SIGMOD Officers, Committees, and Awardees

Chair
Juliana Freire
Computer Science & Engineering
New York University
Brooklyn, New York
USA
+1 646 997 4128
juliana.freire <at> nyu.edu

Vice-Chair
Ihab Francis Ilyas
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA
+1 519 888 4567 ext. 33145
ilyas <at> uwaterloo.ca

Secretary/Treasurer
Fatma Ozcan
Google
Sunnyvale
California
USA
fozcan <at> google.com

SIGMOD Executive Committee:
Juliana Freire (Chair), Ihab Francis Ilyas (Vice-Chair), Fatma Ozcan (Treasurer), K. Selçuk Candan, Rada Chirkova, Chris Jermaine, Wang-Chiew Tan, AnHai Doan, Leonid Libkin, and Curtis Dyreson

Advisory Board:
Yannis Ioannidis (Chair), Phil Bernstein, Surajit Chaudhuri, Rakesh Agrawal, Joe Hellerstein, Mike Franklin, Laura Haas, Renee Miller, John Wilkes, Chris Olsten, AnHai Doan, Tamer Özsu, Gerhard Weikum, Stefano Ceri, Beng Chin Ooi, Timos Sellis, Sunita Sarawagi, Stratos Idreos, and Tim Kraska

SIGMOD Information Director:
Curtis Dyreson, Utah State University

Associate Information Directors:
Huiping Cao, Georgia Koutrika, Wim Martens, and Sourav S Bhowmick

SIGMOD Record Editor-in-Chief:
Rada Chirkova, NC State University

SIGMOD Record Associate Editors:
Azza Abouzied, Lyublena Antova, Marcelo Arenas, Vanessa Braganholo, Aaron J. Elmore, Wim Martens, Kyriakos Mouratidis, Dan Olteanu, Divesh Srivastava, Pınar Tözün, Immanuel Trummer, Yannis Velegrakis, Marianne Winslett, and Jun Yang

SIGMOD Conference Coordinator:
K. Selçuk Candan, Arizona State University

PODS Executive Committee:
Dan Suciu (Chair), Tova Milo, Diego Calvanese, Wang-Chiew Tan, Rick Hull, and Floris Geerts

Sister Society Liaisons:
Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE)

SIGMOD Awards Committee:
M. Tamer Özsu (Chair), Stefano Ceri, Yanlei Diao, Volker Markl, Renee Miller, and Sunita Sarawagi

Jim Gray Doctoral Dissertation Award Committee:
Pınar Tözün (co-Chair), Viktor Leis (co-Chair), Peter Bailis, Alexandra Meliou, Bailu Ding, Vanessa Braganholo, Immanuel Trummer, and Joy Arulraj

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Recipients of the award are the following:

Michael Stonebraker (1992)
David DeWitt (1995)
Serge Abiteboul (1998)
Rudolf Bayer (2001)
Ronald Fagin (2004)
Jennifer Widom (2007)
Umeshwar Dayal (2010)
Stefano Ceri (2013)
Gerhard Weikum (2016)
Anastasia Ailamaki (2019)
Jim Gray (1993)
C. Mohan (1996)
Hector Garcia-Molina (1999)
Patricia Selinger (2002)
Michael Carey (2005)
Moshe Y. Vardi (2008)
Surajit Chaudhuri (2011)
Martin Kersten (2014)
Goetz Graefe (2017)
Beng Chin Ooi (2020)
Philip Bernstein (1994)
David Maier (1997)
Rakesh Agrawal (2000)
Don Chamberlin (2003)
Jeffrey D. Ullman (2006)
Masaru Kitsuregawa (2009)
Bruce Lindsay (2012)
Laura Haas (2015)
Raghu Ramakrishnan (2018)

SIGMOD Systems Award

For technical contributions that have had significant impact on the theory or practice of large-scale data management systems.

Michael Stonebraker and Lawrence Rowe (2015); Martin Kersten (2016); Richard Hipp (2017); Jeff Hammerbacher, Ashish Thusoo, Joydeep Sen Sarma; Christopher Olston, Benjamin Reed, and Utkarsh Srivastava (2018); Xiaofeng Bao, Charlie Bell, Murali Brahmadesam, James Corey, Neal Fachan, Raju Gulabani, Anurag Gupta, Kamal Gupta, James Hamilton, Andy Jassy, Tengiz Kharatishvili, Sailesh Krishnamurthy, Yan Leshinsky, Lon Lundgren, Pradeep Madhavarapu, Sandor Maurice, Grant McAlister, Sam McKelvie, Raman Mittal, Debanjan Saha, Swami Sivasubramanian, Stefano Stefani, and Alex Verbitski (2019); Don Anderson, Keith Bostic, Alan Bram, Grg Burd, Michael Cahill, Ron Cohen, Alex Gorrod, George Feinberg, Mark Hayes, Charles Lamb, Linda Lee, Susan LoVerso, John Merrells, Mike Olson, Carol Sandstrom, Steve Sarette, David Schacter, David Segleau, Mario Seltzer, and Mike Ubell (2020)

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)
Jeffrey Ullman (1996)
Raghu Ramakrishnan (1999)
Daniel Rosenkrantz (2001)
Surajit Chaudhuri (2004)
Hans-Jörg Schek (2007)
David Lomet (2010)
H.V. Jagadish (2013)
Samuel Madden (2016)
Ahmed Elmagarmid (2019)
Stratos Idreos (2020)
Dennis Shasha (2020)
Gio Wiederhold (1995)
Avi Silberschatz (1997)
Michael Carey (2000)
Richard Snodgrass (2002)
Hongjun Lu (2005)
Klaus R. Dittrich (2008)
Gerhard Weikum (2011)
Kyu-Young Whang (2014)
Yannis E. Ioannidis (2017)
Philipe Bonnet (2020)
Stefan Manegold (2020)
Yahiko Kambayashi (1995)
Won Kim (1998)
Laura Haas (2000)
Michael Ley (2003)
Tamer Özsu (2006)
Beng Chin Ooi (2009)
Marianne Winslett (2012)
Curtis Dyreson (2015)
Z. Meral Özsoyoğlu (2018)
Juliana Freire (2020)
Ioana Manolescu (2020)

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau. *Honorable Mentions*: Marcelo Arenas and Yanlei Diao
- **2007** *Winner*: Boon Thau Loo. *Honorable Mentions*: Xifeng Yan and Martin Theobald
- **2008** *Winner*: Ariel Fuxman. *Honorable Mentions*: Cong Yu and Nilesh Dalvi
- **2009** *Winner*: Daniel Abadi. *Honorable Mentions*: Bee-Chung Chen and Ashwin Machanavajjhala
- **2010** *Winner:* Christopher Ré. *Honorable Mentions*: Soumyadeb Mitra and Fabian Suchanek
- **2011** *Winner*: Stratos Idreos. *Honorable Mentions*: Todd Green and Karl Schnaitterz

- **2012** *Winner*: Ryan Johnson. *Honorable Mention*: Bogdan Alexe
- **2013** *Winner*: Sudipto Das, *Honorable Mention*: Herodotos Herodotou and Wenchao Zhou
- **2014** *Winners*: Aditya Parameswaran and Andy Pavlo.
- **2015** *Winner*: Alexander Thomson. *Honorable Mentions*: Marina Drosou and Karthik Ramachandra
- **2016** *Winner*: Paris Koutris. *Honorable Mentions*: Pinar Tozun and Alvin Cheung
- **2017** *Winner*: Peter Bailis. *Honorable Mention*: Immanuel Trummer
- **2018** *Winner*: Viktor Leis. *Honorable Mention*: Luis Galárraga and Yongjoo Park
- **2019** *Winner*: Joy Arulraj. *Honorable Mention*: Bas Ketsman
- **2020** *Winner:* Jose Faleiro. *Honorable Mention:* Silu Huang

A complete list of all SIGMOD Awards is available at: **https://sigmod.org/sigmod-awards/**

[Last updated: December 31, 2020]

Editor's Notes

Welcome to the December 2020 issue of the ACM SIGMOD Record!

This issue features three articles in the Vision column. The first article, by Traub, Kaoudi, Quiané-Ruiz, and Markl, presents Agora, which is a vision toward a unified data-oriented asset ecosystem. The authors envision that in Agora, data, algorithms, and other fine-grained data-related units of production would all be first-class citizens that can be offered, discovered, and combined to form novel data-driven applications. Assets in Agora could be shared through marketplaces, combined and used through asset managers, and executed through execution managers. The article presents a carefully thought through and detailed list of open research challenges that need to be addressed to make the Agora vision a reality. The article is a call for action to the database community, which is well positioned to lead the efforts toward the vision of a unified data-oriented asset ecosystem.

The second article in the Vision column, by Liu, Barthels, Blanas, Kimura, and Swart, presents a new initiative in the context of state-of-the-art RDMA communication primitives. In this setting, the authors focus on the challenge created by data-intensive systems requiring higher-level communication abstractions that support more complex interaction patterns over networks than RDMA. The article details the shortcomings of the natural higher-level option of MPI, and then introduces the Remote Direct Memory Operation (RDMO) interface that permits key operations on remote memory to be executed in one roundtrip. The argument in support of the idea of RDMO includes order-of-magnitude speedup results shown by the authors in performing common database operations as RDMOs vs the alternatives.

The third article in the Vision column, by Karumuri, Solleza, Zdonik, and Tatbul, points to current experiences and challenges in the industry around understanding and resolving complex software problems. As exemplified by the use case of Slack, solutions capable of addressing these challenges would need to be able to handle large amounts of information to be analyzed. The authors bring up the notion of observability from control theory, as a requirement of improved visibility and understanding of complex software behaviors based on the information collected at runtime. The article proposes to consider observability as a data-management problem, and calls for database research in this emerging area. Specifically, the authors present a new cloud-native polystore architecture that decouples real-time and historical data-access tiers from the underlying persistent-storage and querying tier. The proposed architecture is designed in ways that enable scaling the tiers independently. The authors are working on the initial prototype of the architecture, with plans to test it with production data from Slack.

The Reports column in this issue features two articles. The first article, by Artikis, Eiter, Margara, and Vansummeren, reports on the seminar on the Foundations of Composite Event Recognition that was held in February 2020 at Schloss Dagstuhl, Leibniz Center for Informatics. The focus of the seminar was on composite event recognition (CER); the term refers to the process of matching patterns in streams of continuously arriving event data over geographically distributed sources. As CER is a key ingredient in many modern Big-Data applications, this research area has lately been attracting high interest from diverse communities. The existing projects have focused more on practical systems and less on formal foundations of CER; as a result, CER can be difficult to understand, extend, and generalize. The Dagstuhl seminar gathered 39 researchers and practitioners working on CER, in an effort to start addressing these challenges. The seminar featured six tutorials and hosted working groups devoted to five topics, including uncertainty in CER, benchmarking, and parallelization. The seminar identified future research challenges on the foundations of CER; more

events focused on CER are on the agenda. More information on the seminar is available in the Dagstuhl report.

The second article in the column is a special extended report on the experiences and lessons learned by EDBT and ICDT organizers from holding the conferences fully online and live in early 2020. These two conferences, held jointly, were among the first that moved to a fully synchronous online experience due to the COVID-19 situation. The authors describe the resulting experiment that came with potentially important lessons for the community and beyond. The important highlight is that overall, the online event ran smoothly and was significantly more enjoyable and successful than expected. The article shares the experiences and lessons learned for the benefit of organizers of other conferences facing similar situations. It includes a list of questions that other organizers are likely to case, outlines decisions and implementation descriptions, and shares the authors' thoughts in retrospect on the conference outcomes.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the December 2020 issue of the SIGMOD Record!

Your submissions to the SIGMOD Record are welcome via the submission site:
https://mc.manuscriptcentral.com/sigmodrecord

Prior to submission, please read the Editorial Policy on the SIGMOD Record's website:
https://sigmodrecord.org/sigmod-record-editorial-policy/

Rada Chirkova

December 2020

Past SIGMOD Record Editors:

Yanlei Diao (2014-2019)	Ioana Manolescu (2009-2013)	Alexandros Labrinidis (2007–2009)
Mario Nascimento (2005–2007)	Ling Liu (2000–2004)	Michael Franklin (1996–2000)
Jennifer Widom (1995–1996)	Arie Segev (1989–1995)	Margaret H. Dunham (1986–1988)
Jon D. Clark (1984–1985)	Thomas J. Cook (1981–1983)	Douglas S. Kerr (1976-1978)
Randall Rustin (1974-1975)	Daniel O'Connell (1971–1973)	Harrison R. Morse (1969)

Agora: Bringing Together Datasets, Algorithms, Models and More in a Unified Ecosystem [Vision]

Jonas Traub Zoi Kaoudi Jorge-Arnulfo Quiané-Ruiz Volker Markl

Technische Universität Berlin DFKI GmbH

ABSTRACT

Data science and artificial intelligence are driven by a plethora of diverse data-related assets, including datasets, data streams, algorithms, processing software, compute resources, and domain knowledge. As providing all these assets requires a huge investment, data science and artificial intelligence technologies are currently dominated by a small number of providers who can afford these investments. This leads to lock-in effects and hinders features that require a flexible exchange of assets among users. In this paper, we introduce Agora, our vision towards a unified ecosystem that brings together data, algorithms, models, and computational resources and provides them to a broad audience. Agora (i) treats assets as first-class citizens and leverages a fine-grained exchange of assets, (ii) allows for combining assets to novel applications, and (iii) flexibly executes such applications on available resources. As a result, it enables easy creation and composition of data science pipelines as well as their scalable execution. In contrast to existing data management systems, Agora operates in a heavily decentralized and dynamic environment: Data, algorithms, and even compute resources are dynamically created, modified, and removed by different stakeholders. Agora presents novel research directions for the data management community as a whole: It requires to combine our traditional expertise in scalable data processing and management with infrastructure provisioning as well as economic and application aspects of data, algorithms, and infrastructure.

1. INTRODUCTION

As data and data science technologies have become production factors, it is clear that they must be accessible by *everyone* [1]. Academia and industry have made progress towards the goal of providing access to data (e.g., [11, 20]), AI algorithms (e.g., [2, 4, 6, 19]), or computational resources [13]. However, the users still require significant expertise to combine all these *data-related assets* (assets, for short) from different marketplaces and cloud providers. For instance, a social scientist, who has no expertise in data science techniques and does not own any data, can hardly validate her assumptions about a social phenomenon, even if the required data and technology are theoretically available.

We envision *Agora*, an ecosystem that enables and eases the creation and composition of data science pipelines as well as their scalable execution. Agora provides *unified* access to all types of assets (e.g., data, algorithms, and compute resources) and treats them as *first-class citizens*: The social scientist in our example would not only find all relevant assets, but also executable compositions of them. This combination of abstraction and accessibility makes Agora attractive even for non-expert users. Agora brings together asset providers and consumers. It allows providers to offer any type of assets to a broader audience. For consumers, Agora provides access not only to data sources but to the entire data value chain.

We envision this ecosystem playing a dual role: (i) It is composed of a set of marketplaces where providers and consumers can exchange assets, and (ii) it provides the means to users to run their tasks (composition of assets) in Agora instead of using their own computing infrastructure. The key aspect of Agora is the fine-grained exchange of *any* asset. Each type of assets corresponds to a specialization of a provider, leading to different user roles. Agora hides the complexity of each role. For example, (i) a researcher can subscribe to an event stream without knowing details about the infrastructure that captures those events; (ii) a company can acquire a classification pipeline without understanding the details of all involved algorithms; (iii) researchers and companies can book a stream processing cluster with uptime guarantees without having knowledge on cluster operations; and (iv) system operators can focus on cluster monitoring and maintenance without knowing details about the running tasks.

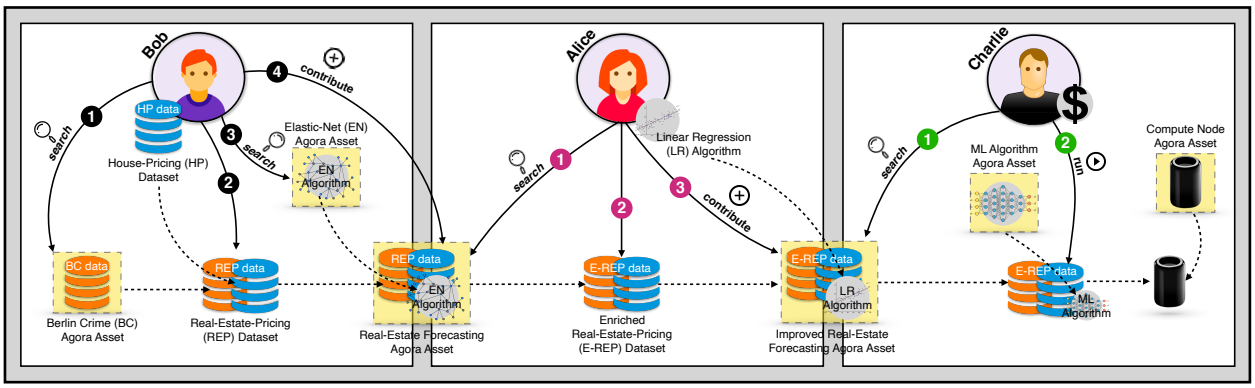


Figure 1: Motivating examples: Bob, Alice, and Charlie use Agora to discover assets, improve them, and contribute them back to the ecosystem. Agora also provides the infrastructure to run asset-based pipelines.

Related work. Most advanced data science pipelines require huge amounts of data, cutting edge data science innovations, and powerful computational infrastructure. Agora aims to connect providers and users of these key assets in an open ecosystem. In contrast, recent works tackle only parts of the solution provided by Agora: For example, OpenAI [21] primarily builds custom solutions and shares them via free software for training, benchmarking, and experimenting. Ocean Protocol [20] has similar goals with Agora but it focuses only on developing a decentralized protocol and network for data sharing. Datum [15] focuses on the privatization and secure storage of data sharing and proposes a network based on blockchain technology. Enigma [13] offers a protocol for computations on encrypted data by enabling computational resources to be shared securely in a decentralized manner. Nebula [18] forms a cloud of edge computers to perform distributed data-intensive computing. Data lake solutions, such as [16, 8], focus only on how to index and find datasets using metadata. In the space of machine learning, ML Bazaar [25] proposes a unified ML API to ease the development and sharing of ML algorithms. Agora goes beyond a simple abstraction to a holistic solution. There are also initiatives in providing marketplaces for sharing data [10, 12] and data science/AI solutions [4, 14, 19, 6, 2] as well as for storing data in a flexible manner [17]. However, their approach is single-facet, which makes it hard for users to combine them. The industry has also brought storage, computational, and cloud resources at the reach of the masses, such as Amazon EC2 and Microsoft Azure. Nevertheless, such cloud-based solutions result in lock-in effects: Users must stick to one provider for the entire pipeline of their solutions. We envision an open ecosystem where one can freely combine resources from different marketplaces. All above efforts go in the right direction for building a data ecosystem, however, it is still hard to combine them for devising end-to-end new solutions. Our work envisions a single data ecosystem where data, data science technologies, and storage/compute resources can seamlessly be combined to extract data insights.

Requirements. To see Agora become a reality, the following requirements should be met: (i) asset sharing and discovering – users should be able to easily provide or consume assets; (ii) asset privacy and security – users must be able to set privacy and security constraints to their assets; (iii) asset interoperability – users should be able to easily combine different (types of) assets; (iv) asset equivalence – users should be able to achieve their desired goals without being concerned about the specifics of the underlying algorithms; and (v) hardware independence – users should be able to run their assets on heterogeneous hardware seamlessly. We see Agora as an umbrella system, uniting all pieces of data management research. We believe that the database community should strive to realize this vision.

2. MOTIVATION

Imagine Bob, a freelance data scientist, who wants to create a machine learning (ML) model for real-estate price forecasting in Berlin (see Figure 1). His dataset is missing the criminality rate of each area, which he knows also affects the prices. He, thus, goes to Agora to find data about the crime rates in Berlin ❶. He finds the data, augments his initial dataset with this feature ❷, and builds an ML model using the elastic-net algorithm ❸. He then decides to provide his composed asset in Agora ❹. Bob's asset consists of the 'real-estate-pricing' dataset for Berlin and the elastic-net algorithm to estimate a potential price of apartments.

Alice, another data scientist, finds Bob's asset in

Agora ❶ and decides to improve it ❷. She enriches the original 'real-estate-pricing' dataset with several feature engineering techniques, adds the 'linear-regression' algorithm for prediction, and contributes it back to Agora to gain some revenue ❸.

Charlie, a consumer who is looking for a real-estate pricing predictor, queries Agora for available assets on price forecasting that yield the average error rate below 5,000 euros ❶. As he does not have the infrastructure to run assets, he decides to use Agora to execute his discovered assets (e.g., train the found ML pipelines) ❷. Although he wants to complete the training as fast as possible, his budget is limited. To overcome his budget limitation, Agora recommends to replace the linear regression algorithm by a logically equivalent neural network implementation having lower license cost. Next, Agora decides to run the resulting asset on an execution node registered within Agora. The latter can be a machine of a cloud provider registered within Agora or an idle cluster of an individual user. This enables users to gain some revenue by offering personal compute resources in competitive prices.

Allowing asset exchange in Agora leads to the following main benefits:

(1) Secondary use of existing assets. Users can reuse any (composed) asset (e.g., data and algorithms) offered in Agora. In most cases, companies own a plethora of highly valuable assets. However, as these assets are fragmented across companies, their economical potential remains unused as secondary asset usage is extremely rare. Similarly individuals can offer any asset, including compute resources, with the goal of gaining revenue. A fine-grained asset sharing would allow for combining existing resources to derive new insights and services.

(2) Leveraging specializations. Agora creates an ecosystem of highly specialized providers who provide assets of a very high quality. Such an ecosystem is comparable with the automotive industry where many companies specialize in certain parts (e.g., brakes, tires, or lights), which get combined to one high-quality car. This enables small and medium-sized companies to offer assets that they would not be able to bring in the market otherwise.

(3) Hiding complexity. Agora hides the complexity and intricacies of assets from the consumers. It is aware of logical equivalence of assets, i.e., assets that yield the same results (e.g., a nested loops join is equivalent with a hash join for equi-joins). Implementations of logically equivalent assets can have very different properties: They may use different programming languages (e.g., C++ and Java), be tailored to different systems (e.g., Flink and Spark),

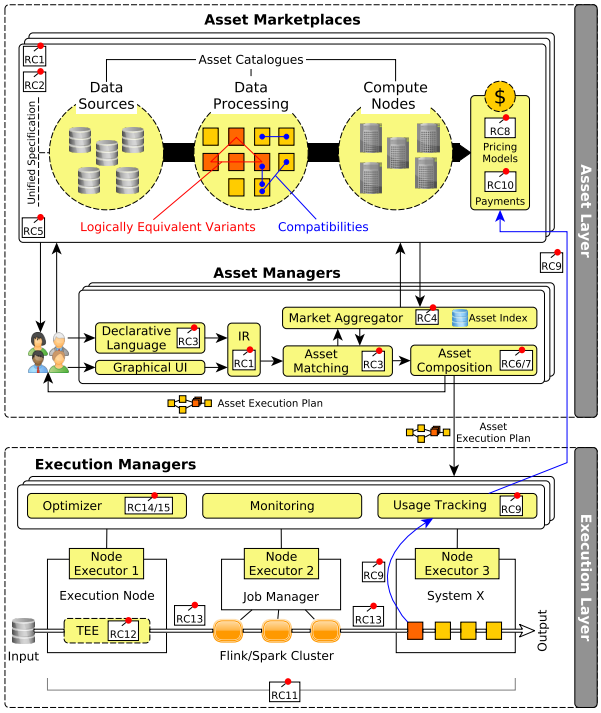


Figure 2: An overview of the architecture of Agora with 15 selected *Research Challenges (RCs)*.

be optimized for specific hardware (e.g., CPU and GPU), and run in a parallelized, distributed, or sequential setting. In addition, each provider can define different pricing for her implementation. To optimize execution, Agora chooses the best combination out of the available implementations based on the requirements of the incoming task.

3. AGORA ARCHITECTURE

Agora builds around assets which we define as *any data-related unit of production that allows users to exploit the value of data.* We identify six major categories of assets: *data sources*, *algorithms*, *pipelines*, *storage and compute resources*, *systems*, and *applications*, discussed in more detai in [26].

Agora consists of two layers (Figure 2): the *Asset Layer* and the *Execution Layer*. In this way we decouple the execution infrastructure from the platform for asset composition and discovery as they come with different requirements. Still, a major strength of Agora is its seamless connection between these two layers. It goes beyond stand-alone marketplaces, stand-alone execution engines, and cloud services with the goal of facilitating the use of DS tools for a broader group of users.

The **asset layer** constitutes an "intelligent" ecosystem of multiple *asset marketplaces* and en-

ables not only offering and finding assets but also composing them in a smart way via *asset managers*. Recall our motivation example described in Section 2. Bob, who is searching for a dataset, has the choice of going directly to his favorite marketplace or to an asset manager to find his desired dataset. In the former case, he either browses the marketplace or uses keywords to search within it. In the latter case, he specifies his request in a declarative manner and the asset manager responds by potentially accessing multiple marketplaces.

The **execution layer** optimizes and runs asset execution plans via *execution managers* and *node executors*. For instance, Charlie, in our running example, finds his pipeline via an asset manager and decides to execute it in Agora. For this reason, the asset manager translates the pipeline into an execution plan together with its equivalent assets, which are logically equivalent variants satisfying the same request. Logically equivalent variants can be different physical implementations of the same logical operator, alternative compute nodes with similar properties, or alternative data sources, such as weather data from different providers. Next, the asset manager passes the execution plan to an execution manager, which optimizes the plan and finds the best possible asset options that respect Charlie's budget. The execution manager accesses processing nodes through a *node executor*, which is a standardized component to interface arbitrary execution environments with execution managers. For example, NodeExecutor 1 in Figure 2 provides access to a Trusted Execution Environment (TEE), such as an Intel SGX Enclave [9], and NodeExecutor 2 provides access to a Flink or Spark cluster.

In Table 1, we show an overview of research challenges (RCs) present in the two layers and also illustrated in Figure 2. We offer a detailed discussion of all RCs in [26]. In the next sections, we present the two layers of Agora referring to all our RCs.

Table 1: Overview of Research Challenges in Agora. A detailed discussion of all challenges can be found at: https://arxiv.org/pdf/1909.03026.pdf [26]

Asset Layer	**RC1: Unified specification.** The design of a unified specification (a standard) for assets. Such a specification is the precondition for sharing, searching, and discovering assets in the ecosystem.
	RC2: Automated specification generation. Asset specifications should automatically be extracted to prevent overhead from providers. Such an automated extraction opens up a new research direction.
	RC3: Matchmaking. Agora should effectively and efficiently identify required assets based on consumer requests. It is a challenge to formalize such requests and match them with assets.
	RC4: Market aggregator. Agora is composed of independent asset marketplaces. It is important for users to be aware of the different marketplaces and their assets through a market aggregator.
	RC5: Constraint specification. Enable the declarative specification of constraints which define the rules for asset sharing and execution.
	RC6: Constraint satisfaction. Efficiently process queries in a manner compliant with respect to asset constraints. Our initial implementation allows expressing constraints on shipping data across geographical borders using extended-SQL statements.
	RC7: Capturing asset provenance. In order to audit compliance with respect to data usage and its sharing policies, we need a technique for capturing provenance in an asset-centric marketplace.
Execution Layer	**RC8: Pricing models.** Our ecosystem should allow providers to define prices of their assets based on different pricing models. Agora should also propose prices based on market monitoring.
	RC9: Asset usage tracking. To ensure fair asset payments, the execution manager needs to track the usage of the assets. Tracking fine-granular operations in a set of assets (e.g., in a pipeline), which may run in parallel, is a challenging task.
	RC10: Payments. Ensuring a safe way to charge and pay the use of assets is crucial for the ecosystem health. A payment process should be distributed such that components can receive payments and split them among their sub-components.
	RC11: Establishing trust through certificates. We need a way to establish trust through certificates in an ecosystem with an extremely large number of actors.
	RC12: Trusted execution environments. A Trusted Execution Environment (TEE) provides a solution for secure computation, which does not require to trust the owner of a compute node. We explore TEE-based solutions in the context of a large data ecosystem.
	RC13: Secure data transfer. We need to enable secure data exchange in Agora. Therefore, we investigate techniques which enable data trading, ensure encryption, and guarantee data integrity.
	RC14: Heterogeneous asset deployment. We want to determine the ideal deployment environment, i.e., the processing system for deploying each asset. For example, if the asset is a stream processing algorithm, Agora might decide to run it on Apache Flink
	RC15: Heterogeneous asset execution. We want to automatically assign assets to compute resources. E.g., matching processor-specific algorithm implementations with the respective processors.

4. ASSET LAYER

The asset layer consists of many connected *asset marketplaces* and *asset managers*. The former allow for sharing assets, while the latter allow for discovering assets across multiple marketplaces.

Each **asset marketplace** contains catalogues that keep track of the available assets and their properties. To make this possible, Agora unifies assets under a common specification which enables easy asset discovery and composition across all the marketplaces in the ecosystem. Providers should conform with this unified specification when they offer new assets to the marketplaces. This can be

a barrier for new asset providers. Therefore, it is crucial that Agora provides the means for automatically generating asset specifications from more intuitive user inputs, such as query and programming languages or graphical interfaces. Defining such a specification and determining ways for its automated extraction is challenging due to the the large heterogeneity of assets (*RC1, RC2*). Our initial efforts [22] towards a unified specification is a declarative intermediate representation of data science assets which is automatically extracted from Python code using static code analysis.

Providers can also define a pricing model (e.g., subscriptions or pay-per-use) for their assets usage (*RC8*). Ideally, Agora proposes a pricing model and a price based on monitoring the current trend of the market. When a provider chooses a pay-per-use pricing model, Agora ensures to track the asset's usage and report usage counters back to marketplace (*RC9*). Marketplaces then perform the invoicing and initiate (micro-)payments between users (*RC10*).

Asset managers are the entry point for users who want to declaratively: find assets across different marketplaces; combine multiple assets into execution plans; and run asset execution plans. An asset manager provides a graphical user interface and/or a declarative language for finding and composing assets (*RC3*). A user request is then converted to an intermediate representation (IR), which allows for matching asset specifications with user requests (*RC1*). The asset manager matches user requests to assets that are compatible with each other and satisfy the requests (*RC3*). For this, it needs to aggregate the assets of all marketplaces and build an asset index (*RC4*). Next, the asset manager composes all the relevant assets (with their equivalent assets) such that they fulfill the request. As a result, the asset manager outputs an asset execution plan, which allows the execution layer to further optimize, deploy, and run the plan.

Asset provenance and usage constraints are crucial in this context: We address these points in RC5, RC6, and RC7. Providers can specify usage constraints to their assets, such as location requirements (e.g., private data may not be moved out of a country) or vendor requirements (e.g., my algorithm may not be used by a competitor). Identifying the best way to describe constraints over assets is an interesting research challenge because of the asset heterogeneity and different constraint granularity (*RC5*). This opens up a completely new dimension of "compliant query (asset) processing" that aims at finding efficient ways to process requests in a compliant manner respecting the assets constraints (RC6). In our efforts towards realizing Agora we provide support for compliant geo-distributed query processing. Our initial implementation allows expressing constraints on shipping data across geographical borders using our extended-SQL statements. A query optimizer aims at finding distributed execution plans that are compliant with respect to shipping of intermediate data between compute sites. This concept can be generalized to model other types of constraints as well. For example, a constraint could require an ISO certification for compute nodes, which would prevent assets from being moved out of ISO-certified data centers.

In RC7, we address the challenge of capturing asset provenance in an open asset ecosystem. This is important to ensure compliance with regulations such as *the right to be forgotten* in GDPR and CCPA. We envision a solution based on a provenance graph which allows for tracing back the source(s) of each asset. Combined assets will thereby inherit the graphs of their sub-assets, forming a new, joined graph. By navigating in the provenance graph, one will be able to trace the use of each asset and to withdraw assets if needed. To ensure that withdrawn assets cannot be accessed any more, we plan to explore techniques, which allow for expiring digital assets using encryption keys [5].

5. EXECUTION LAYER

Agora's execution layer consists of *execution managers*, which receive execution plans, and *node executors*, which run consumers' assets.

An **execution manager** is a core component of the execution layer. It is responsible for optimizing an asset execution plan, deploying it on compute nodes, and monitoring its execution. As the plan may contain different variants of operations, the execution manager can schedule an operation of an execution plan on different execution environments (node executors). Achieving this multi-environment execution of a plan is very challenging as the search space of all possibilities to execute a plan becomes very large (*RC14* and *RC15*). The selection of existing variants and the selection of node executors goes hand-in-hand with possible algorithm adaptations, which increases the performance on a particular target system. We already made the first step towards this direction with Rheem [3] and have shown that using multiple data processing platforms significantly decreases the execution time of a single processing task. However, considering highly diverse assets is still an open research problem.

In addition to determining which processing sys-

tem to execute an asset, Agora also determines how to allocate the asset to compute resources. Agora must adapt algorithms to the specific processor they run on to exploit the full potential of heterogeneous computing resources. For this, we must automatically generate such processor-specific algorithm implementations. Our previous work [7, 23, 24] demonstrates that this is indeed feasible: Data processing systems can learn processor-specific implementations during installation or at runtime.

A **node executor** is Agora's interface component to connect arbitrary execution environments with execution managers. For example, in Figure 2 the asset execution plan is deployed to three node executors with different characteristics: NodeExecutor 1 provides access to a trusted execution environment (TEE), which provides additional security because the owner of the node has no access to the executed source code nor the processed data (*RC12*); NodeExecutor 2 provides access to a Flink or Spark Cluster; and NodeExecutor 3 provides direct access to hardware resources on a dedicated server. When dealing with multiple node executors, Agora provides a secure way to transfer data among nodes to validate data integrity and to pay for data that is traded as an asset. This is hard to achieve especially when data is large or data streams have high bandwidth (*RC13*). Note that we provide secure data transfer for transmitting data in processing pipelines. However, in general, Agora tries to avoid data transfer if possible. For example, consider a classification task over a data stream. Agora can compute a model on one provider's infrastructure, then only transfer the model to another provider, and use the model to classify items from a high bandwidth stream at this other provider. The size of the transferred model would be small, but the underlying data processed at each provider can be large.

Node executors and execution managers are responsible for tracking the usage of assets, which is crucial to ensure fair payments. This is a challenging task because it also assumes tracking fine-granular operations in a composition of assets (*RC9*). Agora adopts certificates to ensure transparency and trust between consumers and providers. For example, one can certify the physical location of a node, security standards, compliance with asset usage tracking, or energy efficiency. The main challenge remains in the standardization of certificates and assets requirements (*RC11*).

6. CONCLUSION

We presented Agora, our vision towards a unified asset ecosystem where assets are fine-grained data-related units of production, such as data and algorithms. Agora provides the technical infrastructure for offering, discovering, and combining assets to form novel data-driven applications. One can share assets through marketplaces, use and combine them through asset managers, and execute them through execution managers. We pointed out multiple open research challenges that need to be addressed to make such an asset ecosystem a reality. This paper is a call for action as we believe that the database community is well positioned to lead the efforts towards such a unified asset ecosystem.

Acknowledgments: This work has been supported by the German Ministry for Education and Research as BIFOLD (01IS18025A, 01IS18037A)

7. REFERENCES

[1] D. Abadi et al. The Seattle Report on Database Research. https://db.cs.washington.edu/events/other/2018/Seattle_DBResearch_Report-Full.pdf.
[2] Acumos. https://www.acumos.org.
[3] D. Agrawal, S. Chawla, B. Contreras-Rojas, et al. Rheem: enabling cross-platform data processing – may the big data be with you! In *PVLDB*, 2018.
[4] AWS Marketplace. https://aws.amazon.com/ marketplace.
[5] S. Blumenau and M. Barnes. Systems and methods for expiring digital assets using encryption key, May 18 2006. US Patent App. 11/282,463.
[6] Bonseyes. https://bonseyes.com/.
[7] S. Breß, B. Köcher, H. Funke, et al. Generating custom code for efficient query execution on heterogeneous processors. *VLDB Journal*, 27(6):797–822, 2018.
[8] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, pages 1001–1012, 2018.
[9] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, (086):1–118, 2016.
[10] Data Space. https://www.datapace.io.
[11] Datahub. https://datahub.io.
[12] Datawex. https://www.dawex.com.
[13] Enigma. https://enigma.co.
[14] G Suite Marketplace. https://gsuite.google.com/marketplace.
[15] R. Haenni. Datum network - the decentralized data marketplace - white paper v15. 2017. https://datum.org/assets/Datum-WhitePaper.pdf.
[16] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD*, page 795–806, 2016.
[17] A. Jindal, J. Quiané-Ruiz, and J. Dittrich. WWHow! Freeing Data Storage from Cages. In *CIDR*, 2013.
[18] A. Jonathan et al. Nebula: Distributed edge cloud for data intensive computing. *IEEE TPDS*, 2017.
[19] Microsoft Azure. https://azure.microsoft.com.
[20] Ocean Protocol. https://oceanprotocol.com.
[21] OpenAI. https://openai.com.
[22] S. Redyuk. Automated documentation of end-to-end experiments in data science. In *ICDE*, 2019.
[23] V. Rosenfeld, S. Breß, S. Zeuch, T. Rabl, and V. Markl. Performance analysis and automatic tuning of hash aggregation on gpus. In *DaMoN*, 2019.
[24] V. Rosenfeld, M. Heimel, C. Viebig, and V. Markl. The operator variant selection problem on heterogeneous hardware. In *ADMS*, 2015.
[25] M. J. Smith et al. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. In *SIGMOD*, 2020.
[26] J. Traub, Z. Kaoudi, K. Beedkar, et al. Agora: A unified asset ecosystem going beyond marketplaces and cloud services. https://arxiv.org/abs/1909.03026, 2020.

Beyond MPI: New Communication Interfaces for Database Systems and Data-Intensive Applications

Feilong Liu[1]*, Claude Barthels[2]*, Spyros Blanas[1], Hideaki Kimura[3], Garret Swart[3]
[1]The Ohio State University, [2]ETH Zurich, [3]Oracle Corp.
[1]{liu.3222, blanas.2}@osu.edu, [2]claudeb@inf.ethz.ch, [3]{hideaki.kimura, garret.swart}@oracle.com

ABSTRACT

Networks with Remote Direct Memory Access (RDMA) support are becoming increasingly common. RDMA, however, offers a limited programming interface to remote memory that consists of read, write and atomic operations. With RDMA alone, completing the most basic operations on remote data structures often requires multiple round-trips over the network. Data-intensive systems strongly desire higher-level communication abstractions that support more complex interaction patterns.

A natural candidate to consider is MPI, the de facto standard for developing high-performance applications in the HPC community. This paper critically evaluates the communication primitives of MPI and shows that using MPI in the context of a data processing system comes with its own set of insurmountable challenges. Based on this analysis, we propose a new communication abstraction named RDMO, or Remote Direct Memory Operation, that dispatches a short sequence of reads, writes and atomic operations to remote memory and executes them in a single round-trip.

1. INTRODUCTION

Popular high-speed interconnects such as InfiniBand offer Remote Direct Memory Access (RDMA) support to accelerate communication. Using RDMA in a database system is not straightforward: database systems need to organize their data in such a way that direct data access mechanisms can be employed [5]. This entails redesigning data processing algorithms [2], data structures [17], and communication mechanisms [10, 15]. The Oracle database has been using RDMA for performance and scalability, but its use has been limited to operations that do not require many rounds of messaging, such as accessing the Commit Cache and the Undo blocks during transaction logging [16]. The fundamental challenge is the limited programming surface of RDMA consisting only of message passing, read/write, and single-word atomic operations.

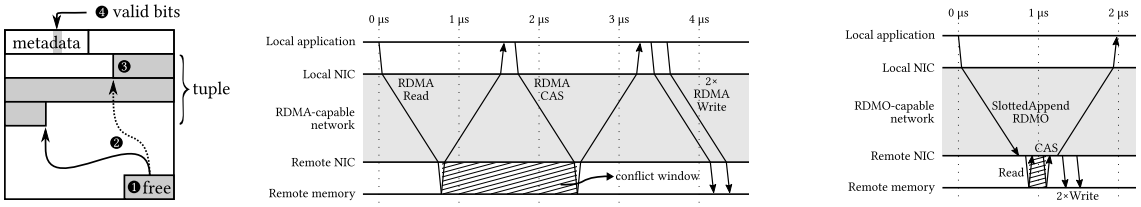
*Contributed equally.

A natural higher-level alternative to consider is MPI, the de facto standard for developing portable and highly parallel applications in the HPC community. MPI has been used in distributed database systems [6] and to implement scalable join algorithms [3]. At a first glance, one is inclined to believe that MPI would be a good fit for data processing. However, Section 2 shows that MPI is profoundly unsatisfactory for database systems:

- (§2.1) MPI has a process-centric model that lacks the concept of a thread for multi-threading.
- (§2.2, §2.3) Many MPI operations can only be performed synchronously (collectively) between a group of processes. Implementations often use barrier-like synchronization which exposes communication delays and underutilizes the CPU. In addition processes cannot dynamically join and leave groups, which is needed for operations whose communication pattern is determined by the data.
- (§2.4) For correctness, remote memory operations need to either acquire coarse-grained locks or manually serialize the execution of operations, akin to flushing I/O buffers to persistent storage.
- (§2.5 – §2.7) MPI cannot (1) notify the remote side of the completion of a memory operation, (2) convey quality of service (QoS) and traffic differentiation information, and (3) support elasticity, fault tolerance and high availability.

Instead of adopting MPI, our idea is to augment RDMA to support the dispatch of simple data processing logic in one "unit" to the remote side which will be executed in one round-trip. We term this a Remote Direct Memory Operation, or an RDMO.

The RDMO abstraction overcomes limitations of RDMA when manipulating data structures. Consider the common database operation of inserting a tuple in a slotted page, shown in Figure 1(a). A latch-free algorithm checks if there is enough free space, then atomically modifies the pointer to the "free" segment of the page, writes the data, and fi-



(a) Append to slotted page. (b) Append using RDMA. (c) Append using RDMO.

Figure 1: A Remote Direct Memory Operation, or RDMO, dispatches data processing logic to the remote side in one "unit".

nally marks the entry as valid to read. An RDMA implementation issues the sequence of requests shown in Figure 1(b). The RDMO implementation issues a single `SlottedAppend` RDMO as shown in Figure 1(c). In this example, the RDMO abstraction reduces the number of transmitted messages by 4× and reduces latency by as much as 2×. Furthermore, the window of a conflict for the atomic operation is now reduced by 10×. Section 3 introduces more RDMO examples.

Prior research has proposed other programming models and abstractions to overcome the shortcomings of MPI. Global Arrays [7] is a programming model that presents the view of consistent global memory. Like MPI, it is limited to one-sided Get, Put, Accumulate and atomic operations that are oblivious to the payload. A more recent effort is DPI [1], an interface definition for modern networks with "flow" and "memory" operations. Like MPI, DPI does not define mechanisms for pushing computation into the network. We envision that RDMOs will be one such mechanism.

2. A CRITIQUE OF MPI

The Message Passing Interface (MPI) is a communication interface that allows HPC applications to communicate in an efficient and portable manner. We evaluate two popular implementations of MPI, MVAPICH 2.2 [12] and OpenMPI 2.0 [13], on a cluster connected with InfiniBand FDR (56 Gbps). We focus on instances where the MPI abstractions are unsatisfactory for data management.

2.1 Multi-threading support

The unit of parallelism in MPI is a process. The MPI library controls the binding of processes to compute cores and hides this binding behind a namespace where processes are addressed solely through their rank number. However, in a database system, the sender of a message often targets a recipient based on its position in the local compute topology, such as when transmitting to a specific NUMA node or thread. No mechanism in MPI can address a message to a specific thread of an MPI process.

The performance of multi-threaded programs suffers due to contention inside the MPI library. Figure 2 shows the throughput of MVAPICH on a cluster with two nodes, where one node keeps sending data to the other with both one-sided and two-sided functions. In the "multi-process" result, each node runs 20 separate MPI processes. In the "multi-thread" result, each node runs 1 MPI process with 20 threads. The multi-threaded MPI performance is 2× to 20× slower than multi-process MPI.

2.2 Communication groups

Every communication in MPI targets a specific *communication group*. Within a communication group, the unit of parallelism is a *process* which is identified by a group-specific integer called a *rank*. To manage communication, database systems have to use different communication groups per operation. Furthermore, the communication patterns in a database system are often data-dependent. A weakness of MPI is that it does not permit processes to join or leave a communication group at runtime. Creating new groups at runtime is inefficient: the creation time is about 30ms with 4 nodes (60 processes) per group. As a result, MPI groups need to be generated conservatively to include every process that could potentially contribute to the result. MPI processes that have no data to contribute still have to participate in remote operations with zero-sized payloads for synchronization purposes.

2.3 Blocking collective operations

Many MPI operations must be executed synchronously by every process. MPI refers to these operations as *collective operations*. Some collective operations are implemented as blocking calls, meaning that a process is only allowed to continue once all other processes have executed the operation as well. Examples of collective operations include reduce, gather, scatter, and broadcast. Management operations, such as window allocation and deallocation, are also implemented as collective operations.

In the context of data management, using blocking collectives is problematic due to the inability

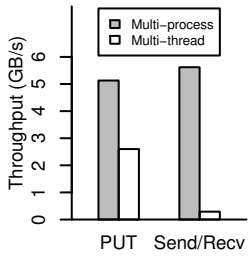


Figure 2: MPI performs poorly with multiple threads.

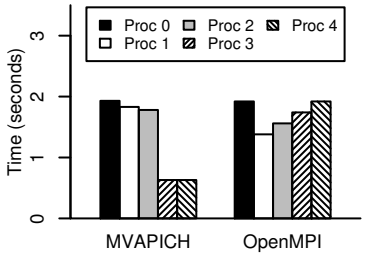


Figure 3: Collective MPI calls lead to unpredictable performance.

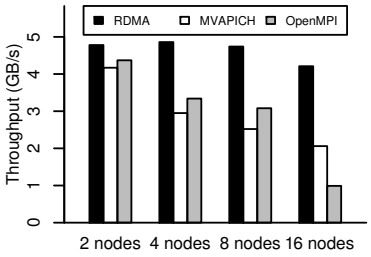


Figure 4: Bypassing MPI and directly using RDMA has better scalability.

of MPI implementations to interleave communication with computation. This is exacerbated if processes manipulate different amounts of data due to skew. Consider an `MPI_Gather` collective operation among 4 processes. Assume processes 1–4 call `MPI_Gather` to send 1 GB of data to process 0. However, processes 1–2 are late in calling `MPI_Gather`, while processes 3–4 call `MPI_Gather` immediately. Figure 3 shows the time when each process returns from the collective call. In the MVAPICH implementation, `MPI_Gather` is not blocking, hence process 3 and 4 exit the collective call earlier than process 1 and 2. However, in the OpenMPI implementation, `MPI_Gather` is blocking, so processes 3–4 are blocked — in fact, they finish after processes 1–2 complete despite starting first. This wastes valuable cycles waiting for the collective operation.

2.4 Synchronizing concurrent accesses

RMA operations in MPI are performed on memory regions referred to as *window* objects. A window is a collection of memory regions on which write ("put") and read ("get") operations can be executed. MPI provides two mechanisms to synchronize concurrent accesses to the same window.

In active target synchronization, both the origin and the target synchronize the remote memory access by adding a fence ("epoch") between RMA operations. Pending operations are only guaranteed to be visible after the epoch completes.

In passive target synchronization, MPI provides a coarse-grained locking mechanism. Before a remote memory access can take place, the entire window must be locked. These accesses are guaranteed to have completed only when the window is unlocked. MPI supports two locking modes: an exclusive lock will only grant access to a specific process to modify or read the content of the window, whereas a shared lock allows for concurrent accesses.

The problem of both concurrency control mechanisms is their granularity: locking an entire window is too coarse-grained and defining every operation in terms of epochs is too fine-grained. MPI lacks the sophistication of multi-level locking [8] that lets the system determine the granularity of access control.

2.5 Composability and low-level access

MPI assumes total control of the underlying hardware and it cannot co-exist harmoniously with low-level network access mechanisms. Many low-level optimizations are mutually exclusive with using MPI. Two examples are:

1. **Unreliable datagrams:** A database system may not require ordered data delivery if it is evaluating a query that does not rely on a sorted order. In this case, it is acceptable for the network to deliver messages out of order. Database systems can unlock better performance and scalability by using the Unreliable Datagram (UD) transport [9, 10], a datagram-based protocol similar to UDP in Ethernet. Figure 4 shows the throughput of a repartitioning operation that transmits each tuple to a destination node based on the hash value of its key. Using the UD transport in InfiniBand has better performance and scalability than the Reliable Connection (RC) transport that is used by many MPI implementations.
2. **Notified one-sided access:** When executing a query or a transaction, the database system can perform one-sided RMA operations to place data at specific locations. For many algorithms, merely writing the data to remote memory is insufficient and specific operations need to be triggered on the remote side after the data has been transmitted. Many network interfaces support notified one-sided RMA accesses to solve this problem. Exposing this functionality in MPI is a topic of active research [4], as notified one-sided accesses are not supported by the MPI standard.

2.6 Traffic differentiation

Many interconnects have the capability to provide better service to selected network traffic. For example, InfiniBand includes quality of service (QoS)

Table 1: MPI does not differentiate traffic.

	MVAPICH	OpenMPI	RDMA
Short message latency (msec)	186.08	0.06	0.03
Bulk transfer latency (msec)	186.08	167.92	188.99

mechanisms inherently and offers flow prioritization. Traffic differentiation is crucial for database systems as some communications are latency-critical, such as messages for algorithmic coordination or transaction processing, while many analytical workloads can tolerate higher latency in exchange for high-bandwidth transfers. However, MPI lacks mechanisms to associate quality of service information to different requests. Database systems are thus relying on the intricacies of the specific implementation of MPI they use for timely message delivery.

To quantify this limitation, we initiate a bulk data transfer (1 GB) and a small latency-critical message transmission (32 KB) using asynchronous send/receive calls. As seen in Table 1, in MVAPICH the small message is transmitted after the large data transfer, while both OpenMPI and RDMA transmit the small message earlier. This experiment shows that the order in which concurrent messages are delivered is implementation-dependent and cannot be controlled by the database system.

2.7 Elasticity, availability, fault tolerance

The degree of parallelism in MPI is specified when the application starts and it is fixed for the entire duration of the program. MPI cannot scale in or scale out a parallel database deployment, as it lacks the functionality to add and remove processes from a running application.

In addition, many HPC applications are started with a specific lifespan in mind which rarely exceeds several hours. MPI does not offer fault-tolerance features as it is expected that every program would terminate at some point anyway. (In fact, the default error handler in MPI terminates the entire program when a single process fails.) This application-centric execution model is radically different from the service-centric model that expects nodes to fail during the lifespan of a typical deployment. In other words, a database system should outlast the hardware it is currently running on. In this model, failures have to be contained and recovered from, and the communication interface needs to support fault tolerance and high-availability mechanisms.

3. A NEW ABSTRACTION: RDMO

This section introduces a new abstraction for data-intensive applications, the Remote Direct Memory Operation (RDMO) interface. An RDMO is a request that triggers the execution of a short sequence of reads, writes and atomic memory operations that will be transmitted and executed at the remote node without interrupting its CPU.

The RDMO communication pattern bridges the gap between direct memory accesses (RDMA) and remote procedure calls (RPC). Unlike RDMA and its fixed set of verbs, the RDMO interface can both transmit data and execute simple operations on a remote node. Unlike an RPC call, the RDMO interface limits the ability of the remote operation to execute too many instructions, access too much data, or block for too long.

3.1 Implementing RDMOs

The RDMO interface builds on RDMA and reuses the queue pair (QP) and completion queue (CQ) objects. Like one-sided RDMA verbs, RDMOs are transported over a Reliable Connection. Transport errors and certain operation completion codes can trigger the server to close the connection.

The RDMO interface deviates when it comes to the processing of each message, which is shown in Figure 5. Instead of targeting a memory location with byte-level reads and writes, an RDMO request targets an abstract data type (ADT) which is identified using a capability (a secret identifier). The RDMO operation must map to a valid method of the abstract data type. Each operation accepts an input byte array, and returns a completion code and an output byte array to the CQ. The format and meaning of the byte arrays are determined by the abstract data type method.

When an RDMO-capable endpoint receives the RDMO request, it is placed in a queue of incoming operations. If the queue is full, the RDMO is rejected and the client CQ receives an error. The RDMO is processed when it reaches the head of the queue. Processing an RDMO starts by mapping the capability to an instance of an abstract data type. If there is no such instance, or the identifier does not map to a method of the abstract data type, the operation returns with an error. Otherwise, the appropriate ADT method is invoked.

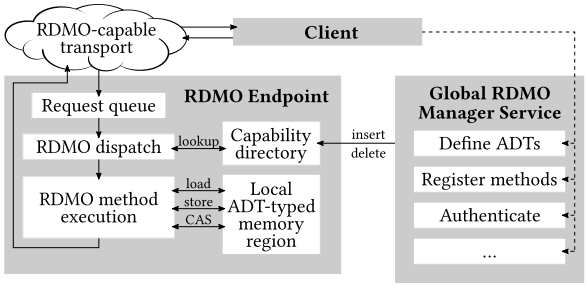


Figure 5: Implementing RDMOs.

The implementor of an RDMO method must certify that it is non-blocking. Each RDMO method has a fixed maximum resource consumption (in cycles) that is provided by the implementor when the RDMO method is registered on the server. Knowledge of the maximum RDMO resource consumption allows the RDMO-capable endpoint to bound the RDMO wait and service time. This avoids the need of timer signals or call progress acknowledgements as part of RDMO transport and execution, and reduces the overhead on the client and the server.

Limiting RDMO methods in this way during registration achieves higher availability, because the completion of an RDMO does not depend (wait) on other activities of the system. In addition, since the RDMO implementation can be fully determined and inlined, it can run outside of the typical application environment. This acts as a layer of abstraction between the endpoint capabilities and the endpoint implementation: an RDMO-capable endpoint can run as a separate user process, an interrupt handler in the OS kernel, a separate VM sharing memory, an FPGA, or even a special purpose chip.

We emphasize that the RDMO interface is not meant to replace existing RDMA or RPC interfaces. To the contrary, the RDMO interface depends on RDMA and RPC: First, the RDMO interface reuses key components of the RDMA mechanism. Second, the RDMO interface requires a higher-level RPC facility that defines new ADT instances, authenticates agents, and authorizes the use of a given ADT by creating a RDMO capability.

3.2 Attributes of RDMOs vs. MPI

RDMOs are embedded into a larger context of a network interface with the following properties:

Multiple Threads: As shown in Section 2.1, multi-threaded MPI programs suffer from poor performance. MPI connections are bound to processes, thus making the process rank both a unit of computation and an address for data accesses. The RDMO interface implements endpoints that are independent of the compute components instead.

Non-Blocking Operations: Section 2.3 shows that the behavior of MPI depends on the specific implementation and the blocking implementations fails to interleave the communication and computation. All functions of the RDMO interface are non-blocking such that the initiator of an operation can interleave communication and computation.

Quality of Service: Section 2.6 shows that MPI cannot prioritize different communication flows. However, database systems have to support many types of workloads that exhibit different communication patterns and some flows need to be prioritized. The RDMO interface provides QoS functionality using the RDMA QoS semantics.

Fault Tolerance: Section 2.7 highlights the lack of fault tolerance mechanisms in MPI. However, database systems offer services that must be highly available. The RDMO interface has a clearly-defined failure model, inherited by the RDMA interface, and allows applications to react to failures.

Schema awareness: Database systems operate on structured data. Pushing down schema information to the network using the abstract data type (ADT) support of RDMOs enables novel in-network processing applications and operations.

Conditional operations: Conditional operations allow the developer to evaluate simple *if-then* operations in the remote node. For an RDMO operation with condition check, the remote network card first evaluates if the remote data is in the specified state before applying the operation. This eliminates several round trips and reduces the need for running expensive synchronization or agreement protocols.

Table 2: Attributes of RDMO operations.

RDMO	Schema aware?	Conditional?	Message savings
SlottedAppend	×	✓	4× or more
ConditionalGather	✓	✓	Up to 30×
SignaledRead	×	✓	3× or more
WriteAndSeal	×	×	2×
ScatterAndAccumulate	✓	✓	Up to 30×

3.3 Five database RDMOs

This section presents five common operations in database systems that can be accelerated using the RDMO interface. The attributes of the five RDMO functions are summarized in Table 2. "Message savings" represents the number of messages saved compared with an RDMA implementation of the same function, if one assumes the same number of scatter/gather entries per request (30) as provided by modern InfiniBand network cards. These operations are used to demonstrate the potential of the RDMO interface and are not an exhaustive list.

1. SlottedAppend. This is the common operation of appending a tuple in a buffer, which has been highlighted in Figure 1(a) in Section 1. In case of contention, lock conflicts will increase exponentially. Performing this operation as an RDMO shrinks the conflict window and permits more concurrency under contention.

2. ConditionalGather. Traversing common data structures often requires following pointers. This requires several lookups over RDMA. This RDMO traverses pointer-based data structures, evaluates a user-defined predicate and gathers the matching

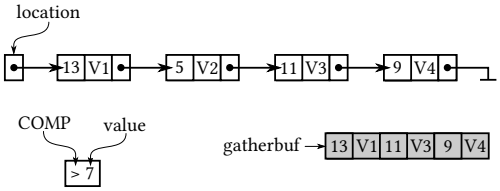


Figure 6: `ConditionalGather` **follows a linked list of versions, performs visibility checks and returns the matching tuples in one operation.**

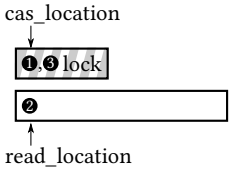


Figure 7: `SignaledRead` **can elide a lock when reading any lock-based data structure in one request.**

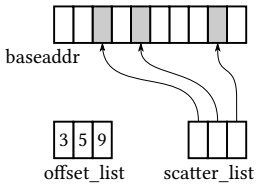


Figure 8: `ScatterAndAccumulate` **accumulates elements in the list at user-defined offsets.**

elements in a buffer that is transmitted back in a single request. Up to 30 elements (the gather list) can be retrieved in a single RDMO request, instead of one message per element with RDMA.

The `ConditionalGather` RDMO would be useful in OLTP workloads when reading versioned tuples, as shown in Figure 6, where it compares timestamps for visibility checking and only returns visible versions in one operation. In an OLAP workload that involves a join, this RDMO retrieves all tuples in a hash bucket that match a key in one round-trip. Short-circuiting the conditional operation performs projection in the network.

3. SignaledRead. Many data structures use locks to serialize concurrent operations. Exposing lock-based data structures over RDMA, however, requires at least three RDMA requests: two operations target the lock and one performs the intended operation. This RDMO saves at least two messages by "eliding" these lock operations, akin to speculative lock elision in hardware [14]. As shown in Figure 7, the `SignaledRead` RDMO attempts a compare-and-swap operation. If the swap fails, the RDMO retries the compare-and-swap a few times and returns the value of the last read. Else, the RDMO reads the requested data and resets the flag.

4. WriteAndSeal. This RDMO first writes data to a buffer, then writes to the seal location to mark the completion of the write. This would require two messages in an RDMA implementation. This RDMO will be used in lock-based synchronization to update data and release the lock in one operation.

5. ScatterAndAccumulate. This RDMO performs a scatter operation that involves indirect addressing to the destination through a lookup table, as shown in Figure 8. Instead of overwriting the data at the destination, this RDMO accumulates the transmitted values to what is already present in the destination address. `ScatterAndAccumulate` reduces the substantial network cost of hash-based parallel aggregation for high-cardinality domains [11].

4. CONCLUSIONS

The RDMA communication primitives offered by fast networks are too low-level and verbose for common data processing operations. One way to overcome the complexity of RDMA is to use MPI. Our analysis shows that MPI lacks many desirable attributes. This paper introduces the Remote Direct Memory Operation (RDMO) interface which permits a sequence of reads, writes and atomic operations on remote memory to be executed in one round-trip. Performing five common database operations as RDMOs cuts down the number of network round-trips by as much as one order of magnitude.

5. REFERENCES

[1] G. Alonso, C. Binnig, et al. DPI: the data processing interface for modern networks. In *CIDR*, 2019.
[2] C. Barthels et al. Rack-Scale In-Memory Join Processing Using RDMA. In *SIGMOD*, 2015.
[3] C. Barthels et al. Distributed Join Algorithms on Thousands of Cores. *PVLDB*, 2017.
[4] R. Belli and T. Hoefler. Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization. In *IPDPS*, 2015.
[5] C. Binnig et al. The End of Slow Networks: It's Time for a Redesign. *PVLDB*, 2016.
[6] A. Costea et al. VectorH: Taking SQL-on-Hadoop to the Next Level. In *SIGMOD*, 2016.
[7] Global Arrays. `http://hpc.pnl.gov/globalarrays`.
[8] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
[9] A. Kalia et al. Using RDMA Efficiently for Key-value Services. SIGCOMM, 2014.
[10] F. Liu et al. Design and Evaluation of an RDMA-aware Data Shuffling Operator for Parallel Database Systems. In *EuroSys*, 2017.
[11] F. Liu et al. Chasing similarity: Distribution-aware aggregation scheduling. *PVLDB*, 12(3):292–306, 2018.
[12] MVAPICH. `http://mvapich.cse.ohio-state.edu/`.
[13] OpenMPI. `https://www.open-mpi.org/`.
[14] R. Rajwar and J. R. Goodman. Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution. In *MICRO*, 2001.
[15] W. Rödiger et al. High-speed Query Processing over High-speed Networks. *PVLDB*, 2015.
[16] K. Umamageswaran et al. Exadata Deep Dive: Architecture and Internals. Oracle OpenWorld, 2017.
[17] E. Zamanian et al. The End of a Myth: Distributed Transaction Can Scale. *PVLDB*, 2017.

Towards Observability Data Management at Scale

Suman Karumuri
Slack Technologies
skarumuri@slack-corp.com

Franco Solleza, Stan Zdonik
Brown University
{fsolleza,sbz}@cs.brown.edu

Nesime Tatbul
Intel Labs and MIT
tatbul@csail.mit.edu

ABSTRACT

Observability has been gaining importance as a key capability in today's large-scale software systems and services. Motivated by current experience in industry exemplified by Slack and as a call to arms for database research, this paper outlines the challenges and opportunities involved in designing and building Observability Data Management Systems (ODMSs) to handle this emerging workload at scale.

1 INTRODUCTION

On May 12, 2020 at 4:45pm PST, the cloud-based business communication platform Slack experienced a total service disruption [9]. To its millions of users, the outage lasted for 48 minutes; within Slack, the cascade of events that led to this outage began at 8:45am PST. It all started with a software performance bug that was caught and immediately rolled back during a routine code deployment. This triggered the auto-scaling of Slack's web tier, ramping it up to more instances than the hard limit allowed by the load-balancer. This in turn exposed a bug in how Slack updates the list of hosts in the load-balancer: some load-balancers had a mix of old, stale, and new unregistered host instances. Eight hours later, the only active host instances were the oldest ones still registered to the load-balancers. When the auto-scaling program started scaling hosts down for the night, it shut down these old instances. Since all remaining instances registered to the load-balancers were either stale or new and unregistered, the service experienced a total outage.

While this description of the Slack incident lays out the logical sequence of events that led to the outage, identifying the root cause of the problem required "all hands on deck" [9]. As soon as the alert was raised, engineers from multiple teams got together and explored several possible hypotheses based on operational data visible through their monitoring, dashboarding, and alerting infrastructure. This incident illustrates *Observability* in action, a critical capability not only at Slack, but at many other large software companies [2].

Today's web-scale, user-facing software systems and services (e.g., Slack, Google, Facebook, Twitter) are built and operated on micro-services managed by highly elastic and shared infrastructures. This *cloud-native* software ecosystem is increasingly more distributed, heterogeneous, and complex, making it challenging to predict their behavior in the face of failures and varying load [1]. Observability is emerging as a key capability for monitoring and maintaining cloud-native systems to ensure their quality of service to customers [25]. Borrowed from control theory, the notion of observability brings better visibility into understanding the complex behavior of software using telemetry collected from the system at run time [14]. Beyond simple black-box monitoring, observability provides deeper contextual insight about the correctness and performance of systems. Its goal is to minimize *time to insight* - a critical measure of understanding what is happening in the system, and why. As such, observability is inherently a data-intensive and time-sensitive process that involves humans in the loop (e.g., DevOps teams) [27].

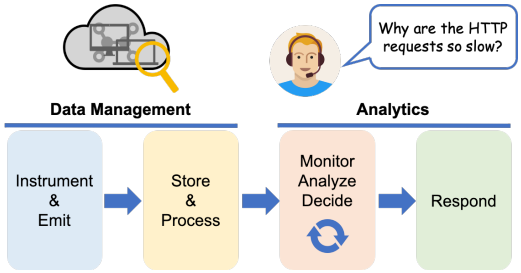


Figure 1: Observability

We see observability as a data management problem. Systems are instrumented to generate large volumes of heterogeneous time series data that must indexed, stored, and queried in near-real time. Instrumentation can emit four types of data: (i) numeric data like gauges and counts, (ii) highly structured data on system events, (iii) logs of unstructured strings, and (iv) a graph of the execution path of a request. In industry, these are referred to as *Metrics, Events*[1], *Logs, and Traces*.

Since observability is critical for meeting service-level objectives (SLOs) for web companies with millions of users, there is a lot of industrial activity in this domain. However, current solutions consist of a patchwork of various specialized tools to cater to the different needs of each time series data type. These ad-hoc solutions do not scale well and incur high performance overheads, operational complexities, and infrastructure costs [13]. There is a growing need to rethink the current design of data and software infrastructures to enable observability data management at scale.

In this vision paper, we analyze the data management requirements of observability workloads (§2) and challenges

[1]Many observability practitioners represent events as highly structured strings and a subset of logs. In §2, we describe how the structured nature of events warrants separate consideration from logs.

Data	Type	Queries	Storage	Volume	Retention
Metrics	numeric string metadata	aggregations filters on metadata	compressed time series hybrid column store	4B time series/day 12M samples/second 12 TB/day (compressed)	30 days
Events	highly structured strings or binary	filters on exact string matches	column store	250 TB/day (raw)	3-24 months
Logs	semi-structured strings	approximate string search	inverted index	90 TB/day (raw)	7 days
Traces	DAGs of durations of execution	disassociated graph search	columnar / inverted index	2 TB/day (raw)	14 days

Table 1: The four categories of observability time series (MELT) widely differ in their characteristics and needs

experienced by today's systems using Slack's observability infrastructure as an exemplar (§3). Then we identify the general design principles for building web-scale ODMSs and provide the blueprint of a new architecture to realize them (§4).

2 METRICS, EVENTS, LOGS, TRACES

The fundamental challenge of ODMSs is timely insight into a system's state in the face of massive data volumes and heterogeneity. This section analyzes observability data in four categories: Metrics, Events, Logs, and Traces (MELT). We first discuss unique characteristics of these categories, followed by their common requirements for data management.

2.1 Metrics

metrics	tags	timestamp	value
http	dc="dc1" host="h1" path="/"	1574260177	124
http	dc="dc1" host="h2" path="/"	1574260170	109
http	dc="dc1" host="h1" path="/"	1574260215	116
http	dc="dc2" host="h1" path="/"	1574260236	105

Figure 2: A Metric for number of HTTP requests/second

Metrics provide quantitative measurements of system performance and availability at a specific point in time. They encompass three types of numeric data: (i) *counters* are values that can only increase or be reset to zero (e.g., total number of HTTP requests received); (ii) *gauges* are values that can go up or down to reflect system state (e.g., the number of HTTP requests waiting to be responded to); and (iii) *histograms* sample observations over a fixed time interval, counting them in configurable buckets (e.g., HTTP request durations or response sizes). In addition to a numeric value and a timestamp, metrics also include metadata as a set of key-value pairs, called *tags*. Tags identify a specific instance of a metric. A unique combination of a metric and tags is called a *time series*. A pair of a timestamp and a value is called a *sample*. Figure 2 provides an example metric, *http*, measuring the number of HTTP requests per second. Each measurement is tagged with the data center, host, and path metadata for the request's origin. There are three (color-coded) time series in this example, each with monotonically increasing timestamps.

Metrics are used in two ways: (i) to generate alerts on unexpected system state, or (ii) for analytical and dashboarding queries. Alerts are generated using small queries on the most current data (e.g., total number of HTTP requests per host per minute). Analytical queries may involve data from arbitrary times to observe system-wide trends (e.g., total number of HTTP requests per host per minute in dc1 over the last day). More complex analytics (e.g., similarity search or clustering [18, 21]) may also be performed.

Metrics require a hybrid storage engine. Prometheus [20], a widely used metric storage engine, employs a compressed storage strategy for the values of the metric and an inverted index for associated tags. This strategy results in high compression rates and fast filtering operations. Other storage strategies (e.g., data series storage [17]) have also been proposed. Slack generates about 4 billion time series per day, at the rate of 12 million samples per second, collecting 12TB (compressed) of metrics data every day. These are retained for 30 days.

2.2 Events

Events are highly structured data emitted during run time. Frequently, they come from a finite set of possible values. For example, there are 9 HTTP request methods (e.g., GET, POST) and a finite set of response status codes (e.g., "404: Not Found"). Events may also be used for high-cardinality data with higher dimensions (e.g., customer-ids and network addresses). Because events are emitted as structured strings or in a compact binary format [25], prior literature on observability typically considers events as a subcategory of logs (discussed in §2.3) [12, 25]. We categorize them separately, because the data model, queries, and access patterns for events are substantially different from those for logs.

Figure 3 shows an excerpt of raw events emitted by a system handling HTTP requests and responses.

```
Nov 20 17:35:23 2019: 192.168.100.11 POST PATH:/ 200 OK
Nov 20 17:35:24 2019: 192.168.100.10 GET PATH:/ 200 OK
Nov 20 17:35:27 2019: 192.168.100.10 DC=2 HOST=2 505 ERROR
Nov 20 17:35:28 2019: 192.168.101.11 POST PATH:/ 200 OK
Nov 20 17:35:28 2019: 192.168.101.10 GET PATH:/ui/ 404
```

Figure 3: Events emitted by an HTTP server

In addition to computing trends, events are typically used to identify specific instances of unexpected system state. For example, a SQL query like `SELECT * from Events WHERE ip=192.168.100.11 and method="POST"` would show all POST requests from the queried IP address `192.168.100.11`.

The structured nature of the data and the low selectivity filter queries make column stores ideal for events. The data in an event store is usually accessed via SQL queries over a column store. Slack generates over 250TB of raw events data per day and stores over 70 PB data at any one time. Event data is retained for relatively longer periods (3-24 months) for archival or legal audit purposes.

2.3 Logs

Logs are collections of semi-structured or unstructured strings. They expose highly granular information with rich local context. This flexibility makes logs crucial to understanding *why* unexpected behavior occurred in a service. For example, when responding to the outage incident, Slack relied on logs to identify the bug in updating the host list for their load-balancer. Similarly, for an HTTP request, an application developer might include a stack trace along with an exception showing the state of the application in an error log (see Figure 4).

```
Wed Nov 20 17:35:22 2019: GET / ERROR 500
InternalServerErrorException: HTTP 500 ...
    at ServiceUnavailableException ...
    at RedirectionException ...
```

Figure 4: Log entry with a stack trace

Unlike metrics, logs usually contain contextual information that can provide more detailed answers to questions like: "What server error caused the response to have a status of 500?". These *needle-in-a-haystack queries* are fundamentally different from the exact-match queries posed over events. They are best served using inverted index-based storage solutions due to the need for approximate string matches. Slack collects about 90TB of log data/day to be retained for 7 days.

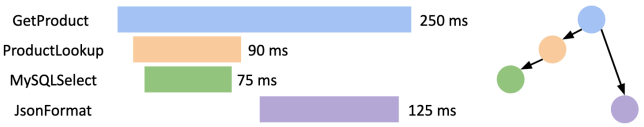
2.4 Traces

Traces encapsulate information about the execution path of a request similar to call graphs [8, 11, 23, 24]. In microservices and distributed settings, traces are call graphs across distributed services and include RPC invocations, asynchronous queues, and other inter-service communication. Traces are represented as directed acyclic graphs (DAGs) where a vertex represents a unit of execution (e.g., a function call) called a *span* and edges indicate a causal ordering (i.e., Lamport's *happens-before* [10]) from one vertex to another. This definition is increasingly accepted with industry-wide efforts like OpenTelemetry [14].

Figure 5 shows an excerpt of a trace of a product lookup request through the *GetProduct* service. To respond to this request, *GetProduct* calls the *ProductLookup* service which in turn makes a database call named *MySQLSelect*. *GetProduct* then formats the result and finally responds to the request. The trace and its spans encapsulate the structure of the execution path of this request, and information about this path. The spans capture the duration of the execution and metadata stored as *tags* in the form of key-value pairs.

Trace ID	Span ID	Parent ID	Start Time (ms)	Duration (ms)	Name	Tags
1234	0		1574260177000	250	GetProduct	api=/getproduct, product=7
1234	1	0	1574260177020	90	ProductLookup	product=7, query=lookup
1234	2	1	1574260177030	75	MySQLSelect	query=select
1234	3	0	1574260177110	125	JsonFormat	encoding=json

(a) A table of spans in the execution of the "GetProduct" request.



(b) The trace visualized as a Gantt chart and as a directed graph

Figure 5: An example Trace for an HTTP request

There are usually two steps to accessing a specific trace: (i) *findTraces:* a user searches for traces that match a certain criteria (e.g., HTTP requests since yesterday over 200ms where a DB call returned at least 2 rows); (ii) *getTrace:* among the list of traces returned, a user selects a specific trace to view as a Gantt chart.

Although tracing has been used in distributed systems for decades [7], there is little research on the storage and management of trace data.

At Slack, trace data volumes are lower than others (e.g., 2 TB/day) because of the challenge in instrumenting and managing trace data. To manage this complexity, Slack represents spans in a format shown in Figure 5a, similar to industry efforts like OpenTelemetry's tracing API [14].

2.5 Common Characteristics of MELT Data

As discussed in the previous subsections and summarized in Table 1, the data, query, and storage models for the four types of time series in an ODMS differ widely. However, MELT data also shares several important characteristics that influence how they should be managed overall.

Data characteristics. Fundamentally, all MELT data is *immutable* and *append-heavy*. Furthermore, because of the dynamic distributed environment, the volume of data generated over time is *highly variable (i.e., bursty)*. For example, a major auto-scaling event or a new log exception can increase input data volume rates by 10-fold for a short period of time.

Query characteristics. A *bias to freshness* accurately summarizes the general nature of queries on all observability data. To illustrate, Table 2 shows the percent distribution of Slack queries by data age, indicating that >97% of all queries are on data that is less than 24 hours old. During an incident, fresh data is needed not only to understand the current state of the system, but also to quickly see if the remediation is having the

Data Age	Logs (19.6M)	Metrics (17M)	Traces (46K)
<1 hour	92.5	94.7	85.2
<2 hours	92.6	95.2	94.8
<4 hours	94.5	97.5	95.0
<1 day	99.8	99.8	97.3

Table 2: % of queries by data age (with total query volumes in parentheses) for 1+ months of querying statistics over logs, metrics, and traces at Slack. More than 97% of the queries are for data produced in the last 24 hours.

intended effect. This implies that fresh data typically needs to be more accessible and available than older data. Furthermore, during major incidents, it is not uncommon to have twice the normal number of users interacting with the dashboards and writing custom queries against MELT data. Most queries are for *dashboards and alerts* that typically query only a small percentage of the collected data, but also require *sub-second latencies* to support real-time decision making [15].

Life-cycle management. Along with the bias to freshness, historical data is still required for a smaller percentage of queries when looking for longer term trends or for legal/business purposes. For example, during the aforementioned outage, Slack engineers looked at data over the last few weeks to check for any seasonal patterns. Hence, MELT data commonly requires data life-cycle management for fresh and historical data side by side, indicating that a *tiered storage strategy* based on data age should be designed into the ODMS.

3 TODAY'S CHALLENGES

Current ODMS solutions deployed in industry are custom-built approaches that only partially address the requirements of MELT data management. These systems are built as a result of reactive implementation [13], without guiding design principles and therefore face similar practical challenges. In this section, we present Slack's current observability system infrastructure as a case study to reveal the key practical challenges to be considered when architecting a scalable ODMS.

3.1 Observability at Slack

Figure 6 depicts Slack's current observability system infrastructure. Like many industry solutions, it consists of multiple tools, ingestion pipelines, and storage engines to collect, store, and serve MELT data generated by Slack applications.

Slack uses Prometheus [20], a single-node pull-based system, to store and serve metrics. It scrapes metrics data from HTTP endpoints exposed by Slack applications. For high availability, Slack uses a pair of Prometheus servers, each maintaining an independent copy of the data. Each application is allocated a pool of 2 to 64 Prometheus servers. In total, Slack runs about 100 such pools.

Events, logs, and traces (ELT) are pushed to Apache Kafka. For durability, Secor [19] consumes and writes the raw data

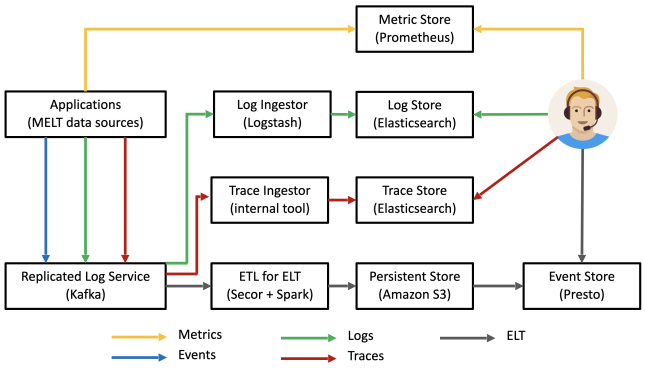


Figure 6: In Slack's current observability infrastructure, an engineer manages and queries four independent systems, using different query APIs and tools.

to Amazon S3. A custom Apache Spark job copies and transforms this data into a columnar Parquet file, also stored into Amazon S3 [26, 28]. These Parquet files can be queried via SQL using Presto [22]. In addition, Slack uses Logstash to ingest logs into Elasticsearch, and an internal tool to ingest traces into a vendored tracing solution and an in-house trace store backed by Elasticsearch [4]. Logs are retained for 7 days and traces for 14 days. Logs and traces are also written to a data warehouse (Presto) for historical querying.

3.2 Practical Challenges

Slack's current ODMS architecture faces a number of practical challenges unique to the requirements of growing observability workloads. Although we use Slack as a case study, we believe that almost all ODMSs currently deployed in industry face one or more of these practical challenges. We group these challenges into three overarching categories:

High Operational Complexity. The infrastructure to serve MELT data at Slack contains over 20 separate software components. Each component has a unique architecture and, as a result, needs custom operations for cluster management, security, and capacity planning. This heterogeneity leads to complex solutions. For example, Prometheus is a single-node system and maintains independent copies of the data to meet high-availability requirements. Meanwhile, although Elasticsearch has built-in replica management, at Slack's scale, it is challenging to manage and operate. During the May 12 incident, data volumes spiked to 4 times the normal peak volume. In such scenarios, the monitoring team performs complex scale-up operations to continue ingesting fresh telemetry data.

Maintaining Low Query Latency. Queries on observability data are highly skewed. Over 97% of queries access data < 24 hours old for near-real-time alerting. Dashboards and major incidents or product changes result in significant spikes in the number of queries putting significant pressure on the ODMS. During the May 12 incident, Slack looked at the data

for the prior 8 hours while handling 2 times the historical peak load. The high operational complexity in Slack's ODMS makes scaling in response to overall query workload and maintaining low query latency a significant challenge.

High Infrastructure Cost. At the petabyte scale, meeting ever-increasing data retention requirements and ensuring data availability in bursty workloads quickly becomes costly. At Slack, significant time is spent in balancing the cost of infrastructure, performance, durability, and availability of the data. For example, Slack duplicates processing ELT in Presto for availability and durability, at the risk of slower performance and higher infrastructure costs. Slack also provisions based on historical peaks to handle spiky workloads such as during the May 12 incident. The new peak load is now 2 times the peak load prior to the incident. This provisioning strategy adds to the infrastructure cost.

4 ODMS DESIGN PRINCIPLES

We now propose a set of design principles that address the real-world requirements and challenges described in prior sections. Considering the heterogeneous nature of MELT data and the need for reducing the complexity of managing this data while meeting the query performance requirements in a scalable manner, we believe that an ODMS should adhere to four core design principles:

Decouple real-time and historical data management. §2 and Table 2 showed that the ODMS's workload is significantly different from hybrid database workloads that combine OLAP and OLTP (e.g., HTAP [16]). It ingests petabytes of potentially bursty immutable writes and queries are biased to data < 24 hours old. ODMSs designed with this workload in mind decouple real-time and historical data management. This decoupling maintains fast ingestion rates, low query latency on recent data, and minimizes the cost of storing and accessing historical data.

Unify MELT data life-cycle management. Time ties all MELT data management to a common framework. The unique real-time and historical data management strategies should be governed in a unified data life-cycle based on data age. Doing so should abstract away the movement of data from real-time to historical storage, and decrease the cost and complexity of transparently accessing all data over arbitrary periods of time.

Provide a single query interface for MELT data. Observability queries commonly require data from different MELT types. A single query interface abstracts the individual storage and processing requirements of MELT data, decreasing the complexity of writing these queries, and providing opportunities for optimization across the storage strategies. This principle lends itself to a "polystore-like" pluggable storage engine architecture [3].

Support cloud-native, distributed deployment. Because of the highly bursty nature of both reads and writes, various tiers

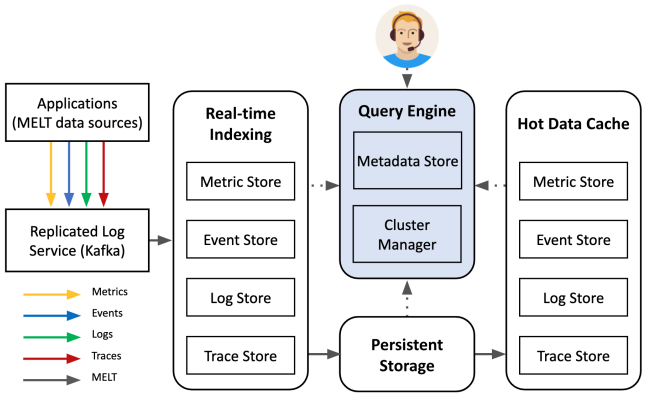


Figure 7: In the new architecture based on ODMS design principles, a user manages and queries a single system.

of the system should be distributed and elastic by design, so that the system can scale with the changing workload. A cloud-native ODMS allows the observability team to make a flexible trade-off between cost and performance for serving MELT data based on workload and the data life-cycle.

These four principles must be integrated into the design process of architecting an ODMS. Unlike current industry practice where ODMSs are built as a patchwork of features responding to ad-hoc requirements, following these principles results in a coherent architecture that provides control over an ODMS's complexity, performance, and cost.

Figure 7 shows an overview of such an architecture that realizes these design principles. It unifies the data life-cycle for MELT data into a single ODMS infrastructure. Unlike Slack's current system, which is influenced by modern Lambda architectures [5], this design is influenced by the design of modern polystore architectures [3]. This decreases the storage requirements and complexity of maintaining multiple data pipelines, and addresses the need for transparently coordinating multiple storage engines.

Data ingestion and storage. All MELT input data from instrumented applications are ingested into a *Replicated Log Service* (e.g., Apache Kafka [6]). The *Real-time Indexing* tier comprised of storage engines specific to each data type pulls data from the log service and indexes them for fast access. Periodically, indexed data files are migrated to the *Persistent Storage* tier similar to Amazon S3. This migration is coordinated by the *Cluster Manager*. At query time, the *Query Engine* coordinates the *Real-Time Indexing* and *Persistent Storage* tiers using the *Metadata Store*. When necessary, the *Query Engine* pulls hot historical data from the *Persistent Storage* tier into the *Hot Data Cache* to maintain query performance.

Query processing. The *Query Engine* services all queries. Using the *Metadata Store*, it coordinates between the three data tiers (*Real-Time Indexing*, *Persistent Storage*, and *Hot*

Data Cache) to filter data based on time-range and to optimize queries across multiple data types (akin to joins by time). It determines optimal data placement based on expected query distributions and costs. For example, ad-hoc queries that access data in *Persistent Storage* may not need to be cached. However, when accessing historical data repeatedly while resolving an issue, the query engine might decide to copy data to the *Hot Data Cache* to minimize query latency.

Distribution and availability. The ODMS's distribution strategy should maintain high availability during bursty periods of new data and during periods with particularly heavy query load. The *Real-Time Indexing* and *Hot Data Cache* tiers are natively elastic, where independent replicas are spun-up on-demand. The *Query Engine*'s *Cluster Manager* monitors the *Real-Time Indexing* tier's workload to determine whether to scale up any of the component stores. During periods of particularly heavy data writes (e.g., from instrumented applications) or queries on fresh data, it scales up the *Real-Time Indexing* tier. During particularly heavy reads of historical data, it scales up the *Hot Data Cache*. The two-tier design also helps support variable availability of data (e.g., higher availability guarantees for more recent/valuable data).

The architecture is generalizeable to settings where observability telemetry is unified in a centralized ODMS that makes the MELT data easily accessible to users. An ODMS architecture following these principles is intended to help during peak loads such as the one during Slack's May 12 incident. The **Query Engine** provides a single query interface and a unified view of the MELT data. It provides an abstraction over the *complexity* of querying and managing the heterogeneous types and their life-cycles across the **Real-Time Indexing** and the historical **Persistent Storage** tiers. The elastic **Hot Data Cache** responds to peak loads based on data access patterns to *maintain low query latency*. Finally, the **Cluster Manager** manages data life-cycles and operational complexity in this system. It also scales or shrinks based on the workload, thereby decreasing the system's overall *infrastructure cost*. These principles result in shorter time to insight during peak loads and decrease overall complexity and costs.

5 CONCLUSION

Observability data management is an emerging area of research that requires more attention from the database community. In this paper, we discussed real-world experience with observability data and its use cases at Slack – a cloud-based team collaboration service. The heterogeneous nature of time series data as well as varying workload characteristics call for a new observability data management architecture. In response, we proposed a new cloud-native polystore-like architecture that decouples real-time and historical data access tiers from the underlying persistent storage and the querying tier in a way that enables scaling them independently. We are currently working on building an initial prototype of this design to test with production data from Slack.

REFERENCES

[1] R. H. Arpaci-Dusseau et al. 2018. Cloud-Native File Systems. In *USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*.

[2] C. Chan et al. 2020. Debugging Incidents in Google's Distributed Systems. *ACM Queue* 18, 2 (2020).

[3] J. Duggan et al. 2015. The BigDAWG Polystore System. *ACM SIGMOD Record* 44, 2 (2015), 11–16.

[4] C. Gormley et al. 2015. *Elasticsearch: The Definitive Guide*. O'Reilly Media.

[5] M. Hausenblas et al. 2017. Lambda Architecture. http://lambda-architecture.net.

[6] N. Narkhede et al. 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*. O'Reilly Media.

[7] J. Jeffrey et al. 1987. Monitoring Distributed Systems. *ACM Transactions on Computer Systems (TOCS)* 5, 2 (1987), 121–150.

[8] J. Kaldor et al. 2017. Canopy: An End-to-End Performance Tracing And Analysis System. In *SOSP*. 34–50.

[9] R. Katkov. 2020. All Hands on Deck. https://slack.engineering/all-hands-on-deck-91d6986c3ee.

[10] L. Lamport. 1976. The Ordering of Events in a Distributed System. *Communications of the ACM* 21, 7 (1976), 558.

[11] J. Mace et al. 2015. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In *SOSP*. 378–393.

[12] S. More. 2018. *A Practical Observability Primer*. mStakx.

[13] S. Niedermaier et al. 2019. On Observability and Monitoring of Distributed Systems – An Industry Interview Study. In *ICSOC*.

[14] OpenTelemetry. 2019. The OpenTelemetry Open-Source Observability Framework. https://opentelemetry.io/.

[15] J. O'Shea. 2020. Building Dashboards for Operational Visibility. https://aws.amazon.com/builders-library/building-dashboards-for-operational-visibility/.

[16] F. Özcan et al. 2017. Hybrid Transactional/Analytical Processing: A Survey. In *ACM SIGMOD Conference*. 1771–1775.

[17] T. Palpanas. 2015. Data Series Management: The Road to Big Sequence Analytics. *ACM SIGMOD Record* 44, 2 (2015), 47–52.

[18] T. Palpanas et al. 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshops. *ACM SIGMOD Record* 48, 3 (2019), 36–40.

[19] Pinterest. 2017. Pinterest Secor: A Service for Implementing Kafka Log Persistence. https://github.com/pinterest/secor.

[20] Prometheus. 2012. Prometheus Documentation. https://prometheus.io/docs/concepts/metric_types/.

[21] J. Rodrigues et al. 2017. Sieve: Actionable Insights from Monitored Metrics in Distributed Systems. In *ACM Middleware Conference*. 14–27.

[22] R. Sethi et al. 2019. Presto: SQL on Everything. In *IEEE ICDE*.

[23] Y. Shkuro. 2019. *Mastering Distributed Tracing: Analyzing Performance in Microservices and Complex Systems*. Packt Publishing.

[24] B. H. Sigelman et al. 2010. *Dapper: A Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google, Inc.

[25] C. Sridharan. 2018. *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media.

[26] D. Vohra. 2016. Apache Parquet. In *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. 325–335.

[27] A. Wiedemann et al. 2019. The DevOps Phenomenon. *ACM Queue* 17, 2 (2019).

[28] M. Zaharia et al. 2010. Spark: Cluster Computing with Working Sets. In *USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*.

Dagstuhl Seminar on the Foundations of Composite Event Recognition

Alexander Artikis[1,2] Thomas Eiter[3] Alessandro Margara[4]
Stijn Vansummeren[5]
[1]University of Piraeus, GR, [2]NCSR Demokritos, GR, [3]TU Wien, AT
[4]Polytechnic University of Milan, IT, [5]Hasselt University and transnational University of Limburg, BE
a.artikis@unipi.gr, eiter@kr.tuwien.ac.at, alessandro.margara@polimi.it
stijn.vansummeren@uhasselt.be

ABSTRACT

Composite event recognition (CER) is concerned with continuously matching patterns in streams of 'event' data over (geographically) distributed sources. This paper reports the results of the Dagstuhl Seminar "Foundations of Composite Event Recognition" held in 2020.

1. INTRODUCTION

Composite Event Recognition (CER) refers to the activity of matching patterns in streams of continuously arriving 'event' data over (geographically) distributed sources. CER is a key ingredient of many modern Big Data applications that require the processing of such event streams to obtain timely insights and implement reactive and proactive measures. Traffic management in smart cities, for instance, requires the analysis of data from an increasing number of sensors, both mobile (e.g. mounted on public transport vehicles and private cars) and stationary (installed at intersections). Using such data streams, CER systems detect or even forecast traffic congestions, thus enabling one to proactively change traffic light policies and speed limits, with the aim of reducing carbon emissions, optimising public transportation, and improving the quality of life and productivity of commuters [3].

Numerous CER systems and languages have been proposed in the literature, cf. [1, 7, 9]. While these systems have a common goal, they differ in their architectures, data models, pattern languages and processing mechanisms, resulting in heterogeneous implementations with sometimes fundamentally different capabilities. Because the research focus in the established literature has been on practical system aspects, and less on formal foundations, CER can be difficult to understand, extend and generalise. As a concrete example, so-called *selection strategies* [7, 9] are supported by numerous systems, but with sometimes incompatible implementations. In this respect, it helps to model the semantics formally, so that these differences and the trade-offs they entail become clear, demonstrating the benefits of formal models compared to others [10].

To start addressing these issues, a seminar on the Foundations of Composite Event Recognition was held at Schloss Dagstuhl, Leibniz Center for Informatics during February 9-14, 2020.[1] The seminar gathered 39 researchers and practitioners working in diverse domains strictly related to CER. The first days put a focus on tutorials and talks that gave an overview of the approaches, techniques, methodologies and vocabularies used in different communities to refer to CER problems. Subsequently, the seminar continued by alternating sessions with focused research talks and working group discussions, on the topics that the participants identified as the most relevant for future investigations and research efforts. This paper gives an overview of the tutorials and outcomes of the discussions. Due to space limitations, the exposition is necessarily brief; more information is available in the Dagstuhl report [4].

2. TUTORIALS

Six tutorials aimed at introducing CER-related research in different communities.

Applications & Requirements of CER. Sabri Skhiri presented key requirements of CER systems from an applications perspective, focusing on four questions: (1) Which industrial applications does CER have? (2) What are the key requirements of CER concerning data models, recognition language expressiveness and performance (latency, throughput, predictive accuracy)? (3) How do existing approaches address the requirements? (4) Which classes of applications can benefit from CER techniques? To answer these questions, a typical

[1]https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=20071

streaming architecture where CER systems may be deployed was presented. The CER challenges were then illustrated on industrial applications in crowd management, banking, telecommunication, security & surveillance, and microservice architectures. The key requirements for them are as follows. CER systems must scale to streams of millions of events per second while handling states or partial matches lasting from a few days to weeks. Next, the CER language should support temporal iteration, negation and sub-patterns. Moreover, imprecise patterns should be supported, as experts cannot always define target composite events precisely.

CER in Data Management. Martin Ugarte and Cristian Riveros presented a theoretical perspective of the most common features of CER languages. They started with a basic setting for CER that served to discuss the fundamental properties from a Data Management perspective: well-defined syntax and semantics, composability, and denotational, declarative semantics. These properties were then exemplified on *complex event logic* [10], which was also used to present the main operations in CER systems. Next, they discussed the challenges of formally defining the CER operations while satisfying the aforementioned properties. Moving on to evaluation of CER languages, they discussed the relevant notions of efficiency and complexity, and presented the types of lower bounds obtainable for evaluating CER patterns. Finally, they outlined CER challenges on concrete examples: What are the relevant complexity classes? What are the classes of queries that may be evaluated efficiently? How does the change from push-based to pull-based semantics affect complexity?

Distributed Event-Based Systems. Avigdor Gal and Ruben Mayer introduced Distributed Event-Based Systems (DEBS), which may be viewed as pipelines from sources of low-level events to sinks with applications, via an operator graph (event processing middleware). They presented event recognition languages and pointed out the differences between composite event processing, stream processing and rule-based event processing. Then they considered windowing in depth, parallelism and time aspects, such as event and processing time, late arrival of events, and the trade-off between latency and accuracy. Finally, they addressed uncertainty associated to event occurrence and attribute values and discussed approaches to deal with uncertainty in the matching process.

Stream Reasoning. Stream reasoning deals with incremental reasoning over rapidly changing information [8]. Jacopo Urbani and Fredrik Heintz gave an overview of the area, first presenting some application scenarios. The key ingredients for stream reasoning are temporal data (time management), various sources (data integration), intelligent decision making (AI), and scalability and efficiency (data management). Urbani and Heintz presented requirements for query answering over streams and matched them against the four V's in Big Data (volume, velocity, variety and veracity). A variety of approaches were then briefly discussed, viz. CSPARQL, CQELS, EP-SPARQL, LARS, Laser, Ticker, BigSR and metric temporal logic (MTL) based reasoning. Open challenges mentioned are handling massive data and uncertainty, combining symbolic and sub-symbolic knowledge, and benchmarking stream reasoning systems.

CER in Logic and AI. Diego Calvanese presented formalisms relevant for CER that have been developed in the areas of knowledge representation and reasoning, formal verification and database theory. Such formalisms typically rely on combining variants of temporal logics with logics used in knowledge representation and reasoning, which poses challenges for semantics and computability. The challenges have to a certain extent been addressed by a variety of techniques and under various assumptions; however, the area is fragmented and there is no unifying or consolidated framework.

CER in Business Process Management. In many application scenarios, business processes may be viewed as consumers as well as producers of events. Common process modelling languages, therefore, contain constructs to incorporate events. At the same time, event-based systems can be used as a basis for process execution and analysis. Against this background, Matthias Weidlich reviewed the relation between the fields of business process management and CER. Furthermore, he outlined opportunities for research at the intersection of the two fields, as regards CER integration with process modelling, event abstraction for process analysis, event pattern derivation from process models, and event-based process execution infrastructures.

3. WORKING GROUPS

The working groups were devoted to five topics.

Expressiveness & Common Model. As CER systems and languages have originated from many different communities, the CER field is broad and diverse, and this in turn makes understanding the relationships between various approaches difficult. The discussions in this working group aimed to clarify whether a "core" of existing systems and lan-

guages can be captured in a common formal model. Such a model could ease the comparison of the expressiveness and capabilities of different approaches as well as improve system interoperability.

The first session revealed sometimes widely different views about many essential CER aspects. For example: what kind of problem is CER, abstractly speaking? Is it a model checking, a monitoring, or a synthesis problem? What kind of object do CER systems produce as output? Is it a sequence of time-annotated facts from the input, a sequence of sets of such facts, or an arbitrary sequence of tuples? Is a notion of time essential in CER? Arguments in favor and against all proposed views were discussed; we refer the interested reader to [4].

As finding a single common model seemed to be difficult, the participants considered then establishing an abstract *meta-model* for CER. Ideally, such a meta-model incorporates key elements of CER systems and focuses on *what* CER does rather than *how* this is done. By introducing conceptual components that can be instantiated in different ways, the meta-model could allow a *common way of thinking* about CER, thus facilitating the discussions in the community. A first abstract candidate meta-model was proposed during the seminar [4]. A natural next step is to investigate how it can be instantiated to capture the existing CER literature, and to identify commonalities among these instantiations.

Uncertainty in CER. CER systems must deal with various types of uncertainty [1]. The events of the input stream may be noisy, e.g. due to inaccuracy of sensors or distortion in a communication channel. Moreover, a sensor may fail to report certain events, due to e.g. hardware malfunction. Even if the hardware infrastructure works as expected, the characteristics of the environment could prevent events from being recorded; consider, for instance, an occluded object in video monitoring. Data uncertainty may also be by intention, e.g. by an event publisher to prevent complete disclosure of an event stream to its subscribers.

Besides uncertainty in the input data, event patterns can be imprecise or incomplete, as identified in the 'Applications & Requirements' tutorial; due to lack of knowledge or inherent complexity of a domain, it is sometimes impossible to capture exactly all the conditions that a pattern should satisfy.

The participants outlined the following three open challenges. (1) Identify possible sources of uncertainty and classify them according to the impact they can have on the recognition process. (2) Define suitable models to represent different types of uncertainty. (3) Define a conceptual framework and algorithms to consider and propagate uncertainty in the composite event recognition process.

Benchmarking. While use-cases, key performance indicators and relevant benchmarking challenges have been identified [2, 6, 11], CER performance evaluation is still not homogeneous. In the absence of sufficient real-world event streams, researchers adapted analytic benchmarks like Linear Road [2], or benchmarks for Message-Oriented Middleware. Such hand-crafted approaches limit experiment reproducibility. Moreover, maintaining these benchmarks is a burden for individual research groups with long-term support hard to guarantee.

The working group focused on identifying a sustainable path to the design of a domain-specific benchmark for CER maintainable by the community. First, interesting types of benchmarks were discussed. As major ones, macro- (aka use-case driven) benchmarks and micro-benchmarks emerged, which focus on evaluating systems w.r.t. specific workloads, typically inspired by real-world scenarios, resp. the performance of single operators. Macro-benchmarks directly relate with the ongoing effort behind the DEBS Grand Challenges, which yearly provide interesting use-cases and workloads. Micro-benchmarks relate to a common CER model via a core algebra of operations that CER engines must support. Second, the discussion highlighted the lack of standard data and query models (and formats) for CER, which are crucial to develop and maintain a benchmark suite for the community. (Streaming extensions of SQL are towards the right direction [5].) Finally, the discussion focused on technical supports, emphasizing the need for a FAIR (i.e., *findable*, *accessible*, *interoperable* and *reusable*) community benchmark.

Towards establishing a CER benchmark, a two-step approach was proposed: (1) Provide a systematic review of existing benchmarks and systems experiments to identify their dimensions of interest, and a repeatability study that tries to replicate existing results. The insights gained provide input for a new benchmark. (2) Form a working group to create the CER benchmark.

Process strategies, parallelization, and distribution. CER applies to heterogeneous scenarios, with different requirements and deployment settings. A CER framework should adapt its processing and deployment strategies to optimise the use of resources for achieving the application goals. The participants identified here several research and engineering challenges: (1) Identify suitable metrics and constraints to express application requirements in terms of performance (e.g. latency and through-

put) as well as use of resources, security, privacy and fault tolerance. (2) Identify suitable models to capture the relevant characteristics of the deployment infrastructure, e.g. in terms of computation power, hardware architecture, memory, network connections, geographical locations and ownership. (3) Understand the trade-offs that exist between expressiveness and optimisation opportunities, e.g. to identify functionalities that limit the applicability of parallel processing. This could lead to the design of various language fragments that offer the best balance between generality, expressiveness, and performance in a given scenario. (4) Define flexible process and deployment mechanisms to allocate operators to physical nodes, and adopt the most suitable processing algorithms and communication techniques for a given deployment infrastructure. (5) Define monitoring mechanisms to promptly detect critical situations, such as failures and node overloads. Design and implement adaptation algorithms to change the deployment at runtime and restore from such critical situations.

Event pattern induction and composite event forecasting. Manual event pattern authoring is error-prone and time-consuming, as is manual fine-tuning of event patterns to optimise their predictive performance, which should be done whenever it deteriorates, e.g. when the statistical properties of a stream are modified. As machine learning techniques support event pattern construction and refinement, they start attracting attention by the CER community.

Composite Event Forecasting (CEF) refers to the ability of a system to provide forecasts about the possible occurrence of composite events in the future [3]. Notably, CEF is less mature than and *orthogonal* to pattern induction, as the underlying patterns for CEF may be manually constructed or automatically extracted from data.

Pattern induction and CEF have several challenges. (1) Enhance machine learning techniques with domain knowledge curated by experts, to reduce the search space and produce patterns with higher predictive accuracy. (2) Provide automatically constructable models that humans can understand, thus supporting explainability, and expressive enough to effectively capture the temporal phenomena of an application. (3) Provide *online* learning algorithms for CER systems that can construct an event pattern set in a single-pass over the input stream, while efficiently dealing with stream changes; to achieve decent performance, distributed learning may be necessary. (4) As for CEF, identify ways for online accurate forecasting of composite events that may take place (far) in the future, e.g. by combinations of probabilistic reasoning and (extended symbolic) automata. (5) Identify ways to effectively inform proactive decision-making as a result of CEF. For instance, if a traffic congestion is forecast for an intersection, re-direct traffic trying to avoid traffic congestion in other intersections.

4. CONCLUSION

Complex Event Recognition (CER) is an area of growing interest that draws from diverse communities. The Dagstuhl seminar served to share their views and identify the relevant topics with future research challenges on the foundations of CER; establishing a common view and (meta-)model of CER is the biggest among them. First steps have been made, but more efforts are necessary. A workshop on reasoning about actions and events over streams (RACES) at KR 2020 and a planned workshop on CER benchmarking are on the agenda.

5. REFERENCES

[1] E. Alevizos, A. Skarlatidis, A. Artikis, and G. Paliouras. Probabilistic complex event recognition: A survey. *ACM Comp. Surv.*, 50(5):71:1–71:31, 2017.

[2] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *VLDB*, pp. 480–491, 2004.

[3] A. Artikis, C. Baber, P. Bizarro, C. Canudas-de-Wit, O. Etzion, F. Fournier, P. Goulart, A. Howes, J. Lygeros, G. Paliouras, A. Schuster, and I. Sharfman. Scalable proactive event-driven decision making. *IEEE Technol. Soc. Mag.*, 33(3):35–41, 2014.

[4] A. Artikis, T. Eiter, A. Margara, and S. Vansummeren, editors. *Foundations of Composite Event Recognition: Report from Dagstuhl Seminar 20071*. Dagstuhl Reports. 2020.

[5] E. Begoli, T. Akidau, F. Hueske, J. Hyde, K. Knight, and K. Knowles. One SQL to rule them all - an efficient and syntactically idiomatic approach to management of streams and tables. In *SIGMOD*, pp. 1757–1772. ACM, 2019.

[6] P. Bizarro. Bicep - benchmarking complex event processing systems. In *Event Processing*, *Dagstuhl Seminar Proc.* 07191. IBFI, Schloss Dagstuhl, 2007.

[7] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comp. Surv.*, 44(3):15:1–15:62, 2012.

[8] D. Dell'Aglio, E. D. Valle, F. van Harmelen, and A. Bernstein. Stream reasoning: A survey and outlook. *Data Sci.*, 1(1-2):59–83, 2017.

[9] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.

[10] A. Grez, C. Riveros, and M. Ugarte. A formal framework for complex event processing. In P. Barceló and M. Calautti, editors, *ICDT*, *LIPIcs* vol. 127, pp. 5:1–5:18, 2019.

[11] T. Scharrenbach, J. Urbani, A. Margara, E. D. Valle, and A. Bernstein. Seven commandments for benchmarking semantic flow processing systems. In *ESWC*, *LNCS* 7882, pp. 305–319. Springer, 2013.

Holding a Conference Online and Live due to Covid-19

Experiences and Lessons Learned from EDBT / ICDT 2020

Angela Bonifati
Lyon 1 University

Giovanna Guerrini
University of Genova

Carsten Lutz
University of Bremen

Wim Martens
University of Bayreuth

Lara Mazilu
University of Manchester

Norman W. Paton
University of Manchester

Marcos Antonio Vaz Salles
University of Copenhagen

Marc H. Scholl
University of Konstanz

Yongluan Zhou
University of Copenhagen

ABSTRACT

The joint EDBT/ICDT conference (International Conference on Extending Database Technology / International Conference on Database Theory) is a well established conference series on data management, with annual meetings in the second half of March that attract 250 to 300 delegates. Three weeks before EDBT/ICDT 2020 was planned to take place in Copenhagen, the rapidly developing Covid-19 pandemic led to the decision to cancel the face-to-face event. In the interest of the research community, it was decided to move the conference online while trying to preserve as much of the real-life experience as possible. As far as we know, we are one of the first conferences that moved to a fully synchronous online experience due to the COVID-19 outbreak. By fully synchronous, we mean that participants jointly listened to presentations, had live Q&A, and attended other live events associated with the conference. In this report, we share our decisions, experiences, and lessons learned.

1. INTRODUCTION

Three weeks before EDBT/ICDT 2020 was planned to take place in Copenhagen, the rapidly developing Covid-19 pandemic reached a state in which it became clear that the face-to-face event had to be canceled. We, the organizers, then decided to move the conference online while trying to preserve as much of the real-life experience as possible. Given the very short notice, we had to be pragmatic and could not prepare the online event as carefully as we would have done otherwise. In fact, we considered the whole thing an interesting experiment suggested by the circumstances, with potentially important lessons to be learned for the community and beyond. We were delighted to receive significant support for our decision from the attendees and keynote speakers and by the community spirit that developed in its course. The online event ran smoothly and was much more enjoyable and successful than we had expected. We received a lot of positive feedback, both informally and in the survey that we sent to our participants after the conference. The purpose of this document is to share our experience and the lessons learned, so that other conference organizers facing a similar situation can benefit from it.

Related Work. EDBT/ICDT 2020 was one of the first Computer Science conferences to shift to an online mode and the first Data Management conference to do so. Our experience was quickly shared with several other conferences that contacted us (among them SIGMOD/PODS 2020, ECAI 2020, and KR 2020). A pointer to our April 2020 *Communication of the ACM* blog article[1] has been included in the ACM Virtual Conferences Report.[2] The only report available when we published the CACM blog article was the report from ASPLOS 2020.[3] In contrast to ASPLOS, which was asynchronous (participants watching the online videos at any time), EDBT/ICDT 2020 was one of the first fully synchronous online conferences, which means: all sessions (keynotes, research sessions, demo sessions, short paper sessions, tutorials and receptions) were jointly attended by the participants at the same time and the schedule was similar to that of a face-to-face conference. This general setup has since also been used by other conferences, such as for instance VLDB 2020. In fact, many conferences have moved

[1] https://cacm.acm.org/blogs/blog-cacm/244379-holding-a-conference-online-and-live-due-to-covid-19/fulltext
[2] https://docs.google.com/document/d/1XsGDOHzBhY9S-D4Smp2p9JgqdI0umZ0IZVi7Nhm0gYg/
[3] https://cacm.acm.org/blogs/blog-cacm/243882-the-asplos-2020-online-conference-experience/fulltext

online now and reports with useful guidance for different communities are becoming available. A list is maintained in the ACM Virtual Conferences Report.

2. DECISIONS

We go over a list of questions that other conference organizers are likely to face when considering a move online, share our decisions, describe how we implemented them in practice, and what we think about the outcome in retrospect.

Should the conference be synchronous (i.e., live) or asynchronous? We decided to run the event in a fully live mode to simulate a face-to-face event as much as possible. Participants jointly listened to presentations, had live Q&A, and attended other live events associated with the conference. This was achieved by combining the Zoom video conference software with the Slack communication platform. Sessions took place on the dates originally planned for the face-to-face event and we used one Zoom meeting for each of the planned sessions, with a password provided to delegates. Holding an online conference asynchronously, in contrast, could mean putting videos of the presentations online to be watched by participants at the time that fits them best and then having synchronous Q&A via Zoom. This mode was adapted, for example, by the joint 2020 editions of LICS and ICALP. Of course, one could also be fully asynchronous and also have Q&A in Slack or a similar tool that does not require participants to be online at the same time. This mode was used by ASPLOS 2020.

In retrospect, we are satisfied with choosing the synchronous approach: interaction and discussion are key features of conferences and although these cannot all be fully preserved online, we managed to preserve them to a reasonable extent. Indeed, interesting discussions emerged after many presentations and, to a lesser extent, also in Slack. With more time to prepare and better tool support, we believe that even more interaction can be fostered.

How do we deal with time differences? It seems difficult to deal with time differences when participants are evenly distributed around the globe. In fact, participants whose time zone is not synchronized with the conference schedule might feel significantly detached and handicapped. In EDBT/ICDT, however, the bulk of participants are from Europe, followed by North and South America, and Asia. To accommodate the relevant time zones, we opted for shorter days, about 5 to 6.5 hours, rather than the expected length (8-9 hours) in a face-to-face meeting. The days were centered around noon CET, which allowed attendees from other continents to attend most sessions without major hassles. For some talks, we made adjustments to the schedule in order to let the speaker present live, e.g. from North America. For example, keynote talks took place at different times on different days as keynote speakers were from different time zones.

In the future, one could try to adapt the program even more carefully to speakers' time zones. In a Eurocentric conference such as EDBT/ICDT, centering around noon CET is a natural thing to do, but speakers from remote time zones could still not attend early and late sessions. This could be alleviated by having even shorter conference days, at the expense of stretching the conference over a longer time period. Apart from careful scheduling, an important support for participants from 'remote timezones' is to record all presentations and to promptly make them available for asychnronous viewing.

How long should sessions and presentations be? Research sessions generally spanned an hour with the net talk length for each paper being 10 and 12 minutes for EDBT and ICDT, respectively. In comparison, EDBT and ICDT allot 20 and 25 minutes for presentations in face-to-face conferences. The main reason for having shorter talks was that we expected longer online sessions to be very tiring for attendees. Another reason is that shorter talks help in achieving shorter conference days to deal with time zone issues as discussed above.

In retrospect, we were happy with the shorter presentations and the sessions didn't seem too long. This impression was confirmed by the participants in our survey (Section 3).

Should talks be recorded or given live? We opted for pre-recorded talks that we then streamed live from a central place with a high capacity internet connection. Our aim was to minimize the probability of technical problems that might result from participants having differing internet connection quality and not being sufficiently familiar with the Zoom software. Some presentations, including 2 keynotes and 1 tutorial were presented live. We suggested using Zoom to record videos, with the speaker visible, which also helped to get participants acquainted with Zoom. We considered using Microsoft CMT for video upload, but ended up using a simple sFTP solution as CMT has a file size limit of 100MB. We checked the quality of the videos beforehand. After the conference, we made the videos (for which we got the author's permission) publicly available on the proceedings web sites.

Our approach mostly worked well, with many good quality videos being submitted. On the one hand, pre-recording talks seemed to result in presentations that were well planned, to the point, and with almost no slips of the tongue.

Therefore, sessions are less likely to run over time. On the other hand, the talks tended to be more monotonous and less dynamic. Since EDBT/ICDT 2020 took place, many online conferences have decided to use prerecorded talks. Some others, such as KR 2020, have dared to mostly have live streamed presentations. While the number of hickups increased, also this model has been proved to be entirely feasible. There is thus no clear answer on whether live presentations or pre-recorded talks are to be preferred, since they both have advantages and disadvantages. Live presentations are usually better received and allow more interaction with the audience especially if the speaker decides to take questions at regular intervals. The latter cannot be easily done with pre-recorded videos unless they are divided into smaller chunks, which we highly recommend for longer talks.

Pre-recorded presentations on the other hand have the advantage that they can in principle be made available *before* the conference.[4] This gives delegates the opportunity to better prepare for the conference and have more productive discussions.

Independent of whether one wants to pre-record talks or not, we feel that making videos of the talks available after the conference is very valuable for the scientific community. This includes recordings of live presentations and the Q&A afterwards.

How can questions be managed? Zoom has two modes, the *meeting* mode and the *webinar* mode. In the beginning of the conference, we used the webinar mode. This mode has a text-based Q&A facility that allows participants to type their questions and to upvote questions asked by other participants. We then had the people with the most popular questions ask them face-to-face. Some participants asked questions in the Zoom chat, but this rendered the chat (which sometimes uses pop-ups) distracting during presentations. Especially when talks are very short, the smallest distraction can bring listeners off-track. We also generated one Slack channel per session, where question and discussion could continue "offline". Slack was a welcome technological addition, which speakers also used to post their slides after the talk.

The Q&A facility in Zoom is quite good in principle, but it is only available in webinar mode. For smaller, parallel sessions we preferred a more informal approach based on Zoom's meeting mode, which we switched to on the second day. In that mode, participants can see a list of the names of the other participants and they can activate their sound and video (but there is still a meeting host who can mute everyone, e.g. when a talk starts). In meeting mode, we simply asked participants to switch on their camera and raise their hand to indicate that they want to ask a question, as in a real conference. We encouraged people to also switch on their camera after each talk even if they did not want to ask a question, the aim being to create a community feeling, which was quite successful. Many people also switched on their cameras at the beginning and end of each session. We still used webinar mode for larger audiences, such as the keynotes.

Retrospectively, we strongly prefer meeting mode and an informal approach to Q&A whenever the audience is of moderate size, say up to 50 participants. This brought much more interaction. Interestingly, quite intense discussions emerged after some talks, probably even more intense than in a face-to-face meeting. This might be due to the group feeling created by Zoom meeting mode when several people have switched on their camera, whereas in a face-to-face event, the few people who are interested in an in-depth discussion of a presentation might sit far apart from each other, with much less of a group feeling.

How can sessions be chaired? We started with only a Zoom host and no session chair. It turned out, however, that the single host is rather busy with running the session, playing the videos, and monitoring the chat. As the event went by, the technology was holding up, and internet connections seemed to be sufficiently stable, session chairs were introduced to manage discussions while a separate Zoom host was technically managing the Zoom session, and this was felt to be successful.

Can there be a social programme? The programme only had very short coffee breaks of 15 minutes, to make the conference days shorter. There was no joint activity during the coffee breaks apart from using the Slack channels. We held two "Bring your own beer" receptions, one on the opening evening of ICDT, and one on the opening evening of EDBT, where evening refers to the CET time zone. In the receptions, people arrived in an online session and were assigned at random to Zoom breakout groups, to allow smaller group interactions.

There obviously could not be a joint conference dinner. Applying a best-effort principle, we published recipes for home cooking on our web page that are relatively easy to make and that use ingredients that we believed would be available to most of our delegates, even in the early Covid-19 lockdown phase.

Retrospectively, we would suggest planning longer coffee breaks to make the conference days less exhausting. The receptions seemed to work well, given the circumstances, and this was also confirmed in our survey (Section 3). There might well be scope for having more sessions with

[4]Given the deadline-driven nature of our research community, it will require extra organizational efforts to bring authors to finish the video and its publication paperwork early.

opportunities for extended, informal, interactions. It might be interesting to use other, technically more sophisticated tools for this, such as Online Town or Gather Town. The latter has been used by SIGMOD/PODS 2020 and VLDB 2020 for social events, such as parties during the conference or simply to foster networking and reproduce the hallway chat.

How should short/poster papers be handled? We decided to waive short advert videos, and use Slack for asynchronous discussions. We didn't have a clear idea how to run more interactive sessions that would simulate a poster session, so the short paper session turned into a collection of short videos (26 in all) back-to-back with no intermediate Q&A. The session was well attended, with over 50 people there throughout. However, it was hard work to sit through so many, diverse, videos, and the Slack channel was not especially busy.

Retrospectively, we would be tempted to simulate a poster session in a more realistic way to enable deeper interactions. One way of doing this could be to have each poster participant create their own Zoom meeting from within Slack such that participants can use Slack to easily switch between the rooms. Another option is to use Gather Town or a similar tool.

What should be the approach to demos? We adopted the conference model of videos of the demo in 15 minute slots. Each demo was a 10 minute video, with 5 minutes for questions. In retrospect, the videos were alright, but there wasn't the chance for extended discussions that are associated with demo sessions. Given that technology and internet connections were more stable than we expected, we would be tempted to try having each demo participant create their own Zoom meeting from within Slack such that participants can use Slack to easily switch between the rooms. Again, this more closely reflects the experience at a face-to-face demo session.

What should be the approach to keynotes and tutorials? Keynotes and tutorials were in 1-hour slots, some of them with gaps every 15 minutes for questions. A complete hour of presenting seemed rather long. Furthermore, in an online event everyone feels close to the presenter, so some people are more inclined to ask questions. As such, one may want to plan more discussion time.

Two keynotes used videos, two were presented live. Concerning tutorials, three of them used pre-recorded videos, one was presented live. The division of talks into parts was felt to have been a success for both keynotes and tutorials in order to let people chime in with questions.

What do we do if the meeting host has technical problems? Before the conference, we tested how the software platform reacts if the meeting host drops out (e.g. by losing the internet connection). In our case, we observed that the meeting can still continue, but the host's video freezes. In order to avoid major technical problems, we reached out to back-up hosts for each session, who received a crash course on how to handle the software platform about one week before the conference. We also wrote a general guide for session hosts on how to set up all the parameters to make the sessions work the way we wanted them to. This technical aspect of running a conference certainly requires some practice and we highly recommend thorough preparation.

3. ATTENDEE FEEDBACK SURVEY

We ran a feedback survey after the conference, which was answered by 114 participants (over 42% of the registered participants). Due to space constraints, we cannot include the results of the survey here, but they are available in the full version of our report [Bonifati et al. 2020].

An important take-away for us as organizers was that, although the participants found the online experience indeed somewhat less than the physical experience, it was better than they expected from a virtual conference. Furthermore, we need to keep in mind that this edition was planned and organized in just three weeks, without any external guidelines. There is room for improvement.

Another important take-away was in terms of CO2 reduction of conferences. There was more support from the community than we anticipated in favor of virtualizing (or partly virtualizing) conferences to reduce CO2 emissions. In particular, the broader research community was very supportive of hybrid (both physical and virtual) conferences in order to reduce CO2 footprint (72% of respondents). Over 50% of the respondents supported the idea of alternating physical and virtual conferences to reduce CO2 emission.

4. CONCLUSIONS AND ADVICE FOR FUTURE EVENTS

First-time organization of an online conference is a complicated matter, especially under tight time constraints. In our case, a great effort of coordination was needed and a task force (formed by the people co-authoring this report) made the executive decisions and carried out the required work. On the other hand, once the executive decisions have been made, the organizational amount of work is reasonable. We therefore encourage other conferences to try out the transition to an on-line mode in the short term. In the medium to long term, on-line and/or hybrid conferences may help the community reduce its

CO2 footprint.

With a long-standing experience and within a large time window, things can be arranged more carefully. For instance, one could think about the following issues:

- Carefully choosing the underlying technological platforms on which the conference has to be hosted. People are aware of security issues around Zoom but there is no available equivalent open-source tool that can host the same number of participants. Since a high number of participants is needed for plenary sessions, hopefully such open-source tools will be available in the long run. Dedicated platforms for scientific conferences are urgently needed in that respect.
- Several sessions that require tighter interactions, such as poster and demonstration sessions, need to be planned carefully. For instance, for demonstrations and posters, one could rely on the breakout rooms in Zoom to let people gather around a demo booth or a poster (with a limited number of participants). If the poster or the demonstrated tool can be shared with the participants beforehand, the sessions can be also prepared in advance and be more fluent and interactive. Other sophisticated solutions, such as virtual reality and avatar-based video and chat tools, may be needed in the long run. These tools would help reproducing the physical interactions needed for poster and demo sessions along with the serendipity of meeting people with similar interests at these sessions. Tools like Online Town or Gather Town already take steps in the right direction, but still need to improve to be able to achieve the same level of effectiveness as the real-life experience of a demo or poster session. For instance, physically walking through a room with poster or demo stands gives visitors a very time-efficient overview of the material being presented, which is not yet being matched by virtual tools.
- Networking would greatly benefit from having dedicated online sessions that are scheduled alongside the normal scientific sessions of the conference. Networking is truly the pitfall of an online event and this is especially deleterious for the junior members of our community. An idea would be to prepare networking well in advance and to pin interesting topics or discussions with colleagues of other universities and research teams (a sort of Pinterest specialized for scientific conferences).

Finally, we are pleased to share our experience at online EDBT/ICDT 2020 and eager to learn more about virtual scientific events in the near future. During the climate change session, which has been hosted by the conference this year, we had a lively and stimulating discussion about adopting CO2 plans for conferences. One of the options there is to allow alternate virtual and in-person events or hybrid (simultaneously virtual and in-person) events and thus contribute to reducing the environmental footprint of scientific conferences. Our on-line survey gives us two hopeful signs. First, there is a significant support of the community for going on-line in order to reduce CO2 footprint, and second, attendees clearly found this year's conference better than what they expected a virtual conference to be like.

In the spirit of moving open science and open access forward, the videos of the conference talks have been made accessible directly from the proceedings.[5]

5. REFERENCES

[Bonifati et al. 2020] Angela Bonifati, Giovanna Guerrini, Carsten Lutz, Wim Martens, Lara Mazilu, Norman W. Paton, Marcos Antonio Vaz Salles, Marc H. Scholl, and Yongluan Zhou. 2020. Holding a Conference Online and Live due to COVID-19. *CoRR* abs/2004.07668 (2020).

[5]The EDBT 2020 proceedings are available at https://openproceedings.org/html/pages/2020_edbt.html and the ICDT 2020 proceedings at https://www.dagstuhl.de/dagpub/978-3-95977-139-9