Bulletin of the Technical Committee on

# Data Engineering

September 2019 Vol. 42 No. 3

IEEE Computer Society

## Letters

## Opinions

## Special Issue on Data fairness, Diversification, and Responsible Data Management

## Conference and Journal Notices

**The TC on Data Engineering**

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is `http://tab.computer.org/tcde/index.html`.

**The Data Engineering Bulletin**

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at `http://tab.computer.org/tcde/bull_about.html`.

# Letter from the Editor-in-Chief

Letter from Haixun

Haixun Wang
WeWork Corporation

# Letter from the Special Issue Editor

Letter from Alexandra Meliou

Alexandra Meliou
University of Massachusetts, Amherst

# Resurrecting Middle-Tier Distributed Transactions

Philip A. Bernstein
Microsoft Research, Redmond, WA 98052

## 1 Introduction

Over the years, platforms and application requirements change. As they do, technologies come, go, and return again as the preferred solution to certain system problems. In each of its incarnations, the technology's details change but the principles remain the same. One such technology is distributed transactions on middle-tier servers. Here, we argue that after a 15-year decline, they need to return to the mainstream.

In the 1980's, Transaction Processing (TP) monitors were a popular category of middleware product that enabled customers to build scalable distributed systems to run transactions. Example products were CICS (IBM), Tuxedo (AT&T for Unix), ACMS (DEC for VAX/VMS), and Pathway (Tandem for Guardian) [4]. Their main features were multithreaded processes (not supported natively by most operating systems), inter-process communication (usually a crude form of remote procedure call), and a forms manager (for end users to submit transaction requests). The TP monitor ran on middle-tier servers that received transaction requests from front-end processors that communicated with end-user devices, such as terminals and PC's, and with back end database servers. The top-level application code executed on the middle-tier and invoked stored procedures on the database server.

In those days, database management systems (DBMS's) supported ACID transactions, but hardly any of them supported distributed transactions. The TP monitor vendors saw this as a business opportunity and worked on adding a transaction manager feature that implemented the two-phase commit protocol (2PC). Such a feature required DBMS's to expose Start, Prepare, Commit, and Abort as operations that could be invoked by the TP monitor. Unfortunately, most of them didn't support Prepare, and even if they did, they didn't expose it to applications. They were willing to do so, but they didn't want to implement a different protocol for each TP monitor product. Thus, the XA standard was born, which defined TP monitor and DBMS interfaces (including Prepare) and protocols that allowed a TP monitor to run a distributed transaction across DBMS servers [17].

This middle-tier architecture for distributed transactions was popular for about 20 years, into the late 1990s. Then TP monitors were replaced by Application Servers, which integrated a TP monitor with web servers, so it could receive transaction requests over HTTP, rather than receiving them from devices connected by a local area or terminal network. Examples include Microsoft Transaction Server, later renamed COM+, and Java Enterprise Edition (JEE), implemented by IBM's WebSphere Application Server, Oracle's WebLogic Application Server, and Red Hat's JBoss Application Server [12]. The back end architecture was the same as before. Each transaction started executing on a middle-tier server and invoked stored procedures to read and write the database.

Although this execution model is still widely used, starting in the early 2000's it fell out of favor for new application development, especially for applications targeted for cloud computing. More database vendors offered built-in support for distributed transactions, so there was less need to control the distributed transaction from the middle tier. A larger part of database applications executed on data that was cached in the middle tier. And the NoSQL movement argued that distributed transactions were too slow, that they limited scalability, and that customers rarely needed them anyway [11]. Eventual consistency became all the rage [18].

The critics of distributed transactions had some good points. But in the end, developers found that mainstream

**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

programmers really did need ACID transactions to make their applications reliable in the face of concurrent access to shared data and despite server failures. Thus, some NoSQL (key-value) stores added transaction support (e.g., CosmosDB [2], DynamoDB [8]). Google, which had initially avoided support for multi-row distributed transactions in Bigtable [5], later introduced them in Spanner [6]. There are now many cloud storage services and database products that support distributed transactions.

Like product developers, database researchers have also focused on distributed transactions for back-end database systems. Almost universally, they assume that transactions execute as stored procedures and that middle-tier applications invoke those stored procedures but do not execute the transaction logic themselves.

## 2 Stateful Middle-Tier Applications

This focus on stored procedures is well justified by the needs of traditional TP applications. However, stored procedures are not a good way of encapsulating application logic for a growing fraction of stateful applications that run on the middle tier. These include multi-player games, datacenter telemetry, Internet of Things, and social and mobile applications. Objects are a natural model for the entities in these applications, such as games, players, datacenter hardware, sensors, cameras, customers, and smart phones. Such applications have a large number of long-lived stateful objects that are spread over many servers and communicate via message passing. Like most new applications, these applications are usually developed to run on cloud computing platforms.

These applications typically execute on middle-tier servers, rather than as stored procedures in database servers. They do this for many reasons. They need large main memory for the state of long-lived objects. They often have heavy computation needs, such as rendering images or computing over large graphs. They use a lot of computation for message passing between objects so they can scale out. And they need computation to be elastic, independent of storage requirements. These needs are satisfied by compute servers that are cheaper than database servers because they have less storage. Hence, these apps run on compute servers in the middle tier.

### 2.1 Requirements for Mid-Tier Cloud Transactions

Some middle-tier applications need transactions because they have functions that read and write the state of two or more stateful objects. For example, a game may allow users to buy and sell virtual game objects, such as weapons, shields, and vehicles. A telemetry application may need to process an event exactly once by removing it from a queue and updating telemetry objects based on that event. A social application may need to add a user to a group and modify the user's state to indicate membership in that group. Each of these cases needs an ACID transaction over two or more objects, which may be distributed on different servers. Since these applications are usually developed to run on cloud computing platforms, distributed transaction support must be built into the cloud platform, a capability that is rarely supported today for cloud computing.

Distributed transactions for middle-tier applications on a cloud computing platform have four requirements that differ from those supported by the late-1990's products that run transactions on the middle-tier. First, like all previous transaction mechanisms, they need to offer excellent performance. But unlike previous mechanisms, it's essential that they be able to scale out to a large number of servers, leading to the first requirement: The system must have high throughput and low transaction latency, at least when transactions have low contention, and in addition must scale out to many servers.

To scale computation independently of storage, these applications typically save their state in cloud storage. The developers' choice of cloud storage service depends on their application's requirements (e.g., records, documents, blobs, SQL), their platform provider's offerings (e.g., AWS, Azure, Google), their employer's storage standards, and their developers' expertise. Thus, we have this second requirement: The transaction mechanism must support applications that use any cloud storage service.

The transaction mechanism needs persistent storage to track transaction state: started, prepared, committed, or aborted. Like the apps themselves, it needs to use cloud storage for this purpose, which is the third requirement: The transaction mechanism must be able to use any of the cloud storage services used by applications.

The traditional data structure for storing transaction state is a log. The transaction manager relies on the order of records in the log to understand the order in which transactions executed. Although cloud vendors implement logs to support their database services, they do not expose database-style logging as a service for customers, leading to a fourth requirement: The transaction mechanism cannot rely on a shared log, unless it implements the log itself, in which case the log must run on a wide variety of storage services.

Due to the latency of cloud storage, requirements (2)-(4) create challenges in satisfying requirement (1).

The above requirements are a first cut, based on today's applications and platforms. It is also worth targeting variations. For example, requirement (1) could include cost/performance, which might require a tradeoff against scalability. And (4) might go away entirely if cloud platforms offer high-performance logging as a service.

# 3 An Implementation in the Orleans Framework

The rest of this paper sketches a distributed transaction mechanism that satisfies the above requirements [9]. Our group built it for Microsoft's actor-oriented programming framework, called Orleans, which is open source and runs on both Windows and Linux [16]. The distributed transaction project is part of a longer-term effort to enrich Orleans with other database features to evolve it into an actor-oriented database system that supports geo-distribution, stream processing, indexing, and other database features [3].

## 3.1 Two-Phase Commit and Locking

For ACID semantics, Orleans transactions use two-phase commit (2PC) and two-phase locking (2PL). Our first challenge was to obtain high throughput and scalability despite the requirement to use cloud storage. In our runs, a write to cloud storage within a datacenter takes 20 ms and has high variance. With 2PC, a transaction does two synchronous writes to storage. Therefore, if 2PL is used, a transaction holds locks for 40ms, which limits throughput to 25 transactions/second (TPS). Low-latency SSD-based cloud storage is faster, but still incurs over 10 ms latency, plus higher cost. To avoid this problem, we extended early lock release to 2PC [1, 7, 10, 13, 14, 15]. After a transaction T1 terminates, it releases locks before writing to storage in phase one of 2PC. This allows a later transaction T2 to read/update T1's updated objects. Thus, while T1 is writing to storage, a sequence of later transactions can update an object, terminate, and then unlock the object. To avoid inconsistency, the system delays committing transactions that directly or indirectly read or overwrite T1's writeset until after T1 commits. And if T1 aborts, then those later dependent transactions abort too. Using this mechanism, we have seen transaction throughput up to 20x that of strict 2PL/2PC.

## 3.2 Logging

Our initial implementation used a centralized transaction manager (TM) per server cluster [9]. It ran on an independent server and was multithreaded. Since message-passing is a potential bottleneck, it batched its messages to transaction servers. It worked well with throughput up to 100K TPS. However, it had three disadvantages: it was an obvious bottleneck for higher transaction rates; a minimum configuration required two servers (i.e., primary and backup TM) in addition to servers that execute the application; and it added configuration complexity since TM servers did not run Orleans and thus had to be deployed separately from application servers.

These disadvantages led us to redesign the system to avoid a centralized TM. Instead, we embed a TM in each application object. Each TM's log is piggybacked on its object's storage. This TM-per-object design avoids the above disadvantages and improves transaction latency by avoiding roundtrips to a centralized TM. However, it doesn't work for objects that have no updatable storage. For example, an object that performs a money transfer calls two stateful objects, the source and target of the transfer, but it has no state itself. We allow such an object to participate in a transaction by delegating its TM function to a stateful participant in the transaction, that is, one that has updatable storage.

Orleans transactions write object state to a log to enable undo when a transaction aborts. This is impractical for large objects and is a poor fit for concurrency control that exploits operation commutativity. We therefore developed a prototype that logs operations.

## 4 Summary

Many new cloud applications run their logic on the middle tier, not as stored procedures. They need distributed transactions. Thus, cloud computing platforms can and should offer scalable distributed transactions.

## 5 Acknowledgments

I'm grateful to the team that built the initial and final implementations of Orleans transactions: Jason Bragg, Sebastian Burckhardt, Tamer Eldeeb, Reuben Bond, Sergey Bykov, Christopher Meiklejohn, Alejandro Tomsic, and Xiao Zeng. I also thank Bailu Ding and Dave Lomet for suggesting many improvements to this paper.

## References

[1] Athanassoulis, Manos ; Johnson, Ryan ; Ailamaki, Anastasia ; Stoica, Radu, Improving OLTP Concurrency through Early Lock Release, EPFL-REPORT-152158, https://infoscience.epfl.ch/record/152158?ln=en, 2009.

[2] Azure CosmosDB, https://azure.microsoft.com/en-us/services/cosmos-db/

[3] Bernstein, P.A., M., T. Kiefer, D. Maier: Indexing in an Actor-Oriented Database. CIDR 2017

[4] Bernstein, P. A., E. Newcomer: Chapter 10: Transactional Middleware Products and Standards, in Principles of Transaction Processing, Morgan Kaufmann, 2nd ed., 2009.

[5] Chang, F., J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber: Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26(2): 4:1-4:26 (2008)

[6] Corbett, J.C. et al: Spanner: Google's Globally Distributed Database. ACM Trans. Comput. Syst. 31(3): 8:1-8:22 (2013)

[7] DeWitt, D.J., R.H. Katz, F. Olken, L.D. Shapiro, M. Stonebraker, D.A. Wood: Implementation Techniques for Main Memory Database Systems. SIGMOD 1984: 1-8

[8] DynamoDB, https://aws.amazon.com/dynamodb/

[9] Eldeeb, T. and P. Bernstein: Transactions for Distributed Actors in the Cloud. Microsoft Research Tech Report MSR-TR-2016-1001.

[10] Graefe, G., M. Lillibridge, H. A. Kuno, J. Tucek, A.C. Veitch: Controlled lock violation. SIGMOD 2013: 85-96

[11] Helland, P., Life beyond Distributed Transactions: an Apostate's Opinion. CIDR 2007: 132-141

[12] Java EE documentation, http://www.oracle.com/technetwork/?java/javaee/documentation/index.html

[13] Larson, P-A, et al.: High-Perf. Concurrency Control Mechanisms for Main-Memory Databases. PVLDB 2011

[14] Levandoski, L.J., D.B. Lomet, S. Sengupta, R. Stutsman, R. Wang: High Performance Transactions in Deuteronomy. CIDR 2015

[15] David B. Lomet: Using Timestamping to Optimize Two Phase Commit. PDIS 1993: 48-55

[16] Orleans, http://dotnet.github.io/orleans

[17] The Open Group, Distributed Transaction Processing: The XA Specification, http://pubs.opengroup.org/onlinepubs/009680699/toc.pdf.

[18] Vogels W., Eventually Consistent. ACM Queue 6(6): 14-19 (2008)

Data Engineering

It’s FREE to join!

# TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

## Join TCDE via Online or Fax

**ONLINE**: Follow the instructions on this page:

**www.computer.org/portal/web/tandc/joinatc**

**FAX:** Complete your details and fax this form to **+61-7-3365 3248**

Name ____________________

IEEE Member # ____________________

Mailing Address ____________________

____________________

Country ____________________

Email ____________________

Phone ____________________

### TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

### Membership Questions?

**Xiaoyong Du**
Key Laboratory of Data Engineering
and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

### TCDE Chair

**Xiaofang Zhou**
School of Information Technology and
Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314