Boosting Performance and Scalability in Cloud-deployed Databases

J. E. Armendáriz-Iñigo[1], J. Legarrea[1], J. R. González de Mendívil[1], A. Azqueta-Alzuaz[1], M. Louis-Rodríguez[1], I. Arrieta-Salinas[1] and F. D. Muñoz-Escoí[2]

[1]*Dpto. de Ing. Matemática e Informática, Univ. Pública de Navarra, Campus de Arrosadía, 31006 Pamplona, Spain*

[2]*Instituto Tecnológico de Informática, Univ. Politècnica de València, 46022 Valencia, Spain*

Keywords: Uncritical Data, Databases, Transactions, Replication, Scalability.

Abstract: Eventual consistency improves the scalability of large datasets in cloud systems. We propose a novel technique for managing different levels of replica consistency in a replicated relational DBMS. To this end, data is partitioned and managed by a partial replication protocol that is able to define a hierarchy of nodes with a lazy update propagation. Nodes in different layers of the hierarchy may maintain different versions of their assigned partitions. Transactions are tagged with an allowance parameter *k* that specifies the maximum degree of data outdatedness tolerated by them. As a result, different degrees of transaction criticality can be set and non-critical transactions may be completed without blocking nor compromising the critical ones.

1 INTRODUCTION

Distributed architectures have been proposed as a solution to deal with diverse issues of web systems, like component development to service an increasing number of users, replicating dynamic web content, achieving fault-tolerance and performance by replica proximity. So far, however, there has been little discussion about web data critical nature influence in the replication techniques. Our objective is to determine whether it is possible to work in a replicated environment and manage web data criticality to increase the system performance, its scalability and availability.

We consider systems with two types of data: critical and non-critical. For example, information about hotels and their rates can be found on some websites. One can think that the information about their availability and rates is non-critical while the booking process is critical. This paper emphasizes the treatment of such non-critical data in a special way to boost the system throughput.

Data is partially replicated; i.e., no replica holds the entire state of the database. Data consistency is managed with a hybrid replication protocol following the ideas presented in (Arrieta-Salinas et al., 2012). This system assigns a certain number of replicas to each partition and these replicas are placed following an onion structure. The core layer will have the most recent data version while outer layers will have stale data versions, though still consistent. Thus, we implement adjustable consistency so it provides strong consistency at the core and eventual consistency in the outer layers.

This model is extended to handle the critical/non-critical data. The client sets up which data is non-critical. The system partitions the data in a smart way (Curino et al., 2010). Some non-critical data is inside each partition and the critical data is accessed in the core while non-critical data will be modified and accessed in the outer layers. The latter also forms another set of multiversioned layers where the original core (critical data) of the partition behaves as an outer layer of non-critical data. Thus, we are generating different data cores inside each partition.

We set some replication rules to infer if non-critical data will need to process updates or defer them in order to provide the consistency demanded by the application. Clients will identify their transaction as non-critical. This kind of transaction will commit but its changes will not be viewed in other nodes until a given time or on demand. This increments the system throughput by delaying non-critical transactions.

The rest of the paper is organized as follows: Sections 2 and 3 deal with the motivation and model of our proposal. Section 4 describes how to include the features of critical and non-critical data in a system.

E. Armendáriz-Iñigo J., Legarrea J., R. González de Mendívil J., Azqueta-Alzúaz A., Louis-Rodriguez M., Arrieta-Salinas I. and D. Muñoz-Escoí F..
Boosting Performance and Scalability in Cloud-deployed Databases.
DOI: 10.5220/0004405605250528
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 525-528
ISBN: 978-989-8565-52-5

2 MOTIVATION

Observing the increase of web services, we can notice that some of the data managed by those systems could be classified as really critical information, like money management transactions or on-time auction transactions but some of the data in the same system could be classified as uncritical information, like static information featured by your bank or auction site. Our approach tries to maximize the scalability and availability of the whole system by taking this novel approach.

Web applications choose to store data on databases with Snapshot Isolation (SI) (Berenson et al., 1995). This is due to the non-blocking nature of each read operation executed under that isolation level, as it reads from a snapshot of committed update transactions up to that transaction beginning.If we extend the notion of SI to a replicated environment we obtain the Generalized Snapshot Isolation (GSI) level (Elnikety et al., 2005): the snapshot got by a transaction could be any of the previous history of committed transaction up to its start; thus, SI is a particular case of GSI. This will cause a benefit in replicated databases, as clients access their closest replicas and reduce their latency.

Replicated databases run a replication protocol to manage transactions. It is well known that replication protocols perform differently depending on the workload characteristics. For instance, a read intensive partition may provide a higher throughput with a primary-backup scheme (Wiesmann and Schiper, 2005). On the contrary, a partition whose items are frequently updated might benefit from an update everywhere replication solution based on total order broadcast such as certification based replication (Wiesmann and Schiper, 2005). However, update everywhere protocols suffer from a serious scalability limitation, as the cost of propagating updates in total order to all replicas is greatly affected by the number of involved replicas.

We take the system presented in (Arrieta-Salinas et al., 2012) as a basis to provide higher scalability and availability. Hence, data is partitioned (Curino et al., 2010) and each partition is placed in a set of replicas, say M, where K of them run a given replication protocol (either update everywhere or primary backup) and the rest $(M-K)$ are placed in a replication tree whose depth and composition depends on the application. Several, or all, of the K replicas act as primaries for other backup replicas (those of the first level in the tree) which would asynchronously receive updates from their respective primaries. At the same time, backup replicas could act as pseudo-primaries for other replicas placed at lower layers of the hierarchy, thus propagating changes along the tree in an epidemic way. If we augment the replication degree of a given partition, then we can forward transactions to different replicas storing it, and thus, transactions will be more likely to obtain old, though consistent, snapshots (GSI) and alleviating the traditional problem of scalability in the core.

The novelty in our approach is to take advantage of this replication hierarchy and place non-critical data along the hierarchy tree and proceed with it in a similar way as with normal data. We will run a primary copy protocol based on the principles given in the COLUP (Irún-Briz et al., 2003) algorithm. The resulting protocol increases the performance and reduces the abort rate of non-critical update transactions, by re-partitioning (apart from its original partitioning based on graphs or any other approach) the data according to its critical nature. Hence, having a traditional partitioning schema where critical and non-critical data are placed in the same partition, we establish that a certain replica should handle the updates of non-critical data while critical data is updated at another replica. Under this assumption, critical transactions can be executed faster and not interleaved with transactions accessing uncritical data. Those uncritical transactions may access older data. Meantime, other critical transactions could be scheduled, increasing the age of the snapshots accessed by uncritical transactions. However, every uncritical transaction is characterized by a threshold on the age of the data it needs to access. As a result, accessing old data is tolerated by these transactions in the regular case and such situation will not necessarily lead to their abortion.

Compared with traditional GSI, what our model does is to try to anticipate when a transaction is going to impact (i.e., to present a write-write conflict) with other transactions and when a non-critical transaction is going to impact, then we control an alternative mechanism. In that case, a transaction A is aborted in the validation phase only when at least one of its conflicting transactions is critical (or uncritical but with an allowing threshold lower than that of A).

Uncritical transactions are characterized by an *allowance* parameter k. The value of k indicates the number of missed updates tolerated by the uncritical transaction. This value is 1 or greater than 1 for uncritical transactions. Implicitly, critical transactions are those that access at least one critical item and have a zero value for k. When conflicts arise between transactions that access critical data (i.e., critical transactions) and transactions that only access uncritical data (i.e., uncritical transactions), no critical transaction

may be aborted by an uncritical transaction. Additionally, conflicts between uncritical transactions are allowed in some applications (i.e., in those applications admitting different values for the k parameter). For instance, an application may decide that it needs three different kinds of transactions: (a) critical ones with $k = 0$, (b) intermediate uncritical ones with $k = 4$, and (c) relaxed uncritical ones with $k = 10$. In that scenario, a transaction with $k = 4$ is able to tolerate (and overwrite) conflicts generated by transactions with $k = 10$ without being aborted in its validation step. Note that with this validation strategy, a critical transaction C will be never aborted by any concurrent conflicting transaction with $k > 0$ that had been committed while C has been executed.

Nodes maintaining database replicas should monitor the update frequency over their non-primary data partitions. Based on this, they are able to forecast the "age" of their maintained snapshot; i.e., the number of transaction writesets that have been applied in its primary replica but that have not been applied yet in the local database replica. When a transaction B is started with a k value lower than the local forecast snapshot age, a forced update propagation is requested from the primary replica. Transaction B is not started until such update propagation is completed.

3 MODEL

The system model, shown in Figure 1, is divided into: a) a set of *client applications*; b) a *metadata manager* (MM) that holds the system state (stored in the metadata repository) and orchestrates the communication with both clients and replicas in the replication clusters; and, c) a set of *replication clusters* (RC), each storing one data partition.

Client applications interact with the system by using a library, which acts as a wrapper for the management of connections with both the MM and the replicas that serve the transactions. In order to submit a transaction, the client library sends a request with a *non-critical* data threshold to the MM. This information determines the partition (or partitions) involved in the transaction, selects one of the replicas of the replication cluster storing that partition and sends the address of the selected replica(s) to the client. Then, the client directly submits the transaction operations to the indicated replica(s). The client library maintains a cache with replica addresses to avoid performing a request for each transaction.

The MM module is in charge of maintaining the metadata repository, which contains the following information: a mapping between each data item, its

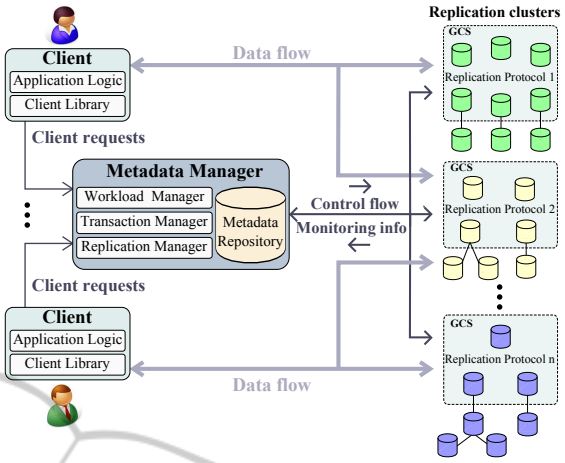

Figure 1: System model.

criticality and the partition it is stored in; a mapping between each data partition (critical/non-critical) and the set of replicas that belong to the RC that handles the partition and their respective hierarchy (for critical partitions, it is needed the replication protocol running on the core level); and, status and metrics of each replica.

This information has stringent consistency and availability requirements. For this reason, the MM can be replicated among a small set of nodes to provide fault tolerance while ensuring consistency with a Paxos algorithm. The information of the metadata repository is used and updated by the following components of the MM:

- Workload manager: it monitors the set of active replicas in the system. Every active replica must periodically send a heartbeat message to let the workload manager know that it is alive. These messages also attach information regarding the status of the sending replica. This component is also responsible for determining the partitioning scheme and deciding when a replica should be upgraded or downgraded in the hierarchy.
- Transaction manager: it assigns partitions to replicas and synchronizes data when a transaction accesses data items stored in several partitions.
- Replication manager: it chooses the replication protocol that best fits for each replication cluster and determines the hierarchy level that corresponds to each replica.

Each RC of Figure 1 consists of a set of replicas organized as a hierarchy of levels. Recall that inside each partition, data is split in critical and non-critical data. Both types of data have their own hierarchy of levels. Let us start with the critical data, the core level comprises a group of replicas that propagate updates among themselves by means of a traditional dis-

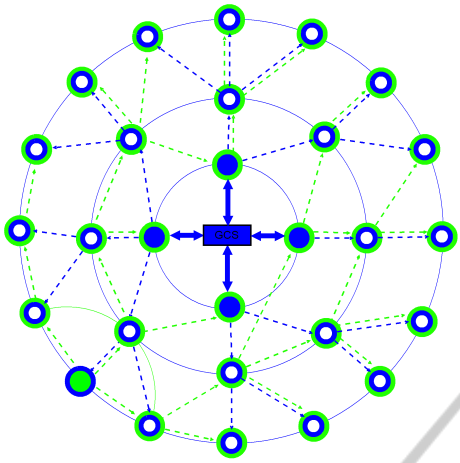

Figure 2: Critical (blue) and non-critical (green) data management and epidemic propagation of updates in our system.

tributed replication protocol (Wiesmann and Schiper, 2005) that makes use of a group communication system to handle the messages among replicas and monitor the set of replicas belonging to the group (blue filled circles in Figure 2). The core level of each RC may be managed by a different replication protocol, which will be determined by the replication manager of the MM depending on the current workload characteristics. On the other hand, the replicas that do not belong to the core level of the hierarchy can be distributed into several levels forming a tree whose root is the aforementioned core level, where a replica that belongs to a given level acts as a backup for a replica of its immediately upper level and may also act as a primary for one or more replicas of its lower level (circles whose line colors are green and blue in Figure 2). These replicas communicate with their respective primaries using reliable point-to-point channels. On the non-critical data side, the MM will choose for each RC one replica along the hierarchy level as the core of these data items (the green filled circle in Figure 2) and the rest of replicas will constitute the non-critical hierarchy tree of this RC. With the aim of exploiting the advantages of in-memory approaches, we assume that every replica keeps all its data in main memory.

4 CONTRIBUTIONS

We are currently developing a partitioning algorithm (Curino et al., 2010) adapted to our system. We have successfully included the definition of the proper replication protocol and the composition of the replication hierarchy tree. We are going to increase its features so as to infer which data should be considered as non-critical (e.g. data hardly updated). However, we also consider that this information can be set in advance by the application. Once the MM establishes the replica that interacts with the client for a given partition, this replica may manage non-critical transactions. Update transactions are ruled by COLUP (Irún-Briz et al., 2003), so these transactions are executed in the primary and lazily propagated to other replicas. We are going to adapt it to our system by way of three different variants: read the data stored locally (pure lazy approach); ask for the last version to the primary copy; or, a hybrid approach. This hybrid approach may be thought as follows, along with the propagation of updates (either critical or non-critical) it can be piggybacked the current version of the non-critical data. Meanwhile, the user can set a threshold at the beginning of the transaction saying that it is fine to read data locally if data is not older than k versions; otherwise, it will need to connect to the primary and retrieve the most recent versions.

ACKNOWLEDGEMENTS

This work has been funded by the Spanish Government and European FEDER under research grants TIN2009-14460-C03 and TIN2012-37719-C03.

REFERENCES

Arrieta-Salinas, I., Armendáriz-Iñigo, J., and Navarro, J. (2012). Classic replication techniques on the cloud. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 268 –273.

Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., and O'Neil, P. (1995). A critique of ANSI SQL isolation levels. *SIGMOD Rec.*, 24(2):1–10.

Curino, C., Zhang, Y., Jones, E. P. C., and Madden, S. (2010). Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, 3(1):48–57.

Elnikety, S., Zwaenepoel, W., and Pedone, F. (2005). Database replication using generalized snapshot isolation. In *SRDS*, pages 73–84. IEEE Computer Society.

Irún-Briz, L., Muñoz-Escoí, F. D., and Bernabéu-Aubán, J. M. (2003). An improved optimistic and fault-tolerant replication protocol. *Lecture Notes in Computer Science*, 2822:188–200.

Wiesmann, M. and Schiper, A. (2005). Comparison of database replication techniques based on total order broadcast. *IEEE Trans. Knowl. Data Eng.*, 17(4):551–566.