

**TRANSACTIONAL CONSISTENCY IN A SECONDARY CLUSTER WHEN TRANSACTIONS ON
DISTRIBUTED DATA ARE PROPAGATED ASYNCHRONOUSLY FROM A PRIMARY CLUSTER**

Priority Claim

5 [0001] The present patent application is related to and claims the benefit of priority to the co-
pending US provisional patent application entitled, "IMPLEMENTING TRANSACTIONS IN
ASYNCHRONOUS REPLICATION OF DATA ACROSS CLUSTERS", Serial No.: 63/591,131,
Filed: 18th October, 2023, which is incorporated in its entirety herewith to the extent not
inconsistent with the description herein.

10

Background of the Disclosure

[0002] Technical Field

[0003] The present disclosure relates to database systems and more specifically to providing
transactional consistency in a secondary cluster when transactions on distributed data are
15 propagated asynchronously from a primary cluster.

[0004] Related Art

[0005] Cluster refers to a group of inter-connected computers (referred to as "nodes") that work
together to provide a single functionality. Examples of such single functionality include databases,
20 web server, application server, etc. However, each node in a cluster may host appropriate software
designed to service different types (e.g., read, write) of requests received from external client
systems.

[0006] A cluster may be designed to store data in a distributed manner, implying one or more parts
(hereafter referred to as shards) of the data is stored on one node of the cluster, while other shards
25 of the data are stored in another node of the cluster. The aggregate data stored in one cluster is
often replicated into multiple clusters for purposes such as maintaining a backup of the data, for
disaster recovery, provide additional access to the data, etc. as is well known in the arts.

[0007] Clusters storing such replicated data are viewed as having a primary cluster and one or
more secondary clusters. A primary cluster is designed to receive and process updates to the stored
30 data, while the secondary clusters may be limited to providing read access.

[0008] Data updates to the primary cluster are often received in the form of transactions. As is
well known, each transaction may contain multiple updates, which are to be effected together
(referred to as a commit) or none at all. Such a requirement is commonly referred to as 'atomicity'
property for transactions. It is therefore required that such committed updates be propagated to
35 the secondary clusters as well.

[0009] Asynchronous propagation may be employed for propagating the transactions to a

secondary cluster. Asynchronous propagation is contrasted with synchronous propagation, in which each transaction is committed to both the primary and secondary clusters together and therefore made available from same time instance to external client systems. Thus, in asynchronous propagation, the transactions are propagated to secondary cluster without being time
5 bound by updates in the primary cluster. The inefficiencies associated with such synchronous approaches may not be acceptable in several environments.

[0010] However, any asynchronous propagation approaches need to provide for transactional consistency in secondary clusters, consistent with the general requirements of implementation of transactions. Aspects of the present disclosure relate to providing transaction consistency in a
10 secondary cluster when transactions on distributed data are propagated asynchronously from a primary cluster.

Brief Description of the Drawings

[0011] Example embodiments of the present disclosure will be described with reference to the
15 accompanying drawings briefly described below.

[0012] Figure 1 illustrates an example environment (computing system) in which several aspects of the present disclosure can be implemented.

[0013] Figure 2A is a block diagram illustrating the manner in which clusters are formed and data replication is implemented between the clusters in one embodiment.

20 [0014] Figure 2B is a block diagram illustrating the challenges with propagating transactions across clusters in one embodiment.

[0015] Figure 3 is a flow chart illustrating the manner in which transactional consistency in a secondary cluster is provided when transactions on distributed data are propagated asynchronously from a primary cluster according to aspects of the present disclosure.

25 [0016] Figure 4A depicts the internal details of a distributed database hosted in a cluster in one embodiment.

[0017] Figure 4B depicts the manner in which distributed transactions are implemented in a distributed database in one embodiment.

30 [0018] Figure 5A depicts the manner in which transactions are propagated from a primary cluster to a secondary cluster in one embodiment.

[0019] Figure 5B depicts the manner in which transactions are propagated across clusters for a single tablet of a distributed database in one embodiment.

[0020] Figures 6A-6C together illustrate the manner in which transactional consistency in a secondary cluster is provided when transactions on distributed data are propagated asynchronously
35 from a primary cluster in one embodiment.

[0021] Figure 7 is a high-level sequence diagram illustrating the manner in which apply safe times and a cluster safe time are determined in one embodiment.

[0022] Figure 8 is a block diagram illustrating the details of digital processing system (700) in which various aspects of the present disclosure are operative by execution of appropriate
5 executable modules.

[0023] In the drawings, similar reference numbers generally indicate identical, functionally similar, and/or structurally similar elements. The drawing in which an element first appears is indicated by the leftmost digit(s) in the corresponding reference number.

10 Detailed Description of the Embodiments of the Disclosure

[0024] 1. Overview

[0025] An aspect of the present disclosure provides transaction consistency in a secondary cluster when transactions on distributed data are propagated asynchronously from a primary cluster. In one embodiment, each of the primary cluster and secondary cluster maintains multiple shards (of
15 data). The primary cluster receives a set of transactions, each transaction containing corresponding updates to at least one shard in the primary cluster, followed by a commit command to indicate end of updates in the transaction. The primary cluster processes each transaction of the set of transactions by storing each update as a corresponding provisional record in a respective shard of the primary cluster, processing the commit command for the transaction at a corresponding commit
20 time and applying the commit command, after the processing, to each of the respective shards, by converting each provisional record of the transaction to a corresponding regular record in the corresponding shard, each regular record being stored associated with the corresponding commit time for the transaction. The primary cluster asynchronously propagates the updates for the shards to the secondary cluster, along with an apply safe time for each shard in the primary cluster. The
25 apply safe time propagated for a shard in the primary cluster indicates a time instance such that all transactions related to the shard and having commit time less than the time instance have been applied in the shard.

[0026] The secondary cluster stores each of the asynchronously received updates in a corresponding shard of the secondary cluster, and determines a cluster safe time as the minimal
30 value of all the apply safe times corresponding to all of the shards in the primary cluster. Upon receiving a read request directed to the secondary cluster for data stored in the secondary cluster, the secondary cluster identifies a set of regular records having corresponding commit times less than or equal to the cluster safe time and provides the set of regular records as a response to the read request.

[0027] According to another aspect of the present disclosure, when no transactions related to the shard are committed by a first time instance, the apply safe time is set to any value up to and including the first time instance.

5 [0028] According to one more aspect of the present disclosure, when a first subset of transactions related to the shard are committed but not applied by a second time instance, the apply safe time is set to a value which is less than or equal to a minimum of commit times of the first subset of transactions.

[0029] According to yet another aspect of the present disclosure, when all transactions committed by a third time instance related to the shard are applied by the third time instance, the apply safe
10 time is set to any value up to and including the third time instance.

[0030] According to an aspect of the present disclosure, the secondary cluster sends multiple get-changes requests corresponding to respective shards of the primary cluster, with each shard in the primary cluster transmitting the corresponding updates in the same order in response to the respective get-changes request asynchronously. The corresponding updates includes provisional
15 records, and regular records with corresponding commit times.

[0031] According to another aspect of the present disclosure, in response to a get-changes request directed to a shard in the primary cluster, a shard leader in the primary cluster identifies a leader safe time and determines one or more transactions to be resolved based the provisional records stored in the shard. The shard leader sends get-transaction-status to one or more transaction
20 coordinators, and receives responses to the get-transaction-status requests indicating statuses of whether the one or more transactions are still in progress, committed, or aborted. The shard leader then generates apply actions to convert provisional records of committed transactions to regular records, and then computes a current end of WAL location. The shard leader then sets the apply safe time for the shard as being equal to the leader safe time if all the records up to the current end
25 of WAL location have been sent to the secondary cluster.

[0032] According to one more aspect of the present disclosure, each of the primary cluster and the secondary cluster implements a distributed database, with each shard corresponding to a tablet in the distributed database, where each tablet stores data corresponding to a unique set of keys.

[0033] Several aspects of the present disclosure are described below with reference to examples
30 for illustration. However, one skilled in the relevant art will recognize that the disclosure can be practiced without one or more of the specific details or with other methods, components, materials and so forth. In other instances, well-known structures, materials, or operations are not shown in detail to avoid obscuring the features of the disclosure. Furthermore, the features/aspects described can be practiced in various combinations, though only some of the combinations are described
35 herein for conciseness.

[0034] 2. Example Environment

[0035] Figure 1 is a block diagram illustrating an example environment (computing system) in which several aspects of the present disclosure can be implemented. The block diagram is shown containing computing infrastructures 110 and 120, network 130, and end user systems 140A-140D.

5 The end user systems are collectively referred to as 140.

[0036] Merely for illustration, only representative number/type of blocks is shown in the Figure 1. Many environments often contain many more blocks, both in number and type, depending on the purpose for which the environment is designed. Each block of Figure 1 is described below in further detail.

10 **[0037]** Each of computing infrastructures 110 and 120 is a collection of processing nodes, connectivity infrastructure, data storages, etc., which are engineered to together provide a virtual computing infrastructure for various customers, with the scale of such computing infrastructure being specified often on demand. The nodes (such as node 150) may be virtual nodes (e.g., virtual machines (VMs), containers containing one or more VMs) operating based on physical nodes,
15 physical nodes themselves, or a combination as well.

[0038] It may be appreciated that the computing infrastructures typically span several continents and are provided by different vendors. In addition, each computing infrastructure may vary substantially from another in terms of interface requirements, scale, technical characters of nodes, hardware/software/network implementation, etc., and thus the computing infrastructures are said
20 to be diverse. Examples of such diverse computing infrastructures include, but are not limited to, public clouds such as Amazon Web Services (AWS) Cloud available from Amazon.com, Inc., Google Cloud Platform (GCP) available from Google LLC, etc., and private clouds such as On-Premises clouds owned by the customers.

[0039] Computing infrastructure (I1) 110 is shown containing nodes (processing or storage, shown
25 as squares such as node 150) located in two different geographical regions R1 and R2. Each region is shown containing multiple availability zones (named as Z1, Z2, etc.), each having independent support infrastructure such as power, networking, etc. Each availability zone (e.g., I1-R1-Z1) can thus operate independent of other zones, such that the availability zone can continue to operate even upon the failure of the other zones (e.g., I1-R1-Z2 and I1-R1-Z3). Computing infrastructure
30 (I2) 120 is similarly shown with regions R1, R3, R4 with respective regional sets of availability zones, with each availability zone containing respective nodes.

[0040] All the nodes of each computing infrastructure 110/120 are assumed to be connected via a corresponding intranet (not shown). Network 130 extends the connectivity of these (and other systems of the computing infrastructures) with external systems such as end user systems 140.
35 Network 130 may be an internetwork (including the world-wide connected Internet), an intranet,

or a combination of internetwork and intranet. Each of the intranets and network 130 may be implemented using protocols such as Transmission Control Protocol (TCP) and/or Internet Protocol (IP), well known in the relevant arts.

[0041] In general, in TCP/IP environments, a TCP/IP packet is used as a basic unit of transport, with the source address being set to the TCP/IP address assigned to the source system from which the packet originates and the destination address set to the TCP/IP address of the target system to which the packet is to be eventually delivered. An IP packet is said to be directed to a target system when the destination IP address of the packet is set to the IP address of the target system, such that the packet is eventually delivered to the target system by network 130. When the packet contains content such as port numbers, which specifies a target application, the packet may be said to be directed to such application as well.

[0042] Each of end user systems 140A-140D represents a system such as a personal computer, workstation, mobile device, computing tablet etc., used by users/customers to generate user requests directed to applications executing in nodes of computing infrastructures 110/120. The user requests may be generated using appropriate user interfaces (e.g., web pages provided by an application executing in the server, a native user interface provided by a portion of an application downloaded from the server, etc.). In general, end user system requests an application for performing desired tasks and receives the corresponding responses (e.g., web pages) containing the results of performance of the requested tasks. The web pages/responses may then be presented to the user at end user systems 140 by client applications such as the browser.

[0043] Some of nodes in computing infrastructure 110/120 may be implemented as corresponding data stores. Each data store represents a non-volatile (persistent) storage facilitating storage and retrieval of data by application services executing in the other systems/nodes of computing infrastructures 110/120. Each data store may be implemented as a corresponding database server using relational database technologies and accordingly provide storage and retrieval of data using structured queries such as SQL (Structured Query Language). Alternatively, each data store may be implemented as a corresponding file server providing storage and retrieval of data in the form of files organized as one or more directories, as is well known in the relevant arts.

[0044] Some of the nodes in computing infrastructure 110/120 may be implemented as corresponding server systems. Each server system represents a server, such as a web/application server, constituted of appropriate hardware, executing application/data services or software components thereof capable of performing one or more tasks. The tasks may be specified as part of user requests received from end user systems 140 or node requests received from nodes of same/other computing infrastructures. A server system, in general, receives a task request and performs the tasks requested in the task request.

[0045] A server system may use data stored internally (for example, in a non-volatile storage/hard disk within the server system), external data (e.g., maintained in a data store) and/or data received from external sources (e.g., received from a user) in performing the requested tasks. The server system then sends the result of performance of the tasks to the requesting end user system (one of 5 140) or node as a corresponding response to the task request. The results may be accompanied by specific user interfaces (e.g., web pages) for displaying the results to a requesting user.

[0046] It may be appreciated that the various nodes of computing infrastructures 110/120 may be grouped together into clusters that provide a common functionality (e.g., databases). As noted above, multiple clusters may be formed using the nodes, with data being replicated between the 10 clusters. The manner in which clusters may be formed and data replication among clusters may be implemented is described below with examples.

[0047] **3. Clusters and Data Replication**

[0048] Figure 2A is a block diagram illustrating the manner in which clusters are formed and data replication is implemented between the clusters in one embodiment. The block diagram is shown 15 containing primary cluster 210 (in turn shown containing nodes 112A-112C) and secondary cluster 220 (in turn shown containing nodes 115A-115C) and async replication 240. Each of the blocks of the Figure is described in detail below.

[0049] Primary cluster 210 represents a group of inter-connected nodes (here three nodes 112A-112C) of computing infrastructure 110. Each node is shown present in a corresponding availability 20 zone (Z1, Z2, Z3) of a single region (R1) in the infrastructure I1. Each of the nodes may communicate with other nodes of the cluster as indicated by the arrows between the nodes. Each node of primary cluster 210 is also designed to serve (receive, process and send corresponding respond to) read and write requests received from end user systems 140.

[0050] Secondary cluster 220 represents another group of nodes (115A-115C) present in 25 corresponding availability zones (Z1, Z2, Z3) of another region (R2) in the infrastructure I1. Though the clusters are shown being formed within a single infrastructure (I1), it may be appreciated that the clusters may contain a fewer/larger number of nodes, may have nodes from different regions (e.g., R1 and R2) in the same infrastructure (I1), and may be formed from different infrastructures (e.g., 210 in I1 and 220 in I2), as will be apparent to one skilled in the 30 relevant arts.

[0051] In one embodiment, primary cluster 210 is designed to act as the primary or active cluster, with nodes 112A-112C designed to serve both read and write requests, while secondary cluster 220 is designed to act as the secondary or standby cluster designed to serve only read requests. In addition, the data stored in each of the nodes is split into multiple parts/shards. Shards are typically 35 designed to store non-overlapping portions of data. For example, when the data is stored associated

with corresponding keys, each shard may store an exclusive subset of the keys (and associated data). In the Figure, each node is shown storing data split into 3 shards each and as such each cluster is shown containing 9 shards.

[0052] Async replication 240 causes data/updates to be replicated from nodes/shards of primary cluster 210 to corresponding nodes/shards of secondary cluster 220. Replication entails first the propagation of updates from the primary cluster to the secondary cluster, and storing of the propagated updates in the secondary cluster. The updates in each shard may be propagated independent of those of the other shards, adding to the asynchronous character of the propagation.

[0053] In addition, each shard in the secondary cluster may store the corresponding propagated updates independent of similar storing in other shards, thereby further adding to the asynchronous character of the replication. However, the storing across shards is required to be constrained by the requirements of transactions (e.g., atomicity).

[0054] In one embodiment, for each shard, secondary cluster 220 sends a get-changes request to primary cluster 210 at regular intervals. Upon receipt of a get-changes request, primary cluster 210 provides the new updates in the shard in comparison to the time instance at which the previous get change request was received. Secondary cluster 220 then stores the updates provided by primary cluster 210 in the corresponding shards. In an alternative embodiment, the updates may be proactively pushed from primary cluster 210 to 220 at regular intervals (but not in response to updates being made at the primary cluster).

[0055] T1, T2 and T3 represent transactions that have been committed on primary cluster 210 in that order. However, due to async replication 240 being performed per shard, only transaction T1 and T3 have been replicated to secondary cluster 220. As such, a read request received at secondary cluster 220 may ideally send responses based only up to transaction T1, even though transaction T3 is present already in secondary cluster 220 since transaction T2 had not arrived yet at secondary cluster 220. Some additional challenges with respect to propagating transactions across clusters is described in detail below.

[0056] Figure 2B is a block diagram illustrating the challenges with propagating transactions across clusters in one embodiment. Specifically, 250 illustrates the challenge when a transaction spans multiple shards (referred to as “distributed transaction”), an example of which is shown below:

```
begin transaction
  update A (K1 = V1)
  update B (K2 = V2)
commit transaction
wherein
  K1 and K2 are keys located in different shards
```

V1 and V2 are values

[0057] A and B are shown updated in respective shards 261 and 262 of primary cluster 210. The corresponding replicated updates A' and B' are shown in shards 271 and 272 of secondary cluster 220. It may be appreciated that any read requests received at the primary cluster sees either (1) changes to A and B or (2) changes to neither A and B due to atomicity requirement, since the updates are part of a transaction. However, read requests on the secondary cluster can see (1) changes to A and B; (2) changes to neither A and B; (3) changes to A but not to B; or (4) changes to B but not to A, due to async replication 240 replicating the updates A and B at different time instances. Such a scenario would cause the “atomicity” property of a transaction to be violated.

10 [0058] 280 illustrates another challenge with asynchronous replication of transactions across clusters. Shards S1-S5 of primary cluster 210 are shown containing multiple committed transactions (1 to 7). The corresponding shards S1'-S5' of secondary cluster 220 are shown containing only a subset of the committed transactions due to the asynchronous nature of replication across multiple shards. Any reads on secondary cluster 220 needs to provide a transactionally consistent view of the data. For example, if a read request is directed to shard S1', the data corresponding to transaction 6 cannot be provided as the corresponding data in other shards have not yet been replicated to secondary cluster 220.

[0059] In general, providing transactional consistency in the secondary cluster when transactions are asynchronously replicated across clusters, entails providing:

- 20 • Transactional atomicity guarantee - A transaction spanning shards A and B has to be either fully be readable or not at all on the secondary cluster. In the case of databases, shards A and B can be portions within the same table OR within a table and an index OR within different tables.
- Global ordering guarantee - Transactions have to be visible on the secondary cluster/side in the same order (“commit order”) as the transactions were committed on the primary cluster/side (even if with some time delay).
- 25 • Support for Planned/Unplanned Failover – The transactions have to be made available on the secondary cluster/side in a way to support roles-switch during planned and unplanned failover scenarios.

30 [0060] Aspects of the present disclosure are directed to providing transactional consistency in a secondary cluster (220) when transactions on distributed data are propagated asynchronously from a primary cluster (210), as described below with examples.

[0061] 4. Transactional Consistency in a Secondary Cluster

[0062] Figure 3 is a flow chart illustrating the manner in which transactional consistency in a secondary cluster is provided when transactions on distributed data are propagated asynchronously

35

from a primary cluster according to aspects of the present disclosure. The flowchart is described with respect to the systems of Figures 1, 2A and 2B, merely for illustration. However, many of the features can be implemented in other environments also without departing from the scope and spirit of several aspects of the present invention, as will be apparent to one skilled in the relevant arts by reading the disclosure provided herein.

[0063] In addition, some of the steps may be performed in a different sequence than that depicted below, as suited to the specific environment, as will be apparent to one skilled in the relevant arts. Many of such implementations are contemplated to be covered by several aspects of the present invention. The flow chart begins in step 301, in which control immediately passes to step 310.

[0064] In step 310, primary cluster 210 receives a set of transactions containing updates to shards in primary cluster 210. Each transaction contains one or more updates (to the same or different shards in the primary cluster) followed by a commit command to indicate end of updates in the transaction. The transactions may be received from end user systems 140.

[0065] In step 320, primary cluster 210 stores the updates as provisional records within the respective shards of primary cluster 210. Provisional records are place holders used internally by the cluster. The data/values in the provisional records may not be made available to requests received from end user systems.

[0066] In step 330, primary cluster 210 processes the commit command for each transaction at a corresponding commit time. The commit command is processed only after all updates of the transaction are stored as corresponding provisional records in the nodes of primary cluster 210. Committing a transaction typically implies that all the previously stored updates are made permanent (in other words, made available in the future) in the nodes of primary cluster 210. In one embodiment, processing of the commit command entails sending a commit acknowledgement to the end user system 140 from which the transaction was received, and thereafter making the updates permanent as described below.

[0067] In step 335, primary cluster 210 applies the commit command (for each transaction) by converting the corresponding provisional records (stored in the respective shards) to regular records. The regular records are stored associated with the corresponding commit time for the transaction. It may be noted that in the primary cluster, data is visible to end users as soon as the transaction is committed. Each shard resolves the transaction for the provisional records and return provisional records to an end user if the transaction is committed. The shard does not have to convert them to regular records first. The regular records are stored with the commit time of their transaction, so that they can be made visible without having the need to resolve the transactions.

[0068] Thus, steps 320, 330 and 335 process each transaction in the set of transactions received at step 320. Control passes to step 310 for processing subsequently received sets of transactions. It

may be appreciated that step 335 may be performed asynchronously, such that control passes to step 310 even before step 335 can start. Also, while step 320 is being performed, other transactions may be performing step 310, so each transaction can be at a different state out of steps 310, 320, 330 and 335 at the same time, and operate independently.

5 [0069] In step 340, primary cluster 210 asynchronously propagates the updates for the shards to secondary cluster 220, along with an apply safe time for each shard in primary cluster 210. The propagated updates may include provisional records and regular records with associated commit times. The apply safe time for a shard in primary cluster 210 indicates a time instance such that all transactions related to the shard and having commit time less than the time instance have been
10 applied in the shard.

[0070] It may be appreciated that steps 330, 335 and 340 are typically performed concurrently. In other words, while some of the transactions are being committed and applied in primary cluster 210, simultaneously, primary cluster 210 may transmit the details noted above to secondary cluster 220.

15 [0071] In step 350, secondary cluster 220 stores the asynchronously received updates (provisional and regular records) in corresponding shards of secondary cluster 220. In step 360, secondary cluster 220 determines a cluster safe time as the minimal value of all the apply safe times corresponding to all of the shards in primary cluster 210. The cluster safe time represents the time instance before which the data in secondary cluster 220 is transactionally consistent. In other
20 words, the data in secondary cluster 220 prior to/at the cluster safe time corresponds to the actual data (and also the commit order) of the transactions that have been committed on primary cluster 210 at a specific time instance.

[0072] In step 370, secondary cluster 220 receives a read request for data stored in secondary cluster 220. The read request may be received from end user systems 140. In step 380, secondary
25 cluster 220 identifies a set of regular records based on the cluster safe time, specifically, the regular records having corresponding commit times less than or equal to the cluster safe time. In step 390, secondary cluster 220 provides the identified set of regular records as a response to the read request. Control passes to step 370, whereby subsequent read requests are received and processed.

[0073] It may be again appreciated that the steps of determining the cluster safe time in step 360
30 may be concurrently performed along with the loop of steps 370, 380, 390. At different time instances, different cluster safe times may be identified based on the apply safe times received from primary cluster 210. As the cluster safe time changes, different updates (regular records) may be provided in the responses. As demonstrated in the below sections, the different updates provided in response to the read requests are provided consistent with the commit order in which the
35 transactions were committed in primary cluster 210, thereby providing transaction consistency in

secondary cluster 220.

[0074] Thus, aspects of the present disclosure provide transactional consistency during asynchronous propagation of transactions across clusters 210 and 220. In the following sections, several aspects of the present disclosure are illustrated with respect to a distributed database hosted on clusters 210 and 220. However, the features of the present disclosure may be implemented with respect to other data services (e.g., file server) as well, as will be apparent to one skilled in the relevant arts by reading the disclosure herein. The manner in which a distributed database may be deployed in clusters 210/220 is described below with examples.

[0075] 5. Distributed Database

[0076] Figure 4A depicts the internal details of a distributed database hosted in a cluster in one embodiment. As is well known, a distributed database is often implemented based on multiple nodes (clusters 210/220) that cooperatively provide a unified view of database interfaces, while shielding the users from the underlying storage and processing of data. Distributed databases thus provide for fault tolerance (of nodes or storage), enhanced performance, data redundancy (by a replication factor), etc., as is well known in the relevant arts.

[0077] Distributed database 400 is accordingly shown containing a cluster of nodes such as VMs, physical machines or containers (also referred to as a “universe”) that collectively function as a highly available and resilient distributed database. For illustration, the universe of distributed database 400 is shown containing three nodes 112A-112C of primary cluster 210, though in alternative embodiments, distributed databases contain fewer or more nodes depending on the purpose for which the database is designed.

[0078] Distributed database 400 is a system-of-record/ authoritative database that geo-distributed applications can rely on for correctness and availability. Distributed database 400 allows applications to easily scale up and scale down across multiple regions in the public cloud, on-premises data centers or across hybrid environments without creating operational complexity or increasing the risk of outages.

[0079] Distributed database 400 may be deployed in a variety of configurations depending on business requirements, and latency considerations. Some examples are single availability zone (zone/rack/failure domain), multiple availability zones in a region, multiple regions (with synchronous and asynchronous replication choices), etc. An example of such a distributed database is Yugabyte DB available from YugabyteDB, Inc.

[0080] In one embodiment, the universe of distributed database 400 consists of one or more keyspaces, with each keyspace being a namespace that can contain one or more database tables. Distributed database 400 automatically splits the data in each database table into multiple parts/ shards referred to as “tablets”, replicates the tablets and load-balances these database tables/tablets

across the nodes in the universe, while respecting user-intent such as cross-AZ or region placement requirements, desired replication factor, and so on. Distributed database 400 automatically handles failures (e.g., node, availability zone or region failures), and re-distributes and re-replicates data back to desired levels across the remaining available nodes while still respecting any data placement requirements. The components of distributed database 400 are described in detail below.

[0081] Distributed database 400 has three components - Master process, T-Server process and data storage. The Master (Server) processes are responsible for keeping system metadata/records, such as what tables exist in the system, where their data is live, what users/roles exist, the permissions associated with them, etc. Master processes also are responsible for coordinating system-wide operations such as create/alter drop tables and initiating maintenance operations such as load-balancing or initiating re-replication of under-replicated data. The Master process' executing in the different nodes (112A-112B) are not in the critical path of IO against user tables (which is handled by T-Server processes as described below).

[0082] T-Server processes 420A-420C are responsible for hosting/serving user data (e.g., database tables). Each T-Server process does the actual IO for end user requests received from user applications (executing in end user systems 140). The user requests may be according to the various protocols supported by distributed database 400. Query Layer, executing as part of each T-Server process, implements the server-side of multiple protocols/ APIs that distributed database 400 supports such as Apache Cassandra CQL, Redis APIs, SQL API, etc. The user data/database tables are maintained in the database storage (e.g. Postgres/Postgre SQL databases) in each node.

[0083] As noted above, each database table is split into one or more shards/tablets based on groups of primary keys. Each tablet is composed of one or more tablet-peers depending on the replication factor, with each T-Server process hosting one or more tablet-peers. The manner in which a table having one or more tablets with a replication factor of 3 (that is, 3 peers) may be maintained in nodes 112A-112C is depicted in Figure 4A.

[0084] Each T-Server process also coordinates operations across tablets (by sending messages via paths 421-424) hosted by it by using techniques such as server-global block cache (leading to highly efficient memory utilization in cases when one tablet is read more often than others), throttled compactions (to prevent high foreground latencies during a compaction storm), small/large compaction queues to keep the system functional even in extreme IO patterns, server-global memstore limits, auto-sizing of block cache/memstore, striping tablet load uniformly across data disks, etc.

[0085] In one embodiment, the Master and T-Server processes use Raft, a distributed consensus algorithm, for replicating changes to system metadata or user data respectively across a set of

nodes. The detail of the Raft consensus algorithm is available in the paper entitled “In Search of an Understandable Consensus Algorithm (Extended Version)” by Diego Ongaro and John Ousterhout of Stanford University. Specifically, the Master process’ executing in the different nodes (112A-112B) forms a Raft group with its peers, while the tablet-peers (e.g. “tablet 1, peer 1”, “tablet 1, peer 2”, etc.) corresponding to each tablet (e.g. “tablet 1”) hosted on different T-Servers (in nodes 112A-112C) form a corresponding Raft group and replicate data between each other.

[0086] Thus, the Master and T-Server processes along with the operation of Raft groups provides for a transactional, high performance distributed database (400) for planet-scale applications. It may be appreciated that a similar distributed database may be hosted in nodes 115A-115C of secondary cluster 220 for operating as a standby for the universe of primary cluster 210. In addition, multiple distributed databases similar to database 400 may be hosted by computing infrastructures 110 and 120.

[0087] By default, a distributed database such as YugabyteDB provides synchronous replication and strong consistency across geo-distributed data centers such as clusters 210 and 220. However, many use cases do not require synchronous replication or justify the additional complexity and operating costs associated with managing three or more data centers/clusters. As such, aspects of the present disclosures facilitate a cross-cluster (xCluster) deployment that provides asynchronous replication across two data centers or clusters.

[0088] For better understanding, the description is continued with the manner in which distributed/multi-shard transactions are implemented in a distributed database. Thereafter, the manner in which such distributed transactions are asynchronously propagated across clusters is described with examples.

[0089] 6. Implementing Distributed Transactions in a Distributed Database

[0090] Figure 4B depicts the manner in which distributed transactions are implemented in a distributed database (400) in one embodiment. The block diagram is shown containing T-Server 1, T-Server 2, etc. which may correspond to one of T-Server processes 420A-420C of Figure 4A. The manner in which the T-Servers operate to implement a distributed transaction is described in detail below.

[0091] T-Server 1 is shown receiving (via path 451) a distributed transaction from one of end user systems 140. For illustration, it is assumed that the distributed transaction noted above with respect to Figure 2B (containing the updates $K1 = V1$ and $K2 = V2$) is received via path 451. In response to receiving the transaction, a transaction manager (TXN MGR) executing in T-Server 1 creates a transaction status record for the received transaction by sending (path 452) a request to T-Server 2, which in response allocates a new transaction identifier (461) in transaction status tablet for the

received transaction. After receiving the provisional identifier from T-Server 2, the transaction manager identifies the different tablets containing the keys K1 and K2 specified in the received transaction.

[0092] Transaction manager then writes the values V1 and V2 as provisional records to the different tablets by sending appropriate requests (paths 453) to the T-Servers managing the tablets, assumed to be T-Server 3 and T-Server 4 for illustration. The provisional records are indicated by 462 (K1 = V1) and 463 (K2 = V2). Other updates (key = value) specified in the received transaction may be similarly performed to cause provisional records to be written to (stored in) the distributed database (400).

[0093] Upon reaching the commit instruction in the transaction and identifying that all provisional records have been successfully stored, transaction manager sends (path 454) a commit indication to T-Server 2 and immediately sends (via path 455) an acknowledgment to the requesting end user system 140. Thereafter, T-Server 2 marks the transaction identifier as being committed (471), and then asynchronously sends (paths 456) an Apply indication to the tablets storing the constituent provisional records (corresponding to the transaction). Each T-Server receiving the Apply indication thereafter converts the corresponding provisional record (462/463) as a corresponding regular record (472/473). It may be appreciated that the Apply indication may be sent periodically for a batch of committed transactions.

[0094] Similarly, the T-Servers of distributed database 400 may receive and process a large number of distributed transactions. The manner in which distributed transactions (on distributed data) are propagated asynchronously from a primary cluster to a secondary cluster is described below with examples.

[0095] 7. Asynchronous Propagation in Distributed Database

[0096] Figure 5A depicts the manner in which transactions are propagated from a primary cluster (210) to a secondary cluster (220) in one embodiment. The events occurring at primary cluster 210 is shown above timeline 500, while the events occurring at secondary cluster 220 are shown below timeline 500. Provisional records are written in primary cluster 210 at time instance 521, with the same records being replicated to secondary cluster 220 by time instance 522, with “nd” representing a network delay (for the propagation). At time instance 523, the transactions are committed in primary cluster 210. As described above, an Apply indication for the provisional records is thereafter sent at time instance 524. The delay between the transaction commit and the Apply indication is typically less than 250 ms (milliseconds). By time instance 525, the Apply indications are propagated to secondary cluster 220.

[0097] It should be appreciated that the propagation of the updates is asynchronous, that is the provisional records stored/apply indications corresponding to different tablets may be propagated

at different time instances. In other words, the time instances at which the updates are transmitted/propagated to the secondary cluster has no correlation to the time instances at which the updates were stored in the primary cluster.

[0098] As propagation is performed asynchronously, there may be several tablets that may not have completed the propagation of the Apply indications. According to an aspect of the present disclosure, a cluster safe time is computed at time instance 526 and distributed to the various nodes of the secondary cluster (220) by time instance 527. The cluster safe time is the minimum of the apply safe times received for the different tablets in primary 210.

[0099] Thus, transactions are propagated from primary cluster 210 to a secondary cluster 220. According to an aspect, secondary cluster 220 sends a plurality of get-changes requests corresponding to respective shards/tablets of primary cluster 210, with each shard in primary cluster 210 transmitting the corresponding updates in response to the respective get-changes request asynchronously. The manner in which such get-changes requests are used for propagation of transactions/updates is described in detail below.

[0100] Figure 5B depicts the manner in which transactions are propagated across clusters (210 to 220) for a single tablet of a distributed database (400) in one embodiment. Timeline 550 is shown containing various time instances such as 100, 101, 102, etc. 571-575 represent various operations being performed at a primary cluster (210) such as writes of provisional records (WRITE_OP) and Apply indications (APPLYING). Each operation is shown having an operation identifier (Opid), a hybrid time (HT) representing a unique time that is used as the reference time across multiple nodes, a type of operation, the key value written, and an associated transaction identifier (TxnId).

[0101] At time instance 105, a GetChanges request is received from a secondary cluster (220). In response to the request, primary cluster 210 determines the changes/updates made in primary cluster 210 from the previous receipt of the get change request (assumed to be 100 for illustration) to the time instance at which the response to the change request is sent. In other words, primary cluster 210 determines the updates between 100 and 108. It may be observed that the GetChanges response includes provisional records, regular records and also apply indications. In addition, the GetChanges response includes an apply safe time which is the time instance (105) of the last apply indication received in the tablet.

[0102] According to an aspect, the apply safe time is a hybrid time per tablet that is picked which guarantees that all single-shard writes and distributed transaction intents (provisional records) which were committed before this time have already been applied\marked as regular records. By itself, the secondary cluster can compute an extremely conservative apply safe time, using the start time of the oldest active intent it knows about. But this will result in long running transactions holding up the apply safe time and consequently the cluster safe time.

Instead the primary cluster can compute a more recent apply safe time by resolving its pending intents.

[0103] In one embodiment, apply safe time for a shard/tablet in primary cluster 210 indicates a time instance such that all transactions related to the shard and having commit time less than or equal to the time instance have been applied in the shard. In other words, apply safe time for a primary shard at time T is a time instance A such that if the corresponding secondary shard applies all the primary changes timestamped at or before time T, then it will be safe for secondary cluster 220 to read regular records consistently from its local storage at time instance A or earlier.

[0104] Assuming C to be a set of transactions effecting a shard/tablet (that is, they have an update stored in the shard) that have been committed at or before time T. Remove from C all the transactions whose Apply indications have been received by the shard at or before T. If C is (now) empty, then the apply safe time is T (or any value prior to T).

[0105] Otherwise, when C is not empty, find the transaction with the earliest commit time in C, breaking ties arbitrarily, and then set the apply safe time to a time instance before the earliest commit time. The otherwise part may be restated as the apply safe time is (the minimum commit time of any transaction remaining in C) – delta, where the delta part is the minimum possible determent of a time value, but is assumed to be 1 for illustration.

[0106] From the above, it may be appreciated that the apply safe time indicates a time instance such that all transactions related to the shard and having commit time less than the time instance have been applied in the shard. In an alternative embodiment, the condition of “less than or equal to” the time instance may be employed. The above technique has the property that the computation of the apply safe time at time T is not related to the computation of any of the previously sent apply safe times.

[0107] For convenience, the above noted technique for computing the apply safe time (AST) at time T may be represented by the following predicates:

P1: Shard without any updates => Set AST = T (or any time instance prior to T)

P2: Shard with updates, but no corresponding transactions are committed => Set AST = T (or any time instance prior to T)

P3: Shard with updates, and has transactions committed at or before T but have not been applied to the shard => AST = minimum (commit time of all unapplied committed transactions) – 1

P4: Shard with updates, and all committed transactions have been applied => Set AST = T (or any time instance prior to T)

[0108] Cluster safe time is the latest hybrid time available across all tablets in a given database at which reads are transactionally consistent. In one embodiment, cluster safe time is the minimum

of apply safe times across all tablets (per database). On the secondary cluster, reads are transactionally consistent across tablets and tables only when read as of a time less than the cluster safe time. So even within a tablet, the system only needs to guarantee transactionality and ordering within the cluster safe time. Such usage may resolve the problem that provisional records and
5 apply indication may be propagated out of order across the clusters.

[0109] With such a design, the system achieves transactionality and consistency without the need to replicate the transaction status tablet. It should be noted that there is no effect of long running transactions on the system. A long running transaction will just result in provisional records staying on the secondary cluster for a long time, just like on the primary cluster. The primary cluster picks
10 its safe time as the apply safe time, and as such the moving forward of the apply safe time on the primary cluster is not stopped due to long running transactions.

[0110] The description is continued with an example data illustrating the manner in which several aspects of the present disclosure are provided in one embodiment.

[0111] **8. Illustrative Example**

15 [0112] Figures 6A-6C together illustrate the manner in which transactional consistency in a secondary cluster is provided when transactions on distributed data are propagated asynchronously from a primary cluster in one embodiment. Specifically, Figures 6A and 6B depict the data and operations performed in a primary cluster (210), while Figure 6C depicts the data and operations performed in a secondary cluster (220). The term "Base" herein refers to a common point in time
20 where the data in both the primary cluster and the secondary cluster are transactionally consistent.

[0113] T11, T12, etc. are unique identifiers associated with each transaction. Each transaction includes a Begin statement (indicating the start of the transaction), a set of updates (indicating the tablet such as A, B, C where the update is to be performed) and a commit statement. It should be noted that the updates shown here refer to any change to the underlying data and may correspond
25 to addition of new data (INSERT statement in SQL) or update of existing data (UPDATE statement in SQL), as will be apparent to one skilled in the relevant arts.

[0114] Referring to Figure 6A, column 610 indicates the hybrid time (HT) and forms the reference time (time sequence) for all the other columns in Figures 6A-6C. Column 620 indicates the operations performed in primary cluster 210 from the view point of (or in interfaces with) a user.
30 Column 630 shows the transactions that are visible/made available at primary cluster 210 at the various hybrid times. It may be observed that each transaction is made available at the primary cluster at the time of performance of the commit operation (e.g. T12 at 110), and not after the corresponding updates have been applied to the different tablets (Tablet A at 111 and tablet B at 112). Such availability is consistent with the implementation of distributed transactions described
35 above with respect to Figure 4B. From column 630, it may be appreciated that the commit sequence

is [T1, T12, T14, T13, T15] and the corresponding commit times are [106, 110, 115, 118, 124].

[0115] Referring to Figure 6B, columns 641, 644 and 647 respectively indicate the operations performed in three tablets (A, B and C) of primary cluster 210 at the corresponding hybrid time (indicated in column 610). It may be appreciated that each of the multi-shard updates (e.g., “T11 Update A, B”) from the point of view of the user has been converted to individual updates/writes (e.g., “T11 Update”) in the corresponding tablets. In addition, the commit commands/operations (“T11 Commit”) have been converted to respective apply operations/indications (“T11 Apply”, “T12 Apply”, etc.) in columns 641, 643 and 645. Each apply operation is shown along with a commit time (CT). The commit time indicates the time that is stored associated with the regular record generated by converting the corresponding provisional record in response to receiving the apply operation/indication.

[0116] It may be observed that the operations in columns 641, 644 and 647 may be performed at the same time or later than the time at which the operation is performed in column 620. For example, the update “T11 Update A, B” occurring at 102 in column 620 is shown as corresponding updates “T11 Update” at the same time under tablets A and B in columns 641 and 644 respectively. However, though the “T11 Commit” operation is performed at 106 in column 620, the corresponding apply operations in tablets A and B are shown occurring at time instances 114 and 109 respectively. Such a delay may be due to the distributed architecture of the primary cluster (210).

[0117] Columns 642, 645 and 648 respectively indicate the apply safe times (AST1, AST2 and AST3) corresponding to tablets A, B and C. Columns 643, 646 and 649 indicate the predicates (P1 to P4 noted above) that have been used to compute the apply safe times AST1, AST2 and AST3 respectively. For example, it may be observed that AST2 is set to the current time instance (that is HT/column 610) at time instances 100 and 101, in view of predicate P1 (tablet B having no updates) and at time instances 102 to 105 in view of predicate P2 (table B has updates corresponding to transactions T11 and T12 that have not yet been committed).

[0118] At HT=106, AST2 is computed in view of predicate P3 as there is an unapplied committed transaction T11 for tablet B. As such AST2 is set to minimum (commit time of T11) – 1, that is, $106-1 = 105$. The same predicate is applicable until HT=109. However, at HT=110, the unapplied committed transaction is T12, and as such $AST2 = \text{commit time of T12} - 1 = 110-1 = 109$. At HT = 112, all the committed transactions until then, that is T11 and T12, have already been applied to tablet B, and accordingly predicate P4 is applied to set AST2 to the current time instance (HT) of 112. Similar logic is applied to determine the apply safe times AST1 and AST3 respectively at different time instances.

[0119] Referring to Figure 6C, columns 661-666 indicates the operations received and stored in

the three tablets (A, B and C) of secondary cluster 220, along with the apply safe time (AST) received for each tablet at the corresponding hybrid time (indicated in column 610). Each tablet is shown associated with a corresponding delay, which indicates the down shift of the entries for the corresponding tablet. For example, tablet A in column 661 having a delay of 4 has entries down
 5 shifted by 4, that is, operations performed at 102, 105, 106, etc. in tablet A in column 641 are now shown being received at 106, 109, 110, etc. in tablet A in column 661. Similarly, the other tablets are shown receiving the updates and indications with a corresponding delay.

[0120] Column 670 indicates the cluster safe time computed at different hybrid times as the minimum of the apply safe times, that is as minimum (AST1, AST2, AST3). Column 680 shows
 10 the transactions that are visible/made available at secondary cluster 220 at the various hybrid times. The specific transactions that are to be made available are determined based on the cluster safe time. For example, at HT=118, secondary cluster 220 determines that the cluster safe time is 109, and accordingly makes visible (sends in response to read requests) the regular records that have a commit time (T11 in tablets A and B) less than equal to the cluster safe time. In other words, the
 15 data corresponding to transaction T11 is made available at secondary cluster 220. It may be observed that the transactions are made available in the same sequence as the commit sequence noted above, though with a corresponding delay.

[0121] Thus, aspects of the present disclosure preserve and guarantee transactional atomicity and global ordering when propagating change data from a primary cluster to a secondary cluster. Due
 20 to the asynchronous nature of the replication, such a deployment comes with non-zero RPO (Recovery Point Objective) in the case of the primary cluster outage. The actual value depends on the replication lag, which in turn depends on the network characteristics between the regions. RTO (Recovery Time Objective) is very low, as it only depends on the applications switching their connections from one cluster to another. The applications may be designed in such a way that the
 25 switch happens as fast as possible. The description is continued with the manner in which optimizations may be performed for the determination of apply safe times, and by extension, the cluster safe time in alternative embodiments.

[0122] 9. Optimizations

[0123] Figure 7 is a high-level sequence diagram illustrating the manner in which apply safe times
 30 and a cluster safe time are determined in one embodiment. The sequence of steps is shown occurring between consumer (secondary cluster 220), producer tablet leader (one of the nodes in primary cluster 210) and transaction coordinator(s) (T-Server(s) coordinating the transaction such as T-Server 2 for the example noted above with respect to Figure 4A).

[0124] At 721, the producer tablet leader receives a GetChanges request (noted above at time
 35 instance 105) from the consumer. At 722, the producer tablet leader gets its leader safe time. As is

well known, the tablet leader in a cluster refers to the node that receives and processes user requests for a tablet, and as such the leader safe time represents the hybrid time that forms the reference time for other nodes in the cluster. The producer tablet leader then determines a set of transactions that are to be resolved based on the provisional records stored in the tablet.

5 [0125] At 724, the producer tablet leader sends get-transaction-status requests (1...N, where N is the number of transactions in the determined set) to the transaction coordinator(s), and receives (at 725) corresponding get-transaction-status responses indicating the status of the transactions (such as “Commit”, “In-progress”, “Aborted”, etc.). As such, the producer tablet generates apply actions that converts the provisional records into regular records.

10 [0126] At 727, the producer tablet computes the current end of WAL (Write-ahead Log) location to identify the set of records that can be transmitted. At 728, the producer tablet sends a GetChanges response containing all records in the commit order. If the sent response includes the end of WAL location computed at 727, it computes an apply safe time, specifically, as being equal to the leader safe time computed at 722, and sends it along with the records to the consumer. If the
15 records could not all be transmitted in the current GetChanges response, then on the next GetChanges request, the actions at 722 to 727 are skipped and the previous values computed at 722 and 727 for the previous GetChanges request is used for the next set of GetChanges requests, until the WAL computed at 728 is sent to the consumer.

[0127] After receiving the apply safe times for all the producer tablets (in primary cluster 210),
20 the consumer (secondary cluster 220) at 729, determines a cluster safe time as the minimum of the apply safe times received corresponding to all the producer tablets. If the consumer does not get a apply safe time in a GetChanges response, the consumer uses the previously received value(s) for the apply safe times, till a new one is sent to it.

[0128] An example implementation of the above described actions for the computation of the
25 apply safe times is described below. The primary cluster sets the time at which the intents were resolved as apply safe time and sends it to the consumer after all the APPLYING operations that were generated have been sent. On the primary cluster, for each stream (N per tablet) the following variables are initialized (reset to) `last_consistent_op_id = -1` and `last_apply_safe_time = kInvalid`.

30 [0129] In GetChanges RPC (remote procedure call):

1. If `last_apply_safe_time` is `kInvalid`
 - a. `last_apply_safe_time` = Tablet/Leader Safe time
 - b. Resolve all intents and apply all the committed intents.
 - c. Set `last_consistent_op_id` = max raft-replicated opId

35 2. If all records including `last_consistent_op_id` are sent in the GetChanges response

- a. Set `response.apply_safe_time` to `last_apply_safe_time`
- b. Reset `last_apply_safe_time` and `last_consistent_op_id`
3. Else don't set any `response.apply_safe_time`. The next GetChanges RPC will reuse the computed `last_apply_safe_time` and `last_consistent_op_id`

5 [0130] Thus, the apply safe time for a producer tablet/shard (of primary cluster 210) is computed. It may be appreciated that any intent commits during the GetChanges request is also handled in view of the order in which the steps are performed. The primary cluster first picks the value that will be used as apply safe time. After this, the primary cluster resolves the intents, without caring about the intents that committed after the picked apply safe time. It may have an APPLYING
10 operation that has been replicated but since it is conservative, the primary cluster does not move the apply safe time. The next GetChanges request may bump the apply safe time to include these commits that are already replicated. It should be noted that the primary cluster may also have updates newer than the apply safe time. The primary cluster replicates the data to the secondary cluster as fast as it can. The next GetChanges request will move the apply safe time to include
15 those updates.

[0131] Thus, apply safe times for individual shards/tablets in primary cluster 210 are computed and sent to secondary cluster 220, which in turn determines a cluster safe time based on the apply safe times received from primary cluster 210.

[0132] Further optimizations may be performed for the above-described techniques. One
20 optimization may be to not have to block the GetChanges request. The system can resolve and apply intents and compute the `last_apply_safe_time` and `last_consistent_op_id` on a background thread. Other optimizations may be for the transaction coordinator to compute and send an apply safe time regularly, or writing the apply safe time to the log frequently.

[0133] It may be appreciated that the very first-time replication is set up, apply safe time will not
25 move until all log up to the replication setup time is caught up. However, such an operation is as per the expectation of the user. In addition, the primary cluster may need to run a few more RPC to resolve intents. The specific additional RPCs to executed is typically workload dependent, but will be $O(\text{node count})$, where node count is the number of nodes in the primary cluster. If the workload reads recent writes, the proposed solution does not introduce any additional RPC. Since
30 such a pattern happens in the read path, the RPC call is very optimized as well. There is no extra IO or CPU or impact to the user workload. It may be noted that above possible disadvantages of the proposed solution may be resolved using one or more optimizations noted above.

[0134] It should be further appreciated that the features described above can be implemented in
various embodiments as a desired combination of one or more of hardware, software, and
35 firmware. The description is continued with respect to an embodiment in which various features

are operative when the software instructions described above are executed.

[0135] 10. Digital Processing System

[0136] Figure 8 is a block diagram illustrating the details of digital processing system (700) in which various aspects of the present disclosure are operative by execution of appropriate
5 executable modules. Digital processing system 800 may correspond to any of nodes in primary cluster 210 and secondary cluster 220, or in general, to any node in computing infrastructures 110/120.

[0137] Digital processing system 800 may contain one or more processors such as a central processing unit (CPU) 810, random access memory (RAM) 820, secondary memory 830, graphics
10 controller 860, display unit 870, network interface 880, and input interface 890. All the components except display unit 870 may communicate with each other over communication path 850, which may contain several buses as is well known in the relevant arts. The components of Figure 8 are described below in further detail.

[0138] CPU 810 may execute instructions stored in RAM 820 to provide several features of the
15 present disclosure. CPU 810 may contain multiple processing units, with each processing unit potentially being designed for a specific task. Alternatively, CPU 810 may contain only a single general-purpose processing unit.

[0139] RAM 820 may receive instructions from secondary memory 830 using communication path 850. RAM 820 is shown currently containing software instructions constituting shared
20 environment 825 and/or other user programs 826 (such as other applications, DBMS, etc.). In addition to shared environment 825, RAM 820 may contain other software programs such as device drivers, virtual machines, etc., which provide a (common) run time environment for execution of other/user programs.

[0140] Graphics controller 860 generates display signals (e.g., in RGB format) to display unit 870
25 based on data/instructions received from CPU 810. Display unit 870 contains a display screen to display the images defined by the display signals. Input interface 890 may correspond to a keyboard and a pointing device (e.g., touch-pad, mouse) and may be used to provide inputs. Network interface 880 provides connectivity to a network (e.g., using Internet Protocol), and may be used to communicate with other systems connected to the networks.

[0141] Secondary memory 830 may contain hard drive 835, flash memory 836, and removable
30 storage drive 837. Secondary memory 830 may store the data (e.g., data portions shown in Figures 6A-6C) and software instructions (e.g., for performing the actions of Figures 3), which enable digital processing system 800 to provide several features in accordance with the present disclosure. The code/instructions stored in secondary memory 830 may either be copied to RAM 820 prior to
35 execution by CPU 810 for higher execution speeds, or may be directly executed by CPU 810.

[0142] Some or all of the data and instructions may be provided on removable storage unit 840, and the data and instructions may be read and provided by removable storage drive 837 to CPU 810. Removable storage unit 840 may be implemented using medium and storage format compatible with removable storage drive 837 such that removable storage drive 837 can read the data and instructions. Thus, removable storage unit 840 includes a computer readable (storage) medium having stored therein computer software and/or data. However, the computer (or machine, in general) readable medium can be in other forms (e.g., non-removable, random access, etc.).

[0143] In this document, the term "computer program product" is used to generally refer to removable storage unit 840 or hard disk installed in hard drive 835. These computer program products are means for providing software to digital processing system 800. CPU 810 may retrieve the software instructions, and execute the instructions to provide various features of the present disclosure described above.

[0144] The term "storage media/medium" as used herein refers to any non-transitory media that store data and/or instructions that cause a machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical disks, magnetic disks, or solid-state drives, such as storage memory 830. Volatile media includes dynamic memory, such as RAM 820. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid-state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge.

[0145] Storage media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between storage media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 850. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0146] Reference throughout this specification to "one embodiment", "an embodiment", or similar language means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present disclosure. Thus, appearances of the phrases "in one embodiment", "in an embodiment" and similar language throughout this specification may, but do not necessarily, all refer to the same embodiment.

[0147] Furthermore, the described features, structures, or characteristics of the disclosure may be combined in any suitable manner in one or more embodiments. In the above description, numerous specific details are provided such as examples of programming, software modules, user selections, network transactions, database queries, database structures, hardware modules, hardware circuits,

hardware chips, etc., to provide a thorough understanding of embodiments of the disclosure.

[0148] 11. Conclusion

[0149] While various embodiments of the present disclosure have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the
5 breadth and scope of the present disclosure should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

[0150] It should be understood that the figures and/or screen shots illustrated in the attachments highlighting the functionality and advantages of the present disclosure are presented for example
10 purposes only. The present disclosure is sufficiently flexible and configurable, such that it may be utilized in ways other than that shown in the accompanying figures.

[0151] Further, the purpose of the following Abstract is to enable the Patent Office and the public generally, and especially the scientists, engineers and practitioners in the art who are not familiar with patent or legal terms or phraseology, to determine quickly from a cursory inspection the nature
15 and essence of the technical disclosure of the application. The Abstract is not intended to be limiting as to the scope of the present disclosure in any way.

What is claimed is:

1. A distributed computing system comprising:

a primary cluster and a secondary cluster, each cluster maintaining a plurality of shards,
said primary cluster operable to:

5 receive a set of transactions, each transaction comprising corresponding
updates to at least one shard in said primary cluster, followed by a commit
command to indicate end of updates in the transaction;

 process each transaction of said set of transactions by:

 storing each update as a corresponding provisional record in
10 a respective shard of said primary cluster;

 processing said commit command for the transaction at a
corresponding commit time; and

 applying said commit command, after said processing, to
each of said respective shards, by converting each provisional record
15 of the transaction to a corresponding regular record in the
corresponding shard, each regular record being stored associated
with the corresponding commit time for the transaction, and

 asynchronously propagate the updates for the shards to said secondary
cluster, along with an apply safe time for each shard in said primary cluster,

20 wherein said apply safe time propagated for a shard in said primary cluster
indicates a time instance such that all transactions related to the shard and having
commit time less than said time instance have been applied in the shard;

said secondary cluster operable to:

 store each of the asynchronously received updates in a corresponding shard
25 of said secondary cluster;

 determine a cluster safe time as the minimal value of all the apply safe times
corresponding to all of said plurality of shards in said primary cluster;

 receive a read request directed to said secondary cluster for data stored in
said secondary cluster;

30 identify a set of regular records having corresponding commit times less
than or equal to said cluster safe time; and

 provide said set of regular records as a response to said read request.

2. The distributed computing system of claim 1, wherein no transactions related to said shard are committed by a first time instance, wherein said apply safe time is set to any value up to and including said first time instance.

5 3. The distributed computing system of claim 1, wherein a first subset of transactions related to said shard are committed but not applied by a second time instance, wherein said apply safe time is set to a value which is less than or equal to a minimum of commit times of said first subset of transactions.

10 4. The distributed computing system of claim 1, wherein all transactions committed by a third time instance related to said shard are applied by said third time instance, wherein said apply safe time is set to any value up to and including said third time instance.

15 5. The distributed computing system of claim 1, wherein said secondary cluster sends a plurality of get-changes requests corresponding to respective shards of said primary cluster, and wherein each shard in said primary cluster transmits the corresponding updates in response to the respective get-changes request asynchronously,

 wherein the corresponding updates includes provisional records, and regular records with corresponding commit times.

20

6. The distributed computing system of claim 5, wherein in response to a get-changes request directed to a shard in said primary cluster, wherein a shard leader in said primary cluster is operable to:

 identify a leader safe time;

25 determine one or more transactions to be resolved based on the provisional records stored in said shard;

 send get-transaction-status requests to one or more transaction coordinators;

 receive responses to said get-transaction-status requests indicating statuses of whether said one or more transactions are still in progress, committed or aborted;

30 generate one or more apply actions to convert the provisional records of committed transactions to corresponding regular records;

 compute a current end of WAL (write-ahead log) location; and

 set said apply safe time for said shard as being equal to said leader safe time if all records until said current end of WAL location have been sent to said secondary cluster.

35

7. The distributed computing system of claim 1, where each of said primary cluster and said secondary cluster implements a distributed database,

wherein each of said plurality of shards corresponds to one of a plurality of tablets in said distributed database, wherein each tablet stores data corresponding to a unique set of keys.

5

8. A method for providing transactional consistency in a secondary cluster, said method being performed in said primary cluster, said method comprising:

receiving a set of transactions, each transaction comprising corresponding updates to at least one shard in said primary cluster, followed by a commit command to indicate end of updates
10 in the transaction;

processing each transaction of said set of transactions by:

storing each update as a corresponding provisional record in a respective shard of said primary cluster;

processing said commit command for the transaction at a corresponding
15 commit time; and

applying said commit command, after said processing, to each of said respective shards, by converting each provisional record of the transaction to a corresponding regular record in the corresponding shard, each regular record being stored associated with the corresponding commit time for the transaction, and

20 asynchronously propagating the updates for the shards to said secondary cluster, along with an apply safe time for each shard in said primary cluster,

wherein said apply safe time propagated for a shard in said primary cluster indicates a time instance such that all transactions related to the shard and having commit time less than said time instance have been applied in the shard.

25

9. The method of claim 8, wherein no transactions related to said shard are committed by a first time instance, wherein said apply safe time is set to any value up to and including said first time instance.

30 10. The method of claim 8, wherein a first subset of transactions related to said shard are committed but not applied by a second time instance, wherein said apply safe time is set to a value which is less than or equal to a minimum of commit times of said first subset of transactions.

11. The method of claim 8, wherein all transactions committed by a third time instance related to said shard are applied by said third time instance, wherein said apply safe time is set to any value up to and including said third time instance.

5 12. The method of claim 8, further comprising:
 receiving, from said secondary cluster, a plurality of get-changes requests corresponding to respective shards of said primary cluster; and
 transmitting asynchronously, from each shard in said primary cluster, the corresponding updates in response to the respective get-changes request,
10 wherein the corresponding updates includes provisional records and regular records with corresponding commit times.

 13. The method of claim 12, wherein in response to a get-changes request directed to a shard in said primary cluster, said method further comprises:
15 retrieving a leader safe time from a leader node in said primary cluster;
 determining one or more transactions to be resolved based on said leader safe time and the provisional records stored in said shard;
 sending resolve intent requests to shards participating in said one or more transactions;
 receiving responses to said resolve intent requests indicating statuses of whether the
20 provisional records corresponding to said one or more transactions already committed, have been converted to regular records; and
 computing said apply safe time for said shard as being equal to said leader safe time if said statuses indicate that the conversion is successful.

25 13. The method of claim 12, wherein in response to a get-changes request directed to a shard in said primary cluster, said method further comprising:
 identifying, by a shard leader in said primary cluster, a leader safe time;
 determining, by said shard leader, one or more transactions to be resolved based on the provisional records stored in said shard;
30 sending, by said shard leader, get-transaction-status requests to one or more transaction coordinators;
 receiving, by said shard leader, responses to said get-transaction-status requests indicating statuses of whether said one or more transactions are still in progress, committed or aborted;
 generating, by said shard leader, one or more apply actions to convert the provisional
35 records of committed transactions to corresponding regular records;

computing, by said shard leader, a current end of WAL (write-ahead log) location; and
setting, by said shard leader, said apply safe time for said shard as being equal to said leader
safe time if all records until said current end of WAL location have been sent to said secondary
cluster.

5

14. The method of claim 8, where each of said primary cluster and said secondary cluster
implements a distributed database,

wherein each of said plurality of shards corresponds to one of a plurality of tablets in said
distributed database, wherein each tablet stores data corresponding to a unique set of keys.

10

15. A non-transitory machine-readable medium storing one or more sequences of
instructions providing transactional consistency in a secondary cluster, wherein execution of said
one or more instructions by one or more processors contained in a node of said secondary cluster
causes said node to perform the actions of:

15

receiving asynchronously from a primary cluster, updates to a plurality of shards
maintained in a primary cluster, along with an apply safe time for each shard in said primary
cluster,

wherein said apply safe time propagated for a shard in said primary cluster indicates a time
instance such that all transactions related to the shard and having commit time less than said time
instance have been applied in the shard;

20

storing each of the asynchronously received updates in a corresponding shard of said
secondary cluster;

determining a cluster safe time as the minimal value of all the apply safe times
corresponding to all of said plurality of shards in said primary cluster;

25

receiving a read request directed to said secondary cluster for data stored in said secondary
cluster;

identifying a set of regular records having corresponding commit times less than or equal
to said cluster safe time; and

providing said set of regular records as a response to said read request.

30

16. The non-transitory machine-readable medium of claim 15, further comprising one or
more instructions for:

sending a plurality of get-changes requests corresponding to respective shards of said
primary cluster,

wherein said receiving receives the updates in response to said plurality of get-changes requests.

17. The non-transitory machine-readable medium of claim 15, where each of said primary
5 cluster and said secondary cluster implements a distributed database,
wherein each of said plurality of shards corresponds to one of a plurality of tablets in said distributed database, wherein each tablet stores data corresponding to a unique set of keys.