WedgeDB: Transaction Processing for Edge Databases

Abhishek A. Singh, Faisal Nawab
 {abasingh,fnawab}@ucsc.edu
 University of California, Santa Cruz

ABSTRACT

Wide-area Edge Database (WedgeDB) span globally and store data closer to the users. We term this the Global Edge Data management problem. In such an environment, aspects such as data storage, retrieval, transaction processing, and protection from malicious actors need to be addressed by any data management system that aims to consider itself a viable solution. Although blockchain technology (both permissioned and permissionless) has provided a way to address these concerns, transaction processing in these environments is still challenging. WedgeDB is an attempt to address these security problems in edge-cloud data systems [2].

The main goals of WedgeDB are to support distributed transaction processing coupled with secure transaction execution. Data stored in WedgeDB is partitioned into clusters with each cluster handling a unique set of keys. WedgeDB is a collection of such clusters. This is shown in figure 1. In figure 1, each cluster C_x stores a unique set of keys. This partitioning scheme allows us to build a distributed transaction model which runs transactions on a subset of clusters in the network. Transactions in WedgeDB are serializable. In WedgeDB, clients perform read operations as part of a transaction via read requests which can be sent to any of the WedgeDB nodes. The read operations are added to the transaction's history. Write operations are cached by the client until a commit is called which sends the transaction object containing the read history and write operations to a WedgeDB node to be committed.

Transactions are processed in batches called *Epochs*. Each cluster maintains a leader which receives transactions and groups them into epochs. A cluster in WedgeDB contains $3f + 1$ nodes (where f is the number of tolerable faulty nodes) and PBFT[1] is used to attain consensus among the nodes when executing transactions. Keys modified during an epoch are added to a Merkle tree which is used to verify changes to keys handled by the cluster. During transaction execution, proof of transaction execution is generated in the form of signed data blocks by the nodes in the cluster. At least $f + 1$ signed messages must be gathered before an epoch can be committed. These data blocks along with the root of the Merkle tree are stored in an SMR log where each entry in the SMR log corresponds to an epoch. Transactions that contain keys from different clusters are executed via two-phase commit. During the prepare phase a remote cluster executes PBFT within its local cluster and checks for dependency violations before moving ahead with the

commit phase. Committed epoch may not have committed transactions and therefore an additional parameter is used to indicate the last committed epoch. This parameter combined with the dependency vector help in finding out serializability violations and abort transactions.

Figure 1: WedgeDB system architecture; The node A^* in each cluster acts as a leader for transaction processing.

With WedgeDB, transactions that affect only a few clusters do not require global consensus to commit. Transactions that read keys from a number of clusters do not require consensus at the server nodes and allow data to be read from a consistent snapshot of the WedgeDB network. We thus, present WedgeDB as a viable solution to the problem of global edge data management.

CCS CONCEPTS

• **Information systems** → **Data management systems**; • **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

transaction processing, distributed systems, distributed databases, edge computing

ACM Reference Format:

Abhishek A. Singh, Faisal Nawab. 2019. WedgeDB: Transaction Processing for Edge Databases. In *ACM Symposium on Cloud Computing (SoCC '19)*, November 20–23, 2019, Santa Cruz, CA, USA. ACM, New York, NY, USA, 1 page. <https://doi.org/10.1145/3357223.3365444>

ACKNOWLEDGMENTS

This research is supported in part by the NSF under grant CNS-1815212.

REFERENCES

- [1] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *OSDI*. 173–186. <http://dl.acm.org/citation.cfm?id=296806.296824>
- [2] Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. 2018. Dpaxos: Managing data closer to users for low-latency and mobile applications. In *SIGMOD*. ACM, 1221–1236.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
 SoCC '19, November 20–23, 2019, Santa Cruz, CA, USA
 © 2019 Association for Computing Machinery.
 ACM ISBN 978-1-4503-6973-2/19/11...\$15.00
<https://doi.org/10.1145/3357223.3365444>