution	1400	1400	64	74000	10	30
driver execution	$0.4n$	$0.4n$	$0.4n$	$13n$	n	30

Fig. 14. TPC-H component resources depending on SF and number n of pods

4 Discussion

We used Kubernets to successfully set up a cloud-native benchmarking environment (RQ1). The presented approach allows us to benefit from key features of this pattern (RQ2), since it provides

1. automation. We run scripts that perform predefined experiments from starting cluster nodes to evaluation of results.
2. flexibility. The same pattern can be used for different benchmarks and tools. This implies we can reduce human workload, since we can reuse most of the components like installing SUT, adjusting monitoring and collecting results.
3. elasticity. We predefine a cluster and use AWS `eksctl` to scale nodegroups per experiment to match requirements. Limitations we have noticed are
   - The SUT node should be fixed, otherwise results are not reproducible.
   - We cannot access parameters of the hypervisor in a simple way by just relying on predefined node groups and this may affect performance.
4. scalability. Kubernetes jobs are suitable to run concurrent synthetical workloads. It is important however to synch pods of a job, for example by using a Redis counter. Otherwise there might be a noticable shift between pods and expressiveness is reduced.
5. observability. We monitor hardware metrics of all components and additionally collect information from container logs. This provides a significant amount of information that sheds light on the performance of the system. Bexhoma is Python-based, so we have access to a variety of evaluation packages. We also found performance limitations, that cannot be explained by the data we collected. It may be helpful to also include application metrics of the DBMS to explain all aspects.

Except for the loading phase of HammerDB and Benchbase, all tools and phases support scale-out (RQ3). We assume that these two steps would also

benefit from scale-out, since the data generation is computationally intensive. HammerDB's server-side and Tcl-based design is very lightweight and low in resources. Benchbase client-side logic and Java implementation is more flexible, but also uses more resources. The effect of scaling-out (RQ4) depends on the use-case.

We reported some rough numbers of hardware metrics without variation. This should give the reader an idea of the resource consumption to be expected. Further experiments would be necessary for fixing the validity ranges of the values. We did not pay attention to the size of the Docker containers. Small components are suitable for a serverless approach. We also ignored the key feature of resilience in this report. Care must be taken to ensure that a distributed system responds to all possible disturbances like lost connections or pods failing to start or running out of resources.

For the transactional workloads of YCSB and TPC-C, we split operations across parallel K8s pods. We primarily observed nice scaling, predictable performance and similar results for split / not-split setups. As a general trend, scale-out shifts compute from SUT to driver. Simply adding power to the driver in terms of threads or processes may lower performance of the SUT. At YCSB the SUT may detect scaling-out vs. scaling-up, depending on the workload. HammerDB is very lightweight, so we could only compare 1 and 2 processes, showing almost the same results. The SUT is put to a limit quickly. Benchbase showed a slightly decreasing performance that drops significantly at 8 processes.

The analytical benchmark TPC-H differs in that it supports loading from stored records. The plain vanilla PostgreSQL ingestion mechanism we used here requires the driver to have RAM up to the scaling factor, that is the size of the data source. On the one hand, this also allows huge amounts of data to be read in if we have enough nodes, which can even be small. On the other hand, PostgreSQL does not load the data "all at once", so RAM in the size of the data source is certainly exaggerated and the mechanism is therefore not optimal in terms of resource management. The generator uses CPU and RAM depending on the number of pods, not the scaling factor. We did not follow all TPC specifications. For example we did not implement the update streams that change the database during benchmarking. We are convinced that the presented approach can also be used for more complex workflows. The throughput test, although we used identical streams, takes a lot of RAM. The driver DBMSBenchmarker, although Python-based, uses only little resources.

All results surely depend on the concrete PostgreSQL configuration, which we did not include. PostgreSQL is a general purpose DBMS, that is suitable for all regarded workloads, but it is rather not optimal. The intention of this research was not to find optimal configurations of PostgreSQL, although this is an interesting aspect. We chose smaller scales to be sure that we can hit and assess performance plateaus, with both horizontal and vertical scaling of drivers. Nevertheless it will be interesting to test more specialized DBMS like distributed SQL for transactional workloads or NewSQL for sharded analytical workloads.

We have implemented benchmarks in the cloud-native pattern, but only in the classic form as is. The next steps could be to further exploit the features, for example to mix drivers for transactional and analytical workloads for a HTAP scenario or to include cloud-native DBMS and to measure how the scale-out behavior of the DBMS reacts to a scale-out of the driver. However, it is important to compare scale-out and scale-up before running extensive benchmarks. At least with the JDBC-based drivers we use, the DBMS can be sensitive to the difference.

5 Conclusion

The cloud-native pattern can be applied fruitfully to different clouds and various DBMS, as in [7,8], and also to various benchmarks and driver tools as presented here. We deduced some research questions to design and evaluate a cloud-native implementation. We presented a solution and how to benefit from the key features of the pattern in terms of scalability, elasticity, observability, flexibility and automation. We implemented and discussed the benchmarks and tools YCSB, TPC-C (HammerDB, Benchbase) and TPC-H. We have shown that we can collect comprehensive data on the performance of the drivers. Our considerations were guided by the idea that scale-out benchmarking is an interesting approach because it better represents a distributed system than a monolithic driver. The presented pattern allows a practically infinite scaling in a relatively simple way. We pursued the question of what effects one has to deal with and what to measure. In the end this also affects the costs. Hopefully this is helpful for the classical use cases presented, but it may also be helpful for more complex applications and the development of future benchmarks and tools. This project has been partly supported by an AWS Research Grant.

References

1. Boncz, P., Neumann, T., Erling, O.: TPC-H analyzed: hidden messages and lessons learned from an influential benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2013. LNCS, vol. 8391, pp. 61–76. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04936-6_5
2. CNCF: Cloud native definition v1.0. https://github.com/cncf/toc/blob/main/DEFINITION.md. Accessed 7 Oct 2022
3. Cooper, B.: YCSB GitHub repository. https://github.com/brianfrankcooper/YCSB. Accessed 11 Dec 2022
4. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 143-154. SoCC '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1807128.1807152
5. Difallah, D.E., Pavlo, A., Curino, C., Cudre-Mauroux, P.: OLTP-bench: an extensible testbed for benchmarking relational databases. Proc. VLDB Endow. **7**(4), 277–288 (2013)
6. Dreseler, M., Boissier, M., Rabl, T., Uflacker, M.: Quantifying TPC-H choke points and their optimizations. Proc. VLDB Endow. **13**(10), 1206–1220 (2020). https://doi.org/10.14778/3389133.3389138

7. Erdelt, P.K.: A framework for supporting repetition and evaluation in the process of cloud-based dbms performance benchmarking. In: Nambiar, R., Poess, M. (eds.) TPCTC 2020. LNCS, vol. 12752, pp. 75–92. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84924-5_6
8. Erdelt, P.K.: Orchestrating DBMS benchmarking in the cloud with Kubernetes. In: Nambiar, R., Poess, M. (eds.) TPCTC 2021. LNCS, vol. 13169, pp. 81–97. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94437-7_6
9. Erdelt, P.K., Jestel, J.: DBMS-benchmarker: benchmark and evaluate DBMS in python. J. Open Source Softw. **7**(79), 4628 (2022). https://doi.org/10.21105/joss.04628
10. Fruth, M., Scherzinger, S., Mauerer, W., Ramsauer, R.: Tell-tale tail latencies: pitfalls and perils in database benchmarking. In: Nambiar, R., Poess, M. (eds.) TPCTC 2021. LNCS, vol. 13169, pp. 119–134. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94437-7_8
11. Grohmann, J., Seybold, D., Eismann, S., Leznik, M., Kounev, S., Domaschka, J.: Baloo: measuring and modeling the performance configurations of distributed DBMS. In: 2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 1–8 (2020). https://doi.org/10.1109/MASCOTS50786.2020.9285960
12. HammerDB: About HammerDB. https://www.hammerdb.com/about.html. Accessed 11 Dec 2022
13. HammerDB: HammerDB. https://www.hammerdb.com/index.html. Accessed 11 Dec 2022
14. Henning, S., Hasselbring, W.: A configurable method for benchmarking scalability of cloud-native applications. Empir. Softw. Eng. **27**(6), 143 (2022). https://doi.org/10.1007/s10664-022-10162-1
15. Henning, S., Wetzel, B., Hasselbring, W.: Reproducible benchmarking of cloud-native applications with the Kubernetes operator pattern. In: Symposium on Software Performance 2021, CEUR Workshop Proceedings (2021)
16. Henning, S., Hasselbring, W.: Theodolite: scalability benchmarking of distributed stream processing engines in microservice architectures. Big Data Res. **25**, 100209 (2021). https://doi.org/10.1016/j.bdr.2021.100209
17. Henning, S., Hasselbring, W.: Demo paper: benchmarking scalability of cloud-native applications with theodolite. In: 2022 IEEE International Conference on Cloud Engineering (IC2E), pp. 275–276 (2022). https://doi.org/10.1109/IC2E55432.2022.00037
18. Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., Zieliński, S.: Towards the observability of cloud-native applications: the overview of the state-of-the-art, pp. 1–1. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3281860
19. Kostopoulos, C., Mouchakis, G., Troumpoukis, A., Prokopaki-Kostopoulou, N., Charalambidis, A., Konstantopoulos, S.: KOBE: cloud-native open benchmarking engine for federated query processors. In: Verborgh, R., et al. (eds.) ESWC 2021. LNCS, vol. 12731, pp. 664–679. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77385-4_40
20. Nikolaidis, F., Chazapis, A., Marazakis, M., Bilas, A.: Frisbee: automated testing of cloud-native applications in Kubernetes. arXiv preprint arXiv:2109.10727 (2021)
21. Poess, M., Rabl, T., Frank, M., Danisch, M.: A PDGF implementation for TPC-H. In: Nambiar, R., Poess, M. (eds.) TPCTC 2011. LNCS, vol. 7144, pp. 196–212. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32627-1_14

22. Rabl, T., Poess, M.: Parallel data generation for performance analysis of large, complex RDBMS. In: Proceedings of the Fourth International Workshop on Testing Database Systems. DBTest '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1988842.1988847
23. Ranganathan, K.: TPC-C benchmark: 10,000 warehouses on YugabyteDB. Yugabyte (2020). https://www.yugabyte.com/blog/tpc-c-benchmark-10000-warehouses-on-yugabytedb/#sharding-the-tpc-c-tool-to-benchmark-at-scale
24. Seybold, D.: An automation-based approach for reproducible evaluations of distributed DBMS on elastic infrastructures, Ph.D. thesis, Universität Ulm (2021). https://doi.org/10.18725/OPARU-37368
25. Seybold, D., Keppler, M., Gründler, D., Domaschka, J.: Mowgli: finding your way in the DBMS jungle. In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, pp. 321–332. ICPE '19, Association for Computing Machinery, New York, NY, USA (2019).https://doi.org/10.1145/3297663.3310303
26. Seybold, D., Volpert, S., Wesner, S., Bauer, A., Herbst, N., Domaschka, J.: Kaa: evaluating elasticity of cloud-hosted DBMS. In: 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 54–61 (2019). https://doi.org/10.1109/CloudCom.2019.00020
27. Seybold, D., Wesner, S., Domaschka, J.: King Louie: reproducible availability benchmarking of cloud-hosted DBMS. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp. 144–153. SAC '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3341105.3373968
28. Souza., P., et al.: Isabel: infrastructure-agnostic benchmark framework for cloud-native platforms. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science - CLOSER, pp. 482–489. INSTICC, SciTePress (2020). https://doi.org/10.5220/0009581004820489
29. Taipalus, T.: Database management system performance comparisons: a systematic survey. arXiv preprint arXiv:2301.01095 (2023)
30. Tchana, A., De Palma, N., El-Rheddane, A., Dillenseger, B., Etchevers, X., Safieddine, I.: A scalable benchmark as a service platform. In: Dowling, J., Taïani, F. (eds.) Distributed Applications and Interoperable Systems, pp. 113–126. Springer, Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38541-4_9
31. The Kubernetes Authors: Jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job. Accessed 3 May 2023
32. Transaction Processing Performance Council: TPC-C Homepage. https://www.tpc.org/tpcc. Accessed 11 Dec 2022
33. Transaction Processing Performance Council: TPC-H - Homepage. https://www.tpc.org/tpch. Accessed 2 Apr 2019
34. Yang, Z., et al.: Oceanbase: a 707 million TPMC distributed relational database system. Proc. VLDB Endow. **15**(12), 3385–3397 (2022). https://doi.org/10.14778/3554821.3554830

Author Index

R. Nambiar and M. Poess (Eds.): TPCTC 2023, LNCS 14247, pp. 143–144, 2024.
https://doi.org/10.1007/978-3-031-68031-1