Efficient Multiset Synchronization

Lailong Luo, Deke Guo, *Member, IEEE, ACM*, Jie Wu, *Fellow, IEEE*, Ori Rottenstreich, Qian He, Yudong Qin, and Xueshan Luo

***Abstract*— Set synchronization is an essential job for distributed applications. In many cases, given two sets A and B, applications need to identify those elements that appear in set A but not in set B, and vice versa. Bloom filter, a space-efficient data structure for representing a set and supporting membership queries, has been employed as a lightweight method to realize set synchronization with a low false positive probability. Unfortunately, bloom filters and their variants can only be applied to simple sets rather than more general multisets, which allow elements to appear multiple times. In this paper, we first examine the potential of addressing the multiset synchronization problem based on two existing variants of the bloom filters: the IBF and the counting bloom filter (CBF). We then design a novel data structure, invertible CBF (ICBF), which represents a multiset using a vector of cells. Each cell contains two fields, id and $count$, which record the identifiers and number of elements mapped into them, respectively. Given two multisets, based on the encoding results, the ICBF can execute the dedicated subtracting and decoding operations to recognize the different elements and differences in the multiplicities of elements between the two multisets. We conduct comprehensive experiments to evaluate and compare the three dedicated multiset synchronization approaches proposed in this paper. The evaluation results indicate that the ICBF-based approach outperforms the other two approaches in terms of synchronization accuracy, time-consumption, and communication overhead.**

***Index Terms*— Multiset synchronization, counting bloom filter, invertible bloom filter, invertible counting bloom filter.**

I. Introduction

CONSIDER a pair of hosts $Host_A$ and $Host_B$, each holding a set A and B. The goal of set synchronization for $Host_A$ and $Host_B$ is to calculate the differences between the two sets A and B then deduce the union $A \cup B$. In fact, set synchronization is a common and fundamental task in a variety of systems [1]. For example, in a distributed file system, files usually need to be duplicated for disaster recovery via set synchronization. In peer-to-peer networks [2], any pair of peers only needs to exchange those missing blocks of a file from each other. For wireless sensor networks [3], the sink node only needs to collect those unobserved results from other hosts. For software-defined networks (SDN) [4], flow tables generated by the controller must be synchronized with the corresponding switches in a timely manner. In this case, only those updated flow entries should be delivered. In cloud computing applications, local devices (smartphones, laptops, robotics, and wearable equipment) only upload or download the nonexistent data from the Cloud [5].

In the above cases, a straightforward method of set synchronization between any two hosts is to exchange all elements with each other. The amount of transferred data is proportional to the total number of elements at the two hosts [1]. This method is inefficient when the two sets differ in just a few elements. Moreover, such a method also incurs non-trivial and unnecessary communication overhead and additional latency. If the hosts cannot identify the difference between the involved sets, unnecessary transmission of the shared elements may burden the networks. Especially for the latency-sensitive applications and the bandwidth-scarce networks, the increasingly frequent synchronization can be fatal. What's worse, since the volume of the common elements in the sets is unpredictable, the hosts cannot manage the additional transmission cost.

Essentially, set synchronization can be classified into two categories. The first one is simple set synchronization, where both A and B are simple sets. By contrast, multiset synchronization concerns two multisets, each of which allows an element has multiple duplicates, i.e., the multiplicity of each element can be larger than one. Note that a multiset is a generalization of a set, and a simple set is a special case of a multiset where all elements only appear once.

Simple set synchronization is relatively easy since any element in the set is unique; hence, the difference between two simple sets stems only from the diverse elements that appear in exactly one of the sets. However, for multisets A and B, the difference between them stems from two sources. The first kind of difference is d_E, which denotes that the elements only exist in either A or B. Specifically, d_{E_A} denotes the elements that only exist in A, and d_{E_B} denotes that the elements that only exist in B. Thus, $d_E = d_{E_A} \cup d_{E_B}$. The second kind of difference, denoted as d_M, includes the elements which appear in both A and B, but with diverse multiplicities, i.e., number of

Manuscript received January 6, 2016; revised June 12, 2016 and September 26, 2016; accepted October 3, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Chen. Date of publication November 7, 2016; date of current version April 14, 2017. This work was supported in part by the National Natural Science Foundation for Outstanding Excellent Young Scholars of China under Grant 61422214, in part by the National Basic Research Program (973 program) under Grant 2014CB347800, in part by the National Natural Science Foundation of China under Grant 61661015, in part by the Program for New Century Excellent Talents in University, in part by the Hunan Provincial Natural Science Fund for Distinguished Young Scholars under Grant 2016JJ1002, and in part by the Research Funding of NUDT under Grant JQ14-05-02 and Grant ZDYYJCYJ20140601.

L. Luo, D. Guo, Y. Qin, and X. Luo are with the Science and Technology Laboratory on Information Systems Engineering, National University of Defense Technology, Changsha 410073, China (e-mail: luolailong09@nudt.edu.cn; dekeguo@nudt.edu.cn; qinyudong12@nudt.edu.cn; xsluo@nudt.edu.cn).

J. Wu is with the Department of Computer and Information Science, College of Science and Technology, Temple University, Philadelpha, PA 19122 USA (e-mail: jiewu@temple.edu).

O. Rottenstreich is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: orir@cs.princeton.edu).

Q. He is with the Key Laboratory of Cloud Computing and Complex System, Guilin University of Electronic Technology, Guilin 541004, China (e-mail: heqian@guet.edu.cn).

Digital Object Identifier 10.1109/TNET.2016.2618006

TABLE I
THE APPLICABILITY OF EXISTING SIMPLE SET SYNCHRONIZATION METHODS

Data structure	BF	ComBF	CBF	SBF	ShBF
Representation	No	No	Yes	Yes	Yes
Inverse decoding	No	No	No	No	No
Identifying d_E	No	No	No	No	No
Data structure	CM	DCF	IBF	IBLT	ICBF
Representation	Yes	Yes	No	No	Yes
Inverse decoding	No	No	Yes	Yes	Yes
Identifying d_E	No	No	No	No	Yes

duplicates of an element. Typically, d_{M_A} denotes the common elements such that their multiplicities in A are larger than those in B. Meanwhile, d_{M_B} denotes the common elements such that their multiplicities in A are less than those in B. Thus, $d_M = d_{M_A} \cup d_{M_B}$. An efficient multiset synchronization method should identify d_E and d_M. Then, to save bandwidth, only the elements in d_E should be transmitted.

Some lightweight methods are proposed for efficient set synchronization. The insight is to employ bloom filters (BF) [6] and their variants, e.g., counting bloom filter (CBF) [7], compressed bloom filter [8], invertible bloom filter (IBF) [9], and invertible bloom lookup table (IBLT) [10]. Such methods represent all elements of a set using a vector of cells, each of which can be one bit or a dedicated data structure. After exchanging the resultant bloom filters of two sets, those unique set elements can be identified via a process of operations, such as query and subtracting. Consequently, large numbers of common elements are not required to be delivered to each other.

We argue that the existing BF-like synchronization methods for simple sets are inapplicable to the multiset synchronization problem. Table I shows the details. Among BF-like structures, only CBF and its variants (including Spectral Bloom Filter (SBF) [12], Dynamic Count Filter (DCF) [13], [14], Shifting Bloom Filter (ShBF) [15], and Count Min Sketch (CM) [16]), can represent element multiplicity in a multiset by augmenting the bit in each cell to be an integer, but they are not inversely decodable, and cannot distinguish d_E from d_M. IBF and IBLT can be decoded inversely, but they fail to represent multiset, since the XOR operations will eliminate it when the element is mapped into the same cells again.

Accordingly, in this paper, we first confirm the potential of synchronizing multisets with CBF and IBF by following a common framework. CBF can record the multiplicity of each set element and a query-based decoding process can finally discover the different elements between two multisets. However, the CBF-based method cannot decode the elements inversely. For the IBF-based method, the resultant IBF after subtracting one IBF from another one may decode the different elements in a recursive manner. In each round, the IBF only decodes those elements in the root sets (see Definition 2) of the current two multisets. After each round of decoding, the two multisets will be updated via eliminating the root sets from them. In this recursive method, multiset synchronization can be realized with high probability. However, the IBF-based method invokes the IBF processes round by round, and thus suffers from a massive computation time. Note that both the IBF-based method and the CBF-based method cannot distinguish d_E and d_M, and thus they will suffer from vast communication overhead. The intrinsic reason is that they must query all elements to uncover d and treat d_E and d_M with no difference.

To avoid the inherent weakness of the CBF and IBF-based methods, we design a novel data structure, invertible counting bloom filter (ICBF), which consists of a vector of cells. Each cell contains two fields, i.e., the ***id*** and ***count***, which record the element mapped into that cell and its multiplicity, respectively.

We then propose an efficient method based on ICBF for the multiset synchronization problem. Our method depends on three operations for ICBF: the encoding, subtracting, and decoding. For the encoding operation, a family of independent hash functions are utilized to map each element of a multiset into the cell vector, and a special identifier mechanism is introduced to identify this element. For two ICBFs, the subtracting operation eliminates those common elements, and results in a new ICBF. Accordingly, the resultant ICBF can decode all elements from its cell vector via referring to the local ***id table***, which records the mapping relationship of ***id*** and the real content of each element. To be specific, ICBF records the multiplicity of an element with its $count$ field, decodes the elements inversely from the cells with the help of id field and the local $id\ table$, and distinguishes d_E from d_M with joint consideration of $count$ and id in $ICBF_C$ and $ICBF_{C'}$. Consequently, only the elements in d_E will be transmitted to the other host.

Furthermore, we conduct comprehensive experiments to evaluate the performance of the ICBF-based and IBF-based methods. The results indicate that our ICBF-based method achieves better accuracy and incurs much less time-consumption than the IBF-based method. We also measure the additional communication overhead when the multiplicities follow different distribution patterns. We find that our ICBF-based method outperforms the IBF-based method, and requires much less communication overhead. The major contributions of this paper can be summarized as follows:

- We propose a novel data structure called ICBF to represent a multiset. Furthermore, we design an efficient method based on ICBF to synchronize a pair of multisets.
- We reveal that the existing CBF and IBF can theoretically realize multiset synchronization. Accordingly, we propose dedicated multiset synchronization methods based on CBF and IBF, respectively.
- Comprehensive experiments demonstrate that the ICBF-based method outperforms the IBF-based methods in terms of the synchronization accuracy, time-consumption, and communication overhead.

The remainder of this paper is organized as follows. Section II summarizes preliminaries about multisets and bloom filters. Section III reports the CBF-based and IBF-based methods for multiset synchronization. Section IV introduces a new data structure, ICBF, and accordingly designs an efficient and accurate multiset synchronization method. Section V evaluates the performance of proposed synchronization methods. We then elaborate on the related works in Section VI,

TABLE II
SYMBOLS AND NOTATIONS

Term	Definition
A, B	Two multisets that need synchronization
$Host_A$	Hosts that hold multiset A
A^*, B^*	Root set of A and B
n_A, n_B	Number of elements in A^* and B^*
$C(A)$	Cardinality of A
$m_A(x)$	The multiplicity of x in A
k	Number of hash functions
m	Number of cells in bloom filters and its variants
CBF_A	Encoding result of A for CBF
IBF_A	Encoding result of A for IBF
$ICBF_A$	Encoding result of A for ICBF
IBF_{C*}	Subtracting result of IBF_{A*} and IBF_{B*}
E_A	Elements sent from $Host_A$ to $Host_B$
E_B	Elements sent from $Host_B$ to $Host_A$
d_E	Difference due to the diverse elements
d_M	Difference due to the distinct multiplicities

and discuss several practical concerns in Section VII. Finally, we conclude this paper in Section VIII.

II. PRELIMINARIES

We introduce the basic concept of the multiset and then summarize the standard bloom filters and two of its variants in this section. The important symbols and notations in this paper are given in Table II.

A. Multiset

Unlike a simple set, a multiset allows an element to appear multiple times [18]. To characterize the features of a multiset, several parameters can be employed to describe a multiset.

Definition 1: Let x be an element of a multiset A. The multiplicity of x is denoted by $m_A(x)$, which denotes the number of instances of x in A.

Definition 2: Given a multiset A, a simple set A^* is defined as the root set of A such that $A^* = \{x \in A | m_A(x) > 0\}$. Hence, different multisets might have the same root set.

Definition 3: Let $C(A)$ denote the cardinality of a multiset A, i.e., the sum of multiplicity of each element. We have $C(A) = \sum_{i=1}^{n} m_A(x_i)$, where x_i is an element of A^* and n denotes the cardinality of the root set A^*.

According to such definitions, a multiset A can be characterized as its root set and the multiplicity of each element. That is, A can be represented as a set of pairs, i.e., $A = \{\langle x_1, m_A(x_1)\rangle, \cdots, \langle x_i, m_A(x_i)\rangle, \cdots\}$. For example, $A = \{\langle x, 3\rangle, \langle y, 2\rangle, \langle z, 1\rangle\}$ stands for $\{x, x, x, y, y, z\}$.

B. Bloom Filters

Given a set $A = \{x_1, x_2, ..., x_n\}$ with n elements, a bloom filter (BF) [6] represents such n elements with a bit vector of length m. All of m bits in the vector are all initially set to 0. A group of k independent hash functions, $<h_1, h_2, ..., h_k>$, are employed to randomly map each set element into k positions, $<h_1(x), h_2(x), ..., h_k(x)>$, in the bit vector. Those bits at such k positions in the vector are all set to 1. In the same way, all the elements can be represented by the same BF. According to the m-bit vector and the k hash functions, we can realize the membership query against any element. If any bit at the k hashed positions of the element is set to 0, the BF judges that this element does not belong to the set. Otherwise, the BF believes that the queried element belongs to the set with a low probability of false positive. That is, for an element not in the set, all of its k hash positions in the bit vector may be 1, due to the unavoidable hash conflicts.

Bloom filters have been used in many fields [19]–[21]. Regarding various applications, several variants are proposed to make them more effective and efficient. We further discuss two mainstream variants, counting bloom filters and invertible bloom filters.

C. Counting Bloom Filters

One drawback of bloom filters is that they are only suitable for static sets, while a dynamic set has to tackle the element insertion and deletion operations. It is clear that bloom filters naturally support the element insertion operation by setting the hashed k bits to 1. It, however, cannot simply reset all the hashed bits to 0 when an element is removed from the corresponding set, since the k hash bits may be shared by other elements in the set, so resetting can lead to false negatives.

To address this issue, the counting bloom filter [7] was proposed to improve the bloom filters. It naturally supports the deletion and insertion of any element by replacing each bit in the vector with a counter consisting of multiple bits. In this way, the value of each cell can exceed 1. Assume element x is hashed into the 4^{th}, 10^{th}, and 15^{th} cells, while element y is hashed into the 5^{th}, 15^{th}, and 24^{th} cells, respectively. Consequently, the $count$ value is 1 for the 4^{th}, 5^{th}, 10^{th}, and 24^{th} cells, but is 2 for the 15^{th} cell. If the element x is deleted from the set, the values of the 4^{th}, 10^{th}, and 15^{th} cells are decreased by 1, but the value of the 15^{th} cell is still positive other than 0. That is, a query of y would result in a correct positive indication, since the membership information of element y is still kept in the updated CBF.

Several variants of CBF have been proposed to optimize the size of each cell, or enhance the query accuracy, e.g., SBF [12], DCF [13], ShBF [15], and CM [16]. SBF and DCF adjust the size of the used bits in each cell according to the maximum multiplicity of elements. By contrast, ShBF and CM are devoted to a more efficient query. Besides, the false negative problem of CBF has been well discussed [17].

D. Invertible Bloom Filters

It is well-known that a bloom filter cannot decode those elements represented by its bit vector, due to the use of one-way hash functions. To enable the set synchronization, the query-based method is used to identify and then exchange different elements between two sets. The insight is to query a BF of one set against each element in another set. Such a query-based synchronization method is inefficient and time-consuming. Invertible bloom filters (IBF) [9] extend bloom filters from several aspects such that the subtracting operation of IBFs for two sets provides the opportunity to decode those

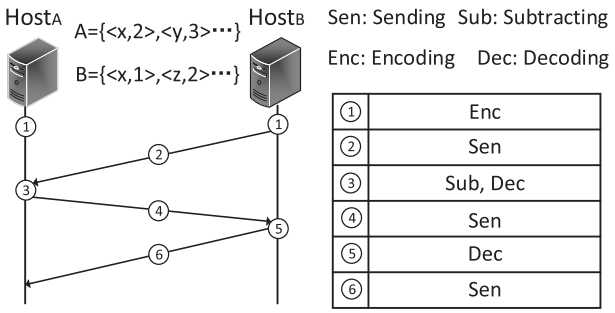


Fig. 1. An illustrative example of the proposed framework.

different elements between the two sets. Each cell of IBF contains three fields:

- $idSum$, the XOR of element ids hashed into that cell.
- $hashSum$, the XOR of elements mapped by an extra hash function g into that cell.
- $count$, the amount of elements hashed into that cell.

To decode the set elements from an IBF with a high probability, its parameters have to be configured carefully. Let d denote the amount of total difference between two sets. The number of cells for an IBF is $m = \alpha \cdot d$, where $\alpha > 1$. It has been also confirmed that 3 or 4 hash functions are sufficient in practice [9]. Consequently, the communication overhead of elements is $O(d)$, and the computation complexity of synchronization is $O(n + d)$, where n is the amount of associated elements, if we do not consider the overhead due to estimating the total number of different elements [9].

III. Multiset Synchronization via Variants of Bloom Filters

In this section, we start with a framework of multiset synchronization. We then explore how IBF and CBF can be used to realize multiset synchronization based on this framework.

A. Framework of Multiset Synchronization

Assume that two hosts, $Host_A$ and $Host_B$, host two multisets, A and B, respectively. Multiset synchronization means to identify and exchange those different elements between A and B such that the two hosts will have the same elements with the same multiplicity at last. The multiset synchronization must handle both the d_E and d_M. For example, for two multisets, $A = \{\langle x, 1\rangle, \langle y, 2\rangle, \langle z, 3\rangle\}$ and $B = \{\langle y, 1\rangle, \langle z, 2\rangle, \langle w, 1\rangle, \langle u, 2\rangle\}$, the result of $A - B$ is a new multiset: $\{\langle x, 1\rangle, \langle y, 1\rangle, \langle z, 1\rangle, \langle w, -1\rangle, \langle u, -2\rangle\}$. Similarly, the result of $B - A$ can be easily derived as: $\{\langle x, -1\rangle, \langle y, -1\rangle, \langle z, -1\rangle, \langle w, 1\rangle, \langle u, 2\rangle\}$.

To enable fast and accurate multiset synchronization, we propose a general framework for the design of multiset synchronization methods as depicted in Fig. 1. The framework contains three processes: encoding, subtracting, and decoding. Given $Host_A$ and $Host_B$, the following six steps are required to accomplish the multiset synchronization.

1) Each host executes the encoding operation, which maps its elements to the cell vector by the k independent hash functions.
2) $Host_B$ sends the encoding result of B to $Host_A$.
3) Given the two encoding results, $Host_A$ executes the subtracting and decoding operations to derive those elements that need to be transmitted to $Host_B$.
4) $Host_A$ sends the subtracting result of the two encoding results, as well as E_A, to $Host_B$.
5) $Host_B$ receives E_A and the subtracting result; it then executes the decoding operation to identify the elements in E_B, which need to be sent to $Host_A$.
6) $Host_B$ sends E_B to $Host_A$ and the synchronization will be accomplished.

Note that, for those elements in d_M, a dedicated number of replicas will be generated to ensure the consistency of A and B. This general framework will guide the design of dedicated multiset synchronization methods. They usually differ in the encoding, subtracting, and decoding operations. In the following sections, we evaluate the possibility of realizing multiset synchronization using the variants of bloom filters. Note that, the standard bloom filter is not feasible since it neither records the multiplicity of each element nor is inversely decodable. For this reason, we only explore IBF and CBF to realize the multiset synchronization.

B. The IBF-Based Synchronization Method

Although IBF is very efficient in enabling the synchronization of simple sets, it cannot be directly used to address the synchronization problem of multisets. Recall that the $idSum$ field of each cell in IBF records the XOR result of those elements, which are mapped into that cell by one of the k independent hash functions. It is easy to find that the XOR operation ensures that each $idSum$ field can record an element only once. If the multiplicity of an element is even, the XOR operation makes the $idSum$ fields of k involved cells eliminate this element. For the same reason, the $idSum$ field of cells will record an element only once if its multiplicity is odd. As a consequence, the IBF cannot directly represent a multiset, not to mention realize the multiset synchronization.

Fortunately, it is reasonable to view a multiset as the "union" of several simple sets. For example, a multiset $A = \{\langle x, 1\rangle \langle y, 2\rangle \langle z, 3\rangle\}$ can be considered as the "union" of three simple sets, i.e., $A = \{x, y, z\} \uplus \{y, z\} \uplus \{z\}$. Inspired by this observation, IBF may successfully enable the multiset synchronization by performing the traditional set synchronization round by round. Basically, the IBF-based method also calls for encoding, subtracting, and decoding operations to identify the different elements during each round.

The functionality of IBF stems from three operations: the encoding, the subtracting, and the decoding. During the encoding process, for any element $x \in A$, k hash functions are employed to map x to the cell vector. The $idSum$ field of the mapped cell demonstrates the XOR of elements that are hashed into this cell. Meanwhile, the $hashSum$ field of the mapped cell records the output of an additional hash function of x for the purpose of checking during the decoding process. To subtract those different set elements, IBF performs the XOR operation on the encoding results of sets A and B

to eliminate the common elements between the two sets. Consequently, the resultant IBF vector just represents the information of those different elements.

Then we can simply decode those different elements. The basic idea is to locate a cell with only one element denoted as x. We then insert x again into the resultant IBF. The XOR operation will eliminate the encoded information of x at the involved k cells. By executing this operation recursively, IBF will finally decode all distinct elements with high probability. However, as discussed in [9], IBF may fail to find a cell with only one element in each round, even after carefully setting parameters. The above method can be embodied in the following steps.

Note that, the IBF-based method also follows the general framework in a recursive manner. $Host_A$ and $Host_B$ map their root sets A^* and B^* into IBF_{A^*} and IBF_{B^*} via k independent hash functions, respectively (Step 1 of the framework). Then $Host_B$ sends IBF_{B*} to $Host_A$ (Step 2 of the framework) and $Host_A$ executes the subtracting and decoding operations to identify the elements in E_A in this round (Step 3 of the framework). The next step is that, according to the framework, $Host_A$ sends the elements in E_A as well as the subtracting result IBF_{C^*} to $Host_B$ such that those elements only appearing in A^* are synchronized to B (Step 4 of the framework).

Upon receiving IBF_{C^*}, $Host_B$ executes the IBF decoding operation [9], thus deriving the elements in E_B of this round (Step 5 of the framework). At last, $Host_B$ delivers the elements in E_B to $Host_A$ such that $A^* = B^*$ eventually (Step 6 of the framework). In this way, the six steps in the framework are accomplished and the IBF-based method synchronizes A^* and B^* with dedicated probability. This method will continue to be performed in the next round, and can only be terminated when at least one set is identified as empty.

For two multisets $A = \{\langle x, 1\rangle, \langle y, 2\rangle, \langle z, 2\rangle, \langle w, 1\rangle\}$ and $B = \{\langle y, 1\rangle, \langle z, 1\rangle, \langle w, 3\rangle, \langle u, 2\rangle\}$, we illustrate an example to clarify the aforementioned synchronizing processes. Their root sets are $A^* = \{x, y, z, w\}$ and $B^* = \{y, z, w, u\}$, respectively. In the first round, the different elements of the two root sets are identified as $\{x, u\}$. After deleting the root sets, the original two multisets are updated as $A = \{\langle y, 1\rangle, \langle z, 1\rangle\}$ and $B = \{\langle w, 2\rangle, \langle u, 1\rangle\}$.

We then recursively execute the synchronization process. After one more round, set A becomes empty while set B is $\{\langle w, 1\rangle\}$. Thus, this method will terminate with the identified difference between the original A and B, i.e., $\{\langle x, 1\rangle, \langle y, 1\rangle, \langle z, 1\rangle, \langle w, 2\rangle, \langle u, 2\rangle\}$. To be specific, the IBF-based method derives $d_{A_1} = \{x\}$ and $d_{B_1} = \{u\}$ in the first round of decoding. In the second round, the decoding operation deduces the different elements as $d_{A_2} = \{y, z\}$ and $d_{B_2} = \{w, u\}$. Note that, the third round will not be executed, since A is already empty, and $Host_B$ will send the remained element, i.e., w, to $Host_A$ directly. In this case, $d_M = \{\langle y, 1\rangle, \langle z, 1\rangle, \langle w, 2\rangle\}$, and $d_E = \{\langle x, 1\rangle, \langle u, 2\rangle\}$. However, the IBF-based method fails to distinguish d_M from d_E, as a result, all the different elements must be transmitted between $Host_A$ and $Host_B$.

This special method, however, still suffers from other weaknesses. First, it is inefficient and must be executed for $\eta = \min\{\eta_A, \eta_B\}$ rounds, where η_A and η_B denote the maximum multiplicity of elements in A and B, respectively. The computation complexity of each round is $O(n + d)$ [9], where n is the amount of associated elements. Moreover, the parameters of IBF have to be reset in each round, according to the estimated size of the difference between updated A and B. Such an estimation process will bring additional computation overhead. Although accurate estimation can improve the decoding probability of this method, it cannot absolutely guarantee the success of the entire decoding process. The reason is that decoding failures may occur during each round. This weakness stems from the IBF mechanism even under the best setting of parameters [9].

Let p_i for $1 \leq i \leq \eta$ denote the probability that the i^{th} round of IBF process successfully decodes the difference. The probability of a successful multiset synchronization is given by $p = \prod_{i=1}^{\eta} p_i$ and is much less than the p_i of any round. For example, even if each round can succeed with $p_i = 0.99$, the probability of a successful multiset synchronization is only $(0.99)^{20} \approx 0.818$ when $\eta = 20$. That is, the IBF-based method is not suitable for high multiplicity multisets.

C. The CBF-Based Synchronization Method

Our research finds that CBF is a candidate for realizing the multiset synchronization, since it offers an indirect way to record the multiplicity of each element. After encoding the element $x \in A$, the minimum $count$ among $CBF_A[h_1(x)], \cdots, CBF_A[h_k(x)]$ will be regarded as an estimation for $m_A(x)$ [7]. Conflicts may occur in such k dedicated cells, because other elements may be mapped into these cells too. However, the probability that all the k cells experience conflict at the same time is negligible. Thus, the minimum value among such cells for an element can tell the correct multiplicity of x with high probability.

Note that, such a CBF-based method still follows the set synchronization framework proposed in Fig. 1. Two hosts, $Host_A$ and $Host_B$, first encode their multisets as CBF_A and CBF_B by using CBF, respectively. For any $x \in A$ with a multiplicity β, the k cells $h_1(x), \cdots, h_k(x)$ in CBF_A will be increased by β. The same result holds for any element in multiset B. Then $Host_B$ sends the resultant CBF_B to $Host_A$. The subtracting of CBF_B from CBF_A can be easily implemented by $Host_A$ since each cell in CBF has only one field, i.e., the $count$. It records how many elements have been mapped into that cell. Let CBF_C denote the new CBF resulting from the subtracting operation. The value of each cell in CBF_C is simply the minus result of the corresponding $count$ in CBF_A and CBF_B.

Apparently, there may be conflicts due to the hash functions, i.e., different elements are mapped into the same cells, with a rather small probability. Note that the resulting CBF_C is significantly affected by the hash conflicts. If there are no conflicts for both CBF_A and CBF_B, a zero cell means that all of elements in A and B are not mapped into that cell,

Algorithm 1 Decoding Operation of CBF at $Host_A$

Require: The subtracting result $CBF_C = CBF(A) - CBF(B)$, any element $x \in A$ and k hash functions.

```
flag ← 0;
for x ∈ A do
    for i = 0 to k − 1 do
        Calculate the hash value of h_i(x);
        if CBF[h_i(x)].count>0 then
            flag++;
    if flag==k then
        add x into E_A for later transmission;
return E_A;
```

or x is mapped into that cell but $m_A(x) = m_B(x)$. When the value of a cell is a positive integer, it demonstrates that for the element x that has been mapped into this cell, $m_A(x) > m_B(x)$. Otherwise, if the value is a negative integer, it implies $m_A(x) < m_B(x)$.

However, if any i-th cell occurs conflict either in CBF_A or CBF_B, the corresponding cells in CBF_C will be meaningless. If both x and y are hashed into the 5^{th} cell in CBF_A and CBF_B, then the value of the 5^{th} cell in CBF_C is decided by the difference of $m_A(x) + m_A(y)$ and $m_B(x) + m_B(y)$. If $m_A(x) + m_A(y) > m_B(x) + m_B(y)$, then $CBF_C[4]$ is a positive integer. Meanwhile, $CBF_C[4]$ is negative when $m_A(x) + m_A(y) < m_B(x) + m_B(y)$ and 0 when $m_A(x) + m_A(y) = m_B(x) + m_B(y)$. As a result, we cannot judge which one is larger among $m_A(x)$ and $m_B(x)$ as well as $m_A(y)$ and $m_B(y)$. This is because the probability of conflict-free encoding of CBF is the same as the invertible counting bloom filters that we will introduce later. We discuss the probability issue and our solution in Section 4.5.

$Host_A$ and $Host_B$ encode the elements of A and B via the same set of hash functions. After receiving the CBF_B, $Host_A$ calculates the result of $CBF_C = CBF_A - CBF_B$ via performing the minus operations on corresponding cells in CBF_A and CBF_B. However, unlike IBF, CBF is not inversely decodable. Accordingly, we cannot derive the difference between multisets A and B just according to the resultant CBF_C. Thus, we still employ the query-based mechanism to identify those different elements.

As depicted in Algorithm 1, at $Host_A$, for an element $x \in A$, the decoding process maps x by the k hash functions. If the values in the resulting cells, i.e., $CBF_C[h_1(x)], CBF_C[h_2(x)], \cdots, CBF_C[h_k(x)]$, are all positive, then x will be added into E_A, which will be further delivered to $Host_B$. In this way, all elements in E_A can be derived. E_A and CBF_C will be sent to $Host_B$ according to the fourth step in our general framework. Similarly, $Host_B$ will execute the same algorithm to distinguish the elements in E_B based on CBF_C. Note that E_B includes any element x which satisfy the constraint that $CBF_C[h_1(x)], CBF_C[h_2(x)], \cdots, CBF_C[h_k(x)]$ are all negative rather than positive.

After identifying E_A and E_B, they should be delivered to $Host_B$ and $Host_A$, respectively. Note that, all of elements in E_A and E_B only need to be transmitted once. To complete the process of multiset synchronization, $Host_B$ queries each element in $x \in E_A$ from CBF_C to derive how many replicas of x should be created to keep accordance with A. For instance, if $Host_B$ queries an element $x \in E_A$ against CBF_C, and finds out that the minimal value among $CBF_C[h_1(x)], CBF_C[h_2(x)], \cdots, CBF_C[h_k(x)]$ is 3, then $Host_B$ will add x and 2 extra replicas of x into B. Similarly, Given CBF_C and E_B, $Host_A$ will execute the query operations to derive how many extra replicas of $x \in E_B$ should be added into A so that we get $A \cup B$ eventually.

Such a method, however, suffers non-trivial time consumption. $C(A^*)$ and $C(B^*)$ queries have to be executed at $Host_A$ and $Host_B$ during the decoding process, respectively. Moreover, the CBF suffers the inherent false positive probability, which may cause a serious negative impact on the performance of multiset synchronization. Given the value of k and the number of elements, the only way to lessen the false positive probability is to utilize more cells. However, more cells will cause more storage overhead. Note that, to minimize the transmission overhead, only those diverse elements should be transmitted, while those common elements with distinct multiplicities can be synchronized by generating a dedicated number of replicas at each host. However, the CBF-based method fails to distinguish d_M from d_E, and thus it incurs unnecessary transmission costs.

For example, given $A = \{\langle x, 1\rangle, \langle y, 3\rangle, \langle z, 1\rangle\}$ and $B = \{\langle y, 1\rangle, \langle z, 2\rangle\}$, $Host_A$ decodes CBF_C and derives that $A - B = \{\langle x, 1\rangle, \langle y, 2\rangle, \langle z, -1\rangle\}$. Then $Host_A$ sends x and y to $Host_B$ because the multiplicity of both x and y are positive in CBF_C. Similarly, $Host_B$ delivers z, whose multiplicity in CBF_C is negative, to $Host_A$ to realize synchronization. However, the fact is that y should not be transmitted to $Host_B$ and z need not to be delivered to $Host_A$ since y and z have already existed in $Host_B$ and $Host_A$, respectively.

In summary, IBF and CBF can indirectly solve the multiset synchronization problem. The CBF based methods are not inversely decodable; hence, a query-based method is the only choice to recognize the elements in the cells. The IBF-based method is inversely decodable but must be executed for multiple rounds to achieve multiset synchronization. So, they suffer from high time complexity and limited accuracy. Besides, the IBF-based method and the CBF-based method fail to distinguish d_E and d_M; thus, they suffer an additional communication overhead. Thus, we need another novel data structure to represent the multiset and detach d_M from d_E to realize fast and accurate multiset synchronization.

IV. Invertible Counting Bloom Filters for Multiset Synchronization

In this section, we propose a novel dedicated data structure, called invertible counting bloom filters (ICBF), to represent a multiset and tackle the multiset synchronization problem.

A. Invertible Counting Bloom Filters

We find that each *count* in CBF is capable of recording how many elements have been mapped into it. That is,

CBF can record the multiplicity of each element in a multiset via the $count$ value in each cell. It, however, fails to decode the elements from the counter vector, due to the well-known "one way property" of hash functions. That is, a CBF requires additional information besides the counter vector to decode all of the elements in a set represented by it. For this reason, we design the invertible counting bloom filters (ICBF) with a new cell vector, each of which contains a ***count*** field and an extra ***id*** field.

For each cell, the ***id*** field is responsible for recording a generated identifier for each element that has been hashed into the cell. The ***count*** field memorizes how many elements have been encoded into that cell, i.e., the multiplicity of a particular element if there are no conflicts. For an element x, we employ the minimal value among $ICBF_A[h_1(x)].count, \cdots, ICBF_A[h_k(x)].count$ as $m_A(x)$. What is more, we also measure the probability of such a conflict-free event, and adjust m and k to ensure high conflict-free probability in Section 4.5.

To decode the elements from a cell vector inversely, the identifier of each element should not only record the element but also identify which set the element belongs to. Thus for the ICBF-based method, the identifier of any set element is constructed as two parts. The prefix part identifies the set this element belongs to, while the suffix part indicates the element itself. The details about the identifier will be introduced later in Section 4.2.

Note that there exist potential conflicts under the encoding process, i.e., multiple diverse elements may be hashed into a same cell with a given probability. If the number of cells is m, the probability that two diverse elements are mapped into a same cell is given by $1/m$. Indeed, bloom filters and their variants always suffer this weakness. ICBF, however, can tackle the potential hash conflicts via the id field. For example, an element x has been hashed into the i^{th} cell of a ICBF. When the ICBF encodes another element y into the i^{th} cell, the $count$ field needs to be increased but the id field remains unchanged. That is, $CBF_A[i].count = m_A(x) + m_A(y)$.

It is clear that ICBF can naturally support the element insertion, fast membership-query. However, ICBF can not only realize the element deletion operation, but also inversely decode the elements. Furthermore, we show in Section 5 that ICBF consumes fewer storage resources and incurs less communication overhead than IBF; this is because it employs only two fields, while IBF needs three fields.

B. The Identifier Generation Mechanism an Element

Given two multisets, A and B, id assignment of each element in A or B only cares about the value of $C(A^*)$ or $C(B^*)$ (the cardinality of the multiset), rather than that of $C(A^*)+C(B^*)$. For any element in set A or set B, we employ the first digit (the leftmost digit) of its id to record the set it belongs to, i.e., the red digit in Fig. 2. We use other binary bits of an identifier to distinguish each element in a multiset. The length of such an identifier is determined by the cardinality of the root set of each multiset. For example, in Fig. 2, a 2-digit identifier (the black digits in the id field) is employed for A,

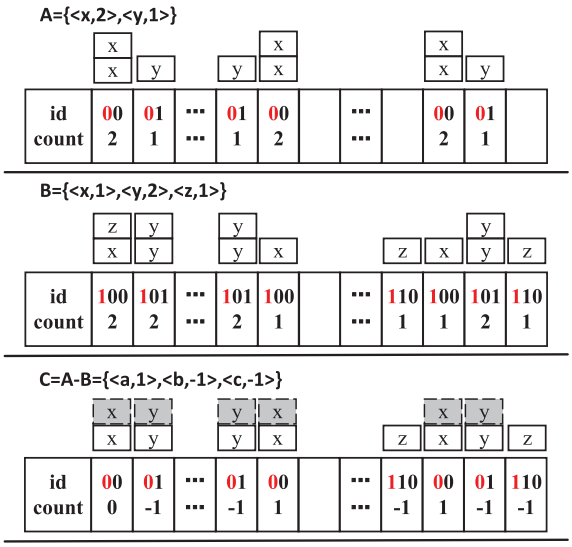


Fig. 2. An example of ICBF encoding and subtracting process. For simplicity, the bits that used to label the empty cells are omitted.

since the root set A^* only has three elements. In this way, our ids can identify each element in any multiset and can be used to realize the subtracting and decoding operations. For instance, in Fig. 2, the red 0 means that all related elements belong to A, while the red 1 declares that this id is used for elements in B.

The identifier of each element is represented as binary bits in memory. Hence, one vital issue is how to distinguish the empty cells with non-empty cells in ICBF. In real implementation, there are two differentiated solutions. One possible strategy is to augment an additional bit as the flag. If the cell is non-empty, this flag will be set as 1; otherwise, it is set as 0. In this way, the algorithm will check the flag during synchronization. In contrast, another solution is to leave a default suffix (e.g., $00\cdots00$) of the identifier as the sign of empty cells. For instance, if there are 3 diverse elements in a multiset, the suffix 00 implies the empty cells. By contrast, 01, 10, and 11 represent the 3 elements, respectively. We prefer the first strategy since only one bit must be checked, which will significantly ease the cost of distinguishing the empty cells.

C. id table at Each Host

To accomplish the synchronization, $Host_A$ ($Host_B$) transmits the elements in E_A (E_B) to $Host_B$ ($Host_A$). However, the ICBF can only derive the ids of these elements. Hence, to know what the content of an id really refers, we maintain an id $table$ at each host to record the mapping relationship between each element and its identifier. When our algorithms need to know the original element behind an id, they can refer to the id $table$ at that host. By maintaining the relationship between ids and elements locally, the synchronization will incur less bandwidth than IBF does, which remotely transmits the original elements between hosts.

Undoubtedly, the introduction of the id $table$ will bring extra storage overhead. However, we argue that, the storage overhead is controllable and acceptable. Firstly, the id $table$

TABLE III
THE SPACE OVERHEAD OF $id\ table$ (MB)

n	1000	3000	5000	7000	9000	11000
Array	0.064	0.192	0.321	0.450	0.580	0.708
Hash Table	1.875	5.625	9.375	13.125	16.875	20.625

only records the information of local elements, and never cares about the elements at other remote hosts. Secondly, the existing key-value storage techniques can be employed to optimize the storage strategy and speed up the query request, e.g., Dynamo [23], Redis [24], and Memcached [25]. Thirdly, instead of saving the original content of elements with the $id\ table$, we prefer to store the location or directory of elements in the local file system.

To quantify the space overhead, we consider two storage strategies, i.e., storing $id\ table$ with an array and storing the $id\ table$ with a hash table (Dynamo, Redis, and Memcached). For the array, the space overhead is $n \cdot (\lceil (log_2 n) \rceil + 1 + \bar{s})/8$ (MB), where n is the number of elements in the root set, and $\bar{s}$ denotes the maximum number of bits that the directory used. As for the hash table, the required space overhead can be calculated as $m \cdot \bar{s}$ (MB), where m is the number of cells the hash table has. Table III records the space overhead of both storage strategies, when $\bar{s} = 500\ bit$ and $m = 30n$. Apparently, with the increase of n, both strategies cost more space overhead, and storing the $id\ table$ with array consumes much less space overhead. But we argue that, the time-complexity of querying an element in an array and a hash table is $O(n)$ and $O(1)$, respectively. Hence, to speed up the synchronization process, storing the $id\ table$ with the hash table is more advisable, and the resulted space overhead is acceptable for nowadays' hosts.

D. The ICBF Operations for Synchronizing Multisets

Note that the synchronization result of two multisets is the union of them at both hosts. For any element x appearing in both multisets, we generate $|m_A(x) - m_B(x)|$ replicas of x at the host that holds the less multiplicity of x, such that $m_A(x) = m_B(x)$. Besides, for those elements appearing only in one multiset, they need to be sent to another host, and a given number of replicas will be generated for consistency. For example, if $m_A(x) = 3$ while x doesn't belong to B, the element x will be delivered from A to B and 2 extra replicas will be generated at $Host_B$. Similarly, our ICBF-based method also follows the proposed synchronization framework, as shown in Fig.1, consisting of three operations: encoding, subtracting, and decoding.

Consider that $Host_A$ and $Host_B$ need to synchronize two multisets A and B. First of all, each host executes the encoding operation and establishes an $id\ table$. Secondly, $Host_B$ sends its encoding result, i.e., $ICBF_B$, to $Host_A$. Thirdly, $Host_A$ performs the subtracting operation to derive $ICBF_C = ICBF_A - ICBF_B$ and $ICBF_{C'} = ICBF_B - ICBF_A$. $Host_A$ employs the decoding operation to identify elements in E_A and d_{M_A}. According to the fourth step in the framework, $Host_A$ sends E_A, $ICBF_C$ and $ICBF_{C'}$ to $Host_B$. In the fifth step, $Host_B$ decodes $ICBF_C$ and $ICBF_{C'}$ to identify the elements in E_B and d_{M_B}. Finally, $Host_B$ sends the decoding result, i.e., E_B to $Host_A$, and thus, the synchronization is accomplished.

Algorithm 2 Encoding Operation of ICBF

Require: A multiset A, any element $x \in A$, k hash functions and a cell vector $ICBF$.
1: Initialize the $ICBF$ vector of cells;
2: **for** $x \in A$ **do**
3: **for** $i = 0$ to $k - 1$ **do**
4: Calculate the hash value of $h_i(x)$;
5: **if** $ICBF[h_i(x)].id$ is not empty **then**
6: $ICBF[h_i(x)].count{+}{+}$;
7: **else**
8: $ICBF[h_i(x)].id \leftarrow x.id$;
9: $ICBF[h_i(x)].count{+}{+}$;
10: **return** $ICBF$;

Encoding: Given a pair of multisets, A and B, when encoding any multiset, each of its elements is mapped into k cells via the k independent hash functions. What is different from the encoding process of IBF is that ICBF need not know the size of the difference between A and B. This will save the additional overhead resulting from estimating the difference, compared to the IBF-based method. Fig. 2 depicts the encoding process for multiset A and B. As reported in Algorithm 2, when an element is mapped into a given cell, if the id field is empty, then the id of such an element will be kept in that id field, and the associated $count$ field will be increased by 1. Otherwise, only the $count$ field needs to be increased by one. Reasonably, according to the minimal value among the k $count$ fields, we can estimate whether an element has been mapped into this cell, and if so, how many times. Note that, in Fig. 2, there is a hash conflict in $ICBF_B$ since x and z are mapped into a same cell. In this case, the id field only records the identifier of x, while the $count$ field counts $m_B(x) + m_B(z)$.

Subtracting: After encoding multisets A and B, the two corresponding hosts will exchange the resultant $ICBF_A$ and $ICBF_B$. So far, the next process of synchronization is to subtract the different elements from the two ICBFs, each of which is a vector of cells. Note that the set of used hash functions and the length of the cell vectors for A and B must be the same, such that any common element will be mapped into the same set of k locations in the two vectors. Algorithm 3 describes the subtracting process. It traverses the cell vectors of A and B from beginning to end. For the i^{th} cell in both $ICBF_A$ and $ICBF_B$, if both $ICBF_A[i].id$ and $ICBF_B[i].id$ are empty, Algorithm 3 just moves towards the next cell. If $ICBF_A[i].id$ and $ICBF_B[i].id$ are not empty, the algorithm only remains $ICBF_A[i].id$ in the resultant cell vector. If either $ICBF_A[i].id$ or $ICBF_B[i].id$ is not empty, the algorithm will remain the nonempty one to identify the corresponding element.

The operation on the $count$ field of each cell in $ICBF_A$ and $ICBF_B$ is relatively simple. It just executes the operation of $ICBF_A[i].count - ICBF_B[i].count$, calculating

Algorithm 3 Subtracting Operation for ICBF

Require: The encoding results $ICBF_A$ and $ICBF_B$, the length of cell vectors m.

Initialize a cell vector, denoted as $ICBF_C$;
for $i = 0$ to $m - 1$ **do**
  if $ICBF_A[i].id$ is not empty **then**
    $ICBF_C[i].id \leftarrow ICBF_A[i].id$;
    $ICBF_C[i].count \leftarrow ICBF_A[i].count - ICBF_B[i].count$;
  if $ICBF_A[i].id$ is empty and $ICBF_B[i].id$ is not **then**
    $ICBF_C[i].id \leftarrow ICBF_B[i].id$;
    $ICBF_C[i].count \leftarrow ICBF_A[i].count - ICBF_B[i].count$;
return $ICBF_C$;

Algorithm 4 Decoding $ICBF_C$ and $ICBF_{C'}$ at $Host_B$

Require: The subtracting result $ICBF_C$ and $ICBF_{C'}$, as well as the number of cells m.

$d_{M_B} \leftarrow NULL$, $E_B \leftarrow NULL$;
for $i = 0$ to $m - 1$ **do**
  if $ICBF_C[i].id.[0]==1$ and the element is not in E_B **then**
    Push the element into E_B;
  if $ICBF_{C'}[i].id.[0]==1$, $ICBF_{C'}[i].count<0$ and the element is not in d_{M_B} **then**
    Push the element into d_{M_A};
return E_B and d_{M_A};

the difference of multiplicity of an element in both A and B. Through the two operations on the id and $count$ fields of each cell, Algorithm 3 can subtract the different elements between A and B. In the resulting cell vector, the id field identifies what the different elements are, and the $count$ field records how many times the elements differ from each other. Fig.2 presents a successful subtracting process between $A = \{\langle x, 2\rangle, \langle y, 1\rangle\}$ and $B = \{\langle x, 1\rangle, \langle y, 2\rangle, \langle z, 1\rangle\}$. The resultant multiset $C = A - B$ can be calculated via the aforementioned rules. The common elements, i.e., x and y, are mapped into the same locations in $ICBF_A$ and $ICBF_B$. In the resulting cell vector $ICBF_C$, the id field of each cell stems from either $ICBF_A$ or $ICBF_B$ (only when $ICBF_A[i].id$ is empty), while the $count$ field of each cell is derived by $ICBF_A[i].count - ICBF_B[i].count$.

During the subtracting process, if both $ICBF_A[i].id$ and $ICBF_B[i].id$ are non-empty or $ICBF_A[i].id$ is non-empty but $ICBF_B[i].id$ is empty, we remain the $ICBF_A[i].id$ in the $ICBF_C$ vector. Only when $ICBF_A[i].id$ is empty but $ICBF_B[i].id$ is non-empty, $ICBF_C$ would record $ICBF_B[i].id$. Note that the decoding process will benefit from this special discipline. When $Host_B$ decodes $ICBF_C$, it can identify those elements that only belong to B. Thus, E_B will be determined since all elements appearing in A can be found by checking the first digit of each identifier, i.e., 0. Also, if $Host_A$ decodes $ICBF_C$, it will be aware of those elements in d_{M_A}, by estimating whether their identifiers begin with 0 and whether their $count$ fields are negative. The specific decoding process will be discussed as follows.

Decoding: In fact, $Host_A$ is also able to derive the set $C' = B - A$ via the designed subtracting process. By decoding $ICBF_C$ and $ICBF_{C'}$, each host can distinguish the elements that should be copied at the local host from the elements that call for transmission to another host. In $ICBF_C$, only those elements appearing in B but not in A are identified with the id fields, beginning with 1. In contrast, in $ICBF_{C'}$, only those elements that appear in A, but never appear in B are identified via those id fields, which start with 0. That is, E_A can be derived from $ICBF_{C'}$ at $Host_A$, while E_B can be derived from $ICBF_C$ at $Host_B$, respectively. Furthermore, $Host_A$ can decode $ICBF_C$ and discover those elements which belong to A, but the $count$ fields are negative. Such elements belong to d_{M_B} and $|count|$ replicas of these elements will be generated at $Host_A$. Similarly, $Host_B$ decodes $ICBF_{C'}$ and recognizes those elements, which belong to B while the $count$ fields are negative, and $|count|$ replicas will be required. Thus, the hosts only exchange the elements appearing in only one set.

To handle the potential hash conflicts, when decoding the elements, ICBF employs the mode value (the value appears most often) of the k $count$ fields as the multiplicity of an element in d. The reason is that, the hash conflicts in $ICBF_A$ and $ICBF_B$ will affect the minimal value of the $count$ field in $ICBF_C$, as well as $ICBF_{C'}$. Consequently, the minimal value of the k $count$ fields in $ICBF_C$ and $ICBF_{C'}$ is not reliable any more. By contrast, even if $k-2$ cells of an element occurs hash conflicts, it is still possible for the mode value of the k cells to infer the correct multiplicity of an element.

As an example, Algorithm 4 reports the decoding operation at $Host_B$. It decodes $ICBF_C$ to find out the elements that need to be transmitted to $Host_A$, i.e., E_B. If the id of a cell in $ICBF_C$ begins with 1, then the associated element belongs to E_B and should be transmitted to $Host_A$. With the help of the local id $table$, $Host_B$ knows the content of each element in E_B. Afterward, $Host_B$ chooses each cell from $ICBF_{C'}$, whose id field starts with 1 and in which the $count$ field is negative, and then generates replicas of the associated elements. In this way, d_{M_A} is derived and synchronized without any additional transmission overhead.

Note that, for the ICBF-based method only, E_A is equal to the root set of d_{E_A} and E_B records the root set of d_{E_B}. Besides, d_M can be calculated as $d_{M_A} \cup d_{M_B}$. But the IBF-based method and the CBF-based method fail to distinguish d_E and d_M. Thus, for these two methods, E_A represents the elements in $d_{E_A} \cup d_{M_A}$, while E_B involves the elements in $d_{E_B} \cup d_{M_B}$. Thus, the ICBF-based method transmits fewer elements than the IBF-based and CBF-based methods.

E. Probability of Correct Representation

In this section, we measure the probability of conflict-free representation of an element x in a given multi-set A, i.e., $min\{C[h_j(x)]\} = m_A(x)$, where $j \in [1, k]$. Indeed, even with a set of random and independent hash

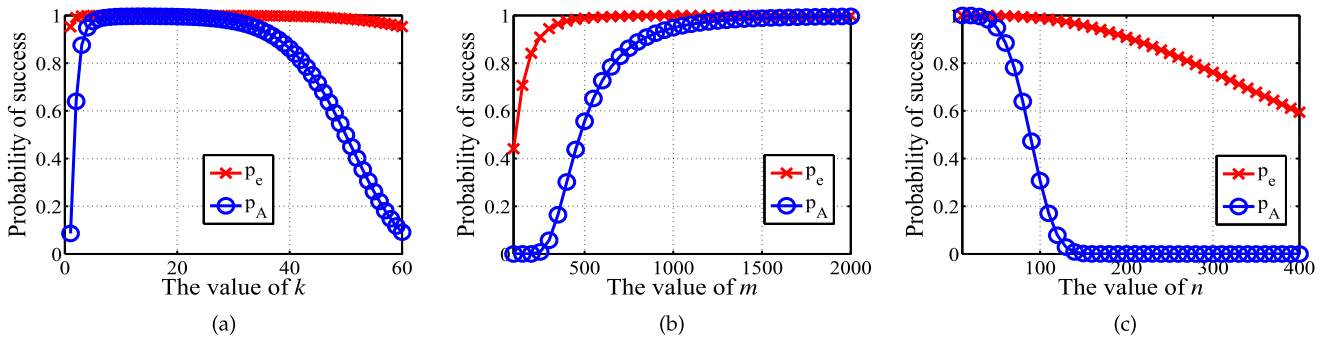


Fig. 3. The changing trends of successful representation probability of an element (p_e) and the entire multiset (p_A), with different parameter options. (a) Impact of k when $m = 1000$, $n_A = 50$. (b) Impact of m when $n_A = 50$, $k = 4$. (c) Impact of n_A when $m = 1000$, $k = 4$.

functions, ICBF may still incur some false behaviors with a given probability during the process of set synchronization. ICBF can accurately represent an element iff the minimal value of the k *count* fields in the mapped cells is equal to its multiplicity. Hence, with given m, n_A and k, we calculate the false-positive probability of representing an element x using an ICBF.

We note that any multiset A is the "union" of several subsets of its root set A^*. For example, $A = \{\langle x,1\rangle \langle y,2\rangle \langle z,3\rangle\}$ can be considered as the "union" of three standard sets, i.e., $A = A^* \uplus A_1 \uplus A_2$, where $A^* = \{x, y, z\}$, $A_1 = \{y, z\}$, and $A_2 = \{z\}$. Hence, hashing all elements in A to the ICBF cells can be realized by mapping the elements in such simple sets round by round, i.e., first A^*, then A_1, and at last A_2.

Theorem 1: Given the value of m, n_A and k, the necessary and sufficient condition of $x \in A$ that is correctly represented by ICBF is that x can be validly represented by ICBF in the first round, i.e., in A^*. Mathematically,

$$p(min\{C[h_j(x)]\} = m_A(x)) = p_r(min\{C[h_j(x)]\} = 1) \quad (1)$$

where $j \in [1,k]$, and $p_r(min\{C[h_j(x)]\} = 1)$ means x is correctly represented by ICBF in the first round.

Proof: We can infer from Theorem 1 that If the element x is correctly represented in the first round, the final ICBF cells satisfy $min\{C[h_j(x)]\} = m_A(x)$ with $j \in [1,k]$, regardless of how many replicas x still has.

On one hand, the element x has been mapped for $m_A(x)$ rounds into ICBF cells. If $min\{C[h_j(x)]\} = m_A(x)$ holds, for any cell whose counter part is $m_A(x)$, at each round only x is mapped into the cell. At each round, k hash functions map x into the cell vector; hence, the minimum *count* filed in these cells record the multiplicity of x, i.e., $m_A(x) = min\{C[h_j(x)]\}$. If any other element y also has been mapped into this cell, the counter part is $m_A(x) + m_A(y)$, which is definitely larger than $m_A(x)$. So, it can be concluded that for the cells whose final state is $m_A(x)$, the counter part is increased by 1 at each round (certainly including the first round which handles the root set of A). Thus, the sufficiency of the conditions is proved.

On the other hand, if in the first round $C[h_j(x)]$'s counter part is only added by 1 due to mapping x with h_j, then in the remaining rounds, no other elements but x will be hashed into this cell. The reason is that, the root set A^* contains all elements that occur in A, and we have $A^* \supseteq A_1 \cdots \supseteq A_i \cdots \supseteq A_\gamma$, where γ denotes the maximum multiplicity of elements in A. Hence, at last, the cell $C[h_j(x)]$'s counter part must be $m_A(x)$. Thus, the necessity of the conditions is proved. □

Theorem 2: Given the value of m, n_A, k, and $j \in [1,k]$, the probability that an element $x \in A$ can be validly represented by ICBF in the first round is:

$$p_r(min\{C[h_j(x)]\} = 1) = 1 - (1 - (1 - \frac{1}{m})^{n_A \cdot k - 1})^k. \quad (2)$$

Proof: In the first round, an element x has definitely been mapped k times by the k independent hash functions. For any position $h_j(x)$ in the cell vector, we have $p_r(C[h_j(x)] = 1) = (1 - \frac{1}{m})^{n_A \cdot k - 1}$. There exist k such positions for x; hence, the probability that x is correctly represented in the first round can be calculated as $p_r(min\{C[h_j(x)]\} = 1) = 1 - (1 - (1 - \frac{1}{m})^{n_A \cdot k - 1})^k$. Thus, Theorem 2 is proved. □

Based on Theorems 1 and 2, the probability that ICBF can rightly represent an element is derived. Then the n_A-th Power of $p_r(min\{C[h_j(x)]\} = 1)$ demonstrates the probability that all elements in A are correctly recorded. Fig. 3 specifies the impact of k, m, and n_A, in terms of the probability of correctly representing a single element (denoted as p_e), and the entire multiset A (denoted as p_A). In Fig. 3 (a), given $m = 1000$ and $n_A = 50$, when k increases from 1 to 60, both p_e and p_A first increase to nearly 1, then drop to a low level. By taking the derivative and equaling to zero, the optimal value of k can be derived. It is difficult to calculate the analysis formula of the optimal k, and we employ the result in [27] as a reference, i.e., $k \approx \frac{m}{n_A} \ln 2$. The reason is that our p_e is much more complicated than the probability in [27]. But we can still determine the optimal k for ICBF by searching around $\frac{m}{n_A} \ln 2$.

Besides, Fig. 3 (b) and Fig. 3 (c) confirm the impact of m and n_A with $n = 50$, $k = 4$ and $m = 1000$, $k = 4$, respectively. Apparently, p_e and p_A benefit from larger m, since more cells can degrade the chance of hash conflicts. The impact of m shows a marginal effect. In the experiment, when m is larger than 1000, p_e and p_A increase more slowly. In contrast, n_A shows a totally opposite influence on the evaluated probabilities. With given m and k, fewer elements

TABLE IV
COMPARISON BETWEEN CBF AND ICBF WITH $n_A = 400$, $n_B = 500$, AND $d = 1000$, IN TERMS OF SYNCHRONIZATION ACCURACY, TIME-CONSUMPTION AND CONFLICT RATE

m	3000	5000	7000	9000	11000	13000	15000
CBF-a	0.912	0.943	0.971	0.978	0.981	0.986	0.992
ICBF-a	0.913	0.940	0.969	0.979	0.981	0.985	0.989
CBF-t	0.265	0.290	0.297	0.293	0.309	0.325	0.327
ICBF-t	0.174	0.194	0.210	0.228	0.238	0.248	0.260
Coll-A	0.172	0.107	0.079	0.059	0.054	0.047	0.039
Coll-B	0.204	0.134	0.095	0.077	0.065	0.055	0.049

result in higher probability of successful representation of elements. This is reasonable, since more elements call for more hash operations and lead to more chances for failure. Note that, p_A is calculated as $p_e^{n_A}$, and thus shows more radical increases or decreases in our experiments. Indeed, with respect to k, according to the means introduced in [27], the best option of m can be derived as $m = -2.081n * \ln(1 - p_e)$; hence, we omit the details here.

Furthermore, Table IV records the performance of synchronization, as well as the conflict rate at $Host_A$ (Coll-A) and $Host_B$ (Coll-A). Note that, the conflict rate means the ratio of the number of hash conflicts to the total times of hash operations. It is clear that increasing the value of m will cause less hash conflicts. $Host_B$ occurs more hash conflicts than $Host_A$, since $n_A = 400$ but $n_B = 500$. Besides, the synchronization accuracy grows with the drop of the conflict rate, irrespective of CBF and ICBF.

V. PERFORMANCE EVALUATION

In this section, we implement IBF and ICBF to evaluate the synchronization accuracy and time-consumption. The synchronization accuracy is defined as the ratio of the number of decoded differences to the amount of real difference. By contrast, the time-consumption records the time lasting from the very beginning of the encoding operation to the end of the decoding operation. We also compare the additional communication overhead caused by transmitting the IBF or ICBF cell vectors.

A. Experiment Methodology

A virtual machine with 2.5 GHz CPU and 8 GB RAM is employed as a host. We augment the strings with lengths and characters that differ from those of elements in multisets. Note that one challenging issue for bloom filter-like data structure is the generation of a group of k independent hash functions. In our experiments, the hash functions are generated with the method employed in [26], i.e.,

$$h_i(x) = (g_1(x) + i \cdot g_2(x)) \ mod \ m \tag{3}$$

where $g_1(x)$ and $g_2(x)$ are two random and independent integers in the universe with a range $[1, m]$. The integer i belongs to the range $[0, k-1]$.

In Table IV, we compare the performance of the synchronization methods based on CBF and ICBF, in terms of the synchronization accuracy and time-consumption, with given $n_A = 400$, $n_B = 500$, and $d = 1000$. Apparently, more cells result in obvious improvement of the synchronization accuracy for both CBF and ICBF, at the cost of more time-consumption. Note that, CBF and ICBF lead to similar accuracy under the same parameter setting. The intrinsic reason is that they follow same encoding, subtracting, and decoding framework; hence they suffer from the same false positive. However, compared with CBF, ICBF costs less time-consumption. Consider that, CBF must query all the elements in A^* and B^* to deduce d, but ICBF only needs the information of elements recorded in $ICBF_C$ or $ICBF_{C'}$. Thus, ICBF outperforms CBF in terms of synchronization time-consumption. More importantly, CBF is not inversely decodable and fails to identify d_E and d_M. Consequently, it cannot minimize the transmission of elements between two hosts. For this reason, we just evaluate the IBF-based and ICBF-based synchronization methods. We also verify the influence of parameter settings on the synchronization performance, and subsequently report all the performance metrics on an average of 100 rounds of tests.

B. Synchronization Accuracy and Time-Consumption

The IBF-based method and ICBF-based method are capable of achieving multiset synchronization, but result in different time-consumptions and communication overheads. In this section, we evaluate their performance in terms of the synchronization accuracy and time-consumption under diverse parameter settings. Typically, the Jaccard similarity of root sets A^* and B^*, i.e., $J = |A^* \cap B^*|/|A^* \cup B^*|$, is employed to measure the similarity of A and B based on their root sets.

We measure the impact of different settings of multisets on the performance of the two methods. Fig. 4(a) and Fig. 5(a) report the synchronization accuracy and time-consumption of the ICBF-based and IBF-based methods, respectively, when J varies from 0 to 1, given $n_A = 500, n_B = 500, d = 2000$. With the increase of J, the synchronization accuracy of the IBF-based method grows up from 0.523 to 0.999, while the time-consumption decreases from 345s to 165s. Also, the ICBF-based method shares the similar changing trends of accuracy and time-consumption.

However, the ICBF-based method achieves a higher synchronization accuracy and incurs less time-consumption, compared with the IBF-based method. Indeed, larger J causes more common elements in the two root sets of A and B, which means that fewer kinds of elements will be involved in the synchronization processes. As a result, the IBF-based method will be executed for more rounds, but involves fewer elements in each round, which leads to better accuracy and less time. This explains why when J increases, the synchronization accuracy of the IBF-based method continues to increase while the time-consumption reduces continuously.

Fig. 4(b) and Fig. 5(b) report the impact of d (the size of the difference among two mulitsets) on the changing trends of performance metrics. Given $n_A = 500$, $n_B = 500$, and $J = 0.8$, we vary the value of d from 500 to 3500. It is clear that the two metrics always fluctuate within a narrow interval with the increase of d. That is, the ICBF-based method

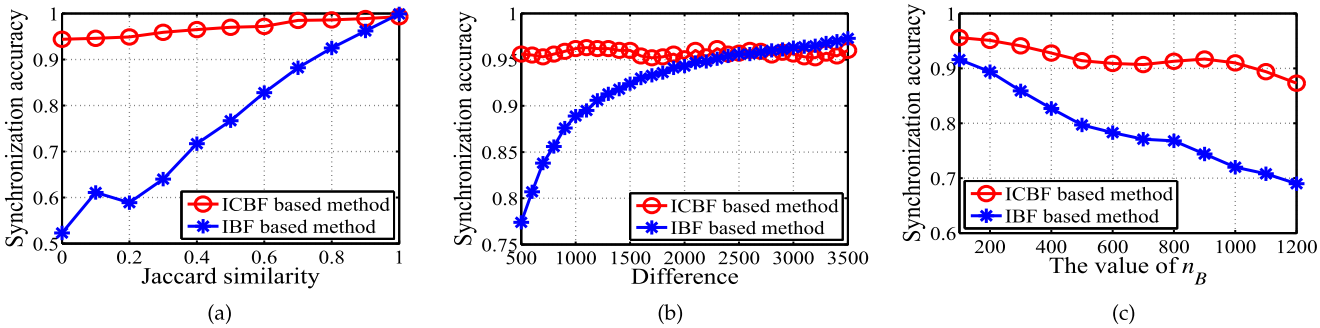


Fig. 4. The synchronization accuracy of ICBF and IBF under different parameter settings. (a) Accuracy when J varies. (b) Accuracy when d varies. (c) Accuracy when n_B varies.

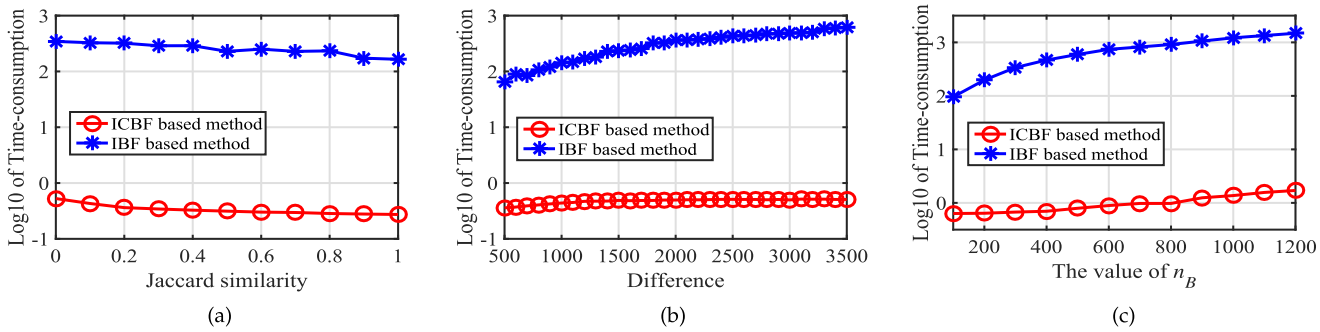


Fig. 5. The time-consumption of ICBF and IBF under different parameter settings. (a) Time-consumption when J varies. (b) Time-consumption when d varies. (c) Time-consumption when n_B varies.

is insensitive to the difference that is caused by different multiplicities of same elements. The root cause is that the two metrics of the ICBF-based synchronization are actually determined by the size of the difference between the two corresponding root sets, irrespective of the multiplicity of each element.

On the contrary, both performance metrics under the IBF-based method grow as d increases. The larger value of d means that the IBF-based method will be executed for more rounds, thus increasing the execution time. In effect, among 100 rounds of failed IBF processes, most failures occur within the first five rounds. Accordingly, the involved elements become scarce as the round order increases; hence, this eases the successful decoding of such elements. In our experiment settings, n_A and n_B are constant. Thus, more differences result in higher multiplicity for elements on average. So the IBF-based method will be executed with a high frequency, and more elements will be identified. Thus, the synchronization accuracy of the IBF-based method keeps increasing and approaches 1.

Let the value of n_B range from 100 to 1200, with $n_A = 600, J = 0.1$, and $d = 6000$. As shown in Fig. 5(c), the ICBF-based and IBF-based methods always result in increasing time-consumption along with the growing n_B. Note that the ICBF-based method saves the consumed time considerably more than the IBF-based method. In Fig. 4(c), the synchronization accuracy of the ICBF-based method decreases from 0.956 to 0.873, while that of IBF-based decreases from 0.916 to 0.690. So the ICBF-based method outperforms the IBF-based method in terms of the synchronization accuracy.

C. Communication Overhead of Cell Vector

As depicted in Fig. 1, three interactions are needed to realize multiset synchronization, and undoubtedly, each round of the communication cost is dedicated to communication overhead. In this section, to reveal the efficiency of different synchronization methods, we assess the communication overhead caused by transmitting cell vectors used by different methods. We do not consider the communication overhead that has been caused by transmitting elements. This is because only our ICBF-based method can distinguish d_M from d_E and thus eliminate unnecessary transmission, while other methods cannot. Compared with the ICBF-based method, other methods definitely suffer from an increased communication overhead of transmitting elements. Generally speaking, 4 bits for the counters are sufficient, but we set the count filed as 16 bits by default in case of the worst situations.

According to the analysis in [27], with respect to the optimal value of k, the best option of m is given as $m = -2.081n \cdot \ln(1 - p_e)$. In our evaluation, the value of p_e is set as 0.999; hence, m can be derived as $14.3776n$. Thus the communication overhead of ICBF, i.e., CO_{ICBF}, can be calculated as:

$$CO_{ICBF} = 14.3776 \cdot (18 + \log_2 n)n. \tag{4}$$

where n denotes the maximum of n_A and n_B, and m denotes the number of cells. Additionally, the *count* field cost 16 bits, while the prefix and the flag (empty or not) need 1 bit, respectively. Meanwhile, the suffix cost $\log_2 n$ bits. Note that, in this parameter setting, the majority of the cells are empty.

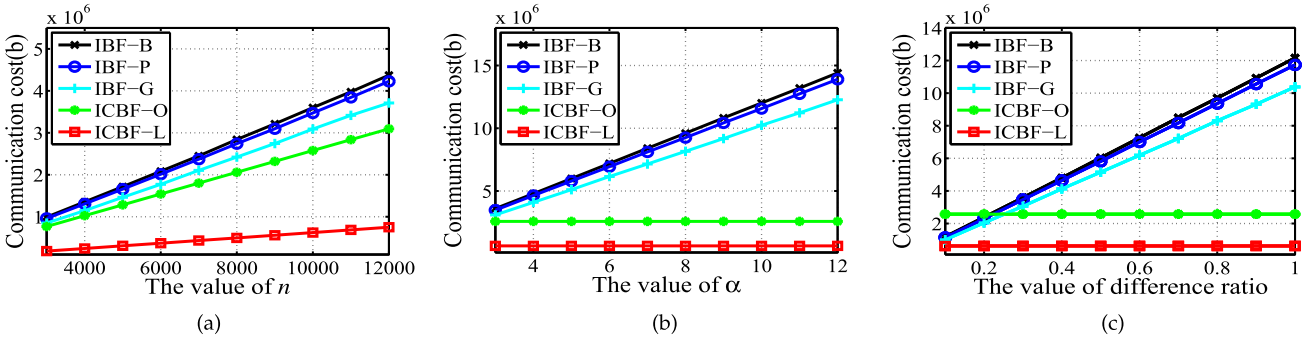


Fig. 6. The communication overhead of IBF and ICBF for multiset synchronization with different parameter settings. (a) Communication overhead when n varies. (b) Communication overhead when α varies. (c) Communication overhead when d/n varies.

To save the consumed memory by ICBF and reduce the resultant communication overhead, we use a double linked list to save the cell vector. Linked list saves those nonempty cells with two pointers, the **prior** and the **next**, to identify the position of each cell in the vector. Here, each pointer is 16 bits. So, when the ICBF cells are stored with linked list, the communication overhead turns to be:

$$CO_{ICBF-l} = (50+\log_2 n) \cdot n. \tag{5}$$

As for the IBF-based method, the communication overhead of each round, i.e., CO_{IBF}, can be calculated by the following equation:

$$CO_{IBF} = \alpha \cdot \bar{d} \cdot (\log_2 n + 2 \cdot \log n + \log \bar{d} + 16) \tag{6}$$

where $\bar{d}$ denotes the size of estimated difference, n denotes the maximum of n_A and n_B in this round, and α is a coefficient that controls the length of the IBF vector. In this equation, $log_2 n$, $2 \cdot \log \bar{d} + \log \bar{d}$ and 16 denote the overhead caused by the $idSum$, $hashSum$ [22], and $count$ fields in an IBF, respectively. The part of $2 \cdot \log n + \log \bar{d}$ is the least bit length of $hashSum$ to identify single-element cells for further decoding use [9].

Fig. 6 plots the communication overhead caused by the two synchronization methods under different parameter settings. In our experiments, three mainstream discrete distributions for the multiplicity in the multiset are employed, i.e., the Binomial, Poisson, and Geometric distributions. We keep the average multiplicity the same under these distributions. In fact, these distributions never affect the communication overhead caused by the ICBF-based method, which is decided by the value of n, i.e., the maximum number of elements in the two root sets. By contrast, IBF decodes all of its elements via round-robin processes, which implies that its communication overhead will be affected by the distribution of multiplicity. Note that, in Fig. 6, the legends "IBF-B", "IBF-P", and "IBF-G" denote the additional communication overhead of the IBF-based method when the multiplicity of elements follow Binomial, Poisson, and Geometric distributions. "ICBF-O" and "ICBF-L" identify the additional communication overhead of ICBF-based method with the original storage strategy and employing double linked list.

In Fig. 6(a), we vary n from 3000 to 12000, while $\alpha = 3$ and $d/n = 0.3$. Compared with the IBF-based method, the ICBF-based method causes less overhead under each multiplicity distribution, since both "ICBF-O" and "ICBF-L" outperform others. By contrast, we keep $n = 10000$, while varying α and d/n in Fig. 6(b) and Fig. 6(c), respectively. To be exact, $d/n = 0.3$ while α increases from 3 to 12 in Fig. 6(b) and $\alpha = 3$ while d/n rises from 0.1 to 1 in Fig. 6(c). Obviously, the communication overhead of ICBF is much less than IBF and stays constant in both Fig. 6(b) and Fig. 6(c). The reason is that the employed linked list only records the nonempty cells in ICBF. Meanwhile, the number of nonempty cells in ICBF can be calculated as $k \cdot n$. Thus, the variables α and d cannot affect the additional communication overhead of the ICBF-based method. Moreover, Fig. 6 also indicates that the double linked list can significantly save storage and degrade the communication overhead of ICBF.

The communication overhead caused by the IBF-based method is very close under the Binomial and Poisson distributions, since they hold the similar statistical characteristics. Furthermore, the IBF-based method incurs the least communication overhead under the long-tailed Geometric distribution, because most multiplicities of elements are less than the average value, and thus, execute fewer rounds of the synchronization processes.

Note that, to execute the subtracting and decoding operations, the received double linked list should be recovered as a vector of ICBF cells. This mission can be easily achieved by using the pointers in the linked list. For example, in the linked list, if $ICBF_l[i].next = 9$ and $ICBF_l[i+1].prior = 4$, then four empty cells ranging from $ICBF[5]$ to $ICBF[8]$ should be added between $ICBF_l[i]$ and $ICBF_l[i+1]$ to recover the ICBF. After recovering all of the m cells, the host will follow the designed steps to deduce the different elements between multisets A and B.

VI. Related Work

As an essential task, set synchronization has been extensively studied in the fields of database, networking, and information theory. We classify the existing work into two categories according to what they synchronize, i.e., specific contents or general elements.

A. Synchronization of Specific Contents

We note that researchers take advantages of the intrinsic features of the content to model the synchronization problems.

Accordingly, the synchronization mechanism will be designed, and the upper and lower bound of the communication complexity can be reasonably derived.

Synchronization of Random Variables: To synchronize two discrete random variables, one model is proposed to calculate the upper and lower bound of the communication complexity [28]. Thereafter, Alon and Orlisky reveal the connection between synchronization of random variables and minimal coloring problem of the corresponding *characteristic graph* [29]. Furthermore, Orlisky also proposes a linear error-correcting codes-based method to realize data synchronization [30]. Based on the well-studied coloring and error-correcting theories, the communication complexity of synchronization can be accurately evaluated.

Synchronization of Files or Strings: The synchronization of files (or strings) on two distributed hosts has been modeled as the well-known edit-distance problem [31]. The error-correcting code-based method has been developed to settle this issue with nearly optimal communication overhead by ranking the strings in sorted order [32]. The efficiency of the error-correcting based method is closely related to the code mechanism [31]. Given two sets, each with a set of bit-strings, a characteristic polynomial is generated to represent each set [33]. By evaluating the rational function of the characteristic polynomials, the different strings will be decoded.

Synchronization of Pictures: Reference [11] denotes itself to synchronize two sets of pictures by evaluating the similarity between two pictures with Earth Mover's Distance (EMD). Based on the observation that close points in the Euclidean space often represent the same element, [11] utilizes the Invertible Bloom Lookup Tables (IBLTs) to record the information of points in the space. After exchanging the IBLTs of the other set, an EMD-enabled decoding algorithm is sufficient to derive the different elements.

These methods realize communication-saving synchronization by modelling the synchronization problem of specific contents; hence, their strength may not be popularized in other fields.

B. Synchronization of General Elements

In fact, this kind of synchronization method employs the bloom filters and their variants to record the information of each set, regardless of what the involved elements are.

Bloom Filter has been employed to realize set synchronization, due to the space efficiency and constant query delay [6]. Each host employs a bit vector to represent all of its elements, and delivers the bit vector to the other host for synchronization [1]. However, if multiple elements have been hashed into any bit in BF, the BF may fail to identify all elements. To settle this problem, the CBF-based set synchronization method records the information of its elements via a cell vector [21]. Note that, the time-complexity of synchronization is $O(n_A+n_B)$ due to a number of n_A+n_B queries. The former two methods failed to decode the elements inversely, thus they need additional query operations to identify the different elements from the vectors. IBF [9], which records the elements via three dedicated fields, i.e., $idSum$, $hashSum$, and $count$, can synchronize two sets and inversely decode the elements from cells. But the parameters need to be carefully designed, and it costs additional computation and space overhead to evaluate the difference.

The aforementioned methods are all inefficient to synchronize multisets, allowing elements to appear multiple times. Thus, for the first time, we propose a novel variant of bloom filters and utilize it to achieve fast and accurate multiset synchronization in this paper.

VII. Discussion

To fully understand the proposed methods, we discuss the following issues further.

Synchronization in Extreme Cases: Undoubtedly, it is true that the ICBF-based method can realize bandwidth-saving synchronization. However, in some extreme cases, representing multisets with ICBFs to discover the different elements between two multisets may be not cost-effective. Firstly, if the amount size of all elements in a multiset is smaller than the size of the used ICBF, it is not advisable to employ the ICBF-based method. Basically, each host calculates the size of ICBF with Equation 5. If the calculated result is no less than the total size of the elements in a multiset, the host will send the multiset to the other host directly. The second extreme case is that there are enormous differences between two multisets. In this case, we believe that the ICBF-based method is still recommendable since it distinguishes d_E from d_M. As a result, only the elements in d_E will be transmitted for once. By contrast, the elements in d_M will be synchronized by generating dedicated number of replicas at the local host. In this way, ICBF-based method transmits the least elements.

The Hash Conflicts: Hash conflicts are unavoidable, when encoding a multiset with BF and its variants. For this reason, many efforts have been done to control and mitigate the impact of hash conflicts, according to different missions. To represent a multiset for query, the minimal $count$ value in the k cells can infer the multiplicity of an element with high probability, as discussed in Section IV-E. By contrast, to synchronize two multisets, the minimum value of the k involved $count$ fields may underestimate the multiplicity during the decoding operation. Hence, it is better to employ the mode value of the k $count$ fields as the multiplicity. For example, in the resulted $ICBF_C$ in Fig. 2, the minimum value of the k cells for both x and z is 0. However, their real multiplicities in $C = A - B$ are 1 and -1, respectively. Hence, the mode value is more reliable than the minimum value for the decoding operation. Indeed, it is complicated to qualify the relationship between the conflict rate and the set synchronization. However, by employing the mode value as the multiplicity of each element, the ICBF can still synchronize this element successfully with high probability.

Future Work: Due to the importance of multiset synchronization, more comprehensive endeavors should be made to further improve the synchronization techniques. Firstly, note that the proposed methods can synchronize multisets with high probability, but fail to realize 100% synchronization accuracy. To address this problem, other outstanding data structures may be useful to represent and synchronize multisets. Moreover,

how to achieve accurate and fast multi-party synchronization is also a challenging issue to be solved in the future.

VIII. Conclusion

In this paper, we focus on the essential problem of multiset synchronization, which has not been addressed in literature. We first examine the potential of the IBF-based method and the CBF-based method for multiset synchronization, using a general framework. However, they are either time-consuming or inefficient. To realize multiset synchronization efficiently, we further propose a novel data structure named invertible counting bloom filters (ICBF). Accordingly, we design the ICBF-based method that synchronizes two multisets both quickly and accurately. Typically, the localized ids are employed to identify each kind of element in A and B. Additionally, an associated *id table* is established in each host to record the mapping relationship between ids and elements. The evaluating results show that, compared with the IBF-based method, our ICBF-based method synchronizes the multisets more accurately within a shorter time period. Moreover, the communication cost of the ICBF-based method is much less than the IBF-based method, and is only decided by the value of n. Accordingly, we believe that our ICBF-based method is effective and practical to synchronize multisets.

References

[1] D. Guo and M. Li, "Set reconciliation via counting bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2367–2380, Oct. 2013.

[2] J. Liu, S. Ahmad, E. Buyukkaya, R. Hamzaoui, and G. Simon, "Resource allocation in underprovisioned multioverlay peer-to-peer live video sharing services," *Peer-to-Peer Netw. Appl.*, vol. 8, no. 3, pp. 399–413, May 2015.

[3] T. Chen, D. Guo, X. Liu, and J. Liu, "BDP: A bloom filters based dissemination protocolin wireless sensor networks," in *Proc. 6th IEEE Int. Conf. Mobile Adhoc Sensor Syst. (MASS)* Oct. 2009, pp. 593–602.

[4] V. Stefano, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[5] K. P. N. Puttaswamy *et al.*, "Docx2go: Collaborative editing of fidelity reduced documents on mobile devices," in *Proc. ACM MobiSys*, 2010, pp. 345–356.

[6] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[8] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[9] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference?: Efficient set reconciliation without prior context," in *Proc. ACM SIGCOMM*, 2011, pp. 218–229.

[10] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *Proc. 49th Annu. Allerton Conf. Commun., Control, Comput.,* Sep. 2011, pp. 792–799.

[11] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, "Robust set reconciliation," in *Proc. ACM SIGMOD* 2014, pp. 135–146.

[12] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD*, Jun. 2003, pp. 241–252.

[13] J. A. Saborit, P. Trancoso, V. M. Mulero, and J. L. L. Pey, "Dynamic count filters," *ACM SIGMOD Rec. Homepage*, vol. 35, no. 1, pp. 26–32, 2005.

[14] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 1, pp. 120–133, Jan. 2010.

[15] T. Yang *et al.*, "A shifting bloom filter framework for set queries," in *Proc. VLDB Endowment*, 2016, pp. 408–419.

[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[17] D. Guo, Y. Liu, X. Li, and P. Yang, "False negative problem of counting bloom filter," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 651–664, May 2010.

[18] D. Singh, A. Ibrahim, T. Yohanna, and J. Singh, "An overview of the applications of multisets," *Novi Sad J. Math.*, vol. 37, no. 3, pp. 73–92, 2007.

[19] D. Guo, Y. He, and P. Yang, "Receiver-oriented design of bloom filters for data-centric routing," *Comput. Netw.*, vol. 54, no. 1, pp. 165–174, 2010.

[20] D. Guo, Y. He, and Y. Liu, "On the feasibility of gradient-based data-centric routing using bloom filters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 180–190, Jan. 2014.

[21] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.

[22] D. Eppstein and M. T. Goodrich, "Straggler identification in round-trip data streams via Newton's identities and invertible bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 2, pp. 297–306, Feb. 2011.

[23] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, and A. Lakshman, "Dynamo: Amazon's highly available key-value store," *ACM Sigops Operating Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[24] "Redis," accessed on Oct. 25, 2015. [Online]. Available: http://redis.io

[25] "About Memcached," accessed on Nov. 21, 2015. [Online]. Available: http://memcached.org

[26] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," *Random Struct. Algorithms*, vol. 33, no. 2, pp. 187–218, 2006.

[27] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 131–155, Feb. 2012.

[28] P. Koulgi, E. Tuncel, S. L. Regunathan, and K. Rose, "On zero-error source coding with decoder side information," *IEEE Trans. Inf. Theory*, vol. 49, no. 1, pp. 99–111, Jan. 2003.

[29] N. Alon and A. Orlitsky, "Source coding and graph entropies," *IEEE Trans. Inf. Theory*, vol. 42, no. 5, pp. 1329–1339, Sep. 1996.

[30] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity," in *Proc. 32nd Annu. Symp. Found. Comput. Sci.* Oct. 1991, pp. 228–238.

[31] M. G. Karpovsky, L. B. Levitin, and A. Trachtenberg, "Data verification and reconciliation with generalized error-control codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 7, pp. 1788–1793, Jul. 2003.

[32] K. A. S. Abdel-Ghaffar and A. El Abbadi, "An optimal strategy for comparing file copies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 1, pp. 87–93, Jan. 1994.

[33] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Trans. Inf. Theory*, vol. 49, no. 9, pp. 2213–2218, Sep. 2003.

Lailong Luo received the B.S. and M.S. degrees from the School of Information System and Management, National University of Defense Technology, Changsha, China, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the College of Information System and Management. His current research interests include data centers and software defined networks.

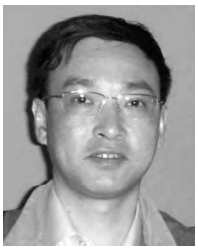
Deke Guo (S'06–M'08) received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of Information System and Management, National University of Defense Technology. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a member of the ACM.

Jie Wu (M'90–SM'93–F'09) was the Program Director with the U.S. National Science Foundation. He is currently the Chair and a Professor with the Department of Computer and Information Sciences, Temple University. He has authored over 450 papers in various journals and conference proceedings. His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has served as an IEEE Computer Society Distinguished Visitor. He is also the Chairman of the IEEE Technical Committee on Distributed Processing.

Ori Rottenstreich received the B.S. (*summa cum laude*) degree in computer engineering and the Ph.D. degree from the Electrical Engineering Department, Technion–Israel Institute of Technology, Haifa, Israel, in 2008 and 2014, respectively. He is currently a Post-Doctoral Research Fellow with the Department of Computer Science, Princeton University. He is a recipient of the Google Europe Ph.D. Fellowship in computer networking, the Andrew Viterbi Graduate Fellowship, the Jacobs-Qualcomm Fellowship, the Intel Graduate Fellowship, and the Gutwirth Memorial Fellowship. He also received the Best Paper Runner Up Award at the IEEE INFOCOM 2013.

Qian He received the B.S. degree from Hunan University, Changsha, China, in 2001, the M.S. degree from the Guilin University of Electronic Technology, Guilin, China, in 2004, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2011. He has done post-doctoral research at the National University of Defense Technology, Changsha, China. He is currently a Visiting Research Associate with the School of Computer Science, The University of Manchester, U.K., and a Professor with the Guilin University of Electronic Technology. His research interests include network security and distribute computing. He is a Senior Member of the CCF.

Yudong Qin received the B.S. degree with the School of Information System and Management, National University of Defense Technology, Changsha, China, in 2016, where he is currently pursuing the M.S. degree with the College of Information System and Management. His current research interests include data centers and software-defined networks.



Xueshan Luo received the B.E. degree in information engineering from the Huazhong Institute of Technology, Wuhan, China, in 1985, and the M.S. and Ph.D. degrees in system engineering from the National University of Defense Technology, Changsha, China, in 1988 and 1992, respectively. He is currently a Professor of Information System and Management, National University of Defense Technology. His research interests are in the general areas of information system and operation research.