Distrib Parallel Databases (2013) 31:377–411
DOI 10.1007/s10619-012-7116-0

Efficiently repairing and measuring replica consistency in distributed databases

Javier García-García · Carlos Ordonez · Predrag T. Tosic

Published online: 27 November 2012

Abstract In a distributed database, maintaining large table replicas with frequent asynchronous insertions is a challenging problem that requires carefully managing a tradeoff between consistency and availability. With that motivation in mind, we propose efficient algorithms to repair and measure replica consistency. Specifically, we adapt, extend and optimize distributed set reconciliation algorithms to efficiently compute the symmetric difference between replicated tables in a distributed relational database. Our novel algorithms enable fast synchronization of replicas being updated with small sets of new records, measuring obsolence of replicas having many insertions and deciding when to update a replica, as each table replica is being continuously updated in an asynchronous manner. We first present an algorithm to repair and measure distributed consistency on a large table continuously updated with new records at several sites when the number of insertions is small. We then present a complementary algorithm that enables fast synchronization of a summarization table based on foreign keys when the number of insertions is large, but happening on a few foreign key values. From a distributed systems perspective, in the first algorithm the large table with data is reconciled, whereas in the second case, its summarization table is reconciled. Both distributed database algorithms have linear communication complexity and cubic time complexity in the size of the symmetric difference between the respective table replicas they work on. That is, they are effective when the network speed is smaller than CPU speed at each site. A performance experimental evaluation with synthetic and real databases shows our algorithms are faster than a previous state-of-the art algorithm as well as more efficient than transferring complete tables, assuming large replicated tables and sporadic asynchronous insertions.

J. García-García (✉)
SEPI-UPIICSA, Instituto Politécnico Nacional, Mexico City, Mexico
e-mail: jagarciag@ipn.mx

C. Ordonez · P.T. Tosic
Department of Computer Science, University of Houston, Houston, TX 77204, USA

Keywords Set reconciliation · Replica consistency · Distributed databases

1 Introduction

Distributed databases are increasingly popular among businesses, as well as government, educational and medical organizations. Fault tolerance is one important reason to architect databases so that they are distributed across different physical locations (sites) in order to ensure uninterrupted continuous availability to end users. Another important reason behind a distributed database architecture is the existence of multiple sources of large amounts of data. Physical distribution of contemporary large-scale databases also enables faster queries (whenever the particular data can be found locally). It also enhances load balancing. For these reasons, it has become common in recent years to distribute a large-scale relational database across several sites under a single database schema. Moreover, providing the most recent data is becoming crucial in many situations. In an active distributed database context, detailed and up-to-date data is required to help decision makers in various types of organizations make better decisions. To guarantee fast response to aggregation queries and to achieve a satisfactory level of fault tolerance, various *replication techniques* are commonly used in the implementation of distributed databases [25].

With recent advances in distributed database technologies, more precise analysis of data consistency across sites is becoming a necessity. In an active distributed database where frequent updates occur, the *freshness condition* of a replica is crucial. In such environments, the value of aggregate queries depends on *data currency.* Several types of data partition and replication have been proposed in the literature; some examples include various physical and virtual partitioning techniques, horizontal and vertical fact replication, complete and partial dimension table replication, and so on [4]. However, most of the existing techniques tend to render the underlying distributed database system too complex to manage [3]. Replication gives rise to several potential problems, such as storage overhead and replica inconsistency. Common causes of replica inconsistency include asynchronous updating at different sites, system or local site failures, and data corruption [25]. Referential integrity can be violated by asynchronously updating replicas of referencing fact tables and referenced dimension tables [28]. Replication can be done by replacing the entire dataset or by changing only the updated records. However, it is important to bear in mind that the possibility of losing replica consistency is in practice always present.

In this paper, we propose several algorithms and optimizations for distributed databases in which replicated tables may be outdated. The proposed algorithms and optimizations focus on how to efficiently compute *the degree of obsolescence* of replicated local tables and then decide whether to evaluate a query remotely or locally, thus avoiding (when possible) the transmission of large tables over the network. The approaches we propose all follow an *asynchronous replication model* [34]. We note that our proposed methods are applicable to both Multi-Master and Master-Slave architectures.

It may appear, at first glance, that with the emergence of high-speed communication networks, the transmission of large volumes of data from one site of a distributed database to another is no longer a big problem. However, when relying on the speed of communication networks such as the Internet, one ought to distinguish between *bandwidth* and *latency*. In today's distributed environments latency due to transmission delays over a network also ought to be given a serious consideration [25]. This latency can be very significant when terabytes or more of data are being transmitted, even if this transmission takes place over the fastest-speed Internet currently available.

Depending on the selected replication model in a given distributed database system, our methods can be used both to update (refresh) and repair tables. These distributed algorithms may be included in active distributed database environments where near-real-time data freshness is needed, but where data availability is also of major importance. For instance, a user may request a *currency evaluation* of the replicated tables in a distributed DBMS environment. One possible solution that could use our algorithms is to automatically compute a query evaluation with tables that meet a certain *currency threshold* in terms of the maximum number of differences between a primary replica and secondary replicas, and then determine whether a local or remote query evaluation is appropriate given resource (especially time) constraints on one hand, and specific freshness requirements, on the other hand. In such distributed database environments, a freshness assessment of the replicas is critical.

In our prior work [24] and [10], we introduce metrics to measure replica consistency and to evaluate referential integrity in a distributed database. Following those initial studies, in this paper we propose evaluating our replica consistency and referential integrity metrics based on computing *symmetric differences between pairs of sets.* In particular, we propose a strategy to update and repair replicas in either a Multi-Master or Master-Slave configuration. Furthermore, related to the referential integrity metrics, we extend our metrics to evaluate *inclusion dependency constraints*. Specifically, when measuring replica consistency and updating/repairing replicas in a Master-Slave configuration, the sets on which symmetric differences are evaluated are the primary and secondary replicas. Insofar as referential integrity and inclusion dependency in Multi-Master configurations are concerned, the sets are the referencing foreign key values and the referenced primary key values. Thus, we show that, with our algorithms, an efficient global evaluation of the replica consistency can be obtained in both Master-Slave and Multi-Master configurations. In this paper, we will primarily emphasize the Multi-Master configuration as the most common architecture in practice for the kind of distributed databases that can benefit the most from our optimizations.

We have conducted extensive experiments to evaluate our methods in a distributed database environment. We have experimented with both some common benchmark (synthetic) databases and "real-world" databases (see Sect. 4). Our techniques are especially useful when comparing large tables with highly similar contents, that is, when a pair of tables differ on a small fraction of rows. Such assumptions hold in an active database where the number of rows that are periodically inserted is expected to be relatively low, but the freshness of a replica is crucial.

The rest of this paper is organized as follows. Section 2 provides the necessary background. Section 3 introduces our approach to evaluate and repair replica consistency and referential integrity in a distributed database. Section 4 provides an extensive experimental evaluation with synthetic and real distributed databases. Related work is discussed in Sect. 5. Lastly, we summarize our contribution and draw conclusions in Sect. 6.

2 Definitions and preliminaries

Let $\mathcal{D} = \{D_0, \ldots, D_{N-1}\}$ be a set of N database replicas, one at each of N sites of a distributed database. Replicas at different sites are updated asynchronously. We consider two general kinds of distributed database architectures:

1. Multi-Master, where any table D_i can be asynchronously updated at any time at a local site, and new records are periodically broadcast to all sites; and
2. Master-Slave, where the primary copy, denoted D_0, is at the master site, and each slave site has its secondary copy; if the slave sites are indexed $1, \ldots, N-1$, we have that each D_i, $i = 1, \ldots, N-1$ is a local replica, that is, a secondary copy of D_0 at the i-th slave site.

To have a uniform framework, all replicas are assumed to have the same table names (i.e., the same structure), but potentially different content (i.e., replicas may be inconsistent).

Definition 1 (Referential integrity constraints) Let D_i be defined as $D_i(\{T_1, T_2, \ldots\}, I)$, where each T_p is a table and I a set of referential integrity constraints. A *referential integrity constraint* [7] between T_r and T_s in I is an assertion of the form $T_r(K) \rightarrow T_s(K)$, where T_r is called the referencing table, T_s is called the referenced table, K is a *foreign key* (FK) in T_r and K is the *primary key* (PK) of T_s. We assume the common attribute K has the same name on both tables T_r and T_s. In a valid database state with respect to I, the following two assertions hold:

(1) $T_r.K$ and $T_s.K$ have the same domain; and
(2) for every tuple $t_r \in T_r$ there must exist a tuple $t_s \in T_s$ such that $t_r.K = t_s.K$.

We also study integrity over *inclusion dependency* (ID).

Definition 2 (Inclusion dependency) An ID constraint between attributes $T_r.A$ and $T_s.A$ holds when $\pi_A(T_r) \subseteq \pi_A(T_s)$. If attribute A is the foreign key K in T_r and the primary key in T_s, then values that violate the ID $\pi_K(T_r) \subseteq \pi_K(T_s)$ are *invalid foreign key values.*

We observe that referential integrity considers tuples, while inclusion dependency considers individual values. We additionally observe that we assume a single replicated database, where each incoming query can be entirely evaluated at a single node. However, in a more complex scenario, distributed queries involving references may be allowed, where the referencing table is located at one node and the referenced table is located at another node as, for example, in *partial replication* approaches; however, partial replication techniques are beyond our current scope.

Since we work with outdated replicas, we assume $D_i.T_p$ may contain rows different from $D_j.T_p$, where $i \neq j$ and $D_i, D_j \in \mathcal{D}$, thereby causing replica inconsistency or violation of referential integrity and inclusion dependency constraints. Let $D_i.T_p$ be a replica of table T_p in the local database D_i in a Master-Slave system. Each local secondary replica table $D_i.T_p$, $i = 1, \dots, N-1$ is periodically refreshed according to updates at the primary replica table $D_0.T_p$. We define replica consistency in this context as follows:

Definition 3 (Replica consistency) Let D_i, $i = 1, \dots, N-1$ be a secondary replica in a Master-Slave distributed database D. D_i is said to meet the *replica consistency* constraint if it is an identical copy of the primary replica D_0. That is: $\forall T_p \in D_i$, $D_0.T_p = D_i.T_p$. The primary replica is (trivially) replica consistent.

It is possible that only a subset of tables in D_i meets this type of consistency, but not the full set. In that case, the replica-consistent subset of tables should be constituted of the identical copies of their corresponding primary copies.

In a Multi-Master configuration, where each table can be asynchronously updated at any site and the new (batched) records are only periodically broadcast to all sites (i.e., it is too expensive to broadcast them after every single local table update), whenever we say *replica consistency*, we will be referring to *global replica consistency* as defined below.

Definition 4 (Global replica consistency) A distributed database is *globally replica consistent* if all of its replicas are identical. That is: $\forall T_p \in D_i$ and $T_p \in D_j$, $i \neq j$, $D_i.T_p = D_j.T_p$.

It is possible that only a nontrivial subset of tables in $\mathcal{D}$ meets this type of consistency, if all its corresponding replicas are identical copies. Specifically, a single table meets the global replica consistency if all of its replicas are identical. Note that the global replica consistency property also applies to the Master-Slave systems.

The following metric is used to determine if a replica of a given table in a Master-Slave architecture, say $D_i.T_p$, meets the replica consistency constraint (i.e., if it is up-to-date).

$$\mathrm{cur}(D_i.T_p) = \frac{|D_i.T_p \ominus D_0.T_p|}{|D_0.T_p|} \tag{1}$$

where $D_i.T_p \ominus D_0.T_p$ is the symmetric set difference between tables $D_i.T_p$ and $D_0.T_p$. That is, it is the set union of the records that are in $D_i.T_p$ but not in $D_0.T_p$ and the records that are in $D_0.T_p$ but not in $D_i.T_p$:

$$D_i.T_p \ominus D_0.T_p = (D_i.T_p - D_0.T_p) \cup (D_0.T_p - D_i.T_p). \tag{2}$$

Definition 5 (Size of symmetric difference) We denote the size of the symmetric difference between two replicas by δ. That is, $\delta = |D_i.T_p \ominus D_0.T_p|$.

We can see that, if metric cur() = 0 for table $D_i.T_p$, then $D_i.T_p$ is replica consistent. We observe that the metric cur() shown in Eq. (1) is somewhat similar to the one presented in [24], which is used to determine if a table in a distributed Multi-Master database has all the latest updates. This metric extends the *Jaccard coefficient* [14], defined as follows:

$$\mathrm{gcur}(T_p) = 1 - \frac{|\mathcal{T}_{p\cap}|}{|\mathcal{T}_{p\cup}|} \tag{3}$$

where $\mathcal{T}_{p\cap} = \bigcap_{i=0}^{N-1} D_i.T_p$ and $\mathcal{T}_{p\cup} = \bigcup_{i=0}^{N-1} D_i.T_p$ are the global intersection and the global union, respectively, of the local copies. If the value of gcur() is zero for T_p, then T_p is globally replica consistent.

This metric is defined for a Multi-Master configuration where we assume each replicated table can be asynchronously updated at any site and (batched) new records are periodically broadcast to all sites. The metric is defined in terms of the intersection and the union of the replicated tables; the idea behind it is that gcur() gives a general idea about the consistency of a given table. In contrast, cur() refers to the consistency of a specific secondary replica and is defined in terms of the symmetric set difference evaluated over the primary and the specific secondary replicated table. That is, cur() gives an idea of δ (Definition 5) with respect to the primary copy. Therefore, given a Master-Slave architecture, if all replicas of a given table T_p are replica consistent, then $\mathrm{cur}(D_i.T_p) = 0$ will hold for all D_i, $i = 0, \dots, N-1$. Also, if these conditions hold, then $\mathrm{gcur}(T_p) = 0$. See Lemma 1 in the Appendix for the formal statement and its proof.

Metric gcur() as defined in Eq. (3) gives a global perspective of the replica consistency of a table. We observe that metric cur() (Eq. (1)) will be close to zero if the secondary replica is nearly identical compared to the primary replica. Also, this metric gives an idea about how different the replicated tables are. As the quotient grows, the number of different records in the two tables will also grow. We note that in a Master-Slave system, $D_0.T_p - D_i.T_p$ is the set of records that are not yet propagated to $D_i.T_p$, that is, the records inserted into $D_0.T_p$ that are not yet in $D_i.T_p$. On the other hand, $D_i.T_p - D_0.T_p$ is the set of records not yet deleted from $D_i.T_p$. As for the updated records in $D_0.T_p$, we will consider in this context an update to be constituted of two elementary operations, that is, a delete and an insert of a record. Consequently, to update/repair a replicated table $D_i.T_p$ in a Master-Slave system, we need to propagate the insert and delete operations from $D_0.T_p$ to $D_i.T_p$.

Definition 6 (Updated (repaired) table with respect to replica consistency) We say that a replicated table $D_i.T_p$ is updated (repaired) with respect to replica consistency if

$$D_i.T_p = \big(D_i.T_p \cup (D_0.T_p - D_i.T_p)\big) - (D_i.T_p - D_0.T_p) \tag{4}$$

The next metric, useful in the context of Multi-Master architectures, gives a relative measure of the global foreign key violations. This metric detects the existence of foreign keys with invalid values across all replicas (that is, *globally*). To simplify exposition, in the rest of the paper we will treat null values as referential integrity violations. The metric we propose below is inspired by the one originally introduced in [24].

Assume that T_r and T_s are the referencing table and the referenced table, respectively.

$$\text{grcom}(T_r.K) = \frac{|\mathcal{T}_{r\cup} \bowtie_K \mathcal{T}_{s\cup}|}{|\mathcal{T}_{r\cup}|} \tag{5}$$

This metric gives a global perspective concerning referential integrity. We observe that, if the same erroneous tuple appears in several replicas of table T_r, it will be accounted for by the above metric as just one erroneous tuple. This is motivated by the fact that, in a distributed database context, an inserted erroneous tuple in a given replica will eventually be propagated to the other sites.

We observe that the evaluations of our global metrics, as given by Eqs. (3) and (5), are based on counts over expressions with the union $\cup$ and the intersection $\cap$. We argue that, in many practical scenarios (see Sect. 4 for some examples), we can expect the number of differences among replicas that participate in a union or intersection operation to be low. In contrast, the total number of rows could be very large in a typical large-scale distributed database. Hence, solutions to the replica consistency problem that communicate just the pairwise replica differences among sites can be expected to be more scalable and feasible in practice than approaches that communicate entire tables.

As for the *join* operator, $\bowtie$, this operator is computed on foreign key-primary key attributes with potential referential integrity violations. In our metrics, this operator is used to count the number of rows that correspond to valid references in the referencing table. In other words, to achieve the same objective of quantifying referential integrity, we can compute the set of referencing values that are not in the set of referenced values. After that step, one can count the rows in the referencing table and then subtract the number of invalid rows from the total number of rows. See Lemma 2 in the Appendix for details.

We observe that, in a partial replication approach, distributed queries involving references could be allowed where the referencing table is located at one node and the referenced table at another. Furthermore, global replica consistency does not guarantee that there are no global foreign key violations.

The question that remains to be addressed is how can we efficiently compute the aforementioned pairwise set differences without transferring or broadcasting large amounts of data. Similarly, in case of the union and the intersection set operations (that is, when evaluating Eqs. (3) and (5)), the challenge is to compute the differences $D_0.T_p - D_i.T_p$ or $D_i.T_p - D_0.T_p$ without transferring the entire replicas. We describe our approach to these challenging problems in detail in the next section.

An alternative to computing the operations involved in our metrics in the Master-Slave context could be to broadcast the master replica from a "central" site to all sites and then to compute the required operations locally. However, this alternative can be expected to be expensive in practice, since in many distributed database environments the tables may contain millions of rows. Although, in the case of the referential integrity violations in Multi-Master configurations, only a projection of the foreign key could be transferred instead of the entire table, the overall amount of data to be transferred would still be prohibitively large. However, we argue that the total amount of data to be transferred can be dramatically reduced, by taking advantage of ideas borrowed from abstract algebra and number theory discussed next.

3 Measuring and repairing replica consistency and referential integrity

In this section, we describe in detail our approach to repair and measure replica consistency, as well as referential integrity in distributed databases, based on the metrics defined in Sect. 2 and some ideas from number theory (described below). Our metrics cur() and grcom(), as expressed in Eqs. (1) and (5) respectively, are based on the computation of the basic set operations, such as union, intersection and symmetric difference. The definition of cur() is directly based on the symmetric difference between two sets. Moreover, as we will show shortly, the metric gcur(), Eq. (3), defined in terms of set unions and intersections, may also be computed using the pairwise set differences.

Computing our metrics in a realistic large-scale distributed database scenario is challenging, since the sets are located at different sites and can be expected to be very large in practice. The communication complexity required to evaluate the proposed metrics by directly applying the definitions from the previous section can be therefore expected to be very high. Our objective, then, is to use advanced algebraic techniques in order to evaluate these metrics efficiently, and so that the communication complexity (that is, how much data is communicated between different sites) is relatively low, rendering the overall replica consistency approach feasible. We note that, once evaluation of the said metrics has been completed, repairing the violations is relatively straight-forward, as explained below.

To compute our metrics efficiently, we propose a *set reconciliation approach* (SRA), which adapts set reconciliation techniques originally described in [21]. We assume the cost of transferring large tables is high; this is a very realistic assumption in a typical distributed database environment. We also assume that the number of different rows across replicas may vary. As there is a great deal of similarity between computing the metric cur() (see Eq. (1)) and the metric gcur() (Eq. (3)), we describe in some detail just how to compute the latter.

An alternative to computing the operations involved in Eq. (1) in a Master-Slave configuration could be to transfer to the slave site the primary replica involved and then evaluate the required computations locally. However, this alternative can be expected to be very expensive in a large distributed DB, especially in terms of communication complexity since the operands could be tables with millions of records. Once the data records have been transferred to the slave site, performing the calculations involved may also be quite expensive. An advantage of this alternative is that it does not depend on any given propagation process. At any given time, the procedure could be triggered in order to verify replica consistency.

Other strategies to maintain consistency assume, at a starting point, that a replicated table is already updated/repaired. At the master site, these strategies keep track of the updated records in a propagation set. At a transfer time point, the master site sends its propagation set to the slave site(s). Upon receiving the propagation set, the freshness condition may be updated taking the previous consistent state into account. The common problem with all proposed solutions based on the general approach outlined above is that they assume an initial consistent state—or, at the very least, a periodic one if the strategy includes updating/repairing the slave replicated tables. If the system loses track of these cyclic evaluations, for example, because of networking delay issues, then the accuracy and therefore reliability of the freshness evaluation may be compromised. To return to a consistent state, the table has to be repaired by other methods that must match both master and slave replicas. In other words, these strategies are not self-contained. A more detailed discussion and a number of references on specific approaches along the general lines above are provided in Sect. 5.

3.1 Overview of proposed set reconciliation approach (SRA)

We start by determining *the maximum size of the symmetric difference between any two replicas*. Since in our asymptotic cost analysis we will actually use that upper bound value (due to the exact size of the symmetric difference set not always being known), we introduce the following definition.

Definition 7 (Maximum size of the symmetric difference) We denote the maximum size of the symmetric difference between two replicas by Δ. In order to minimize new notation, we also denote an upper bound on the possible size of the symmetric difference by Δ.

A given site, c, shares with each of $N-1$ other sites Δ evaluation points that will be used to "guess" the pairwise symmetric differences between its own replica and a replica at each of the $N-1$ other sites. Once the symmetric differences have been determined for each site, we locally compute at site c Eqs. (6) and (7) as follows:

$$\mathcal{T}_{p\cup} = \bigcup_{i=0}^{N-1} D_i.T_p = D_c.T_p \cup \left(\bigcup_{\substack{i=0\\ i\neq c}}^{N-1} (D_i.T_p - D_c.T_p)\right) \tag{6}$$

and

$$\mathcal{T}_{p\cap} = \bigcap_{i=0}^{N-1} D_i.T_p = D_c.T_p - \left(\bigcup_{\substack{i=0\\ i\neq c}}^{N-1} (D_c.T_p - D_i.T_p)\right). \tag{7}$$

After these computations have been completed, we will have all the elements at site c that are needed to evaluate Eq. (3). To compute our metric grcom() according to Eq. (5), we will use the Δ evaluation points with $\pi_K(\mathcal{T}_{r\cup})$ to compute the values in $\pi_K(\mathcal{T}_{r\cup})$ that are not in $\pi_K(\mathcal{T}_{s\cup})$, as if both projections were replicas. The challenge here is to "guess" the pairwise symmetric differences between sets. However, that can be done with a low communication complexity since we are assuming that δ and therefore also Δ, is considerably smaller than the sizes of the replicas (i.e., full tables at each site), that is, much smaller than $|\pi_K(\mathcal{T}_{r\cup})|$. We provide an example of how to apply our replica reconciliation scheme to a real-world distributed database in the next section.

Evaluating Eq. (1) is challenging in practice, due to the size of the replicas in a typical distributed database. In our approach, we assume a *lazy master replication model* [13]. We observe that, at each slave site, the most important (and most expensive) computation is the symmetric difference. Our techniques focus on optimizing this computation. We realistically assume that the cost to transfer a large table is high, and we additionally assume that the expected sizes of the symmetric differences are relatively low compared to the sizes of the involved tables. We make a general assumption that in the primary replica records there may be insertions, deletions and updates.

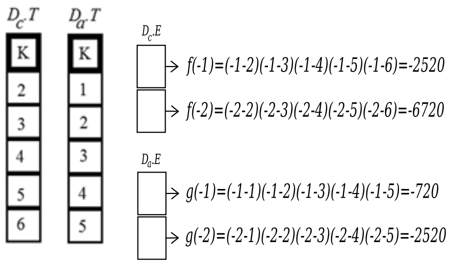
The question that remains is how we can efficiently compute the symmetric difference (Eq. (1)) at a given slave site, without transferring or broadcasting large amounts of data and without assuming a consistent initial state. That is, how can we compute at a remote site the differences $D_0.T_p - D_i.T_p$ or $D_i.T_p - D_0.T_p$, yet without transferring the entire primary replica from the master site? To accomplish this objective, we adapt techniques used to reconcile sets whose differences are small [21], that avoid transferring large amounts of data. In essence, we ensure that the amount of data transferred depends on the number of differences between the tables at different sites, rather than on the size of the involved tables themselves. That is, the communication complexity is $O(\delta)$, following Definition 5 instead of $O(|D_0.T_p|)$. Throughout the article we assume $O(\delta) = O(\Delta)$. This is a great advantage, considering that in a distributed database context, in most scenarios we are aware of, $\delta \ll |D_0.T_p|$. On the other hand, once the data has been transferred, the computational complexity of the algorithms that compute the actual different values is at most cubic in the number of differences, $O(\delta^3)$. Determining the replica consistency of tables in a distributed database where we deal with big sets with proportionally small differences is the general scenario that is particularly suitable for our techniques. Once Eq. (1) has been evaluated, replica consistency can be assessed and then the user can decide either at which site a query should be computed or simply refresh the replica.

3.2 Repairing replica consistency

To compute replica consistency, we adapt the set reconciliation techniques originally proposed in [21]. The basic assumptions and steps of the set reconciliation process are described below. We use the Master-Slave terminology below for simplicity, and remark that the discussion applies to Multi-Master architectures, as well.

- The sets to be reconciled (the records of two replicas) are represented by their *characteristic polynomials*.
- A number of evaluation points that are mutually agreed, in our case, between the master site and the slave site, are used to evaluate the characteristic polynomials. The number of points used for polynomial evaluation must not be less than the maximum number of differences between the two replicas, Δ (Definition 7).
- Instead of transferring one of the reconciling sets from one site to the other, we transfer the evaluation points with their corresponding characteristic polynomial evaluations, thereby considerably reducing the communication complexity, $O(\delta)$.
- The two sets of characteristic polynomial evaluations are used to interpolate a rational function whose numerator and denominator, once the common factors have been canceled out, will have for their roots precisely the symmetric difference between the sets of records of the two replicas (read below).

For a rigorous mathematical treatise on evaluating polynomials and rational functions over finite fields, we refer the reader to [19]. On how to efficiently compute



$D_c.T$
$D_a.T$
K
2
3
4
5
6
K
1
2
3
4
5
$D_c.E$
f(-1)=(-1-2)(-1-3)(-1-4)(-1-5)(-1-6)=-2520
f(-2)=(-2-2)(-2-3)(-2-4)(-2-5)(-2-6)=-6720
$D_a.E$
g(-1)=(-1-1)(-1-2)(-1-3)(-1-4)(-1-5)=-720
g(-2)=(-2-1)(-2-2)(-2-3)(-2-4)(-2-5)=-2520

Fig. 1 Primary replica at site c, an inconsistent replica at site a and computation of evaluation point vectors at each site

characteristic polynomials, we refer the reader to numerical linear algebra books such as [12].

The following two examples give an overview of how the technique outlined above can be used to update/repair a table with respect to replica consistency. We remark that it is not our intention to go into all the mathematical details of the set reconciliation algorithm in this paper. (We refer the interested reader to [21] for those details.)

Example 1 Let us assume we have two replicas as in Fig. 1, the primary replica at site c and the secondary one at site a, with a unique attribute, the primary key K. Assume further that we want to determine the symmetric difference between the two replicas without transferring a whole set of numbers from one site to the other.

Let the characteristic polynomial of the sets at each site be:

– site c: $f(x) = (x-2)(x-3)(x-4)(x-5)(x-6)$
– site a: $g(x) = (x-1)(x-2)(x-3)(x-4)(x-5)$

The roots of the characteristic polynomials $f(x)$ and $g(x)$ are the elements of the corresponding sets.

Observe that

$$h(x) = \frac{f(x)}{g(x)} = \frac{(x-2)(x-3)(x-4)(x-5)(x-6)}{(x-1)(x-2)(x-3)(x-4)(x-5)} = \frac{(x-6)}{(x-1)} \tag{8}$$

The common factors of the rational function $h(x) = \frac{f(x)}{g(x)}$ cancel out and the roots of the numerator and denominator are the symmetric difference between the two sets. The fundamental question here is, how can we compute efficiently the symmetric difference in a distributed database? The following procedure summarizes our solution to this problem.

- Execute asynchronously at each site the following steps:
  - Master and slave sites agree on the maximum size of the symmetric difference, in our Example 2.
  - Both sites agree on the same set of 'evaluation points': $-1, -2$.
  - At each site, evaluate the respective characteristic polynomial at the selected integer values, to obtain the evaluation point vectors $D_c.E$ and $D_a.E$ as in Fig. 1:
- To compute the symmetric difference at one site, say site a, we proceed as follows.
  - Transfer the evaluation point vector $D_c.E$ and the table cardinality to site a.
  - In our example only three numbers are transferred: the evaluation point vector, $(-2520, -6720)$, and the cardinality, 5. Observe that the communication complexity depends on the size of the symmetric difference. Even if one or both of

the original tables were much larger, the size of the transferred data would still remain the same.

– At site a, evaluate $h(-1)$ and $h(-2)$:

$$\begin{aligned} h(x) &\equiv \frac{f(x)}{g(x)} \\ h(-1) &= \frac{-2520}{-720} = 3.50 \\ h(-2) &= \frac{-6720}{-2520} = 2.66 \end{aligned} \tag{9}$$

– We note that the difference between the two cardinalities (in our case, $5 - 5 = 0$) gives us the difference of the degrees of $f(x)$ and $g(x)$, which in this particular example is zero.
– With that information on the difference between the degrees of $f(x)$ and $g(x)$, we can infer the form of the two characteristic polynomials; in this particular example, both characteristic polynomials are of the same degree 1:

$$f(x) = x + a, \qquad g(x) = x + b.$$

– We now solve the following system of equations in two variables:

$$\begin{aligned} -1 + a &= 3.50(-1 + b) \\ -2 + a &= 2.66(-2 + b) \end{aligned} \tag{10}$$

– Now we have: $a = -6$ and $b = -1$, meaning that 6 is the element in the replica at site c which is missing from the replica at site a, whereas 1 is the element in the replica at site a that is missing from the replica at site c.

In Example 2 below, the practical difficulties due to having to work with large integers in the computation of the evaluation point vectors are resolved by evaluating polynomials over a finite field.

Example 2 Suppose now we have a table with 100 rows replicated at two sites, site c for the master replica and site a for the secondary replica. As before, let us assume a unique attribute, the primary key K, which holds 8-bit signed integer values. Suppose we know that the number of elements in the symmetric difference between the two sets of primary key values is bounded by 20. To reconcile the two replicas, first both sites agree on 20 evaluation points. At each site, the *characteristic polynomial* [19] is evaluated at each of the agreed-upon 20 evaluation points. Let k_i, $i = 1, \dots, 100$ be the primary key values of the replica at site c. The corresponding characteristic polynomial would then be $\prod_{i=1}^{100}(x - k_i)$.

Let k_e be one of the 20 evaluation points. The evaluation of the characteristic polynomial at k_e is $\prod_{i=1}^{100}(k_e - k_i)$. To avoid working with very large integers when computing the evaluations of the characteristic polynomials, we need to work over a finite field of a size not less than $v = (2^7 - 1) + 20 = 147$, in order to be able to uniquely map the primary key values and the evaluation points. (Notice that 2^7 is the largest possible absolute value of an integer that can be represented as a signed 8-bit "computer integer", and that the additive factor of 20 is there to take into account for the 20 evaluation points; for algebraic details, see [21].) The selected field could be then $\mathbf{F}_q$, with $q = 149$, since $q = 149$ is the smallest prime greater than v.

We observe that one can adjust the value of q in a sense that a different prime that is greater than 149 would also work, since the domain of the primary key values would not change, and neither would the upper bound on the number of elements in the symmetric difference.[1]

The 20 mutually agreed evaluation points can be selected, for example, to be integers in the closed interval $[-20, -1]$. Since these numbers are not members of the set of possible primary key values, they will never be zeroes of the corresponding characteristic polynomials. Both sites keep their evaluation point vector with 20 pairs of values containing each evaluation point associated to its corresponding evaluation of the characteristic polynomial. Also, each site keeps the cardinality of its replica. To keep the vector updated, whenever a record with a primary key value of for example k_0 is inserted/deleted, all the 20 current evaluations of the characteristic polynomial have to be multiplied/divided (mod q) by $(k_e - k_0)$, where k_e is an integer in the interval $[-20, -1]$. The cardinality is updated as well, by adding or subtracting 1 to/from the current cardinality.

Suppose that, at a particular time, both replicas agree on all their primary key values, and suppose further that these values are positive integers in the closed interval $[1, 100]$. In both evaluation point vectors, the value that corresponds to the evaluation point -1 is

$$110 = (-1 - 1)(-1 - 2) \cdots (-1 - 100) \bmod q$$

where $q = 149$

Suppose site c receives three records with values 101, 102, 103 in its primary key K and the record with value 100 in K is deleted. The updated evaluations of the characteristic polynomial that correspond to the evaluation points -1 and -2 are 15 and 129, respectively. When the freshness evaluation is required at site a, site c then transfers its evaluation point vector to site a, as well as the cardinality of its replica. We again emphasize that the size of transferred data depends on the size of the symmetric difference and not on the table cardinality itself. At site a, at each evaluation point, the two values of the evaluation of the characteristic polynomials are divided (mod q). Then the results of those divisions are used together with the cardinality of the functions to interpolate a reduced rational function resulting from the aforementioned polynomial division. This reduced rational function will have in its numerator and denominator the characteristic polynomials whose roots are precisely the values that are in one set (i.e., replica) but not in the other—that is, precisely those values that are in the symmetric set difference.

[1]In the real-world example, where the actual finite field sizes are many orders of magnitude greater than in our toy-size example above, there are practical advantages to picking a prime q that is close to, i.e., only slightly greater than, the applicable value of v. So, ideally, one would want to pick the smallest prime strictly greater than v, but *any* prime greater than v would mathematically still work out just fine.

Take for example the values of the two evaluation vectors at the evaluation point -1 above. The division between the two values gives (mod q)

$$\frac{15}{110} = \frac{(-1-1)(-1-2)\cdots(-1-99)(-1-101)(-1-102)(-1-103)}{(-1-1)(-1-2)\cdots(-1-99)(-1-100)}$$
$$= \frac{(-1-101)(-1-102)(-1-103)}{(-1-100)} = \frac{-7}{-101} = 34$$

As described above, the common factors of the characteristic polynomials in the numerator and the denominator cancel out, so the resulting rational function has roots of its numerator and denominator that are precisely the values that are in one replica but not in the other.

Next, in Algorithm 1, we summarize how to use the set reconciliation techniques described and exemplified above in order to implement the set reconciliation approach in the distributed database context. We emphasize that, while our illustrative examples in this section and real-world examples in Sect. 4, for simplicity, include just a master site and a single slave site ($N = 2$), clearly the proposed approach and the algorithm below apply to scenarios with multiple slave sites ($N > 2$).

Algorithm 1 Algorithm for set reconciliation approach (c—master site; a—remote slave site; pt—processing time; ct—communication time)

```
0. (Asynchronously): All sites agree on the maximum size of the symmetric
    difference between any two replicas, $\Delta$. All sites also agree on $\Delta$
    evaluation points for their characteristic polynomials. At each site,
    the characteristic polynomial is formed with the values of the attributes in
    table $T$ that are updated in another table, $E$, at each of the $\Delta$
   evaluation points whenever an insert takes place.
   Table $E$ is the evaluation point set. The cardinality $|T|$ is also updated.
1. (ct): Site c broadcasts $|D_c.T|$ and $D_c.E$.
   Table $D_c.E$ has the characteristic polynomial of $D_c.T$ evaluated at
   each of the $\Delta$ evaluation points as explained above.
2. (pt): Site a receives $|D_c.T|$ and $D_c.E$ and together with $|D_a.T|$ and $D_a.E$
   interpolates a reduced rational function, computes $D_a.T \ominus D_c.T$
   and generates tables $(D_a.T - D_c.T)$ and $(D_c.T - D_a.T)$.
   (Note: These are the repairing tables in the Master-Slave architecture.)
3. (ct): Site a transfers to site c tables $(D_a.T - D_c.T)$ and $(D_c.T - D_a.T)$.
4. (pt): Site c receives the results from the remote sites, and computes
   the consistency metrics cur() and gcur().
```

When the maximum number of different values between any two replicas Δ is not known so that we only work with a global bound, the cardinality of the tables together with that global bound can be used to determine upper bounds on the degrees of both characteristic polynomials in order to interpolate and reduce the resulting rational

Table 1 Lineitem example table

pk	*l_quantity*	*l_shipdate*	*l_shipmode*
1	17	1996-03-13	TRUCK
2	36	1996-04-12	MAIL
3	8	1996-01-29	REG AIR
4	10	1996-05-05	AIR
7	38	1997-01-28	RAIL

function. By finding the zeroes of the characteristic polynomials, the symmetric difference between both replicas can be determined at site a. When the type of the key is not integer, the key domain values are mapped to values in a finite field. This can be done by utilizing the key value's binary representation or another similar integer-based representation. If necessary, the tables can be vertically fragmented keeping in each fragment the primary key in order to rebuild the records that belong to the symmetric difference.

Example 3 The following example shows the transformation of rows in the TPC-H [33] *lineitem* table and the evaluation of the characteristic polynomial at five evaluation points, which are the required pre-processing steps needed to apply the proposed set reconciliation techniques. The example is shown on Table 1.

The table with equivalent strings and the mapping of these rows to an integer-based representation table is shown below. To enhance readability, the integer representation of each string is shown as 25 groups of four digits each, where each four-digit group represents one symbol).

117.001996-03-13TRUCK
1 0049 0055 0046 0048 0048 0049 0057 0057 0054 0045 0048 0051 0045 0049···
0051 0084 0082 0085 0067 0075 0032 0032 0032 0032 0032

236.001996-04-12MAIL
20051 0054 0046 0048 0048 0049 0057 0057 0054 0045 0048 0052 0045 0049···
0050 0077 0065 0073 0076 0032 0032 0032 0032 0032 0032

38.001996-01-29REG AIR
30056 0046 0048 0048 0049 0057 0057 0054 0045 0048 0049 0045 0050 0057···
0082 0069 0071 0032 0065 0073 0082 0032 0032 0032

410.001996-05-05AIR
40049 0048 0046 0048 0048 0049 0057 0057 0054 0045 0048 0053 0045 0048···
0053 0065 0073 0082 0032 0032 0032 0032 0032 0032 0032

738.001997-01-28RAIL
70051 0056 0046 0048 0048 0049 0057 0057 0055 0045 0048 0049 0045 0050···
0056 0082 0065 0073 0076 0032 0032 0032 0032 0032 0032

Since the resulting representations are 100-digit integers, we choose a finite field of order shown in the following long integer, which is a prime number of length 110:

7692142110676012528555092924090335496637043182779271492008601148810$\cdots$
039520949691757314599081173759953492458393430

Suppose $\delta = 5$ and evaluation points: $\{-1, -2, -3, -4, -5\}$. Next, we show the evaluation of the characteristic polynomial at each of the evaluation points:

-1:2811853750720991592377042877384902756115272220386130734419496145678$\cdots$
905620077698955743585960336104741510162292

-2:7490612439450685798631958486627572000212519188372346652812208940739$\cdots$
458857161995332415609376352788006326628910

-3:7156107320132739951435015762982733352442465815725075783252161564518$\cdots$
196517993512362782414696137156028076864678

-4:6095918023711878366925701585282502870967982352817409353595692477772$\cdots$
154516667095104636573814634795492794057993

-5:9054817339695482671678338506627804332464893627031094935939989334015$\cdots$
64621695701825945411170229189513182972897

We additionally observe that, in a Master-Slave configuration, after the completion of Step 2 at the slave site a, $D_c.T_p - D_a.T_p$ is the set of records that have not yet been propagated from the master site c to the slave site a. That is, this set difference corresponds to the records inserted into $D_c.T_p$ that are not yet in $D_a.T_p$. As for the set difference $D_a.T_p - D_c.T_p$, this is the set of records not yet deleted from $D_a.T_p$. Upon the completion of Step 2, the slave site a is in the position to locally update/repair replica $D_a.T_p$.

3.3 Measuring replica consistency based on foreign keys with a skewed distribution

We now present an extended algorithm to summarize insertions that complements our previous algorithm when tables cannot be repaired (reconciled). This algorithm measures replica consistency with a data summary based on foreign keys. The algorithm is applicable when foreign keys have a high cardinality and a skewed distribution. That is, most insertions happen in a small set of foreign key values. In this section, we will refer to the tables that cannot be repaired (reconciled) as the main tables.

The idea behind this variation is that all sites maintain a frequency table with values of the foreign key and a column with the number of records for each foreign key value (i.e. a count(*) in SQL) that were inserted into the main tables.

Let $\pi_{K,\text{count}(*)}(T)$ be an extended relational algebra notation to denote the frequency table with values of the foreign key K and the number of rows for each foreign key value that are inserted into the main table T or in SQL

```
SELECT T.K, count(T.K)
FROM T
GROUP BY T.K.
```

To simplify exposition the foreign key K is not allowed to have nulls.

In contrast to the algorithm presented before, the evaluation point vectors are evaluated with the records of the frequency tables and not with the records in the main tables. An insertion in the main table will cause an update in the frequency table. Specifically, there will be an update in the row that corresponds to the foreign key value of the inserted record in the main table to increment the corresponding count by one. An evaluation point vector of this frequency table must be maintained. On the other hand, an update in the frequency table is handled as two elementary operations: a deletion and an insertion of a row, as explained in Sect. 2. We assume that the foreign key values may be distributed accodding to a skewed distribution such as, for example, geometric or Zipf's distributions, we expect the symmetrical differences among the frequency tables to remain low even though the actual differences among the main tables could be large. It is feasible that new records could be inserted in the frequency tables, since records with new foreign key values could be inserted into the main tables. But, again, since we assume foreign keys with a skewed distribution, the symmetrical differences among the frequency tables is expected to remain low.

When the freshness condition of the main tables is requested, we evaluate the symmetrical difference of the frequency tables. This is done as we proceed in the previous section using the evaluation point vectors, but now, the evaluation point vectors correspond to the frequency tables. Algorithm 2 summarizes the steps to measure replica consistency based on the set reconciliation approach in the distributed database con-

Algorithm 2 Algorithm for measuring replica consistency based on foreign keys with a skewed distribution using SRA (c—master site; a—remote slave site; pt—processing time; ct—communication time)

```
0. (Asynchronously): At each site, compute frequency table $\pi_{K,\text{count}(*)}(T)$.
   All sites agree on the maximum size of the symmetric difference between
   any two instances of $\pi_{K,\text{count}(*)}(T)$, $\Delta$. All sites also agree on $\Delta$
   evaluation points for the characteristic polynomials of the frequency
   tables. At each site, the characteristic polynomials is formed
   with the values of the attributes in table $\pi_{K,\text{count}(*)}(T)$ that are
   updated in another table, $E$, at each of the $\Delta$ evaluation points as a result
   of an insertion in the main table $T$, since an insertion will cause
   the corresponding count in the record of the $K$ value in
   $\pi_{K,\text{count}(*)}(T)$ to increase by 1. Table $E$ is the evaluation point set.
   The cardinality $|\pi_{K,\text{count}(*)}(T)|$ is also updated
   and it will be incremented when it corresponds. That is,
   when a record with a new value of $K$ is inserted in the main table.
1. Use Algorithm 1 to compute $\pi_{K,\text{count}(*)}(D_a.T) \ominus \pi_{K,\text{count}(*)}(D_c.T)$
   and generate tables $(\pi_{K,\text{count}(*)}(D_a.T) - \pi_{K,\text{count}(*)}(D_c.T))$
   and $(\pi_{K,\text{count}(*)}(D_c.T) - \pi_{K,\text{count}(*)}(D_a.T))$.
2. (pt): Site c receives the results from the remote sites. Taking into account
   there are only inserts, computing the symmetrical differences will show,
   for each foreign key value, in how many records both main tables, $D_a.T$
   and $D_c.T$, differ. This is done, once the symmetrical differences are known,
   by computing the differences between the counts of each foreign key
   values and/or with the counts of the new foreign key values.
```

text considering a skew foreign key scenario. We again emphasize that to simplify exposition, we include only a master site and a single slave site ($N = 2$).

With these results, the user can have a good idea about the size of the symmetrical differences among the main tables. The foreign key values frequencies will provide useful information to decide if a reconciliation is needed or distributed processing can continue. In Sect. 4 we present experimental evaluation of this summary reconciliation algorithm considering skewed discrete distributions.

3.4 Measuring and repairing distributed referential integrity

To compute our metric grcom() according to Eq. (5) we need to obtain the number of invalid references in the referencing global table. Once the involved global tables are computed, namely, $\mathcal{T}_{s\cup}$ as the referenced table and $\mathcal{T}_{r\cup}$ as the referencing table, if the two tables are at different sites, we proceed as follows. Let site r be the site with the referencing global table, and site s the one with the referenced global table. Suppose that the maximum number of differences between $\pi_K(\mathcal{T}_{r\cup})$ and $\pi_K(\mathcal{T}_{s\cup})$, which are the table with the foreign key values of the referencing global table and the table with the primary key values of the referenced global table, respectively, is less than an upper bound, Δ. (The case when even an upper bound on the maximum size of pairwise set differences between replicas is not known will be addressed in Sect. 3.6.) If Δ is equal to the upper bound used to evaluate T_s metrics using Algorithm 1, we can use the same evaluation points. Otherwise, an additional evaluation point vector should be computed for each replica used to create the global referenced table, and we then proceed using this new table instead.

At site s, we evaluate a temporal evaluation point set of the table $\pi_K(\mathcal{T}_{s\cup})$. As for the referencing global table $\mathcal{T}_{r\cup}$ at site r, the evaluation point vector for $\pi_K(\mathcal{T}_{r\cup})$ has to be computed “from scratch”. A frequency table is built with the values of the foreign key K of the global table $\mathcal{T}_{r\cup}$. The idea is to maintain the number of rows for each foreign key value. We again evaluate the corresponding evaluation point vector at the Δ evaluation points. At tater stages of the process, the frequency table will be used to compute the number of invalid rows. Site s transfers the computed evaluation point vector to site r where the table $\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})$ is computed with the invalid values (see Eq. (5)). Using the frequency table mentioned above, the number of invalid references can be obtained.

To repair referential integrity, we proceed as follows. We first note that in Algorithm 1 the tables $(D_a.T - D_c.T)$ and $(D_c.T - D_a.T)$ are generated on the per-replica basis. Hence, the tagging procedure described below can be applied. We observe that, by computing $(\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup}))$, we obtain the invalid values. We assume that the valid referenced values are in $\pi_K(\mathcal{T}_{s\cup})$. We further assume a partial replication approach, namely, that in general not all tables are replicated and not at all the sites. Other scenarios are less complex and can be addressed in a simpler manner than the more general scenario.

The general idea is to update the invalid foreign key values with an “undefined” or “not available” value (e.g., “NA”), and insert in the corresponding referenced tables an “undefined” row (one for each foreign key) [11]. We need to trace from which sites have the invalid values been received. Let us assume that c is the site that is receiving the differences of table T_r from replicas like the one at site a. To accomplish the aforementioned tracing, the transferred tables received while building $\mathcal{T}_{r\cup}$, that is, $(D_a.T_r - D_c.T_r)$ and $(D_c.T_r - D_a.T_r)$, have to be tagged with the replica's site identification, e.g., 'a'. We note that one value could have come from several sites; consequently, in general, it could have multiple tags. This can be accomplished by transferring from each site not only the values of the symmetric difference, but also the site identification from where these symmetric difference values come from. If an invalid value is not at site c, i.e., it is in $\pi_K(D_a.T_r - D_c.T_r)$, then we need to update that value in the corresponding T_r table. Alternatively, if the invalid value is in $\pi_K(D_c.T_r)$, then we need to update that value (i) in $D_c.T_r$ and (ii) at all other sites where T_r is present and has that value in attribute K.

We notice that we may have the tables where some values are in replica at site c but not in replicas at other sites. That is, we may have tables such as $\pi_K(D_c.T_r - D_a.T_r)$. We additionally observe that, if an invalid value is in one of these latter tables, then the value is in the replica site c and in replicas at all the sites where the table T_r is replicated, with the exception of those sites that are associated with the invalid value in these tables. We also notice that, if the value is not found in any of the two types of tables, that is, neither in $\pi_K(D_a.T_r - D_c.T_r)$ nor in $\pi_K(D_c.T_r - D_a.T_r)$, then the invalid value is in the replicas at all sites where T_r is replicated, including the master site c.

3.5 Communication and processing complexity

The communication complexity using the set reconciliation techniques just described depends on the number of differences among the tables involved and the size of the records, $O(\delta \cdot b)$ (see *Step 1* of Algorithm 1). We use the term processing complexity instead of computational complexity to avoid confusion with communication complexity. At the slave site(s), the time required (i) to interpolate the rational function using the evaluation point vectors by Gaussian elimination, and then (ii) to reduce the obtained rational function by applying Euclid's algorithm, have the overall processing complexity of $O(\delta^3)$ [21].

We are assuming that the tasks described in Step 0 are performed asynchronously at each site. The main reason behind this assumption is that different sites of a distributed database may have different CPU "horse powers" and other computational resources available to them, and consequently, in particular, may perform local computations at very different speeds. Furthermore, the results of these local computations need to be aggregated at a single site, and the communication time to transfer the results from one site to another will depend on the speed of the available communication network. With these constraints in mind, we summarize the asymptotic complexity of the local computations at each site that are necessary to complete Step 0. We recall that a characteristic polynomial of degree k that is in the product-of-monomials form (i.e., in the form $(x - a_1) \cdot (x - a_2) \cdots (x - a_k)$) can be evaluated using k subtractions and $k - 1$ multiplications, that is, in time $O(k)$. Since we have assumed the upper bound Δ on the maximum number of elements in the symmetric set difference between any two replicas, the degrees of characteristic polynomials at all sites are upper-bounded by Δ. Therefore, at each site, evaluating the local characteristic polynomial can be done in $O(\delta)$ elementary steps. Since *Step 0* also includes scanning the table T, the overall complexity of the local computations for this step at each site is bounded by $O(n \cdot \delta)$, where the table T is assumed to have n rows.

Next, in order to compute grcom() according to Eq. (5), we need a frequency table with the number of rows per value for each foreign key to be measured. That table can be computed by several means. One possibility is to do it via a simple scan of the global referencing table if the table containing the different values and their frequencies can be stored in memory or sorting a projection of the referencing global table containing its primary key and its foreign key and then computing the needed table. In the worst case, the cost is $O(n \cdot \log(n))$, where n is the size of the referencing global table. We note that the frequency tables and the evaluation point vectors can be evaluated concurrently. We additionally note that, in order to compute our metric, we can use the frequency table instead of the referencing global table, as the former can be expected to be of smaller size.

3.6 Dealing with unknown upper bound on symmetric difference set size

There are cases where an upper bound of the maximum number of records of the symmetric difference between two arbitrary replicas, Δ, cannot be determined. We next present an overview of a method to apply the set reconciliation techniques described earlier in the paper that enables us to obtain the symmetric difference between two replicas without knowledge of an upper bound on the size of that symmetric difference. This method is an adaptation of the probabilistic verification technique presented in [21].

We estimate the maximum size of the symmetric difference between any two replicas of a given table T above which it would be unfeasible to use the SRA approach. We observe that the actual number of values in the symmetric set difference could be greater than this “maximum size of SRA feasibility number”. This bound can be obtained by taking into account how much time can take the computations to obtain the rational function described in Sect. 3.1 and its roots for a certain number of pairwise symmetric differences. Let us call that number the *Replica reconciliation threshold* (d_T).

Definition 8 (Replica reconciliation threshold) The Replica reconciliation threshold is the maximum size (number of rows) of the symmetric difference between any two replicas of a given table T a user can tolerate, above which using the SRA approach to determine the symmetric difference would be infeasible. We denote this number by d_T, where T is the name of the table.

This number is a user-defined threshold and is defined in a given situation according to the user’s needs and resources. It depends on factors such as record size, network speed, time to compute the roots of the rational function. The replica reconciliation threshold could be the maximum number of records of the symmetric difference between any two replicas, Δ, if this number can be determined and is tolerable according to Definition 8.

We determine a first set of evaluation points mutually agreed between the involved sites, say c and a, with d_T number of points. Let us call this set Q. Next, we need to determine the number of additional evaluation points that will be used to test if, with a probability greater than or equal to the specified threshold probability, an interpolated rational function is indeed the function we are looking for. To accomplish this, we specify a low probability value, $p > 0$, as the maximum probability we can tolerate for erroneously interpolating the desired rational function that captures the symmetric set difference between the replicas at sites a and c. According to this probability, we determine a second set of mutually agreed points that are drawn randomly from a subset of the finite field over which the rational function we are trying to determine is defined. We will use these random evaluation points to determine if the computed rational function is indeed the desired one with the specified probability of $1 - p$ or higher. Let us call this set S. Now our evaluation point vector, the one that corresponds to table E in the discussion of the SRA method presented in Algorithm 1, will hold the evaluations of the characteristic polynomials of Q and S.

When the freshness evaluation is required, that is, when we require to compute metric cur(), then site a (the slave site) receives the evaluation point vector of its counterpart, site c (the master site), with $|Q| + |S|$ evaluation points and its cardinality, and computes the corresponding polynomial divisions. Next, the interpolation of the rational function takes place, assuming the number of rows in the symmetric difference is bounded by $|Q|$. We then test to determine if this function is in fact the rational function we want with a probability of failure of at most p. This is done by comparing the divisions that correspond to the points in S against the values of the interpolated rational function evaluated at those same points. If the test is successful, then the current interpolated function is reduced and its roots are obtained. Otherwise, if at least one point fails, then we decide that the SRA method is not feasible to determine the differences. Each time the set S of random points is used for this purpose, the sites involved compute another mutually agreed random set. We note that this can be done asynchronously. Also, since usually in practice S can be expected to be small, several sets can be maintained simultaneously. Since the interpolation is done only once, and $|S|$ is fixed, the computational complexity at site a is $O(|Q|^3)$ and the communication complexity is $O(|S| + |Q|)$. Bearing in mind this complexity, we can better estimate the size of Q.

What remains to be explained is how to compute the size of S from p. We note that, if the values of a key can be mapped to a field of integers representable by b bits, then we can add an additional bit to enlarge the field and we have 2^b more values of length of b bits, from which we can pick the values of Q and S. Since Q is a fixed set of values, the random values will be chosen from a set of $2^b - |Q|$ possible values in total. According to Theorem 4 in [21], and considering the finite field described above, the probability that two *different* monic rational functions which have the sum of their numerator and denominator degrees less than B agree in one randomly selected point (although the two functions are different) is bounded above by

$$\overline{p} = (B - 1)/\bigl(2^b - |Q|\bigr) \tag{11}$$

We recall that B is an upper bound on the maximum number of different values between the two replicas, i.e., $B = |D_c.T| + |D_a.T|$.

Since the rational function is obtained by the division of the characteristic polynomials, we know from algebra that two different rational functions cannot agree on more than the number of points equal to the size of the symmetric difference of the involved sets minus one. Moreover, if two rational functions agree on more than this number of points, then they must be equivalent, that is, they are the same function up to a multiplicative constant.

With these properties of polynomials and rational functions over finite fields in mind, we can determine an upper bound on the probability that two arbitrary rational functions that agree at e randomly selected evaluation points are actually different functions: that probability is no more than $B\overline{p}^e$. Let that probability be denoted by p; then the size of S is

$$|S| = \lceil \log_{\overline{p}}(p/B) \rceil \geq \lceil \log_{\overline{p}}(p/\Delta) \rceil \tag{12}$$

where, as before, Δ is the maximum number of different values between the two replicas.

3.7 Advantages and limitations

Our approach offers a compromise among the following requirements applied to large distributed databases: replica freshness, computing OLAP queries efficiently, communication efficiency and self containment in the computation of replica consistency. The user of a distributed database that wishes to measure replica consistency (Definition 3) of a replica of table T, only needs to determine the threshold d_T (Definition 8), the maximum size of the admissible symmetric difference between the master and slave replicas. This value is transmitted to the sites where the master and slave replicas reside in order to determine the number of evaluation points, according to Algorithm 1. An important contribution of our work is a novel technique to measure replica consistency in a distributed database context with a communication complexity that depends only on the size of the symmetric difference between the corresponding replicas, and not the size of the tables themselves.

Efficient computations to compare distributed database or data warehouse tables have been proposed, for example, in the context of the snapshot differential problem [17]. However, these techniques in the distributed context have a communication complexity that depends on the size of the tables themselves, which is often prohibitively expensive in practice. In a Master-Slave configuration, our computations lead to a method to update/repair the slave replica in a manner that is independent of the propagation method used by the distributed database system. A scenario particularly suitable for the use of our techniques are distributed databases with very large tables/datasets, yet where the symmetric differences between replicas are relatively small. The computational complexity at the slave site using the set reconciliation techniques to obtain the symmetric difference between replicas, although cubic in the number of differences (which, as discussed earlier, is expected to be low), does not depend on the size of the tables themselves, which is a huge advantage. Furthermore, in Multi-Master configurations, metrics used to determine the global replica consistency (Definition 4, Eq. (3)) and global referential integrity (Eq. (5)) are also well-suited for the set reconciliation approach.

When it comes to addressing referential integrity, these techniques can be combined with other techniques, such as the ones presented in [11], in order to compute aggregation queries over distributed databases that may have referential integrity issues. Query tools for distributed databases generally assume underlying replicated tables are up to date and without referential integrity errors. Unfortunately, this is often not true. Namely, in a distributed database environment, consistency problems frequently arise [1, 7, 16]. Given that it may be acceptable in practice to evaluate queries with slightly outdated replicated tables [10], with our techniques one can efficiently compute the degree of obsolescence of replicated local tables and then, according to the results, decide whether to evaluate a query locally—thereby avoiding the transmission of large tables over the network. Otherwise, the query can be remotely evaluated less efficiently with the master copy of the tables. Evaluation of our metrics can be optionally associated to evaluations of distributed queries at only a modest increase in overall computational cost (under the assumptions discussed earlier), so that the user can assess the degree of consistency of the tables involved in her queries.

We recall that all arithmetic operations involved in computing the characteristic polynomials and obtaining the roots of the resulting rational function are done over a finite field in order to avoid working with large integers. In case that the concatenation of the binary representations of the attribute domain values results in integers that end up being too large, the table can be vertically fragmented. Each fragment should keep the primary key value in order to eventually assemble the records that belong to the symmetric difference. Each fragment is treated as a different table and the computations needed to obtain the characteristic polynomials and the roots of the resulting rational function can be done in parallel at the corresponding site. We observe that the computations to update the evaluations of the characteristic polynomials are done asynchronously, locally at each of the corresponding sites, thus reducing the need to communicate between different sites.

Last but not least, it is important to mention that, when it comes to evaluating our metrics, both of our approaches scale well with respect to the number of sites, since the computations are distributed across all the sites. Equations (6) and (7) allow that all involved sites involved participate in the computations of metrics. In the Master-Slave configuration, the last two steps of Algorithm 1 imply that every site sends two small tables to the central site, and the central site then computes two set unions. Thus, the computational load is balanced across the sites, and the communication between different sites is minimized. We acknowledge that, from a fault-tolerance standpoint, in this scenario the central site remains a single point of failure. We note, however, that other approaches found in the literature also suffer from this problem in the context of Master-Slave configurations; some of those approaches will be discussed in Sect. 5. The Multi-Master configurations, in contrast, do not suffer from this single point of failure problem.

4 Experimental evaluation

The main goal of our experimental evaluation is to compare our proposal to measure replica consistency based on the set reconciliation techniques which, in turn, are based on efficient evaluation of the appropriate characteristic polynomials, (*Set*

Reconciliation Approach—SRA). To evaluate our proposal, we compare it with an efficient hierarchical algorithm, *pg–comparator* [6] (*Hierarchical Approach*—HA), for comparing two remote tables based on the checksum computations, more specifically, on performing a reconciliation merge of partial checksums fetched hierarchically level-by-level. That algorithm, just like our proposed solution, is a viable approach if the expected differences among replicas are relatively small. In particular, the HA algorithm has a communication complexity of $O(\delta \cdot \log(n))$, where δ is the number of differences to be found and n is the replica size.

In our experiments, we investigate some significant differences between SRA and HA. Our following experiments are conducted with table sizes that highlight the efficiency of our SRA-based methods even in scenarios where the tables are not too large. We recall that based on the analytical evaluation presented above, the advantage of our approach can be expected to grow as the sizes of the actual replica tables grow with respect to the sizes of symmetric differences between replicas.

Hardware and software We conducted our experiments on the following two sets of database servers:

- Two servers with two Intel Core Duo CPU at 1.66 GHz with 2 GB of main memory and 160 GB on disk.
- Two clusters each one with three Sunfire X2200 M2 servers with AMD Opteron Quad-Core processors at 2.3 GHz with 4 GB of main memory and 160 GB of disk storage.

We identified each set of servers as SS and FS setups, respectively (which stand for slow server and fast server setup). We used two networks one with a transmission speed of 0.1 MB/s, and the other one with a transmission speed of 10 MB/s. We identified these two types of networks as SN and FN, respectively (for slow and fast networks). We installed Fedora 8 in all servers with gcc 3.6. We adapted the reconciliation functions available from [32]. We simulated a Master-Slave configuration with the relational DBMS PostgreSQL 8.2.

Synthetic and real databases Our synthetic databases were generated by the TPC-H DBGEN program, [33]. The tables we replicated were *customer*, *orders* and *lineitem* with scale factors 1, 5 and 10 and the following primary keys: *c_custkey*, *o_orderkey* and a compound primary key (*l_orderkey*, *l_linenumber*). We used a real-world database that came from an educational institution in Mexico. This distributed database had two replicas. The total number of rows was about 120 million. Each experiment was run five times; we then eliminated the fastest and slowest execution and reported the average of the remaining runs. In each experiment, we recorded separately the total elapsed time to execute our algorithms on the servers (processing time, *pt*) and the total time for the communication (communication time, *ct*). Execution times are shown in seconds.

4.1 Repairing replica consistency: comparing SRA vs HA

In this subsection, we compare the HA approach against the SRA approach. For the SRA approach, set reconciliation techniques are adapted for the purpose of replica

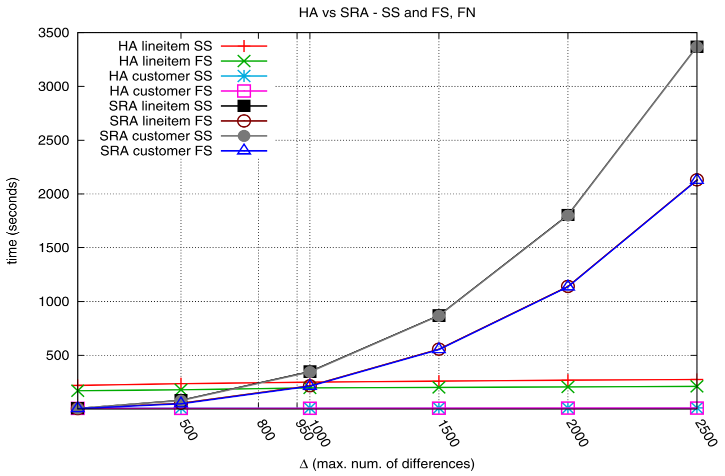


Fig. 2 Comparison between HA variant vs. SRA variant. Scale factor 1 (*lineitem* $n = 6M$, *customer* $n = 150{,}000$)

comparison. We transform each row to a string and then we map the string to a large integer using the class ZZ_p of the NTL library [30, 31]. We use this class to compute the evaluation points. To compute the roots of the polynomials, we adapt the set reconciliation techniques originally described in [21].

Figure 2 shows several executions of the HA and the SRA variants with the SS and FS servers and the FN network with tables *lineitem* and *customer*. We can see that the computational time of the SRA variant depends on the maximum number of differences between any two replicas, Δ. We also observe that, in case of SRA, the computational time is not affected by the size of the replica since the couple of lines that correspond to SRA *lineitem* FS and SRA *customer* FS and the couple that correspond to SRA *lineitem* SS and SRA *customer* SS overlap, meaning the performances are similar for each pair of cases. Similar behavior can be also observed in Fig. 3, where we contrast the performance of the SRA and HA algorithms run over three sizes of table *lineitem*, that is, scale factors 1, 5 and 10 equivalent to $n = 6M$, $30M$ and $60M$, respectively. For algorithm SRA, the performance remains constant.

One can observe how the execution time increases as Δ increases, which is to be expected. Since the evaluations of the characteristic polynomials at each evaluation point are done asynchronously, that is, right after the database operation taking advantage of having the record in memory, this time is not considered in this experiment (however, see the discussion in Sect. 3.5). We corroborate that the characteristic polynomial evaluation adds little to the overall execution time. An insertion in table *lineitem* scale factor 10, that is, $n = 60M$, took 11 ms. and the characteristic polynomial evaluation considering 2000 evaluation points took 8 ms. On the other hand, the computational time of the HA variant depends on the size of the tables. Since the sizes of the symmetric differences in the experiment are much smaller compared to the sizes of the tables, the computational time for each table is almost constant. We observe that the points where the lines that correspond to the SRA variant cross with the ones that correspond to the HA variant, indicate the transition point (as a function

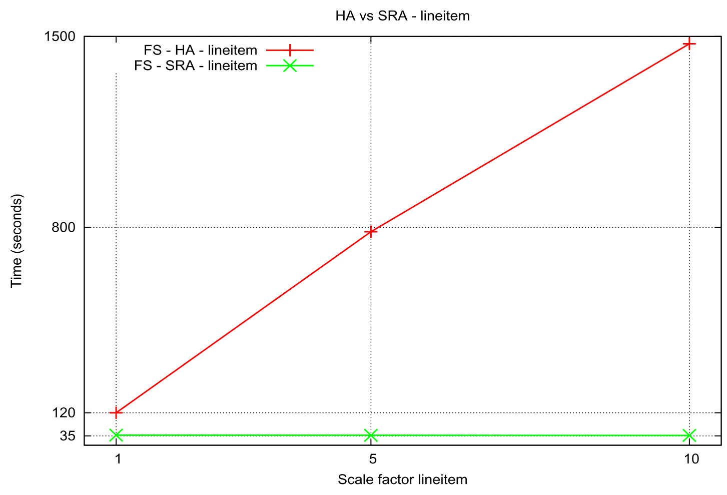


Fig. 3 Comparison between HA variant vs. SRA variant. Scale factors 1, 5 and 10 (*lineitem* $n = 6M$, $30M$ and $60M$ respectively)

Table 2 Computational and communication times of HA and SRA variants

SRA lineitem $6M$ and $12M$			HA lineitem $6M$		HA lineitem $12M$	
Δ	Com.	Roots	Com.	Comput.	Com.	Comput.
100	4.0	4	24.5	102.4	50.4	219.0
200	4.0	12	24.6	103.9	52.0	211.2
300	4.0	24	24.8	106.3	53.7	215.7
500	4.0	54	25.0	110.4	49.8	195.0
600	4.0	76	25.1	110.6	51.5	207.1
700	4.0	103	24.2	108.8	51.8	205.0
800	4.0	135	25.2	107.9	51.3	199.8
900	4.0	172	25.4	108.5	52.7	203.3
1, 000	**4.0**	**216**	**25.6**	**108.7**	**50.6**	**194.0**
1, 500	4.0	557	25.7	109.1	55.1	202.6
2, 000	4.0	1140	25.9	110.2	48.9	199.0
2, 500	4.0	2131	26.2	110.8	53.2	197.7

Com: communication; Roots: interpolation of reduced rational function

of Δ) where one variant becomes better than the other one. These experiments are useful to determine the replica reconciliation threshold (see Definition 8) in order to decide where and when a query should be evaluated.

Table 2 shows the processing and communication times of the SRA and the HA variants for several Δ values. The computations are done with replicas of table *lineitem* scale factor 1 and 2, with $6M$ and $12M$ rows, respectively, executed on the SS servers and the FN networks.

The goal is to show the specific execution times of various sub-tasks. The first two columns refer to the specific tasks of the SRA variant considering the evaluation of the characteristic polynomials is done asynchronously. The columns show the times for (i) the transfer of the evaluation point vector, (Com.), and (ii) the interpolation of the reduced rational function to obtain the roots of the numerator and denominator (Roots). We notice that since for the two table sizes, $6M$ and $12M$, the elapsed times were almost the same, hence we only show one column representing both sizes. As for the HA variant, the next two pairs of columns show the time to transfer one of the replicas (Com.), and the time to evaluate the *pg–comparator* algorithm according to each size of table *lineitem*.

We highlighted the approximate value of Δ which is the phase transition point that determines which approach is better to use. In particular, the highlighted line indicate that for $\Delta < 1{,}000$, the SRA variant is better than the HA variant.

We next briefly discuss Table 2, specifically, the communication times of the two variants. One can immediately see that the SRA variant has a much lower communication time. However, when it comes to processing times, in case of the SRA variant, computing the roots of a rational function becomes expensive as Δ grows.

4.2 Measuring replica consistency based on foreign keys with a skewed distribution

In Sect. 3.3 we introduced Algorithm 2 that computes the freshness condition of replicated tables by reconciling summary frequency tables of their foreign key values. In this section we present an experimental evaluation of this Algorithm.

For this experiment we simulated two sites, D_c and D_a, both with a replica of table *lineitem* scale factor 10, that is, $n = 60M$. Asynchronously, following Algorithm 2 we computed the aggregation query $\pi_{partkey,suppkey,count(*)}(lineitem)$ in both sites. Afterwards, we inserted approximately $10M$ rows in $D_c.lineitem$ where the values of foreign key (*partkey*, *suppkey*) followed different skewed geometric distributions, $(1-p)^{n-1}p$, for different p values.

The idea behind the experiment was to determine for a symmetrical difference of approximately $10M$ rows in *lineitem*, up to how many different foreign key values SRA surpassed HA as a better technique to determine how many insertions were done in *lineitem* using Algorithm 2.

Table 3 shows that for an approximate δ value of $10M$ between the two replicas of *lineitem*, SRA is better than HA.

Table 3 Computational time (sec.) of HA and SRA variants

$\lvert\pi_{partkey,suppkey}(lineitem)\rvert$	δ *lineitem*	*HA*	*SRA*
46	8,388,607	76	0.08
76	9,828,718	74	0.06
132	9,956,784	75	4.16
238	9,777,715	76	7.80
438	9,335,809	75	18.20
820	**9,646,372**	**76**	**71.50**
1542	9,944,855	77	210.30

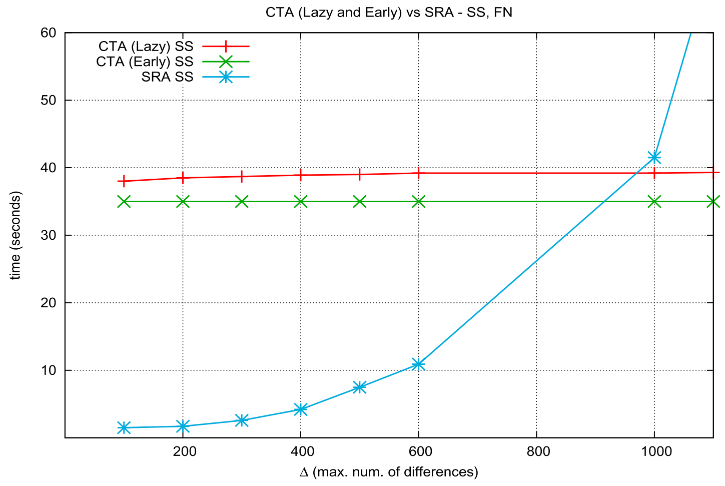


Fig. 4 Three methods to compute referential integrity metric grcom() using *lineitem* as referencing table and *orders* as referenced table: Early and Lazy CTA variants and the SRA variant

4.3 Measuring referential integrity consistency

To compute the referential integrity metric grcom(), as discussed in Sect. 3.4, again we can use the set reconciliation techniques.

In Fig. 4, we compare three methods to compute the referential integrity metric assuming as the global tables *orders* and *lineitem*, the referenced and the referencing tables respectively, and assuming they are computed at different sites. We also assume different number of foreign key value errors—which, in our case, is the number of differences. The early and lazy CTA (Complete Transfer Approach) variants are inspired by the computation of the referential integrity quality metrics presented in [23]. In both variants, first a projection of the referenced global table primary key is transferred to the site where the principal referencing global table resides. The next step is to evaluate the referential integrity errors by joining the transferred referenced table with either a frequency table computed by means of a SQL group by clause by foreign key (eager evaluation), or a referencing table in which the number of referential integrity errors is computed afterwards (lazy evaluation). The SRA variant consists of computing a frequency table with the foreign key values of the referencing global table and then computing the metric as described in Sect. 3.4. We observe that for differences $\Delta < 890$, the SRA variant is more efficient than either of the CTA variants.

4.4 Measuring consistency with SRA on a real-world distributed database

Finally, the following experiment done with a real database highlights how the SRA algorithm outperforms the HA algorithm for rather big tables and marginal differences between replicas.

In Fig. 5, we present our experimental results on a real-world database from a public educational institution. We were able to explore a large historic table with $120M$

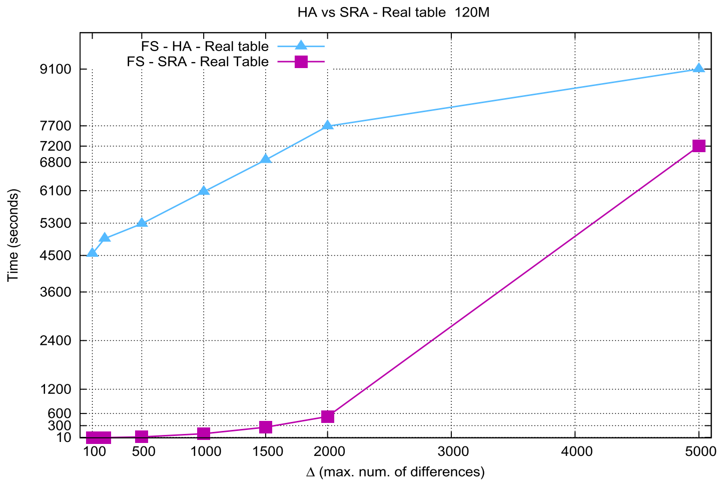


Fig. 5 Comparison between HA and SRA. A real database with $n = 12M$ records from an educational institution

records containing student information, such as student name, year, semester, course names, course grades, credits and grade point average. The goal of our experiments was to compare the performance between the SRA algorithm and the HA algorithm in a production environment where the table periodically receives relatively small chunks of records of different sizes; in case of this particular database, these batches of periodic updates would typically range between 100 and 5000 new or modified records. Upon receiving of each batch, the table was replicated, primarily for fault tolerance reasons.

The key insights from our experiments are as follows (see Fig. 5). We considered Δ in the range from 100 to 5,000, in accordance with the typical numbers of records that would get updated in a single batch of table updates. For this entire range of replica differences, *SRA considerably outperforms HA*. We consider this result to be a major real-world validation of our SRA-based approach to replica consistency. However, as Δ grows, the benefits of SRA shrink. Furthermore, the increase in SRA's overall execution time indeed appears to grow approximately proportionally to Δ^3, in accordance with our theoretical analysis. Therefore, as predicted, SRA indeed offers the maximal benefits when Δ is fairly small (in this case, a few hundred new/modified records out of about 10^8 records total). In contrast, the performance degradation of HA with the growth of Δ is much more gradual, again as predicted by theory. In particular, the overall processing time difference between the two algorithms decreases from 3500 sec (for $\Delta = 100$) to 1000 sec (when up to $\Delta = 5000$).

Results shown in Fig. 5 were also discussed with the Information Technology manager at this educational institution. He stated that since the historic table was constantly updated with new semester course grades, using SRA was a good alternative to verify the consistency of the replica with the database at the master site.

5 Related work

Replica consistency has received a considerable attention in recent years. In [27], the authors propose two update propagation strategies that improve freshness, a concept that assumes that replica consistency in a distributed database can be relaxed. These strategies are based on immediate propagation, without waiting for the commitment of the update transaction in Master-Slave configurations. In [26], the authors propose a refreshment algorithm to maintain replica consistency at a lazy master replicated database based on specific properties of the topology of replica distribution across the nodes. Both works propose strategies toward maintaining replica consistency in a database that is in a Master-Slave configuration. In [9], a transaction router for a database cluster is proposed; that transaction router takes into account the freshness requirements of queries in order to improve load balancing. It considers cluster load, the estimated time to ensure the freshness of replicas at the level required by the incoming queries. Our work assumes an *a posteriori* scenario where replica inconsistency and referential integrity violations are probably present and the user wants to quantify the severity of this problem and (where applicable) to repair it.

We briefly review several other lines of prior research on replica consistency and referential integrity. In [29], the authors study several optimistic replication algorithms. These algorithms allow replica contents to diverge in the short term to support concurrent work practices and tolerate failures in low-quality communication links. Update propagation is usually done by transferring one replica to another site. This approach is very inefficient and turns worse as the replica grows. Different optimizations have been studied but most of them have a complexity that depends linearly or logarithmically on the size of the replicas. In [5], the authors propose two lazy update protocols that can be used in a distributed database. These protocols guarantee serializability, but require that the copy graph be a *directed acyclic graph* (DAG). The authors propose a solution to prevent the lazy replication inconsistency problems in a particular distributed configuration. In [22], the authors propose a new class of replication systems called *TRAPP* (Tradeoff in Replication Precision and Performance). Instead of storing stale exact values in the slave site, the system stores ranges that are guaranteed to bound the updated data values. Users provide quantitative precision constraints along with their queries and then the system selects a combination of locally cached bounds and up-to-date data stored at the master site. The system delivers its response in the form of a bounded range that is no wider than the specified precision constraint that is guaranteed to contain the correct answer and is computed as quickly as possible. In [2], the authors present a protocol whose purpose is to manage efficiently many replicas in a scenario where OLTP and OLAP workloads are supported. Their protocol (i) combines lazy and eager replica maintenance to ensure correct and efficient executions, (ii) does not require data to be fully replicated at all sites, and (iii) supports different replication granularities. In contrast to [2], our system delivers imprecise values that meet the user precision constraints or (when more processing time can be afforded) the exact values, where the data transmission cost, under the assumptions discussed earlier, can be expected to be quite low. In [8], a coherency index to measure replica consistency (coherency) is introduced. The paper examines the tradeoff between consistency and performance, and demonstrates that, in many situations, a slight relaxation of coherency can increase performance. In our work, we focus on efficiently diagnosing replica consistency issues in a distributed database scenario. In contrast, we propose a method to measure the quality of a distributed database with respect to replica consistency and referential integrity and we evaluate how to obtain a fast diagnosis considering different replica scenarios. In [35] and [36], the authors propose several metrics to measure the quality of replicated services where the access to a replicated database is included. They propose a middleware layer that enforces consistency bounds among replicas allowing applications to dynamically trade consistency for performance based on the current service, network, and request characteristics. They measure availability while varying the consistency level, the protocol used to enforce consistency, and the failure characteristics of the underlying network. However, only the quality of the service (access) is evaluated in that paper, and not the quality of the actual replicated data (as in our case).

In [18], the authors propose an approach to repair a crashed site in a distributed database that uses data replication to tolerate machine failures. Their approach uses timestamps to determine which records need to be copied or updated. In contrast, our strategy is an on-demand technique, that also queries sites but does not require timestamps. That is, our approach measures the quality of a distributed database when the user needs a diagnosis of the inconsistencies. A related problem is that of detecting and extracting modifications from information sources—in particular, detecting differences between snapshots of data. In [17], the authors propose several algorithms that address this problem, known as *the snapshot differential problem.* The algorithms perform compression of records and one of them, the window algorithm, works specially well when the number of differences is small. However, the time complexity of the algorithms depends on the size of the snapshots. In particular, the window algorithm needs to read the full snapshots. In contrast, our method using set reconciliation techniques based on the computation of the characteristic polynomials depends on the size of the symmetric differences, and has an additional important advantage that the computation of the evaluation points can be done asynchronously at different sites. In [15], the authors propose *DYFRAM*, a decentralized approach for dynamic table fragmentation and allocation in distributed database systems, based on observation of the access patterns of sites to tables. The method requires that replication be master-copy based, i.e., all updates to a fragment are performed to the master-copy, and afterward propagated to the replicas. The communication complexity of this method depends on the size of the replicated fragment. In contrast, the communication complexity of our method depends on the estimated number of differences between, in a master-copy scenario, the master-copy and the replica. In [20], the authors address the limitation of static physical partitioning by taking advantage of replicas and propose a dynamic query load balancing strategy avoiding the overhead of full replication. Their technique makes it possible to dynamically redistribute tasks from busy nodes to idle nodes that contain replicas of their data fragments. This method is specifically oriented to optimize exploratory aggregation queries. In contrast, our method allows the complete replication of tables no matter how the replicas will be used.

To close our discussion on related work, in [24] we propose referential integrity metrics in distributed databases. The present paper builds on the top of [24] by addressing both referential integrity and replica consistency in a unified framework, and applying advanced algebraic techniques (based on efficient evaluation of polynomials over finite fields) in order to address replica consistency problem in a scalable, that is, processing and communication complexity efficient manner.

6 Conclusions

We propose novel algorithms and optimizations to efficiently repair and measure replica consistency and referential integrity in distributed databases. Our algorithms can help measuring and maintaining distributed consistency, where fast response time and fault tolerance are required. We consider evaluating replica consistency of individual tables in a Master-Slave configuration and repairing inconsistent tables. On the other hand, in a Multi-Master configuration, we consider the evaluation of global replica consistency and referential integrity. Specifically, we investigate how to quickly compute the symmetric difference between two table replicas, adapting set reconciliation algorithms to a distributed database. Our techniques are applicable when the symmetric difference size between table replica pairs is small and network transmission is slower than disk I/O, since the communication complexity is proportional to the number of different records. The processing complexity of our approach is relatively high, namely, it is cubic in difference size. While recognizing the limitations stemming from such cubic complexity, we argue that, in a distributed database where tables are generally large and the symmetric differences between replicas are expected to be small, our techniques represent a promising alternative for to repair and measure replica consistency as well as referential integrity. In summary, we adapt and optimize set reconciliation distributed algorithms to repair and measure table replica consistency in a distributed database. We also consider referential integrity, a prominent consistency problem, especially in Multi-Master configurations. Our algorithms have linear communication complexity and cubic time complexity on the size of the symmetric difference between two sets of values, which is expected to be small. Therefore, our distributed algorithms are highly efficient to repair and measure consistency of table replicas, thereby eliminating the need to track database insertions and deletions with temporary tables or specific indexes. We conduct an extensive experimental evaluation with real and synthetic databases focusing on performance. We compare our methods with a state-of-the-art hierarchical algorithm (*pg–comparator*), which is based on checksums; this algorithm performs a reconciliation merge of partial checksums that are fetched hierarchically, level-by-level. Our experiments confirm the efficiency and effectiveness of our approach.

There are several open research issues for future work. The set reconciliation techniques may be optimized using parallelism, thereby taking advantage of primary key values seen as a set of ordered values. We would like to develop new database algorithms to maintain consistency measures online, as new records are being inserted. We plan to explore methods where the user gives thresholds for the precision of aggregate functions over measure attributes, together with the size of the symmetric difference between replicas. We would like to explore how set reconciliation techniques can solve other problems in distributed database systems, beyond repairing and measuring consistency.

Acknowledgements The second author was partially supported by NSF grant IIS 0914861. The third author is currently working for Microsoft Corp.

We would like to thank Sergio Rajsbaum for an inspiring conversation that derived on an earlier version of this article. We would also like to thank Claudia Morales-Almonte and Rogelio Montero-Campos for programming our algorithms to conduct the experimental evaluation.

Appendix

Lemma 1 *If* $\forall D_i$, $i = 0, \ldots, N-1$, $\mathrm{cur}(D_i.T_p) = 0$ (*i.e.*, *if all replicas of table* T_p *are replica consistent*), *then* $\mathrm{gcur}(T_p) = 0$ (*that is*, *table* T_p *is global replica consistent*).

Proof If $\forall D_i$, $i = 0, \ldots, N-1$, $\mathrm{cur}(D_i.T_p) = 0$, then by Eqs. (1) and (2) that

$$\begin{aligned} &\forall D_i,\ i = 0, \ldots, N-1,\ \bigl|(D_i.T_p - D_0.T_p) \cup (D_0.T_p - D_i.T_p)\bigr| = 0, \\ &\text{so } \bigl|(D_i.T_p - D_0.T_p)\bigr| = 0 \quad \text{and} \quad \bigl|(D_0.T_p - D_i.T_p)\bigr| = 0 \end{aligned} \tag{13}$$

Since $|\bigcup_{i=0}^{N-1} D_i.T_p| = |D_0.T_p \cup (\bigcup_{i=1}^{N-1}(D_i.T_p - D_0.T_p))|$ and $|\bigcap_{i=0}^{N-1} D_i.T_p| = |D_0.T_p - (\bigcup_{i=1}^{N-1}(D_0.T_p - D_i.T_p))|$, by virtue of Eq. (13) we have $|\bigcup_{i=0}^{N-1} D_i.T_p| = |D_0.T_p|$ and $|\bigcap_{i=0}^{N-1} D_i.T_p| = |D_0.T_p|$, and consequently, by definition of gcur() (Eq. (3)), it follows that

$$\mathrm{gcur}(T_p) = 0 \tag{14}$$

□

Lemma 2

$$\begin{aligned} \mathrm{grcom}(T_r.K) &= \frac{|\mathcal{T}_{r\cup} \bowtie_K \mathcal{T}_{s\cup}|}{|\mathcal{T}_{r\cup}|} \\ &= \bigl(|\mathcal{T}_{r\cup}| - \bigl|\bigl(\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})\bigr) \bowtie_K \mathcal{T}_{r\cup}\bigr| \\ &\quad - \bigl|\sigma_{\mathrm{isnull}(K)}(\mathcal{T}_{r\cup})\bigr|\bigr)/|\mathcal{T}_{r\cup}| \end{aligned} \tag{15}$$

Proof By definition $|\mathcal{T}_{r\cup} \bowtie_K \mathcal{T}_{s\cup}|$ is the number of tuples in $\mathcal{T}_{r\cup}$ having their foreign key K values match a primary key K value in $\mathcal{T}_{s\cup}$. $\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})$ is a one attribute relation with values in $\pi_K(\mathcal{T}_{r\cup})$ that are not in $\pi_K(\mathcal{T}_{s\cup})$, so $(\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})) \bowtie_K \mathcal{T}_{r\cup}$ are the tuples in $\mathcal{T}_{r\cup}$ that do not have a foreign key value matching a value in $\pi_K(\mathcal{T}_{s\cup})$. Observe that in these tuples, K is not null.

Consequently, $|\mathcal{T}_{r\cup}| - |(\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})) \bowtie_K \mathcal{T}_{r\cup}|$ is the number of tuples in $\mathcal{T}_{r\cup}$ with their foreign key K values matching a primary key K value in $\mathcal{T}_{s\cup}$ plus the tuples with a null value in foreign key K. The last subtraction, $-|\sigma_{\mathrm{isnull}(K)}(\mathcal{T}_{r\cup})|$, is to discount these tuples. □

To evaluate the above expression, among other computations we need the difference $\pi_K(\mathcal{T}_{r\cup}) - \pi_K(\mathcal{T}_{s\cup})$. This expression evaluates a table with the invalid foreign key values—that is, a table with the inclusion dependency violations. We note that, if the difference is low, this table difference would be relatively small compared to the size of $\mathcal{T}_{s\cup}$ or even $\pi_K(\mathcal{T}_{s\cup})$.

References

1. Abadi, D.: Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer **45**(2), 37–42 (2012)
2. Akal, F., Türker, C., Schek, H.-J., Breitbart, Y., Grabs, T., Veen, L.: Fine-grained replication and scheduling with freshness and correctness guarantees. In: VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 565–576 (2005)
3. Albrecht, J., Lehner, W.: On-line analytical processing in distributed data warehouses. In: IDEAS'98, p. 78. IEEE Computer Society Press, Los Alamitos (1998)
4. Bernardino, J., Madeira, H.: Experimental evaluation of a new distributed partitioning technique for data warehouses. In: Int'l Symposium on Database Engineering and Applications (IDEAS'01), pp. 312–321 (2001)
5. Breitbart, Y., Komondoor, R., Rastogi, R., Seshadri, S., Silberschatz, A.: Update propagation protocols for replicated databases. In: SIGMOD'99, pp. 97–108 (1999)
6. Coelho, F.: Pg comparator. http://pgfoundry.org/projects/pg-comparator (2012), consulted on April 2012
7. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 3rd edn. Addison/Wesley, Redwood City (2000)
8. Gallersdörfer, R., Nicola, M.: Improving performance in replicated databases through relaxed coherency. In: VLDB'95, pp. 445–456 (1995)
9. Gançarski, S., Naacke, H., Pacitti, E., Valduriez, P.: The leganet system: freshness-aware transaction routing in a database cluster. Inf. Syst. **32**(2), 320–343 (2007)
10. García-García, J., Ordonez, C.: Consistency-aware evaluation of OLAP queries in replicated data warehouses. In: ACM DOLAP'09, pp. 73–80 (2009)
11. García-García, J., Ordonez, C.: Extended aggregations for databases with referential integrity issues. Data Knowl. Eng. **69**(1), 73–95 (2010)
12. Golub, G.H., van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins Univ. Press, Baltimore (1996)
13. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: SIGMOD'96, pp. 173–182 (1996)
14. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 1st edn. Morgan Kaufmann, San Francisco (2001)
15. Hauglid, J.O., Ryeng, N.H., Nørvåg, K.: DYFRAM: dynamic fragmentation and replica management in distributed database systems. Distrib. Parallel Databases **28**, 157–185 (2010)
16. Huang, J., Naughton, J.F., Livny, M.: TRAC: toward recency and consistency reporting in a database with distributed data sources. In: VLDB, pp. 223–234 (2006)
17. Labio, W., Garcia-Molina, H.: Efficient snapshot differential algorithms for data warehousing. In: VLDB'96, pp. 63–74 (1996)
18. Lau, E., Madden, S.: An integrated approach to recovery and high availability in an updateable, distributed data warehouse. In: VLDB'06, pp. 703–714 (2006)
19. Lidl, R., Niederreiter, H.: Finite Fields, 2nd edn. Encyclopedia of Mathematics and Its Applications, vol. 20. Cambridge University Press, Cambridge (1997)
20. Lima, A.A.B., Furtado, C., Valduriez, P., Mattoso, M.: Parallel OLAP query processing in database clusters with data replication. Distrib. Parallel Databases **25**)(1–2), 97–123 (2009)
21. Minsky, Y., Trachtenberg, A., Zippel, R.: Set reconciliation with nearly optimal communication complexity. IEEE Trans. Inf. Theory **49**(9), 2213–2218 (2003)
22. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: VLDB, pp. 144–155 (2000)
23. Ordonez, C., García-García, J.: Referential integrity quality metrics. Decis. Support Syst. **44**(2), 495–508 (2008)
24. Ordonez, C., García-García, J., Chen, Z.: Measuring referential integrity in distributed databases. In: ACM First Workshop on Cyber Infrastructure: Information Management in EScience, CIMS, pp. 61–66 (2007)
25. Ozsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn. Springer, Berlin (2011)
26. Pacitti, E., Minet, P., Simon, E.: Replica consistency in lazy master replicated databases. Distrib. Parallel Databases **9**(3), 237–267 (2001)
27. Pacitti, E., Simon, E.: Update propagation strategies to improve freshness in lazy master replicated databases. VLDB J. **8**(3–4), 305–318 (2000)

28. Pu, C., Leff, A.: Replica control in distributed systems: as asynchronous approach. In: SIGMOD'91, Proc. of the 1991 ACM SIGMOD Int'l Conf. on Management of Data. ACM, New York (1991)
29. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. **37**, 42–81 (2005)
30. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, New York (2005)
31. Shoup, V.: NTL: a library for doing number theory. http://www.shoup.net/ntl/ (2008)
32. Starobinski, D., Trachtenberg, A.: Boston University Laboratory of Networking and Information Systems. http://ipsit.bu.edu/nislab/projects/cpisync/download.htm (2008)
33. TPC: TPC-H benchmark. Transaction Processing Performance Council. http://www.tpc.org/tpch (2005)
34. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding replication in databases and distributed systems. In: Proc. of the 20th Int'l Conference on Distributed Computing Systems, pp. 464–474 (2000)
35. Yu, H., Vahdat, A.: Design and evaluation of a continuous consistency model for replicated services. In: OSDI'00, p. 21 (2000)
36. Yu, H., Vahdat, A.: The costs and limits of availability for replicated services. ACM Trans. Comput. Syst. **24**(1), 70–113 (2006)