ITI-SIDI-2012/006

Consistency in Scalable Systems

M. I. Ruiz-Fuertes, M. R. Pallardó-Lozoya, F. D. Muñoz-Escoí

Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, Spain

{miruifue,rpallardo,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2012/006

M. I. Ruiz-Fuertes et al.: *Consistency in Scalable Systems*

Consistency in Scalable Systems

M. I. Ruiz-Fuertes, M. R. Pallardó-Lozoya, F. D. Muñoz-Escoí

Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, Spain

Technical Report ITI-SIDI-2012/006

e-mail: {miruifue,rpallardo,fmunyoz}@iti.upv.es

June 5, 2012

Abstract

While eventual consistency is the general consistency guarantee ensured in cloud environments, stronger guarantees are in fact achievable. We show how scalable and highly available systems can provide processor, causal, sequential and session consistency during normal functioning. Failures and network partitions negatively affect consistency and generate divergence. After the failure or the partition, reconciliation techniques allow the system to restore consistency.

1 Introduction

Scalable systems, such as cloud systems, are composed of multiple data centers, each one of them composed of a set of nodes that are located in the same facility and are locally connected through a high-speed network. Different data centers are geographically distant and connected by some inter-data-center channels, with limited bandwidth and longer transmission delays (when compared to intra-data-center networks). In these systems, each data item should have multiple replicas and these replicas should be spread over multiple data centers in order to increase both availability and fault tolerance. However, such geographical dissemination of replicas entails greater difficulty for maintaining the consistency among them. The CAP theorem [9, 18] states the impossibility of a distributed system to simultaneously provide consistency, availability and partition tolerance. Since availability is the key for scalable cloud systems, a trade-off appears between consistency and network-partition tolerance. As network partitions may be common among remote data centers, scalable systems generally sacrifice consistency, trading it for partition tolerance. In this case, and as a minimum, eventual consistency [31] can be guaranteed.

The aim of this paper is to study the different levels of consistency [27] that can be achieved in these systems, even when network partitions occur and always guaranteeing availability. For this, we take as basis previous results in the interconnection of distributed shared memory consistency models [17, 12] as well as in the interconnection of message passing systems [23, 5, 2].

In distributed shared memory systems (DSM systems), interconnectable memory consistency models are able to *export* intra-group consistency to other connected groups or subsystems, without changing them. To this end, they only require FIFO channels between subsystems in the regular case and so, they are trivially admitted in a network-partitionable system. In case of a partitioned network, the updates to be transmitted are buffered and re-sent when the channel is repaired. Exporting this idea into scalable systems allows different data centers to operate independently during partitions and to update or reconcile replicas as necessary once the network is repaired. The cache [19], FIFO/pRAM [26] and causal [22] consistency models are interconnectable [17, 12]. The sequential model [25], however, cannot be interconnected in a non-intrusive manner [12]. Similar results are achieved when interconnecting message passing systems (groups of processes that exchange messages through a Group Communication System, or GCS) to provide end-to-end delivery semantics: FIFO and causal semantics can be provided among different subsystems [23, 5, 2], but total/atomic semantics require intrusive modifications that penalize performance of individual subsystems [23].

Upon these previous results, this paper studies different possible configurations for scalable database replication systems, analyzing the consistency level provided in each one and proposing recovery and reconciliation techniques that allow the system to restore consistency after failures or network partitions. To this end, we subdivide the persistent data into multiple disjoint sets (data partitioning) and use passive replication. We show that it is possible to provide stronger levels of consistency than the usual eventual consistency.

The rest of this paper is structured as follows. Section 2 summarizes previous work on which this paper is based. Section 3 proposes different scalable configurations and analyzes their consistency guarantees even in case of failures or network partitions. Section 4 discusses some reconciliation techniques that are required to resolve divergence between data centers. Finally, Section 5 concludes the paper.

2 Related Work

Different previous works propose mechanisms for the interconnection of multiple local groups in order to form a global system with certain properties. Fernandez et al. [17] interconnect two causal distributed shared memory systems with a bidirectional reliable FIFO channel connecting one process from each system, in such a way that the resulting DSM system is also causal. As authors highlight, the same mechanism can be used to construct a global causal system by the interconnection of sequential or atomic DSM systems, as these consistency models also respect causality.

Cholvi et al. [12] define a DSM memory model as fast if memory operations in such model require only local computations before returning control, even in systems with several nodes. Otherwise, the model is said to be non-fast. Systems implementing fast models can be interconnected without any modification to the original systems. However, systems implementing non-fast models cannot be interconnected in a non-intrusive manner. Authors propose a system architecture where application processes are executed in the nodes of the distributed system and groups of nodes are provided with the shared memory abstraction by a memory consistency system, or MCS, composed by MCS-processes that also run in the nodes. In order to interconnect two of such distributed systems, an interconnection system, or IS, is used. An IS is a set of processes (IS-processes), one per each system to interconnect. The IS-process of each system is an application process that has access to shared memory as any other application process. IS-processes exchange information between them using a reliable FIFO communication network. Different algorithms are proposed for those IS-processes to interconnect systems whose MCS provide certain memory consistency model: FIFO-ordered pRAM, globally-ordered causal, or cache.

Other authors focus on a similar concept: the interconnection of message passing systems in order to provide end-to-end delivery semantics. In this regard, Johnson et al. [23] propose an architecture in which different process groups are connected by means of inter-group routers. An inter-group router is a special process which is capable of forwarding messages between two or more communication protocols. This way, when a process wants to send an intra-group message it uses the local communication protocol, but inter-group communication requires the collaboration of inter-group routers, which communicate with each other using one or more inter-group communication protocols. With this architecture, two systems locally providing FIFO delivery semantics can be interconnected to form a global system with end-to-end FIFO delivery semantics, by using FIFO delivery for the communication among routers. For two systems locally providing causal delivery semantics, end-to-end causal delivery semantics are also guaranteed by using FIFO delivery for the group of routers. Authors also study the conditions of the interconnection of multiple subsystems with one or more groups of routers and show that end-to-end total order delivery cannot be provided without modifying local communication protocols. Indeed, to achieve end-to-end total order delivery, messages must initially be sent to the inter-group router only, which will then send them to the inter-group communication protocol which must totally order the messages and deliver them back to the inter-group routers. Only then can the routers deliver the messages to their local groups.

Baldoni et al. [5] propose a hierarchical daisy architecture for providing end-to-end causal delivery. In a similar way as previous works, each local group contains a causal server that performs the interconnection and is also part of a group of causal servers. Multiple local groups interconnected in this way by a single group of causal servers form a daisy. Several daisies can also be interconnected by means of hyper servers grouped together in a hyper servers group. Communication inside each group (local group, causal servers group, hyper servers group) is done by means of a causal broadcast primitive provided by a GCS.

Although these last works focus on communication protocols, their results are easily extrapolated to memory consistency models, as usually broadcasts are used to propagate write operations and the delivery of a broadcast message would be equivalent to a read operation. This way, FIFO broadcasts would provide pRAM/FIFO consistency; causal broadcasts would enable causal consistency; and FIFO total order broadcast would be used to ensure sequential consistency.

3 Scalable Configurations

The aim of this paper is to propose different interconnection architectures and protocols in order to connect different data centers to form a geographically extended scalable system where availability and network partition tolerance are major concerns, while analyzing the provided level of consistency among replicas. As seen before, previous related work suggests that causal consistency is possible in the resulting system. Obviously, more relaxed consistency levels could be also provided. Finally, we study if it is possible to add additional constraints that allow the system to guarantee a pseudo-sequential consistency and even session consistency among replicas. Next we describe different scalable configurations providing different levels of replica consistency.

3.1 Basic Configuration: Processor Consistency

The database is partitioned. Following the model of primary copy [10], each data center is the master of one partition and stores backups of the rest of partitions, thus providing full replication of the database.[1] Transactions are restricted to write into at most one partition, although they can read any number of partitions. This is needed to physically serve each transaction in only one data center and thus avoid the delays of forwarding operations of on-going transactions to other data centers. Update transactions to a given partition must be addressed to the data center which is the primary for that partition. However, inside that data center, any node can serve transactions. Those updates are propagated through a FIFO total order broadcast among the nodes of the same data center, which perform a validation process on each transaction. Validated updates are then lazily propagated by a selected node to the rest of data centers through FIFO channels. All communication is assumed to be reliable (if a message is delivered at an available node, it will be delivered at all available nodes). In short, in this basic configuration each data center acts as a single node of a traditional primary-backup system.

To perform the propagation of updates to remote data centers, a node in each data center acts as a router. Routers belong to both the local group of their data center and to the so called group of routers. As a member of the local group, a router receives all the messages broadcast inside the group, i.e., all the totally ordered messages that contain the writesets to validate and (if accepted) apply in the database. The router may or may not serve user transactions or store and update its own copy of the data. But it validates writesets and later *exports* the positively validated ones to the routers group, using a FIFO broadcast. When a router R_i receives an exported writeset, it employs its local FIFO total order broadcast primitive to spread it into its own data center. As such communication primitive respects FIFO order, all imported messages will be delivered in their original order, although they may be alternated with local messages. Nodes can easily distinguish messages that must be validated from messages that correspond to remote, already validated writesets, by simply including a partition identifier in the messages.

The replica consistency perceived by the users (the user-centric replica consistency) in each individual partition is sequential, independently of the data center used to access the partition: either at its primary

[1] This is a simplification without loss of generality. It is also possible that a data center is the primary copy of two or more partitions or none. If several partitions are mastered by the same data center, we can consider all of them as only one partition. If a data center is not the primary copy of any partition, it only stores backups. Thus, it is not necessary to divide the database in a number of partitions equal to the number of data centers.

DC_1		DC_2		
P_1	P_2	P_1	P_2	
0	0	0	0	initial situation
1	0	0	0	T_1 updates P_1 in its primary copy
1	0	0	1	T_2 updates P_2 in its primary copy
1	0	1	1	updates of P_1 arrive to DC_2
1	1	1	1	updates of P_2 arrive to DC_1

Figure 1: Sequence of states at each data center

data center or at any backup, the sequence of states of that partition will be the same (although at a given moment in time, backups may probably be outdated with regard to the primary copy).

With this basic configuration, as long as each transaction accesses to only one partition, either for reading (at any data center) or for writing (at the corresponding primary data center), the image users perceive will be sequential. Of course, reading from backups may result in outdated values (inversions occur [28]), but always complying with sequential consistency. However, as soon as a transaction accesses to two or more partitions, no sequential guarantee is provided regarding the state between partitions. That is the case of Microsoft SQL Azure system [11]. In that system, relational databases are partitioned and each partition is replicated *n* times (*n-safe* property) following the primary-backup model. Transactions inside a partition have full ACID guarantees, but queries that read data from two or more partitions will not achieve full ACID consistency. A similar case occurs in Google Megastore [4], which is a layer placed on top of the key-value Bigtable system. In Megastore, each key-value pair is assigned to an entity group, so each entity group is a database partition with regular ACID properties and SQL-like interface. Entity groups are synchronously replicated and distributed in different data centers, ensuring strong consistency and availability. Otherwise, consistency between entity groups is relaxed, so inter-entity group transactions require a distributed commit protocol.

To illustrate the lack of sequential consistency when multiple partitions are accessed by the same transaction, suppose a system composed of two data centers, DC_1 and DC_2, and a database divided into two partitions, P_1 and P_2. DC_1 is the master of P_1 and backup of P_2. DC_2 is the master of P_2 and backup of P_1. Considering a partition as a single object that changes of version, we can establish the sequence of states each data center goes through as the concatenation of the versions of the stored partitions. Initially, all partitions have version 0. Now transaction T_1 updates P_1 in its primary copy, DC_1. Before those updates arrive to DC_2, transaction T_2 updates P_2 in DC_2. Afterwards, updates from DC_1 are applied in DC_2 and, finally, updates of DC_2 are applied in DC_1. The sequence of states of each data center is depicted in Fig. 1.

The final state of both data centers, when all versions are 1, reflects the eventual consistency generally guaranteed by scalable systems. However, the sequence of states is different at each data center. Each individual partition maintains the sequentiality in all the system (it goes from version 0 to version 1 in that order in every data center), but the database as a whole does not follow the same sequence everywhere. Such lack of global sequentiality can be perceived by any transaction that reads both partitions. For instance, a user can read in DC_1 the state 1-0 and afterwards move to DC_2 and read the state 0-1: partition P_2 has advanced towards a more up-to-date state, but P_1 has gone *backwards* in time.

With the primary-backup approach, only the primary is able to generate updates for a given partition. These updates are later propagated among data centers with FIFO communication, which maintains the original order of updates. As a result, every node in the system sees the same sequence of writes for each data item, thus guaranteeing cache consistency. On the other hand, messages sent by the same node are maintained in FIFO order by the local broadcast primitive providing FIFO total order. The same as before, such ordering is maintained when propagating among remote data centers, thus ensuring pRAM consistency. The combination of cache and pRAM results on a general image of processor consistency [19, 1]. For mono-partition transactions, the image users perceive is guaranteed to be sequential. This is a very interesting result as it ensures a strong consistency level when each transaction accesses to only one partition. In the case of systems based on stored procedures, such condition could be relatively easy to ensure, thus achieving a scalable system with a *pseudo-sequential* level of consistency (the consistency is sequential from the point of view of users, although internally only a processor level is globally guaranteed).

Availability must be always guaranteed: accesses to the data are allowed even in the presence of failures or when the network is partitioned and different data centers are isolated during relatively long time intervals.

In case of network partition, when communication among data centers is interrupted, the updates generated at one data center cannot be sent to some remote data centers, which will maintain outdated backups of those partitions. However, as long as communication with users is maintained, availability is ensured and consistency remains at the processor level. Once the network is repaired, all buffered updates are broadcast to other data centers, thus updating all backups.

Partial failures in a data center, when only some of the nodes of a data center crash, are tolerated and managed by the remaining nodes of the data center. To this end, alive nodes assume the load of the data center and buffer all delivered messages, which are transferred in batch to the failed node as soon as it recovers. However, in order to guarantee availability, should a data center suffer a generalized failure or be completely isolated from other data centers and all its users due to a network partition (two situations that are not distinguishable from the outside), another data center must be promoted as the new master for the partition whose primary copy is stored in the isolated or failed data center. The new master must be chosen deterministically, and this election can be based or not in the state of data centers (e.g., to choose as master the most updated data center or any available data center). Although the transactions that were in execution in the failed data center are lost, the new master will manage the incoming updates to that partition during the downtime of the original master. To avoid divergence, however, the new primary should have such a partition completely up-to-date. If this is not the case, due to the lazy propagation of updates, a reconciliation could be necessary afterwards, once the original master is again accessible. Reconciliation techniques are discussed in Section 4.

A last problem must be considered. Due to a network partition or to a false suspicion of failure, a data center may become the new master of a given partition whose original master is still accessible by some users. This is the case when a network partition isolates a data center from other data centers but not from its users. As a result, not only the new master may start from an outdated copy of the partition but also both data centers accept updates over the same partition, thus increasing divergence. Once the network is repaired and the master duplicity is noticed, one of the involved data centers will stop being master and both will propagate their updates to all data centers. Before returning to normal functioning, the divergence of such updates must be resolved with reconciliation techniques in order to achieve an agreement on the final state of the partition.

Once reconciliation is completed, the state of partitions is agreed and the initial consistency guarantees are restored. During the divergence, however, FIFO is the only consistency guarantee that is ensured. On the other hand, divergence does not always appear during failures or network partitions. Indeed, if there is no master duplicity and new masters start from an updated copy of the partition, then the processor level of consistency is maintained.

3.2 Causal Consistency

While mono-partition transactions already perceive sequential consistency with the basic configuration, we study other configurations that increase the consistency perceived by multi-partition transactions.

Previous work showed that it is possible to interconnect causal DSM systems [12] and to achieve end-to-end causal semantics among multiple message passing systems [5]. In scalable systems, using a causal propagation of updates can raise the consistency for multi-partition transactions from processor to causal. The architecture of the system and its way of functioning is the same as in the basic configuration for processor consistency; the only aspect that changes are the guarantees of the communication used for update propagation among data centers. While the internal communication among the nodes of a data center is based on a FIFO total order broadcast, the communication in the routers group is based on a causal broadcast. Upon the reception of an exported writeset W_1 from the routers group, a router R broadcasts W_1 inside its data center, using the local broadcast primitive. Any writeset W_2 later generated by any node in that data center will be causally ordered after W_1. This order is respected by the causal propagation of updates as the events of importing W_1 and exporting W_2 are from the same processor: the router.

To perform the causal propagation, the group of routers may use a causal primitive from a GCS or manage themselves all causal relationships by adding to each message a vector of integers of a size equal to the number of data centers of the system. The vector of a message M from data center DC_i states, for each other data center DC_j, the identifier (for instance, the delivery position in its local data center) of the last message from DC_j that was imported (delivered by the total order primitive) in DC_i before M was generated. Those messages are thus causally ordered before M. This precedence is trivially respected in the data center that generates M (if a message M_2 is created after the delivery of a previous message M_1 by the total order broadcast, then M_2 is ordered after M_1 by this broadcast primitive), but must be exported to other data centers. The vector is added to a message M by the local router, which signals as causally previous all imported messages that it itself forwarded and were later delivered by the local total order primitive *before* the delivery of M. Once the vector is added, the router broadcasts the message in the routers group. When a router receives a message M, it waits for all causally previous messages to be received and imported into its local data center before it inserts M in the local total order.

Similarly to the previous configuration, in case of network partition, the updates applied at a data center are buffered and will be propagated as usual once the network is repaired. During the partition, the guarantee of causal consistency is maintained. However, in the cases of a generalized failure of a data center and of a duplication of masters due to network partition, divergence is possible and must be resolved with reconciliation techniques as the ones described in Section 4. During periods of divergence, consistency is guaranteed to be FIFO. If divergence does not occur, then causal consistency is ensured even during failures and network partitions.

3.3 Sequential Consistency

To provide multi-partition transactions with sequential consistency, all nodes from all data centers must apply exactly the same sequence of updates, arriving from all partitions. To this end, the order between writesets must be agreed *before* their application at any data center. The previous approach, where already validated writesets were lazily propagated from the primary data center DC_i to remote data centers, is no longer valid as it allows that some nodes of DC_i have already finished the application of such writesets before an order is agreed. Instead, we propagate all writesets as they are generated at each node of each data center, using for this propagation sequential end-to-end semantics that ensure a total order among updates. The validation of a given writeset will take place, as before, only at its primary data center. A second broadcast, without ordering guarantees, will communicate the validation result to remote data centers, thus following a weak voting approach [33]. As a result, if each node validates or waits for the vote of each writeset and consequently applies it or not in the database following their delivery order, every node of every data center will follow the same sequence of states, thus ensuring sequential consistency.

The inconvenience of the weak voting technique is that remote data centers must wait for the vote of update transactions executed in other data centers. Another possibility is that each node is able to validate any writeset, but this requires the maintenance of a global history log at each data center.

Regarding failures, in the case of a generalized crash of a data center, and as long as the message delivery is uniform (if a message is delivered at any faulty or available node, it will be delivered at all available nodes), any other data center is in the position of becoming the new master for that partition, as all of them have received all the writesets generated by the failed data center. If the communication is not uniform, then the new master may start from an outdated copy of the partition and thus create divergence that must be later resolved. Divergences are also possible in the case of network partitions or false suspicious of failure, which may lead to master duplicity, as explained before. In the presence of divergence, consistency guarantees drop to FIFO. Reconciliation techniques as those discussed in Section 4 are necessary to agree on a common database state after divergence.

In case of network partition, when some data centers cannot communicate among them but can still be contacted by users, sequential consistency is no longer guaranteed as it is not possible to send the required end-to-end total order broadcasts to the whole system. As long as there is no master duplicity, multi-partition transactions are guaranteed to perceive, at least, processor consistency. Indeed, each connected subgroup can continue to use end-to-end total order broadcasts inside the subgroup and thus it provides sequential consistency over the set of partitions whose primary copy is stored in the subgroup. Messages that are delivered during the network partition are buffered respecting their total order. Once the network is repaired, a router from each subgroup is selected to send the buffered messages to previously disconnected data centers with FIFO guarantees. Upon the delivery of those messages, all data centers must apply their updates and reconcile their states before accepting new requests. Messages generated after the network reestablishment are broadcast with end-to-end total order guarantees to the entire system.

End-to-end total order broadcasts, however, require mechanisms that hinder high availability and scalability. For a message to have end-to-end total order guarantees, it must first be sent to the local router only, which will send it to the routers group communication protocol, which totally orders all messages and delivers them to the group of routers [23]. Only then can the routers broadcast the message in their local data centers. While this mechanism can provide good response times in low load scenarios, when the load increases it may be necessary to discard end-to-end guarantees in order to comply with the response time defined in the Service Level Agreement (SLA) between the service provider and the final users. Indeed, when a data center serves a high number of users, in order to cope with the rate of incoming requests, it may be forced to process them locally and postpone their propagation. This situation, which is similar to a network partition, is preferable to the violation of the SLA that could result from the delay in the response to users.

Another possibility to increase multi-partition consistency up to the sequential level without requiring end-to-end total order guarantees is to keep eventual consistency as the uniform consistency of the system and follow the synchronized entry model [8] to specifically ask for the update of those partitions that the user wants to access simultaneously in the same transaction. The entry consistency is a DSM memory consistency model that associates each shared variable with some synchronization variable. The same synchronization variable may be used to control multiple shared variables. When a process needs to access a shared variable, it must *acquire* the associated synchronization variable. When an acquire is done, only those shared variables guarded by the synchronization variable are made consistent, i.e., are updated. Each synchronization variable has a current owner: the process that last acquired it. The owner can freely access the shared variables controlled by that synchronization variable. When another process wants to access those variables, it sends a message to the current owner asking for ownership and the updated values of the associated variables. It is also possible for multiple processes to concurrently own a synchronization variable in non-exclusive mode: all those processes may read but not write the associated variables. Exporting this model to our scalable systems requires to associate each partition to a synchronization variable and make each data center the current owner of the synchronization variable corresponding to the partition whose primary copy is stored in that data center. If writes to a given partition are only allowed at its primary copy, then other data centers are only allowed to get non-exclusive ownership of the synchronization variable corresponding to that partition. Such acquire operations must ask for all the updates performed to the primary copy that have not yet been applied in the local copy.

Several scalable systems implement sequential consistency as, for example, Hyder [7] and VoltDB [32]. Unfortunately, both of them are designed to be implemented in cluster environments. The architecture of Hyder is based on multiple servers sharing a single data store, which is composed of networked raw flash disks, where a single log is shared between all servers. For that reason, every server in the system watches the same sequence of updates in the log, but if that architecture was implemented between data centers, SLA would be violated due to network delays.

Regarding VoltDB, scalability, relational schema with ACID SQL transactions and serial consistency are achieved in the whole system following the recommendations made by Stonebraker et al. [29]: horizontal partitioning, main-memory storage, no resource control, no multi-threading and high availability (replication). To achieve this, most updated tables in the database are partitioned so that partitions and read tables are replicated *k+1* times (*k-safety* property) based on the number of processors in the system. Furthermore, since most modern applications are not interactive, their transactions can be implemented as stored procedures and executed in the same order in all partitions, obtaining serial consistency. In addition to the *k-safety* property and in order to achieve high availability, full replicas of the database are maintained in other clusters as slaves in case a primary cluster disaster occurs and, moreover, periodic snapshots of the primary cluster can be saved to disk. Regrettably, due to the main-memory storage restriction, VoltDB databases are only allowed to be implemented in cluster environments.

3.4 Session Consistency

An interesting feature of replication systems is to provide session consistency [30, 14, 15]. Sessions allow database users to logically group sets of their transactions. Transactions from different users belong to different sessions. However, it can be left to the user the decision of using one or multiple sessions to group their transactions. Transactions of the same user session are submitted sequentially, but may be addressed to different nodes or, even, different data centers. This is particularly the case when a user updates, through the same session, different partitions, with different data centers as the primary copy of each partition. In this case, session consistency allows each individual user to perceive an image of atomic consistency from all the transactions that belong to the same session, regardless of the partitions or the data centers accessed.

Session consistency can be built upon sequential consistency [14, 15]. This way, we can extend the previous configuration in order to ensure that the updates of all previous transactions of a given session have been applied in a given node before this node starts to serve the current transaction of such a session. In order to do this, each transaction must be provided with a unique identifier (a trivial identifier is the delivery position in the total order that propagates updates) and each new transaction, except for the first transaction of each session, must provide the identifier of the previous transaction of the same session. The condition to serve this new transaction is to have already applied all the writesets up to the signaled one. Obviously, the first transaction of a session does not need to wait, as well as the transactions that are served by the same data center that served the previous transaction of the session. The price to pay for the increased consistency is the possible latency that transactions may experiment when a single user session is used to update different partitions.

As this configuration is almost identical to the previous one, the problems related to network partitions and failures have similar consequences and can be treated in the same way. If a network partition isolates data centers between them, then users are not ensured to get session consistency, as the sequential consistency taken as basis is no longer guaranteed. However, inside connected subgroups, sequential and thus session consistency is provided for sessions of multi-partition transactions that only access partitions whose primary copies are inside the subgroup. In the general case, during failures or network partitions only FIFO consistency is ensured. Processor consistency can be guaranteed if no divergence arises.

Another possibility is to achieve session consistency based on eventual consistency. For instance, this type of consistency can be managed by the *Consistency Rationing* approach presented by Kraska et al. [24], where authors analyze how to implement database-like facilities on top of cloud storage systems like Amazon S3 [3]. In this approach, data is classified into three different categories (A, B and C). On one hand, category A implements serializability [6], i.e., sequential consistency [25] according to Mosberger [27], and is assigned to critical data for which a consistency violation results in large penalty costs. On the other hand, category C comprises data that tolerates inconsistencies and ensures session consistency [30]. Finally, category B is named *adaptive consistency* because it encompasses data whose consistency requirements vary depending on multiple factors, e.g., time constraints, the availability of that data, their probability of conflicts, etc. For example, adaptive consistency can be used in auction systems, where session consistency may be admitted until the last minutes of the auction, when serializability is required to avoid inconsistencies about who wins the auction.

In addition to manage session consistency on top of Amazon S3, these authors also mention the possibility of achieving session consistency from the *per-record timeline consistency*, which is a stronger type of eventual consistency provided by the Yahoo PNUTS [13] system, using the advanced operations offered by its API.

3.5 Multi-master Configuration

When the number of partitions and data centers is high, it may be expected that each master will have little load and it will respond quickly to users. However, the primary copy configuration may have as a consequence an increase in response times when higher and higher number of users write into the same partition, as all this load must be managed by the master of such a partition. If that data center cannot be longer scaled up or out, then a multi-master configuration may help to improve performance. Similarly, if the database cannot be partitioned into a high enough number of partitions of a reasonable size, multiple masters per partition may be required to comply with SLA guarantees.

A multi-master configuration follows similar principles to those of a primary-backup approach, with the difference that more than one master is assigned to each partition. Multiple masters decrease response times, but they may impair consistency. If the same data item can be updated from two different data centers and updates are propagated lazily to other data centers, a reconciliation process is needed each time an item is concurrently updated with different values, which may result into abortions of previously committed user requests. Moreover, despite reconciliations, the maximum consistency guarantee for both mono- and multi-partition transactions drops to FIFO. Indeed, a multi-master configuration resembles very much the situation of master duplicity, with the attenuating circumstance of having the network available.

In order to get stronger guarantees than FIFO and eventual consistency, the multi-master configuration could be used with end-to-end total order multicasts. Such communication primitives would be used to propagate writesets among the masters of the same partition, in order to validate them with regard to all concurrent transactions executed over that partition at all involved data centers, before applying them at each replica. The multi-master configuration with this partition-wide validation (where the validation of a writeset is performed by all the data centers that are masters of the affected partition, instead of being performed only by the data center that executed the transaction) could be used in conjunction with either (a) a lazy propagation of updates to non-master data centers, thus achieving sequential consistency for mono-partition transactions and processor consistency for multi-partition transactions, or (b) an end-to-end total order broadcast of all writesets to all data centers, achieving sequential consistency for all transactions.

Regarding failures, the generalized crash of a data center is now tolerated by the rest of masters of the same partition. Only the generalized failure of all the data centers that are masters of a given partition would require the election of a new set of masters for that partition and possibly originate divergence if the new masters store outdated versions of that partition. The previously discussed problem of master duplicity is also possible here if the set of masters of a given partition becomes isolated due to a network partition and another set is elected to accept updates to the same data. Moreover, divergence may also occur if different masters of the same set are isolated due to a network partition. All possible divergences must be resolved with adequate reconciliation techniques.

During a network partition that isolates data centers, the messages delivered for update propagation are buffered and retransmitted by routers with FIFO guarantees once the network is repaired. During the partition, and until reconciliation is complete, the consistency guarantees for both mono- and multi-partition transactions are FIFO in the general case, or processor if there is no divergence. Moreover, sequential consistency can be ensured for mono-partition transactions that access partitions whose set of masters is connected. Finally, if end-to-end total order messages are used for both the partition-wide validation and the propagation of updates, then sequential consistency can be ensured for multi-partition transactions that only access partitions whose complete sets of masters are in the same subgroup.

4 Reconciliation Techniques

During a failure or a network partition situation, the state of different replicas of the same partition may diverge. After the failure or the network are repaired, data centers must agree on a common state of each partition, in a process of reconciliation. Reconciliation merges different updates made to the same data item through different data centers. Reconciliation techniques depend on the nature of the operations concurrently performed to different replicas of the same partition. If operations are commutative, reconciliation is trivially achieved by applying at each replica the operations performed in the other one [20, 21]. For example, if items have been added to a shopping cart from two different data centers DC_1 and DC_2, the reconciliation just adds all the items from DC_1 to the cart of DC_2 and vice versa. The aim of reconciliation is to contemplate all performed operations and agree on a final state of the data. When operations are not commutative, including all the performed operations may not be possible, being necessary to undo or discard some operations. The selection of which operations will be successful and which will be discarded can be random or based on a given criterion. In any case, the selection must be the same for all data centers, so deterministic criteria must be followed or random selections must be performed by a single node that later broadcasts the result to the rest of the system.

It is possible to determine different reconciliation criteria based on metadata information. One option is to automatically assign timestamps to updates and reconcile copies with the most recent write, as done by Dynamo [16]. A clock synchronization protocol can be used while the network is up. As network partitions are usually brief, clocks will not noticeably diverge and will allow nodes to mark their writesets with *correct* timestamps. This approach has as an advantage that it can be automatized and performed without human intervention: starting from the lists of writesets from each data center, a unique list can be built following a chronological order. The resulting list incorporates all updates and resolves conflicting writes by prevailing more recent updates over older ones. Timestamps constitute a deterministic criterion for the duplicated masters and for the rest of data centers. However, a highly scalable system may receive thousands of requests per second and the rule of the-most-recent-update-wins may not be always useful.

Another option for metadata-based reconciliation is to prevail some updates over others regarding to their origin. For this, different origins are given different priorities. The origin of an operation may be the system node that performed the operation, the user that made such a request, the geographical zone from which the request came, etc. By including the origin into the metadata and comparing priorities in case of divergence, a deterministic choice can be made unless conflicting operations are of the same origin. In this case, additional criteria must be followed.

Finally, reconciliation can be performed considering only the operation itself. This requires a good comprehension of the application semantics, and thus it is hardly achievable without human collaboration, either from system administrators or from the final users.

In any case, for the users not to lose their confidence in the system, divergences must be detected and reconciled as soon as possible, and the affected users, those that performed the aborted or discarded operations, must be promptly and accurately informed. One way of minimizing the inconveniences suffered by users due to divergences caused by a crashed master and an outdated new master is to wait for the correct application of a writeset at a given number of data centers before reporting the operation as successful to the user. The higher the number of data centers updated before responding to users, the lower the probability of having to abort an already confirmed operation. While this technique increases response times, it tolerates more generalized failures with regard to the problem of the outdated new master: more data centers have to crash to force the system to undo a confirmed operation.

5 Conclusions

Eventual consistency is the usual consistency guarantee provided by scalable and highly available systems. We show that it is possible to provide stronger levels of consistency in such systems, by partitioning the database and following the primary-backup approach. Processor, causal and sequential levels of consistency, as well as session consistency, are achievable during normal functioning. Failures and network partitions pose three main related problems: (a) the possible divergence when a new master is selected to replace a crashed data center but its copy of the affected partition is not up-to-date, (b) the propagation of updates in case of network partitions, and (c) the divergence entailed by master duplicity, when two data centers, isolated from each other, act as masters of the same partition. During failure or partition situations, consistency guarantees can be affected, but FIFO consistency is always ensured. We analyze all cases of failures and network partitions and propose solutions for recovery and reconciliation.

Acknowledgements

This work has been supported by EU FEDER and Spanish MICINN under research grants TIN2009-14460-C03-01 and TIN2010-17193.

References

[1] M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures*, SPAA '93, pages 251–260, New York, NY, USA, 1993. ACM.

[2] A. Alvarez, S. Arévalo, V. Cholvi, A. Fernández, and E. Jiménez. On the Interconnection of Message Passing Systems. *Inf. Process. Lett.*, 105(6):249–254, 2008.

[3] Amazon Web Services LLC. Amazon Simple Storage Service (S3). Website, March 2011. Available from: `http://aws.amazon.com/s3/`.

[4] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, J. Léon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing Scalable, Highly Available Storage for interactive services. In *5th Biennial Conf. on Innovative Data Systems Research (CIDR)*, pages 223–234, Asilomar, CA, USA, January 2011.

[5] R. Baldoni, R. Beraldi, R. Friedman, and R. van Renesse. The Hierarchical Daisy Architecture for Causal Delivery. *Distributed Systems Engineering*, 6(2):71–81, 1999.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[7] P. A. Bernstein, C. W. Reid, and S. Das. Hyder - A Transactional Record Manager for Shared Flash. In *5th Biennial Conf. on Innovative Data Systems Research (CIDR)*, pages 9–20, Asilomar, CA, USA, January 2011.

[8] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The Midway Distributed Shared Memory System. In *Proc. IEEE CompCon Conf.*, 1993.

[9] E. A. Brewer. Towards Robust Distributed Systems (Abstract). In *Proc. ACM Symp. Princ. Distrib. Comput.*, page 7, 2000.

[10] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The Primary-Backup Approach. In S. J. Mullender, editor, *Distributed Systems*, chapter 8, pages 199–216. Addison-Wesley, ACM Press, 2nd edition, 1993.

[11] D. G. Campbell, G. Kakivaya, and N. Ellis. Extreme Scale with Full SQL Language Support in Microsoft SQL Azure. In *Intnl. Conf. on Mngmnt. of Data (SIGMOD)*, pages 1021–1024, New York, NY, USA, 2010. ACM.

[12] V. Cholvi, E. Jiménez, and A. Fernández Anta. Interconnection of distributed memory models. *J. Parallel Distrib. Comput.*, 69(3):295–306, 2009.

[13] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.

[14] K. Daudjee and K. Salem. Lazy Database Replication with Ordering Guarantees. In *Proc. Int. Conf. Data Eng.*, pages 424–435. IEEE-CS, 2004.

[15] K. Daudjee and K. Salem. Lazy Database Replication with Snapshot Isolation. In *Proc. Int. Conf. Very Large Data Bases*, pages 715–726. ACM, 2006.

[16] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *ACM Symp. Oper. Syst. Princ.*, pages 205–220, 2007.

[17] A. Fernández, E. Jiménez, and V. Cholvi. On the interconnection of causal memory systems. *J. Parallel Distrib. Comput.*, 64(4):498–506, 2004.

[18] S. Gilbert and N. A. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News*, 33(2):51–59, 2002.

[19] J. R. Goodman. Cache Consistency and Sequential Consistency. Technical Report 61, SCI Committee, March 1989.

[20] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 173–182. ACM, 1996.

[21] P. Helland and D. Campbell. Building on Quicksand. In *Proc. Bienn. Conf. Innov. Data Syst. Research*. www.crdrdb.org, 2009.

[22] P.W. Hutto and M. Ahamad. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 302–311, May 1990.

[23] S. Johnson, F. Jahanian, and J. Shah. The Inter-group Router Approach to Scalable Group Composition. In *ICDCS*, pages 4–14, 1999.

[24] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann. Consistency Rationing in the Cloud: Pay only when it matters. *PVLDB*, 2(1):253–264, 2009.

[25] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[26] R. J. Lipton and J. S. Sandberg. Pram: A Scalable Shared Memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, sep 1988.

[27] D. Mosberger. Memory Consistency Models. *Operating Systems Review*, 27(1):18–26, 1993.

[28] M. I. Ruiz-Fuertes and F. D. Muñoz-Escoí. Refinement of the One-Copy Serializable Correctness Criterion. Technical Report ITI-SIDI-2011/004, Instituto Tecnológico de Informática, Valencia, Spain, November 2011.

[29] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). In *33rd Intnl. Conf. on Very Large Data Bases (VLDB)*, pages 1150–1160, Vienna, Austria, September 2007. ACM Press.

[30] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. B. Welch. Session Guarantees for Weakly Consistent Replicated Data. In *Proc. Int. Conf. Parallel Distrib. Inform. Syst.*, pages 140–149. IEEE-CS, 1994.

[31] W. Vogels. Eventually Consistent. *Communications of the ACM (CACM)*, 52(1):40–44, January 2009.

[32] VoltDB, Inc. VoltDB technical overview: A high performance, scalable RDBMS for Big Data, high velocity OLTP and realtime analytics. Website, April 2012. Downloadable from: http://voltdb.com/sites/default/files/PDFs/VoltDBTechnicalOverview_April_2012.pdf.

[33] M. Wiesmann and A. Schiper. Comparison of Database Replication Techniques Based on Total Order Broadcast. *IEEE T. Knowl. Data En.*, 17(4):551–566, April 2005.