Efficient Synchronization of Multiple Databases over Broadcast Networks

Muhammad Muhammad, Stefan Erl, and Matteo Berioli

German Aerospace Center (DLR)
Institute of Communications and Navigation
82234, Oberpfaffenhofen, Germany
{Muhammad.Muhammad,Stefan.Erl,Matteo.Berioli}@dlr.de

Abstract. This work deals with the problem of synchronizing multiple distributed databases over a broadcast network, such as satellite networks. The proposed method is based on introducing network coding techniques besides extending the well-known database reconciliation algorithm, the characteristic polynomial interpolation-based synchronization (CPISync). One key element is to elect a master node that manages the operations in a central manner. Performance is shown in terms of completion time for full synchronization, and average number of packets exchange. Compared to point-to-point and traditional broadcasting synchronization methods, the algorithm implementing network coding allows reaching the lower-bound for the number of packets exchange.

Keywords: multiple databases synchronization, network coding, broadcast channels, satellite communications.

1 Introduction

A database consists of an organized collection of related data. A distributed database system (DDBS) is a collection of multiple, logically interrelated databases. These distributed databases are physically spread over a computer network of multiple nodes, where any node can update its database at any time. A database management system (DBMS) consists of software that controls databases. The services provided include storage, access, security, backup and some other facilities. The DBMSs can be categorized according to the database model that they support such as relational databases, the type of computer they support, and the query language that accesses the database such as SQL. Some commonly used DBMSs are MySQL and PostgreSQL.

With the high demand on the usage of these DDBSs and due to the changes that may occur on one or many sites discarding the others, data synchronization algorithms have been deeply studied, see [1] and [2]. Additionally, several synchronization and replication applications, to harmonize information and to keep consistency of data among all points in the network managing these databases, have been developed for the different DDBSs currently available, such as Cybercluster and Maatkit.

R. Dhaou et al. (Eds.): PSATS 2013, LNICST 123, pp. 77–89, 2013.

In spite of the great effort done by the scientific community in the field of database synchronization, especially for CPISync, in terms of reducing the synchronization traffic, this method mainly operates in a point-to-point environment, which limits the applicability of synchronizing multiple databases. In this light, the aim of this work is to present a novel method for synchronizing multiple databases, namely, to make all datasets involved in the reconciliation process have local access to all the data with further minimized traffic load that is achieved by extending CPISync to multiple nodes, by applying network coding principles and by changing the network topology to a star network.

Network coding [3,5] has been also applied to solve the problem of distributed storage [10]. Distributed storage systems often introduce redundancy to increase reliability. When coding is used, the repair problem arises, which addresses the case if a node storing encoded information fails, then in order to maintain the same level of reliability, encoded information need to be created at a new node. A survey of network coding and distributed storage systems can be found in [13]. Some problems of distributed storage systems, like storage allocations, are addressed in [11,12]. However, our problem differs from that of the distributed storage systems in the sense that our proposal does not aim to provide data reliability in case of node failure, but to allow all involved nodes to acquire local access to the complete dataset at any given time.

This paper is organized as follows. The next section gives background information about the CPISync algorithm. Section 3 exploits different methods of differences discovery in database synchronization using CPISync. In Section 4 solutions for updating databases are given. The performance of the new methods is evaluated in Section 5. Finally, a conclusion follows in Section 6.

2 Background

Let us consider a network of nodes (distributed over a wide geographical area, but can be served by a single satellite beam), where each node in the system is maintaining a database. These datasets form a DDBS. The datasets may or may not have common entries. Whenever a synchronization is required; may be on demand, periodically, or after re-establishing the network connection in case of node or link failure, a bidirectional merge of the involved databases takes place. The operation of merging two databases can be separated in two subtasks: (i) the finding of the differences between the two databases, and (ii) the bidirectional data exchange to update the two databases and to guarantee that they are identical.

The former task requires an optimized algorithm for finding the differences. For the work in this paper, CPISync [6] is considered. The latter task may be non trivial in the case of a DDBS, but a lot of work has been done to provide efficient solutions; especially over broadcast networks, where the concepts of reliable multicast can be exploited. In this respect, this work will focus on Network Coding to perform the latter task. We will focus on a particular, yet simple, Network Coding scheme, namely Random Linear Network Coding (RLNC) [4],

which is representative of the general idea to exploit Network Coding for DDBS. The next subsection will introduce the basics of CPISync.

2.1 CPISync

The work in [6] shows that CPISync has close-to-optimum performance, it has a minimum comparison overhead that depends only on the amount of differences between the synchronizing datasets, but not on the dataset size itself. In [7] the authors tested CPISync and it was shown to be a promising solution for synchronizing databases over satellite narrowband links.

A record in a database is a row that is constructed of multiple related fields. In order to use CPISync, each record in the database can be represented by one unique integer, e.g. by means of a hash function. This allows to associate to each database A a set of integers $S_A = \{x_1, x_2, ..., x_n\}$, representing all records in the database A. The key point of the CPISync algorithm is the conversion of a database into a polynomial, which is called the characteristic polynomial of the database. The characteristic polynomial of database A is defined as follows:

$$X_{S_A}(Z) = (Z - x_1)(Z - x_2)(Z - x_3) \cdots (Z - x_n). \tag{1}$$

The idea is to manipulate these characteristic polynomials to discover the differences between two databases, and this is done as explained in the following.

Let us assume we have two databases A and B, for which we can define two characteristic polynomials as indicated in (1) above. Let us also define $\Delta_A = S_A \backslash S_B$, as the set of integers in A but not in B, and symmetrically $\Delta_B = S_B \backslash S_A$, as the set of integers in B but not in A. If we knew all elements of the two databases, we would be able to build the following rational function:

$$f(Z) = \frac{X_{S_A}(Z)}{X_{S_B}(Z)} = \frac{X_{S_A \cap S_B}(Z) \cdot X_{\Delta_A}(Z)}{X_{S_A \cap S_B}(Z) \cdot X_{\Delta_B}(Z)} = \frac{X_{\Delta_A}(Z)}{X_{\Delta_B}(Z)}. \tag{2}$$

This rational function has the interesting property that can be described by only means of the two sets Δ_A and Δ_B. So if it were possible to build this function without complete knowledge of the two databases, then it would be possible to discover the differences between the two databases. This is the core principle of CPISync. Curios readers are forwarded to [6] to find a detailed example on how CPISync works.

The upper bound ($\overline{m}$) of the actual symmetric differences is in reality difficult to know or predict in many applications. For that reason, an improved CPISync algorithm, called Partitioned-CPISync, was proposed in [8] to overcome this drawback. The value of $\overline{m}$ is fixed a priori by the two hosts, and the algorithm recursively divides each set into p partitions until the basic CPISync algorithm can succeed with the pre-agreed upper bound $\overline{m}$ on the number of differences in a single partition. In other words, the set S_A is partitioned into p non-intersecting subsets, and if the basic CPISync fails in a subset, the algorithm keeps dividing each subset into p subsubsets, and so on, until the differences can be discovered by the basic CPISync algorithm.

The complexity of the Partitioned-CPISync was analyzed in [8]. Worst-case bounds for communication and computation complexity are given there. When assuming hashed indexes of the set that are randomly and uniformly distributed, these worst-case bounds could be optimized to the following formulas for synchronizing two databases.

The expected number of rounds $\overline{r}$ needed by the Partitioned-CPISync algorithm for the reconciliation of two databases is at most:

$$\overline{r} = 2\log_p\left(\frac{m}{\overline{m}+1}\right) + \mathcal{O}(1), \tag{3}$$

with m denoting the actual number of differences between the two sets and p denoting the partition factor, representing the number of non-overlapping parts the set p is split up in each round. In one round several Basic-CPISync runs are performed, because the evaluation values of individual parts p of the same level do not depend on each other and can be generated and sent together.

The expected number of overall bits transmitted from one node to another in order to determine the differences is at most:

$$\overline{b} = 8emp(b+1) + \frac{8emkp(b+1)}{\overline{m}+1}, \tag{4}$$

with $e \approx 2.71828183$ being the Euler number. The parameter k is the number of additional evaluation values sent by the host to verify that the rational interpolation with the chosen $\overline{m}$ was correct. The parameter b is the length of the hash value of a row in the database, which is used to calculate the characteristic polynomial.

3 Phase I: Discovering the Differences

In this section, we propose, investigate, and compare different possibilities on how to exploit CPISync to discover the differences inside a DDBS.

Assume a network of N distributed nodes is maintaining a DDBS. Each endpoint is handling a single database. The goal of phase I is that the system knows which node is missing which packets and how all the nodes can be synchronized.

3.1 Solutions for Difference Discovery

Four solutions are proposed: one under the category of mesh network, and three under the category of star network.

Mesh Network. A fully meshed comparison between all nodes is the easiest way to run CPISync. In this scenario, every node is compared and synchronized with every other node in the system in order to achieve a complete system synchronization. There is no central coordination, to synchronize N databases in this scenario, every node has to synchronize (on its own) with all the other $(N-1)$ nodes. This will result in a complete synchronization of the system

after performing $\frac{N(N-1)}{2}$ synchronization procedures by all nodes. This technique considers the absence of a specific nodes set up, i.e., every node synchronizes its database with a randomly chosen another node, such that any synchronization process between any two nodes is performed only once. In this respect, the time required to run the algorithm and end up with a fully synchronized system is:

$$T_{\text{mesh}} = \frac{N\,(N-1)}{2}\,[\overline{r}\,(2T_{\text{s}} + T_{\text{eval}} + T_{\text{calc}}) + 2T_{\text{data}} + T_{\text{s}}]\,, \tag{5}$$

where T_{s} is the one-way signal delay between any two synchronizing nodes, T_{eval} is the time to receive the evaluations, and T_{data} is the time to receive the missing data. The time for the calculation is summarized in T_{calc}. T_{calc} mainly contains the time to perform the mathematical computations of the polynomial interpolation. T_{eval} depends on the chosen upper-bound $\overline{m}$, the length of the hash ID b and on the actual size $|d_i|$ of the differences set. For one complete run of CPISync, T_{eval} can be expressed in dependence of $\overline{b}$, the total number of bits transmitted.

In this scenario, phase I (difference discovery) and phase II (update of the differences) take place one after the other at each pairwise comparison, so they both take place $\frac{N(N-1)}{2}$ times.

Star Network. In a star network, one of the nodes is set to the role of the master node that takes care of the synchronization. In phase I, the master (with set S_0) gains the knowledge of its respective differences with S_i ($i = 1,...,N-1$); and receives the data d_i from the other nodes. In fact the operation of updating the master with the data d_i from all nodes, is already part of phase II; it will be mentioned in this section for the sake of understanding, but will not be considered in the performance evaluation of phase I presented at the end of this section. We present three methods on how to employ the CPISync algorithm within a multiple node environment. The more practical Partitioned-CPISync is used for our analysis as it is likely that several rounds of communication are used, which may drastically increase the time that is needed for discovering the differences between the nodes, especially for satellite networks with long round-trip time (RTT).

Round-Robin. The easiest and most straight-forward approach to obtain global view in the master node, since CPISync is originally suited for two sets, is a round-robin-like synchronization. The master discovers its differences to every node one after the other. After one full cycle, the differences between S_0 and each S_i are known. Combining these differences results in the global set ρ. This approach is shown in Figure 1(a). The figure shows the Partitioned-CPISync algorithm with taking two rounds of communication.

The master first sends its evaluations to the first node. This node tries to interpolate the missing data elements, but in this example, not enough evaluation values were sent. So it requests more evaluations from the master. After the interpolation was successful in the second round, the node knows its d_i and

sends it back to the master, including the request for the missing data (which would be d_0). In general, there are $\overline{r}$ rounds of communication between one node and the master. The same is also done with the remaining $N-2$ slaves. So the number of runs of the CPISync algorithm is $N-1$, and thus the complete number of transmitted bits, the complete amount of rounds needed, and the computation complexity for the CPISync in this scheme are multiplied by $N-1$.

The time to complete the algorithm can be derived from Figure 1(a). In each round, the evaluation values are sent to a node ($T_\mathrm{s} + T_\mathrm{eval}$) and a request is sent back to the master (T_s). In the last round, also the time for sending the data (T_data) is required. The time for the calculation is summarized in T_calc. The complete time results in:

$$T_\mathrm{rr} = (N-1)\left[\overline{r}(2T_\mathrm{s} + T_\mathrm{eval} + T_\mathrm{calc}) + T_\mathrm{data}\right]. \tag{6}$$

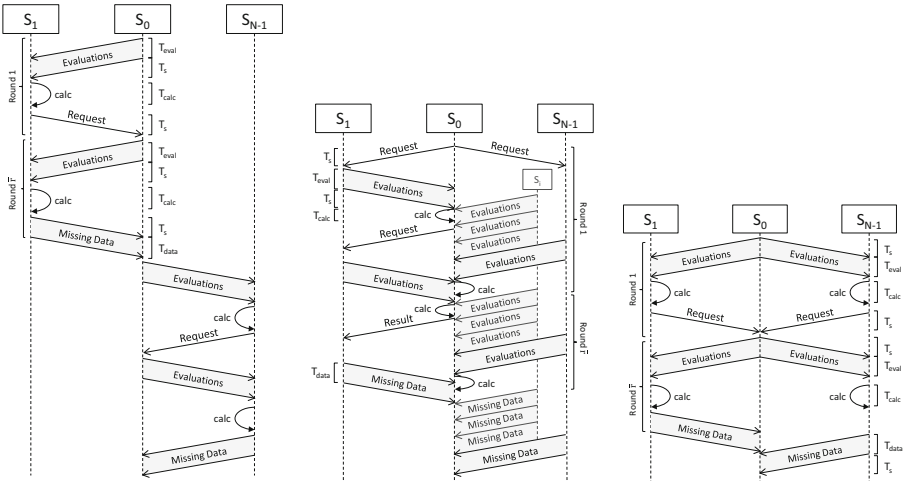
Central Calculation. In the central calculation approach, all the computation is moved from the nodes to the master. Here, the master requests evaluation values from all nodes. The computation for the first node can be started as soon as the evaluation values from the first node have been received. Figure 1(b) demonstrates this procedure. The request for the evaluation values is broadcast to the slave nodes. The first node starts transmitting, and the calculation is started as soon as the transmission is complete. Meanwhile the other nodes are sending their values. If the algorithm takes more rounds, a new request is sent to the corresponding node. After finishing the algorithm, the master has to request the missing data from all the nodes, because it only knows what data is missing, but it still does not have the data itself.

The transmitted bits, rounds and computation are the same like in the round-robin method, but since the single CPISync processes are more parallelized than in the previous method, the total time required to finish the algorithm will be shorter. Depending on the more time-intensive process, the interpolation or the reception of the evaluations, the total time for the algorithm is:

$$T_\mathrm{cc} = \begin{cases} 2\,T_\mathrm{s} + (N-1)(\overline{r}\,T_\mathrm{eval} + T_\mathrm{data}), & \text{if } T_\mathrm{eval} > T_\mathrm{calc} \\ 2\,T_\mathrm{s} + (N-1)(\overline{r}\,T_\mathrm{calc} + T_\mathrm{data}), & \text{if } T_\mathrm{calc} > T_\mathrm{eval} \end{cases} \tag{7}$$

The calculation for one node will be done while other nodes are still transmitting, or the reception is performed while other nodes' interpolations are still in progress, respectively.

Broadcast Evaluations. In contrast to the previous method, in this approach the evaluation values of the master's set are broadcast. Then each node does the interpolation on itself and discovers its differences to the master. Figure 1(c) shows this approach. Evaluation values needed for further rounds can be broadcast as soon as the first request arrives and following request for the same round can be ignored. Other nodes can buffer the further evaluations if their calculation is not yet finished. After the nodes finished their calculations, they send their additional data d_i and a request for the data they are missing to the master.



(a) Round-Robin-like deployment of CPISync. (b) Master computes differences centrally. (c) Evaluations are broadcast.

Fig. 1. Several techniques in obtaining differences in broadcast medium

Because the evaluation values are broadcast and the interpolation is done fully parallel across all nodes, the only parameter that depends on the number of nodes is the transmission time T_{data} when sending the missing data back to the master node. The total required time with this approach is then:

$$T_{\text{bc}} = \overline{r}(2\,T_{\text{s}} + (T_{\text{eval}} + T_{\text{calc}}) + (N-1)\,T_{\text{data}}). \tag{8}$$

3.2 Comparison

In this section, we provide lower and upper bounds on the number of CPISync processes to discover the complete differences between any two synchronizing databases. From the previously discussed methods, it is clear that the number of CPISync processes in master-slave mode ($P_{\text{M-S}}^{\text{CPISync}}$) is reduced due to the deployment of the star network. However, this will increase the complexity of the algorithm at the master node as the number of slaves increases. But, this difficulty will be linked only to the master end-point. Therefore, the number of CPISync processes is always: $(N-1)$.

Alternatively, when more and more nodes use CPISync in a mesh network, the complication will be on each device to track the changes occurring and the next node to connect to. This makes the number of CPISync processes grows up to: $\frac{N(N-1)}{2}$. So the number of CPISync calls grows with the square of the number of databases N, whereas for the master-slave approach it grows linearly with N. In addition, in the mesh-network approach also phase II (update of the differences) is called $\frac{N(N-1)}{2}$ times, whereas in the master-slave approach it can be performed just once and more efficiently, by exploiting network coding principles (this will be explained in the next section).

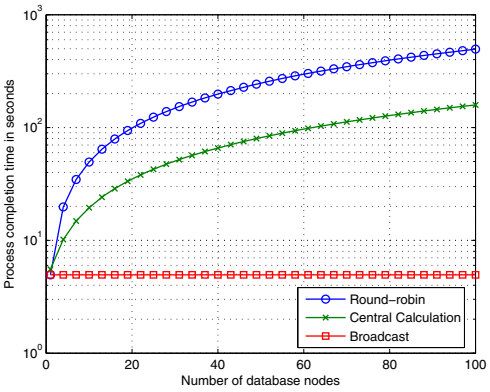


Fig. 2. Time for obtaining the differences using Star network topology

For these reasons the mesh-network approach can be discarded as it is the one with the highest complexity. Now we compare the three different methods for the star network, namely, *round-robin*, *central calculation*, and *broadcast evaluations*. We assume a satellite link with a signal delay $T_\mathrm{s} = 300\,\mathrm{ms}$ and a bit-rate of 1 Mbps. Each node has an additional data $|d_i| = 500$ rows, and each row has a size of 1 KB. The CPISync parameters are set to $b = 32$, $p = 4$, $k = 4$ and $\overline{m} = 50$, which results in an average number of $\overline{r} = 4$ rounds and $\overline{b} = 1.6\,\mathrm{Mbit}$. This results in $T_\mathrm{eval} = 1.6\,\mathrm{s}$. The calculation time is assumed with $T_\mathrm{calc} = 1\,\mathrm{s}$.

For sending and receiving data, a simple ideal channel model (i.e. with no packet error rate (PER)) with time slots is assumed. Only one node is allowed to send at each time slot, and the data sent from several nodes to one receiver arrives one after the other. For transmitting data no specific protocol is assumed and acknowledgments for each packet received or packet sizes are irrelevant.

Figure 2 shows the required time for the three methods with the above parameters as a function of the number of nodes N. Since all the methods include a transfer of the missing data to the master at the end of the algorithm, we ignored T_data in this graph, since this is in fact part of what we defined as phase II (update of the differences).

As expected, the round-robin method performs worst, as all nodes are synchronized serially, resulting in many rounds of communication and a long processing time. The central-calculation approach is a bit faster, because it is more parallelized and, depending on the properties of the channel and the master, either the channel or the processor in the master node are fully utilized. The best performance was given by the broadcast method, which can achieve a similar performance like a two-node synchronization, as the only time dependent on the number of nodes is the transmission of the data itself.

4 Phase II: Update of the Differences

Let us denote the complete set of the database records by $\rho = \{P_1, P_2, ..., P_K\}$, this is the set of packets (or records) that every node should have after synchronization, with $K = |\rho|$. Additionally, let us denote by ϕ_i the set of packets missed by node i. The set of records available at node i is S_i.

As mentioned previously, in a mesh network, the complexity will be at every node by monitoring the nodes to synchronize to; also, the processing power will be higher, because of the polynomial interpolation due to the one-to-one relationship.

For the synchronization process to take place, the nodes are re-arranged in a star network (i.e. Master-Slave). Otherwise, they can have a mesh network communication infrastructure. The rationale behind this topology is to allow network coding to reduce the bandwidth utilization and the delay and to minimize the complexity at the other nodes while keeping it at the central point.

Our algorithm allows to further reduce the traffic load related to the synchronization of multiple databases. This load is already minimized by CPISync for two synchronizing databases. Nevertheless, when harmonizing multiple databases, keeping the one-to-one relation introduces a large overhead, since synchronization will follow a mesh network topology. This overhead is reduced with our technique, which works as follows. First, one of the databases engaged in the synchronization process is selected as a master, say node 0 (whose set is S_0), to coordinate the overall reconciliation activity. The other nodes will be slaves. Secondly, the master will ask each slave to transmit the differences with its respective data by broadcasting its evaluation points to the other nodes, which is the fastest method according to Section 3. At this point, the master has a complete knowledge of what data every other database has and what it needs for a complete system synchronization. The master has now built the set ρ that is the complete database and he knows what every node is missing, i.e. he knows that node i is missing the set of records ϕ_i. The master builds the set of the records that are missed by at least one node, this set can be indicated as $\Psi = \rho \backslash \bigcap_{i=1}^{N-1} S_i$. Let $\Delta = |\Psi|$ be the number of records in this set. Finally, the master node linearly combines the packets required in the set Ψ using the RLNC technique and broadcast these coded packets to the slaves. In this way the number of packets that the master needs to send to update all slaves is drastically reduced; this will be shown in the next section.

5 Performance Evaluation

The performance of the proposed technique is analyzed in this section in terms of the average number of exchanged packets.

5.1 Average Number of Packets Exchange

In this section, the expected number of retransmissions ($\mathbb{E}[\mathcal{P}]$) will be evaluated according to two simulation schemes. In the first scheme (scheme A), the

databases to be synchronized have the same size (K) with a uniformly distributed set of differences. Scheme B, on the other hand, features the possibility that the synchronizing databases are of different sizes also having uniformly distributed differences. In the latter scheme, the different sizes will simulate the nodes being idle for some time.

In the two above mentioned schemes, the performance metric measure will be the expected number of packets exchange (i.e. the packets that have been interchanged during the synchronization process) in order to achieve a complete system harmonization. The simulations consider three techniques for synchronizing N datasets. The first one uses pure CPISync in a mesh network, where every node checks with all its peers about new information until all the nodes have the full set of packets (ρ). The average number of packets exchange grows with the square of the number of nodes (as previously explained):

$$\mathbb{E}\left[\mathcal{P}\right]_{\text{Mesh}} = \frac{\overline{d}}{1-\epsilon}\frac{N(N-1)}{2}, \tag{9}$$

where $\overline{d}$ is the average number of differences between any two synchronizing databases and ϵ is the PER.

The second method uses a star network, but without network coding, see 3.1. Instead, after the master pulls the differences from the other nodes, it broadcasts all the data packets (ρ) and each user will take or drop packets according to its need. The average number of packets exchange in this case is:

$$\mathbb{E}\left[\mathcal{P}\right]_{\text{BC}} = (N-1)\overline{d} + \frac{K}{1-\epsilon}. \tag{10}$$

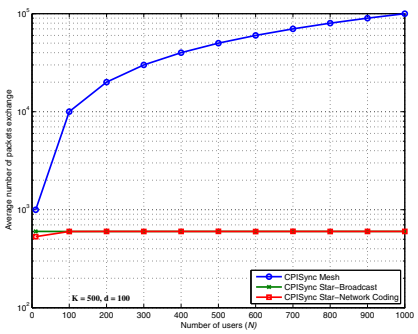
Finally, applying network coding techniques in a star network only on packets that have not been correctly received by all slaves. The expected number of packets exchange can be evaluated using:

$$\mathbb{E}\left[\mathcal{P}\right]_{\text{NC}} = (N-1)\overline{d} + \left[\frac{\Delta}{1-\varepsilon}\right]\delta, \tag{11}$$

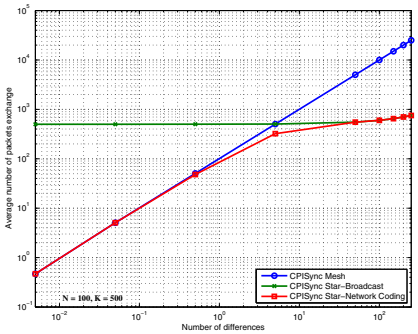
where Δ is the number of linear combinations with elements from a Galois field of $\mathbf{GF}\left(2^q\right)$ and δ is chosen as an optimal value of $\delta = \frac{1}{1-\epsilon}$ to overcome the link erasures. Additionally, a large $\mathbf{GF}\left(2^q\right)$ (with $q = 8$, i.e. a symbol = 8 bits) is chosen in order to reduce the probability of having linearly dependent equations and hence not ask for retransmissions, as proposed by [9]. Although it is very low, but there still exists a decoding failure probability (due to the linear dependency among the linear equations of the system), which is represented by ε.

Let us consider that $\hat{\varepsilon}$ is an upper-bound on ε, i.e., $\hat{\varepsilon} > \varepsilon$. This leads to identify an upper-bound on the average number of packets exchange for the network coding case of synchronizing N databases:

$$\mathbb{E}\left[\mathcal{P}\right]_{\text{NC}} \leq (N-1)\overline{d} + \frac{\Delta}{(1-\hat{\varepsilon})(1-\epsilon)} = \hat{\mathbb{E}}\left[\mathcal{P}\right]_{\text{NC}}. \tag{12}$$



(a) Average number of packets exchange as a function of N



(b) Average number of packets exchange as a function of number of differences

Fig. 3. Average number of packets exchange versus N and PER

5.2 Results

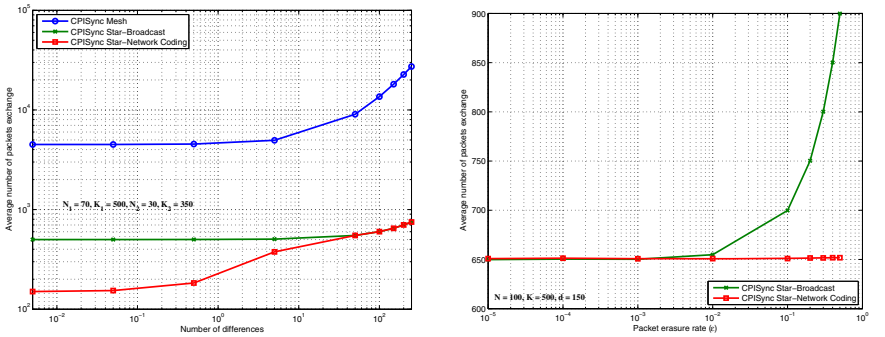
The following results represent the average number of packets exchange for synchronizing multiple databases after 100 runs.

Figure 3(a) plots the simulation results of (9), (10) and (11) as a function of N using scheme A. The graph shows that for small values of N, the performance of network coding is slightly better than the broadcasting scenario. However, using a mesh network the expected number of packets exchange grows logarithmic with the number of users. The simulation used a database size $K = 500$ records with 100 uniformly distributed differences and PER $\epsilon = 0$.

The chart in Figure 3(b) also simulates scheme A. But here, the results are shown as a function of the number of differences. The number of users N is set to 100, the database size $K = 500$ and PER $\epsilon = 0$. As it can be realized, linearly combining packets in a star network forms a lower bound for all other techniques. Further, as the number of differences gets really low or very high, the number of packets exchange is the same for a mesh or broadcast scenarios, respectively.

The plots in Figure 4(a) represent the case when some (say 30 %) of the total $N = 100$ databases were off-line for a period of time. Therefore, they are missing relatively large amount of data in addition to the existence of few differences of the information they acquire with the other databases. It is clear the inefficiency of the point-to-point synchronization in terms of bandwidth utilization. Also, with the network coding approach, the packets exchange is very low for low-to-moderate differences values. However, as the latter value increases, the number of transmitted packets in a network coding scheme is similar to the simple broadcasting scenario. Nevertheless, this behavior is due to the fact that no packet erasures on the channel were considered.

In the last simulation, shown in Figure 4(b), a realistic scenario is implemented using scheme A, where packet erasures with probability ϵ occur on the downlink from the master node. For $N = 100$, $K = 500$ and number of differences of



(a) Average number of packets exchange as a function of number of differences with different database sizes (K_1, K_2)

(b) Average number of packets exchange as a function of PER (ϵ)

Fig. 4. Average number of packets exchange versus number of differences and PER

150 with low-to-moderate PERs, both techniques (i.e. broadcasting and network coding) are comparable. However, for large values of ϵ, network coding is more appropriate and efficient because a single linear combination can reveal information about several missing native packets that have to be all retransmitted in case of simply broadcasting the information.

One last thing, by applying network coding, a coding delay ($D = D_e + D_d$) is added to the synchronization process. On the master side, the packets have to be encoded with delay denoted by D_e. Additionally, on the slave's side, a decoding delay D_d is realized due to the idle time the node has to wait to receive the coded block and to decode using Gaussian elimination techniques. However, compared to the RTT for requesting lost packets, the coding delay D is negligible. This is especially true for satellite networks with a high RTT. Further, it is shown in literature that the decoding complexity of RLNC is $\mathcal{O}\left(\Delta^3\right)$, which is applicable with nowadays technology.

6 Conclusions

In this work, a new approach to synchronize multiple databases was proposed based on network coding and an already state-of-the-art database synchronization mechanism, namely, CPISync. The newly recommended algorithm assumes that the master node, to coordinate the process of synchronizing all the databases involved, pulls from every slave node at a time its respective differences by using CPISync in a broadcast manner. Finally, when the big picture is clear, the master combines the packets that haven't been correctly received by all the users. As it has been clearly seen, using this approach, due to network coding lower capacity utilization was achieved. Further, because of the master-slave organization lesser CPISync processes were initiated, which could be accelerated by our broadcast method.

References

1. Tridgell, A.: Efficient Algorithms for Sorting and Synchronization, Ph.D. dissertation, The Australian National University (2000)
2. Agarwal, S., Starobinski, D., Trachtenberg, A.: On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices. IEEE Network 16(4), 22–28 (2002)
3. Fragouli, C., Le Boudec, J.-Y., Widmer, J.: Network Coding: An Instant Primer. SIGCOMM Comput. Commun. Rev. 36(1), 63–68 (2006)
4. Ho, T., Medard, M., Koetter, R., Karger, D., Effros, M., Shi, J., Leong, B.: A Random Linear Network Coding Approach to Multicast. IEEE Transactions on Information Theory 52(10), 4413–4430 (2006)
5. Ahlswede, R., Cai, N., Li, S.-Y., Yeung, R.: Network Information Flow. IEEE Transactions on Information Theory 46(4), 1204–1216 (2000)
6. Trachtenberg, A., Starobinski, D., Agarwal, S.: Fast PDA Synchronization Using Characteristic Polynomial Interpolation. In: Proceedings IEEE INFOCOM, New York City, NY, USA, vol. 3, pp. 1510–1519 (June 2002)
7. Tang, C., Donner, A., Chaves, J., Muhammad, M.: Performance of Database Synchronization Algorithms via Satellite. In: 2010 5th Advanced Satellite Multimedia Systems Conference (ASMS) and the 11th Signal Processing for Space Communications Workshop (SPSC), pp. 455–461 (September 2010)
8. Minsky, Y., Trachtenberg, A.: Practical Set Reconciliation, Boston University, Tech. Rep. (February 2002)
9. Liva, G., Paolini, E., Chiani, M.: Performance versus Overhead for Fountain codes over Fq. IEEE Communications Letters 14(2), 178–180 (2010)
10. Dimakis, A.G., Godfrey, P.B., Wu, Y., Wainwright, M., Ramchandran, K.: Network Coding for Distributed Storage Systems. IEEE Transactions on Information Theory 56(9) (September 2010)
11. Leong, D., Dimakis, A.G., Ho, T.: Distributed Storage Allocations for High Reliability. In: Proc. of the IEEE International Conference on Communications, ICC (2010)
12. Leong, D., Dimakis, A.G., Ho, T.: Distributed Storage Allocation Problems. In: Workshop on Network Coding Theory and Applications, NetCod (2009)
13. Dimakis, A.G., Ramchandran, K., Wu, Y., Suh, C.: A Survey on Network Codes for Distributed Storage. Proceedings of the IEEE 99(3) (March 2011)