The Bloom Clock for Causality Testing

Anshuman Misra and Ajay D. Kshemkalyani(✉)

University of Illinois at Chicago, Chicago, IL 60607, USA
{amisra7,ajay}@uic.edu

Abstract. Testing for causality between events in distributed executions is a fundamental problem. Vector clocks solve this problem but do not scale well. The probabilistic Bloom clock can determine causality between events with lower space, time, and message-space overhead than vector clock; however, predictions suffer from false positives. We give the protocol for the Bloom clock based on Counting Bloom filters and study its properties including the probabilities of a positive outcome and a false positive. We show the results of extensive experiments to determine how these above probabilities vary as a function of the Bloom timestamps of the two events being tested, and to determine the accuracy, precision, and false positive rate of a slice of the execution containing events in the temporal proximity of each other. Based on these experiments, we make recommendations for the setting of the Bloom clock parameters. We postulate the *causality spread hypothesis* from the application's perspective to indicate whether Bloom clocks will be suitable for correct predictions with high confidence. The Bloom clock design can serve as a viable space-, time-, and message-space-efficient alternative to vector clocks if false positives can be tolerated by an application.

Keywords: Causality · Vector clock · Bloom clock · Bloom filter · Partial order · Distributed system · False positive · Performance

1 Introduction

1.1 Background and Motivation

Determining causality between pairs of events in a distributed execution is useful to many applications [9,17]. This problem can be solved using vector clocks [5,11]. However, vector clocks do not scale well. Several works attempted to reduce the size of vector clocks [6,12,18,20], but they had to make some compromises in accuracy or alter the system model, and in the worst-case, were as lengthy as vector clocks. A survey of such works is included in [8].

The Bloom filter, proposed in 1970, is a space-efficient probabilistic data structure that supports set membership queries [1]. The Bloom filter is widely used in computer science. Surveys of the variants of Bloom filters and their applications in networks and distributed systems are given in [2,19]. Bloom filters provide space savings, but suffer from false positives although there are no false negatives. The confidence in the prediction by a Bloom filter depends on the

D. Goswami and T. A. Hoang (Eds.): ICDCIT 2021, LNCS 12582, pp. 3–23, 2021.
https://doi.org/10.1007/978-3-030-65621-8_1

size of the filter (m), the number of hash functions used in the filter (k), and the number of elements added to the set (q). The use of the Bloom filter as a Bloom clock to determine causality between events was suggested [16], where, like Bloom filters, the Bloom clock will inherit false positives. The Bloom clock and its protocol based on Counting Bloom filters, which can be significantly more space-, time-, and message-space-efficient than vector clocks, was given in [7]. The expressions for the probabilities of a positive outcome and of a false positive as a function of the corresponding vector clocks, as well as their estimates as a function of the Bloom clocks were then formulated [7]. Properties of the Bloom clock were also studied in [7], which then derived expressions to estimate the accuracy, precision, and the false positive rate for a slice of the execution using the events' Bloom timestamps.

1.2 Contributions

In this paper, we first give the Bloom clock protocol and discuss its properties. We examine the expressions for the probability of a positive and of a false positive in detecting causality, and discuss their trends as the distance between the pair of events varies. We then show the results of our experiments to:

1. analyze in terms of Bloom timestamps how the probability of a positive and the probability of a false positive vary as the distance between a pair of events varies;
2. analyze the accuracy, precision, and the false positive rate for a slice of the execution that is representative of events that are close to each other. The parameters varied are: number of processes n, size of Bloom clock m, number of hash functions k, probability of a timestamped event being an internal event pr_i, and temporal proximity between the two events being tested for causality.

Based on our experiments, we

1. analyze the nature of false positive predictions,
2. make recommendations for settings of m and k,
3. state conditions and analyze dependencies on the parameters (e.g., n, pr_i) under which Bloom clocks make correct predictions with high confidence (high accuracy, precision, and low false positive rate), and
4. generalize the above results and state a general principle (the *causality spread hypothesis*) based on the degree of causality in the application execution, which indicates whether Bloom clocks can make correct predictions with high confidence.

Thus our results and recommendations can be used by an application developer to decide whether and how the application can benefit from the use of Bloom clocks.

Roadmap: Section 2 gives the system model. Section 3 details the Bloom clock protocol. Section 4 studies properties of the Bloom clock, discusses ways to

estimate the probabilities of a positive outcome and of a false positive, and predicts the trends of these probability functions as the temporal proximity between the events increases. Section 5 gives our experiments for the complete graph and analyzes the results. Section 6 gives our experiments for the star graph (client-server configuration) and analyzes the results. Section 7 summarizes the observations of the experiments and discusses the conditions under which Bloom clocks are advantageous to use. It also postulates the *causality spread hypothesis* and validates it. Section 8 concludes.

2 System Model

A distributed system is modeled as an undirected graph $(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of processes and $\mathcal{L}$ is the set of links connecting them. Let $n = |\mathcal{N}|$. Between any two processes, there may be at most one logical channel over which the two processes communicate asynchronously. A logical channel from P_i to P_j is formed by paths over links in $\mathcal{L}$. We do not assume FIFO logical channels.

The execution of process P_i produces a sequence of events $E_i = \langle e_i^0, e_i^1, e_i^2, \cdots \rangle$, where e_i^j is the j^{th} event at process P_i. An event at a process can be an *internal* event, a *message send* event, or a *message receive* event. Let $E = \bigcup_{i \in \mathcal{N}} \{e \,|\, e \in E_i\}$ denote the set of events in a distributed execution. The causal precedence relation between events, defined by Lamport's "happened before" relation [10], and denoted as $\rightarrow$, induces an irreflexive partial order $(E, \rightarrow)$.

Mattern [11] and Fidge [5] designed the vector clock which assigns a vector V to each event such that: $e \rightarrow f \iff V_e < V_f$. The vector clock is a fundamental tool to characterize causality in distributed executions [9,17]. Each process needs to maintain a vector V of size n to represent the local vector clock. Charron-Bost has shown that to capture the partial order $(E, \rightarrow)$, the size of the vector clock is the dimension of the partial order [3], which is bounded by the size of the system, n. Unfortunately, this does not scale well to large systems.

3 The Bloom Clock Protocol

The Bloom clock is based on the Counting Bloom filter. Each process P_i maintains a Bloom clock $B(i)$ which is a vector $B(i)[1, \ldots, m]$ of integers, where $m < n$. The Bloom clock is operated as shown in Algorithm 1. To try to uniquely update $B(i)$ on a tick for event e_i^x, k random hash functions are used to hash (i, x), each of which maps to one of the m indices in $B(i)$. Each of the k indices mapped to is incremented in $B(i)$; this probabilistically tries to make the resulting $B(i)$ unique. As $m < n$, this gives a space, time, and message-space savings over the vector clock. We would like to point out that the scalar clock [10] can be thought of as a Bloom clock with $m = 1$ and $k = 1$.

The Bloom timestamp of an event e is denoted B_e. Let $\mathcal{V}$ and $\mathcal{B}$ denote the sets of vector timestamps and Bloom timestamps of events. The standard vector comparison operators $<$, $\leq$, and $=$ [5,11] apply to pairs in $\mathcal{V}$ and in $\mathcal{B}$. Thus, for example, $B_z \geq B_y$ is $\forall i \in [1, m], B_z[i] \geq B_y[i]$. The Bloom clock mapping from E to $\mathcal{B}$ is many-one. $(\mathcal{B}, \leq)$ is a partial order that is not isomorphic to $(E, \rightarrow)$.

Algorithm 1: Operation of Bloom clock $B(i)$ at process P_i.

1 Initialize $B(i) = \overline{0}$.

2 (At an internal event e_i^x):
apply k hash functions to (i, x) and increment the corresponding k positions mapped to in $B(i)$ (local tick).

3 (At a send event e_i^x):
apply k hash functions to (i, x) and increment the corresponding k positions mapped to in $B(i)$ (local tick). Then P_i sends the message piggybacked with $B(i)$.

4 (At a receive event e_i^x for message piggybacked with B'):
P_i executes
$\forall j \in [1, m], B(i)[j] = max(B(i)[j], B'[j])$ (merge);
apply k hash functions to (i, x) and increment the corresponding k positions mapped to in $B(i)$ (local tick).
Then deliver the message.

Proposition 1. *Test for $y \rightarrow z$ using Bloom clocks: if $B_z \geq B_y$ then declare $y \rightarrow z$ else declare $y \not\rightarrow z$.*

4 Properties of the Bloom Clock

We have the following cases based on the actual relationship between events y and z, and the relationship inferred from B_y and B_z.

1. $y \rightarrow z$ and $B_z \geq B_y$: From Proposition 1, this results in a true positive.
2. $y \rightarrow z$ and $B_z \not\geq B_y$: This false negative is not possible because from the rules of operation of the Bloom clock, B_z must be $\geq B_y$ when $y \rightarrow z$.
3. $y \not\rightarrow z$ and $B_z \not\geq B_y$: From Proposition 1, this results in a true negative.
4. $y \not\rightarrow z$ and $B_z \geq B_y$: From Proposition 1, this results in a false positive.

Let pr_{fp}, pr_{tp}, and pr_{tn} denote the probabilities of a false positive, a true positive, and a true negative, respectively. Also, let pr_p denote the probability of a positive. To evaluate these probabilities, we need $pr(y \rightarrow z)$ and $pr(B_z \geq B_y)$. As we do not have access to vector clocks, we cannot evaluate $y \rightarrow z$ as $V_y \leq V_z$. So we estimate $pr(y \rightarrow z)$ as the probability that $B_z \geq B_y$, which is the probability of a positive, pr_p. So the estimate of pr_{fp} is $(1 - pr_p) \cdot pr_p$, from Case (4) above. However, the second term pr_p can be precisely evaluated, given B_y and B_z, as $pr_{\delta(p)}$, where

$$pr_{\delta(p)} = \begin{cases} 1 & \text{if } B_z \geq B_y \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

So $pr_{fp} = (1 - pr_p) \cdot pr_{\delta(p)}$. Also, $pr_{tp} = pr_p \cdot pr_{\delta(p)}$ from Case (1) above. Further, as a negative outcome ($B_z \not\geq B_y$) is always true from Cases (2,3) above and a negative outcome can be determined precisely, $pr_{tn} = 1 - pr_{\delta(p)}$. Thus,

$$\begin{aligned} pr_{fp} &= (1 - pr_p) \cdot pr_{\delta(p)}, \\ pr_{tp} &= pr_p \cdot pr_{\delta(p)}, \\ pr_{tn} &= 1 - pr_{\delta(p)} \end{aligned} \tag{2}$$

If $pr_{\delta(p)}$ were not precisely evaluated but used as a probability, we would have:

$$\begin{aligned} pr_{fp} &= (1 - pr_p) \cdot pr_p, \\ pr_{tp} &= pr_p^2, \\ pr_{tn} &= 1 - pr_p \end{aligned} \tag{3}$$

We now show how to estimate pr_p using Bloom timestamps B_y and B_z.

Definition 1. *For a vector X, $X^{sum} \equiv \sum_{i=1}^{|X|} X[i]$.*

For a positive outcome to occur, for each increment to $B_y[i]$, there is an increment to $B_z[i]$. The number of increments to $B_y[i]$, which we denote as c the *count threshold*, is $B_y[i]$. The probability pr_p of $B_z \geq B_y$ is now formulated. Let $b(l, q, 1/m)$ denote the probability mass function of a binomial distribution having success probability $1/m$, where l increments have occurred to a position in B_z after applying uniformly random hash mappings q times.

$$b(l, q, 1/m) = \binom{q}{l} (\frac{1}{m})^l (1 - \frac{1}{m})^{q-l} \tag{4}$$

Observe that the total number of trials $q = B_z^{sum}$. Then,

$$b(l, B_z^{sum}, 1/m) = \binom{B_z^{sum}}{l} (\frac{1}{m})^l (1 - \frac{1}{m})^{B_z^{sum} - l} \tag{5}$$

The probability that less than the count threshold $B_y[i]$ increments have occurred to $B_z[i]$ is given by:

$$\sum_{l=0}^{B_y[i]-1} b(l, B_z^{sum}, 1/m) \tag{6}$$

The probability that each i of the m positions of B_z is incremented at least $B_y[i]$ times, which gives pr_p, can be given by:

$$pr_p(k, m, B_y, B_z) = \prod_{i=1}^{m} (1 - \sum_{l=0}^{B_y[i]-1} b(l, B_z^{sum}, 1/m)) \tag{7}$$

Equation 7 is time-consuming to evaluate for events y and z as the execution progresses. This is because B_z^{sum} and $B_y[i]$ increase. A binomial distribution

$b(l, q, 1/m)$ can be approximated by a Poisson distribution with mean q/m, for large q and small $1/m$. Also, the cumulative mass function of a Poisson distribution is a regularized incomplete gamma function. This provides an efficient way of evaluating Eq. 7.

For arbitrary event y at P_i, to predict whether $y \rightarrow z$ where events z occur at P_j, there are at first true negatives, then false positives, and then true positives as z occurs progressively later. As $B_z^{sum} - B_y^{sum}$ increases, we can predict the following trends from the definitions of pr_p and pr_{fp}.

1. pr_p, the probability of a positive, is low if z is close to y and this probability increases as z goes further in the future of y. This is because, in Eq. 7, as B_z^{sum} increases with respect to B_y^{sum} or rather its m components, the summation (cumulative probability distribution function) decreases and hence pr_p increases.
   This behavior is intuitive because intuition says that as z becomes more distant from y, the more is the likelihood that some causal relationship will get established from y to z either directly or transitively, by the underlying message communication pattern.
2. pr_{fp}, the probability of a false positive, which is the product $(1 - pr_p) \cdot pr_p$ using Eq. 3, is lower than the two terms. It will increase, reach a maximum of 0.25, and then decrease.
   If Eq. 2 were used, then $pr_{fp} = (1 - pr_p) \cdot pr_{\delta(p)}$ would be higher for a positive outcome. Once $B_z \geq B_y$ becomes true, it steps up from 0 and then as z goes into the future of y, it decreases. Given a positive outcome, if $B_z \geq B_y$ and z is close to y (B_z^{sum} is just a little greater than B_y^{sum}), there are two opposing influences on pr_{fp}: (i) it is unlikely that "a causal relationship has been established either directly or transitively from y to z by the underlying message communication pattern", and thus $1 - pr_p$ and pr_{fp} should tend to be high; (ii) it is also unlikely that "for each $h \in [1, m]$, $B_z[h] \geq B_y[h]$ due to Bloom clock local ticks only (and not due to causality merge for $y \rightarrow z$)", and thus pr_{fp} should tend to be low. As z goes more distant from y, the likelihood of influence (i) that a causal relation has been established increases, resulting in a lower $1 - pr_p$ and hence lower pr_{fp}. This *overrides* any conflicting impact of the likelihood of influence (ii), that $\forall h, B_z[h] \geq B_y[h]$ due to local ticks only and not due to causality merge for $y \rightarrow z$, increasing and thus increasing pr_{fp}.
   Based on the above reasoning, it is not apparent whether Eq. 2 or 3 is better for modeling pr_{fp} behavior. However, Eq. 2 uses the full range of [0,1] (as opposed to [0,0.25]), and uses an approximation only for $pr(y \rightarrow z)$ and not for $pr(B_z \geq B_y)$.

We remind ourselves that these probabilities depend on B_y, B_z, k, and m, and observe that they are oblivious of the communication pattern in the distributed execution.

We are also interested in calculating the accuracy, precision, and false positive rate of Bloom clocks. Accuracy (Acc), precision ($Prec$), recall (Rec), and false positive rate (fpr) are metrics defined over all data points, i.e, pairs of events, in

the execution. Let TP, FP, TN, and FN be the number of true positives, number of false positives, number of true negatives, and the number of false negatives, respectively. Observe that FN is 0 as there are no false negatives. We have:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, Precision = \frac{TP}{TP + FP}, \\ Recall = \frac{TP}{TP + FN}, fpr = \frac{FP}{FP + TN} \tag{8}$$

Recall is always 1 with Bloom clocks. Given events y and z and their Bloom timestamps B_y and B_z, there is not enough data to compute these metrics. So we consider the slice of the execution from y to z and define the metrics over the set of events in this slice.

We observe that many applications in distributed computing require testing for causality between pairs of events that are temporally close to each other. In checkpointing, causality needs to be tracked only between two consistent checkpoints. In fair mutual exclusion in which requests need to be satisfied in order of their logical timestamps, contention occurs and request timestamps need to be compared only for temporally close requests. For detecting data races in multi-threaded environments, a causality check based on vector clocks can be used; however, in practice one needs to check for data races only between events that occur in each other's temporal locality [14,15]. In general, many applications are structured as phases and track causality only within a bounded number of adjacent phases [4,13]. Thus, in our experiments to measure accuracy, precision, and false positive rate, as well as the probability of positives and the probability of false positives, we consider an execution slice that is relatively thin.

There is a trade-off using Bloom clocks. m can be chosen less than n, for space, time, and message-space savings. But for acceptable precision, accuracy, and *fpr*, and a suitable pr_{fp} distribution, an appropriate combination of values for the clock parameters m and k can be determined.

5 Experiments for the Complete Graph

In the complete graph, we assume a logical channel between each pair of processes. This experiment consists of a decentralized system of processes asynchronously passing messages to each other over shared memory. The processes are scheduled in a fair manner and are identical to each other. Even though FIFO channels are not maintained, a majority of messages arrive in order. The parameters of this experiment are *number of processes (n)*, *size of Bloom clock (m)*, *internal event probability* (pr_i), and *number of hash functions (k)*. Each event can be uniquely identified with a *Global Sequence Number (GSN)*. An event is modelled as an object with the following attributes: (i) vector timestamp, (ii) Bloom timestamp, (iii) GSN, (iv) executing process ID, (v) sending process ID, (vi) receiving process ID, (vii) physical timestamp.

The main program establishes shared memory, creates n processes and supplies them with parameters pr_i, k, and m. It then waits for all processes to

complete execution and analyzes the distributed execution log. Shared memory consists of an integer tracking GSN, a message queue containing messages (send events) yet to be received, and an execution log containing all events executed at any point of the distributed execution. All processes maintain a local queue containing messages asynchronously pulled from the shared message queue. Message receive events are executed by processing messages one at a time from the local queue with probability $(1 - pr_i)/2$. Send events are executed with probability $(1 - pr_i)/2$. For each send event the sending process randomly selects a receiving process from the other $n - 1$ processes. Processes execute internal events with probability pr_i. All executed events are pushed into the global execution log. Send events are also pushed into the global message queue.

Each process maintains its own vector clock and Bloom clock which are ticked in accordance to the vector clock and Bloom clock protocols, whenever an event is executed. The event object stores the local process's revised clocks as its vector and Bloom timestamps. In addition to this, upon executing an event, each process increments the global GSN variable by 1 and stores it in the event object. Whenever a process increments the global GSN counter, it has to acquire a lock. This is done to prevent race conditions on the GSN counter as it is stored in shared memory. Other operations that are required to be atomic and around which locks are used include accessing the global message queue in shared memory containing messages that are waiting to be retrieved. Each process continues to iterate and execute events until the GSN reaches n^2. Once all processes terminate, the main program analyzes the execution to compute precision, accuracy, and *fpr* of the Bloom clock protocol from the execution log. The execution log contains approximately n^2 events at the end of the execution.

The main program computes causal relationships of pairs of events in the execution slice beginning with the event with GSN $= 10n$ (to eliminate any startup effects) and until the last event (with GSN $= n^2$) in steps of 100. This means that the sample that we use to check for causality predictions consists of a series of events where two closest events have a difference of 100 in GSN. Further, the number of pairs of events for which we tested for causality was approximately $n^4/10^4$. The main program compares causality predictions of the Bloom timestamps of events with predictions of vector timestamps and classifies the Bloom clock predictions as true positives, false positives and true negatives. The precision, accuracy, and *fpr* are computed over this execution slice. We intentionally chose an execution slice with n events per process because in practice, causality tests are applied to pairs of events in the temporal proximity of each other. Had we chosen a larger execution slice, we expect the metrics would have improved.

Finally, in this section and the next on experiments with the star configuration, each reading reported is the average of at least 3 runs of each setting of the parameters indicated. Also, in Sects. 5.2 to 5.4, where indicated, each reported reading is also averaged over multiple settings of m and/or k for simplicity of presentation of results; the impact of varying each individual parameter is clear when the results of all experiments are considered.

5.1 Number of Processes

We ran the decentralized experiment for $n = 100$ to $n = 700$ in increments of 100 to ascertain scalability of Bloom clocks. Parameters were fixed to maintain uniformity of results with $pr_i = 0$, $k = 2$, and $m = 0.1 * n$. The results are compiled in Table 1. A visual representation of the trend can be seen in Fig. 1. We see that as n increases Bloom clock performance improves considerably. Accuracy increases from 85.2% for $n = 100$ to 95.7% for $n = 700$ and the *fpr* drops from 20.3% for $n = 100$ to 7.4% for $n = 700$. Since Bloom clocks are not prone to false negatives, a critical method of measuring performance is to calculate the ratio of positive predictions that are correct to overall positive predictions. Precision measures exactly that. We observe that precision increases from 64.4% for $n = 100$ to 90.7% for $n = 700$. Overall from Table 1, we conclude that Bloom clocks are highly scalable.

Table 1. Variation of metrics with n

n	Precision	Accuracy	*fpr*
100	0.644	0.852	0.203
200	0.781	0.905	0.145
300	0.833	0.926	0.118
400	0.856	0.935	0.107
500	0.883	0.947	0.089
600	0.897	0.953	0.081
700	0.907	0.957	0.074

5.2 Internal Event Probability

We ran the decentralized experiment for fixed $n = 200$ and averaged metrics over $m = 0.1 * n, 0.2 * n, 0.3 * n$ and $k = 2, 3, 4$ for individual values of pr_i in order to observe the variation of metrics with pr_i. The results are shown in Table 2. We observed that by introducing more relevant (and therefore timestamped) internal events in the decentralized execution, the performance of Bloom clocks deteriorates significantly. So with an increase in send events and thus message-passing, i.e., a relative decrease in the number of relevant timestamped internal events, more causal relationships get established among events across processes, which get captured through the merging of Bloom clocks at receive events. This results in a higher fraction of the number of pairs of events being related by causality and a smaller fraction of the number of pairs of events being concurrent. Bloom clocks performed best at $pr_i = 0$. We generalize this observation as the *causality spread hypothesis* later in Sect. 7.2.

The practical implication of setting $pr_i = 0$ is that most of the relevant events at which clocks tick are send and receive events, and only a few internal events (of interest to the application) cause the clocks to tick. In contrast, with a high

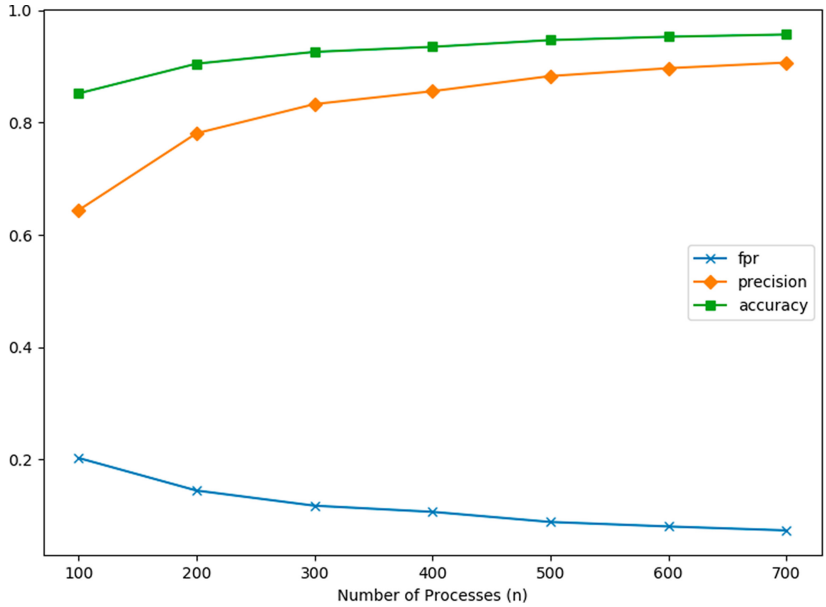

Fig. 1. A plot of metrics vs. number of processes for decentralized execution

Table 2. Variation of metrics with pr_i

pr_i	Precision	Accuracy	*fpr*
0	0.807	0.918	0.125
0.90	0.609	0.847	0.201
0.95	0.311	0.760	0.269
1	0.101	0.773	0.232

value of pr_i (such as 0.9 at which 90% of events at which clocks tick are internal events), accuracy and precision drop significantly, and *fpr* increases significantly. Thus, Bloom clocks are practical only when the percentage of relevant events (where clock ticks) that are internal events is small.

5.3 Number of Hash Functions

We ran the decentralized experiment for fixed $n = 200$ and fixed $pr_i = 0$ and averaged metrics over $m = 0.1 * n, 0.2 * n, 0.3 * n$ for individual values of k to check the variation of Bloom clock performance with respect to k. The results are shown in Table 3. We observe that the effect of changing the number of hash functions does not have a quantifiable effect on Bloom clock performance.

Table 3. Variation of metrics with k

k	Precision	Accuracy	*fpr*
2	0.804	0.917	0.126
3	0.809	0.919	0.124
4	0.808	0.919	0.124

5.4 Size of Bloom Clock

We ran the decentralized experiment for fixed $n = 200$ and fixed $pr_i = 0$ and averaged metrics over $k = 2, 3, 4$ for individual values of m to check the variation of Bloom clock performance with respect to m. The results are shown in Table 4. As expected, Bloom clock performance improves, but by up to 4.3% points, as m increases from $0.1 * n$ to $0.3 * n$. The improvement seems intuitive because with a larger number of indices the probability of hash function outputs mapping to the same indices reduces, due to which there is a lower probability of false positives.

Table 4. Variation of metrics with m

m	Precision	Accuracy	*fpr*
$0.1 * n$	0.784	0.906	0.143
$0.2 * n$	0.811	0.920	0.122
$0.3 * n$	0.827	0.929	0.109

In addition, we ran the experiment with scalar clock ($m = 1$ and $k = 1$) instead of Bloom clock, in order to investigate improvement in metrics for Bloom clock over scalar clock. We compared Bloom clock of size $m = 0.1 * n$ and $k = 2$ to scalar clock at various values of n for $pr_i = 0$. The results are presented in Table 5. We observe significant performance improvements over scalar clock by utilizing Bloom clock at all values of n – precision was 0.06 to 0.11, accuracy was 0.07 to 0.09, and *fpr* was 0.10 to 0.12 better.

Table 5. Bloom clock vs. scalar clock

n	Bloom Clock			Scalar Clock		
	Precision	Accuracy	*fpr*	Precision	Accuracy	*fpr*
50	0.492	0.788	0.266	0.434	0.713	0.368
100	0.644	0.852	0.203	0.542	0.769	0.318
200	0.781	0.905	0.145	0.672	0.835	0.248

5.5 Plots for pr_p and pr_{fp}

We ran the decentralized experiment for fixed parameters $n = 100$, $pr_i = 0$, $k = 2$ and $m = 0.1 * n$ to obtain plots for pr_p, and pr_{fp} computed using Eqs. 2 and 3. These plots demonstrate the behavior of Bloom clocks throughout an execution as the temporal proximity between events y and z varies, using just the Bloom timestamps of the two events being compared for causality. For these plots we fix event y with $GSN = 10 * n$, which is 1000, to allow for any startup transient effects, and compare its Bloom timestamp with all events z with $GSN = 10*n+1$ to $GSN = 4500\,(\sim n^2/2)$. This slice of the execution is adequate to capture all the trends. The x-axis of Figs. 2, 3 and 4 is the GSN of z and the y-axis is the probability being plotted.

Figure 2 shows a plot of pr_p as a function of GSN. We observe that as GSN increases, the probability of a positive prediction increases and flattens to around 1 between GSN = 3500 and GSN = 4000. This is because as the distance between two events increases, there is a higher probability of a causal relationship being established either directly or transitively. The split view of pr_p vs. GSN allows us to observe that most false positives occur in the middle of the distribution while all true negatives occur within the first half of the execution. This is due to the fact that initially the probability of a true negative is very high because the probability of a causal relationship being established is lower.

Figure 3 shows plots for $pr_{fp} = (1 - pr_p) \cdot pr_{\delta(p)}$ (Eq. 2) vs. GSN. We observe that Bloom clocks correctly predict the probability of false positive being 0 for all true negatives in the execution. Most of the false positives are distributed in the middle of the execution slice; the pr_{fp} jumps from 0 to large values once false positives start occurring and then gradually decreases as GSN increases. The (few) false positives that occur towards the end of the execution slice are not captured correctly with low values of pr_{fp}. The probability of false positive for a majority of true positives is below 0.25; however, for the initial few true positives, the pr_{fp} is inaccurately evaluated as being high. This probability pr_{fp} (for the true positives) rapidly decreases to 0 as GSN increases.

Figure 4 shows plots for $pr_{fp} = (1 - pr_p) * pr_p$ (Eq. 3) vs. GSN. As expected, pr_{fp} has values below 0.05 for most true negatives and true positives and reaches a maximum value of 0.25 in the middle of the execution where most of the false positives reside. Thus, the pr_{fp} is inaccurately evaluated as being low for the false positives in the middle of the execution slice.

Thus, Figs. 2, 3 and 4 confirm the theoretical predictions made in Sect. 4. Equation 2 uses a range of [0,1] for pr_{fp}, gives a high pr_{fp} to the initial few true positives, and does not seem to capture the two conflicting influences on pr_{fp} described in Sect. 4 when the GSN of z is just a little greater than the GSN of y. Equation 3 uses a range of only [0,0.25] and inaccurately gives a low pr_{fp} for the false positives in the middle of the execution slice.

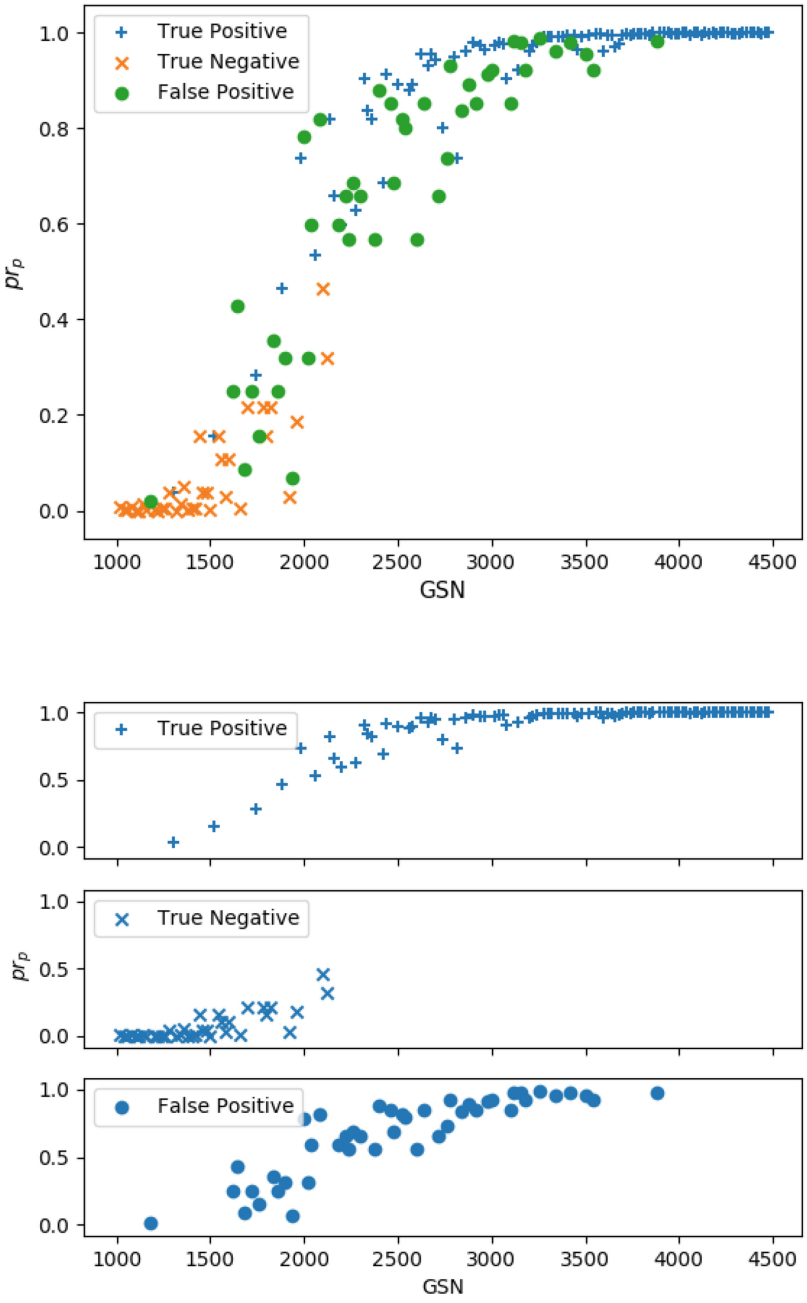

Fig. 2. pr_p vs. GSN, showing combined view and split view

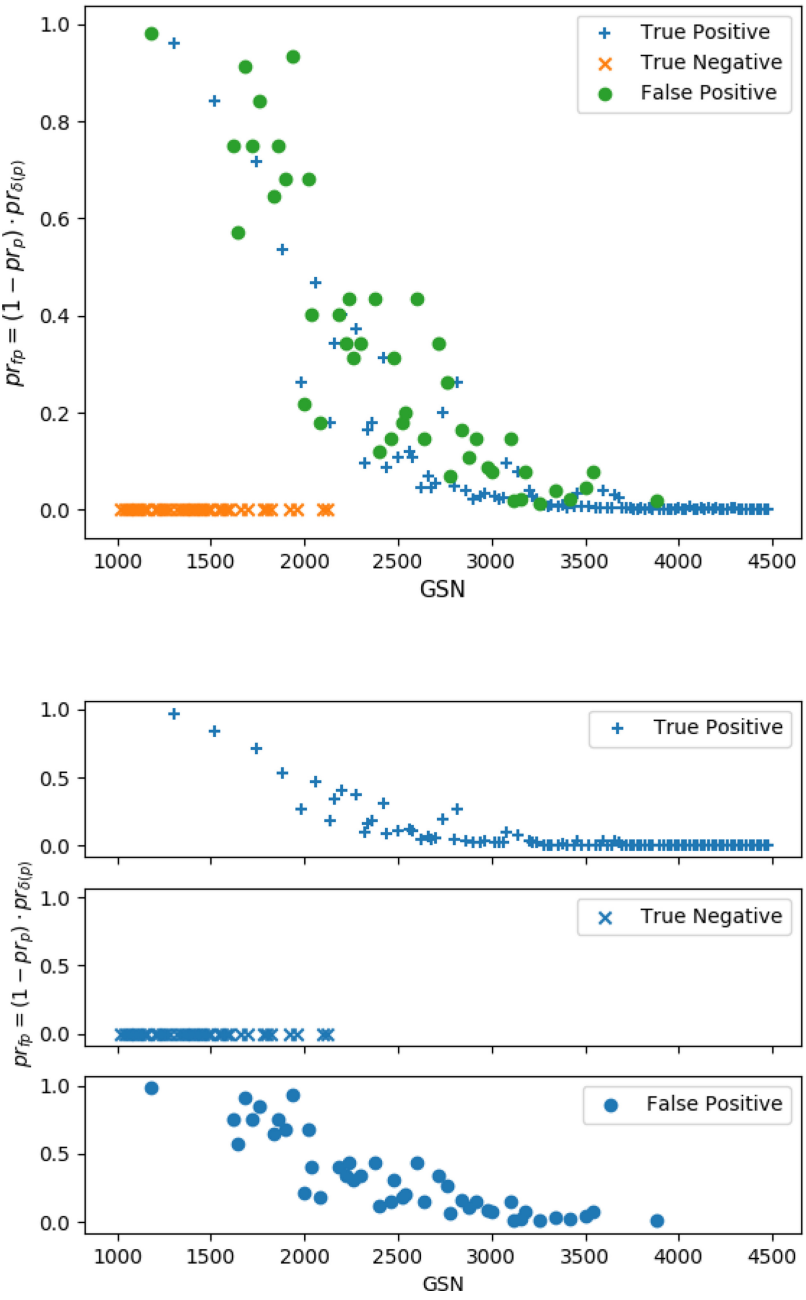

Fig. 3. $pr_{fp} = (1 - pr_p) \cdot pr_{\delta(p)}$ using Eq. 2 vs. GSN, showing combined view and split view

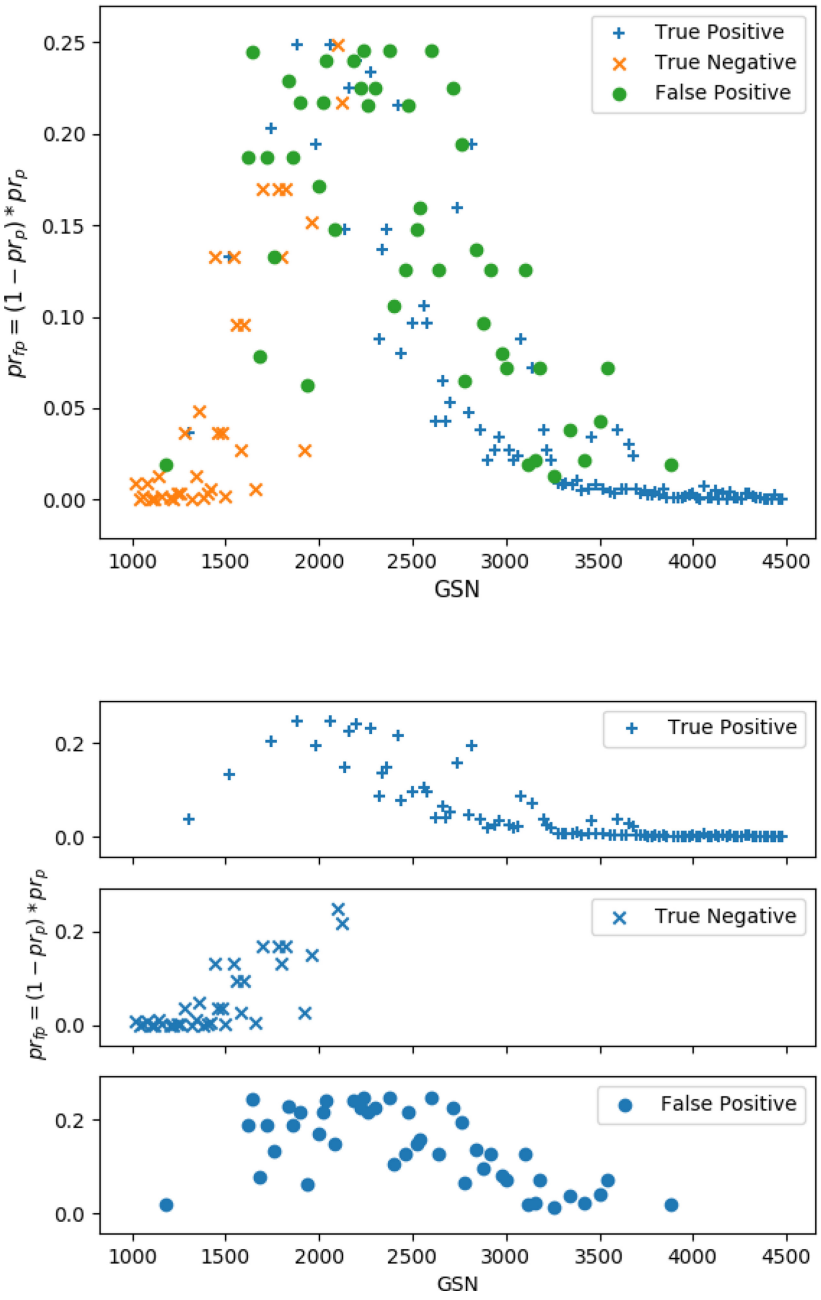

Fig. 4. $pr_{fp} = (1 - pr_p) \cdot pr_p$ using Eq. 3 vs. GSN, showing combined view and split view

6 Experiments for the Star Graph

We set up an experiment with a client-server architecture to investigate how faithfully the Bloom clock determines causality. Client processes connect to a multi-threaded server accepting TCP connections. Each server thread connects to a single client. The internal event probability, pr_i was set to 0. All message sends were synchronous and blocking and receives were blocking. Each client consisted of a process and had its own vector clock and Bloom clock. The server had a single vector clock and a single Bloom clock shared across all threads.

The server threads used a single lock to make sure that there were no race conditions on the vector clock and the Bloom clock while executing events. We did not use locks at the client end because GSN was not maintained. Further, not using locking mechanisms allowed interleaving of client processes.

Each client sent n messages to the server and received n corresponding messages from the server. This resulted in overall $O(n^2)$ events in the execution. Post execution, each 100^{th} event was taken from the execution log containing all the events from the execution to create a sample of events to be compared for causality. Each event y was compared to each other event z to determine if Bloom clock correctly classified whether $y \rightarrow z$ or $y \not\rightarrow z$. The correctness of the Bloom clock prediction was ascertained by comparing it with the prediction from vector clock. The results for the client-server experiment for $k = 2$ are shown in Table 6.

As can be seen from the results, Bloom clock performs quite well with high values of precision and low *fpr*. The first four rows are for $m = 0.1 * n$ and the last four rows are for $m = 0.05 * n$. We observed that for a small Bloom clock of size $m = 3$ for $n = 50$, the accuracy is high at 100% (There was one false positive, but rounding off to three decimal places results in the stated accuracy value). The difference in precision, accuracy, and *fpr* for smaller Bloom clocks as compared to larger Bloom clocks is not significant, therefore it is safe to say that for this configuration, smaller Bloom clocks perform well. The reason for strong performance of Bloom clock is that there are a lot of merge events with a centralized process, and the inherent message pattern at the server resulted in automatic and widespread distribution/broadcasting of information contained in individual Bloom clocks among all client processes. The server is always up to date with a client's Bloom clock after it executes a receive event corresponding to a message send event from the client. We generalize the reasoning behind the good performance of the Bloom clock for the client-server configuration by postulating the *causality spread hypothesis* in Sect. 7.2.

7 Observations and Discussion

7.1 Summary of Results

The results of the experiments are summarized as follows.

1. In predicting the causality between events y and z using their Bloom timestamps, we observe the following.

Table 6. Results for client-server experiment with $k = 2$

n	m	Precision	Accuracy	*fpr*
50	5	0.985	0.992	0.015
100	10	0.990	0.995	0.010
125	13	0.991	0.996	0.009
150	15	0.995	0.997	0.005
50	3	100	100	0
100	5	0.996	0.998	0.004
125	7	0.997	0.998	0.003
150	8	0.997	0.998	0.003

   (a) The probability of a positive pr_p increases relatively quickly from 0 to 1 as z occurs after but in the temporal vicinity of y.
   (b) The probability of a false positive pr_{fp} is 0 or close to 0 except when z occurs later than but in the temporal vicinity of event y. As z occurs later at a process, the probability spikes up from 0 to a high value but soon comes down to 0 as the occurrence of z get temporally separated from the occurrence of y. Some true positives have a non-zero value of pr_{fp}.
2. As the number of processes n increases, the Bloom clock performance improves significantly – the accuracy and precision increase, and the *fpr* decreases.
3. When the number of internal events at which the clock ticks is low relative to the number of send events, precision, accuracy, and *fpr* all improve significantly. Thus, with relatively more send events, performance of Bloom clock improves. With more send events, causality between more pairs of events is established. On the other hand, if the number of internal events being timestamped is high with respect to the number of send events, Bloom clocks do not perform well.
4. The number of hash functions k used in the Bloom clock protocol does not impact much the precision, accuracy, and the *fpr*. Hence, it is advantageous to use a small number (such as 2 or 3) of hash functions.
5. The precision, accuracy, and the *fpr* improved by a few percentage points as the size of the Bloom Clock m was increased from $0.1 * n$ to $0.3 * n$. The impact is noticeable but not much. Hence, this suggests that small-sized Bloom Clocks can be used to gain significant space, time, and message-space savings over vector clocks. As a baseline for comparison, we also measured the precision, accuracy, and *fpr* for Lamport's scalar clocks. The scalar clocks performed noticeably worse.
6. For the client-server configuration, Bloom clocks performed exceedingly well.

Bloom clocks are seen to provide a viable space-, time-, and message-space-efficient alternative to vector clocks when some false positives can be tolerated. Bloom clock metrics improve as the number of processes increases. Bloom clock

sizes can be 10% or even lower of the number of processes, and can handle churn transparently when processes join and leave the system. The probability of a false positive is high only when the two events occur temporally very close to each other. However, Bloom clocks do not perform well when the fraction of timestamped events that are internal events is not very low. In the next section, we generalize this behavior using the *causality spread hypothesis*.

7.2 Causality Spread

After conducting experiments to track causality using the Bloom clock for multiple architectures and varying parameters, we develop a hypothesis to help system engineers and software developers figure out whether the Bloom clock is a good fit for a given application. This hypothesis is stated only from the application's perspective. We hypothesize that with an increase in spread of causality in an execution, i.e., with a larger proportion of events related by causal relationships, Bloom clock performance (i.e., confidence in its predictions) increases. We define and compute the *causality spread*, α, as the ratio of the number of ordered pairs of events that are causally related, that is, *total positives*, to the sum of all ordered pairs of events compared for each execution. The set of events that we include in the computation of causality spread are the *relevant events* for the application.

Definition 2 (Causality spread α)

$$\begin{aligned} \textit{Causality spread } \alpha &= \frac{\textit{Total Positives}}{\#\ \textit{All pairs of events}} \\ &= \frac{\textit{Total Positives}}{\textit{Total Positives + Total Negatives}} \\ &= \frac{TP + FN}{TP + FN + FP + TN} = \frac{TP}{TP + FN + FP + TN} \end{aligned} \tag{9}$$

Hypothesis 1 (Causality spread hypothesis). *The confidence in the predictions of the Bloom clock as measured by precision, accuracy, and* fpr *increases as the causality spread α of the application's set of relevant events increases.*

A higher α signifies more (fraction of) event pairs being related by causality, which are correctly classified as true positives, thereby increasing TP (say, by a), decreasing FP, decreasing TN, and decreasing FP + TN (by a). Theoretically, we expect precision and accuracy will improve (as per some non-linear functions), while the impact on *fpr* depends on the factors by which its numerator FP and its denominator FP + TN change.

This hypothesis is corroborated by our previously stated observation that increased message passing results in superior Bloom clock predictions. In order to quantify this hypothesis, we took a sample of executions from both the decentralized experiment and the client-server experiment and computed α. We observed that precision and accuracy increase and *fpr* decreases as α increases,

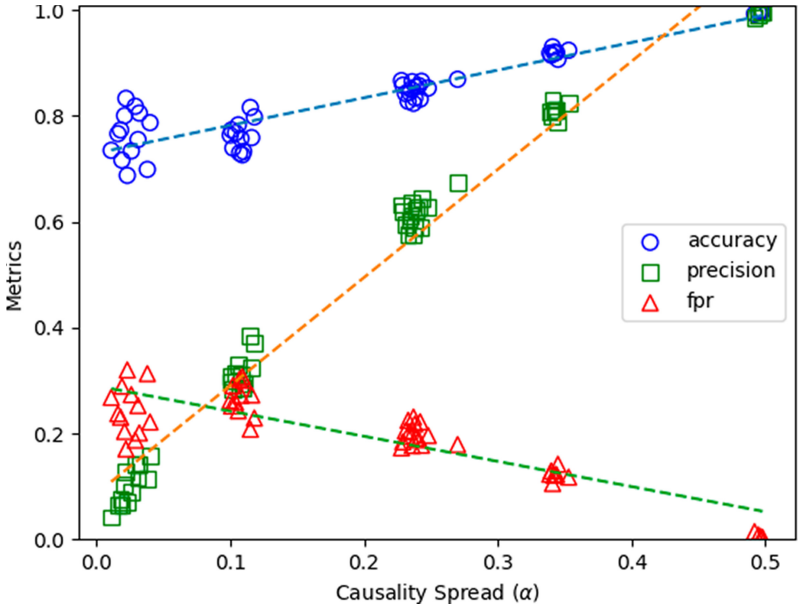

Fig. 5. A plot of metrics vs. *causality spread*

for $0 < \alpha < 0.5$, empirically confirming our hypothesis. A graph showing the increase in metrics as a function of causality spread is shown in Fig. 5.

An important note about causality spread is that it will range between 0 and 0.5 in our experiment because we check for causality between all pairs of events. An extreme case where $\alpha = 0$ would be each process executing only one event. Another extreme case where $\alpha = 0.5$ would be a linear chain of events. In the client-server experiment, α is near 0.5 due to the nature of transmission of causal relationships because of the server behavior. In the complete graph configuration with a high pr_i, the large number of timestamped internal events in the set of relevant events significantly increases the number of pairs of concurrent events and hence decreases α considerably, resulting in poor prediction by Bloom clocks.

We performed an experiment for multicast/broadcast messages to check if it conforms to our causality spread hypothesis. In the broadcast experiment, each process broadcasts a message to all $n - 1$ processes and waits to receive $n - 1$ broadcast messages from the other processes. Here, causality does not spread much because there is only one message send event followed by many receive events for each process. Here the receive events act as internal events that are timestamped (akin to high pr_i), and in effect there are many pairs of events that are concurrent and hence not related by causality, thereby resulting in a low α. In the experiment, $\alpha = 0.005$, precision $= 0.014$, accuracy $= 0.661$, and fpr $= 0.341$. The poor performance of Bloom clock in this experiment can be attributed to a low α as per the hypothesis.

8 Conclusions

Detecting the causality relationship between a pair of events in a distributed execution is a fundamental problem. To address this problem in a scalable way, this paper gave the formal Bloom clock protocol, and derived the expression for the probability of false positives, given two events' Bloom timestamps. We ran experiments to calculate the accuracy, precision, and *fpr* for a slice of the execution. We also ran experiments to calculate the probability of a false positive prediction based on the Bloom timestamps of two events. Based on the experiments, we made suggestions for the number of hash functions and size of Bloom clocks and identified conditions under which it is advantageous to use Bloom clocks over vector clocks. The findings are summarized as follows.

1. Bloom clocks can perform well for small size m and small number of hash functions k.
2. Bloom clocks perform well when the number of internal events considered is low compared to the number of send events (low pr_i).
3. Bloom clocks perform increasingly better as the system size n increases.
4. We also postulated the *causality spread hypothesis* from the application's perspective to determine whether Bloom clocks would give good performance (precision, accuracy, and *fpr*) for the application, and validated it through experiments. A high α indicates good performance.

Thus, Bloom clocks are seen to provide a viable space-, time-, and message-space-efficient alternative to vector clocks for the class of applications which meet the properties summarized above, when some false positives can be tolerated. It would be interesting to study the applicability of Bloom clocks to some practical applications.

References

1. Bloom, B.: Space/time tradeoffs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
2. Broder, A.Z., Mitzenmacher, M.: Survey: network applications of bloom filters: a survey. Internet Math. **1**(4), 485–509 (2003). https://doi.org/10.1080/15427951.2004.10129096
3. Charron-Bost, B.: Concerning the size of logical clocks in distributed systems. Inf. Process. Lett. **39**(1), 11–16 (1991). https://doi.org/10.1016/0020-0190(91)90055-M
4. Couvreur, J., Francez, N., Gouda, M.G.: Asynchronous unison (extended abstract). In: Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, Japan, 9–12 June 1992, pp. 486–493 (1992). https://doi.org/10.1109/ICDCS.1992.235005
5. Fidge, C.J.: Logical time in distributed computing systems. IEEE Comput. **24**(8), 28–33 (1991). https://doi.org/10.1109/2.84874
6. Kshemkalyani, A.D., Khokhar, A.A., Shen, M.: Encoded vector clock: using primes to characterize causality in distributed systems. In: Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN 2018, Varanasi, India, 4–7 January 2018, pp. 12:1–12:8 (2018). https://doi.org/10.1145/3154273.3154305

7. Kshemkalyani, A.D., Misra, A.: The bloom clock to characterize causality in distributed systems. In: Barolli, L., Li, K.F., Enokido, T., Takizawa, M. (eds.) NBiS 2020. AISC, vol. 1264, pp. 269–279. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-57811-4_25
8. Kshemkalyani, A.D., Shen, M., Voleti, B.: Prime clock: encoded vector clock to characterize causality in distributed systems. J. Parallel Distrib. Comput. **140**, 37–51 (2020). https://doi.org/10.1016/j.jpdc.2020.02.008
9. Kshemkalyani, A.D., Singhal, M.: Distributed Computing: Principles, Algorithms, and Systems. Cambridge University Press, Cambridge (2011). https://doi.org/10.1017/CBO9780511805318
10. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
11. Mattern, F.: Virtual time and global states of distributed systems. In: Proceedings of the Parallel and Distributed Algorithms Conference, pp. 215–226 (1988)
12. Meldal, S., Sankar, S., Vera, J.: Exploiting locality in maintaining potential causality. In: Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, PODC 1991, pp. 231–239. ACM, New York (1991). https://doi.org/10.1145/112600.112620
13. Misra, J.: Phase synchronization. Inf. Process. Lett. **38**(2), 101–105 (1991). https://doi.org/10.1016/0020-0190(91)90229-B
14. Pozzetti, T.: Resettable Encoded Vector Clock for Causality Analysis with an Application to Dynamic Race Detection. M.S. Thesis, University of Illinois at Chicago (2019)
15. Pozzetti, T., Kshemkalyani, A.D.: Resettable encoded vector clock for causality analysis with an application to dynamic race detection. IEEE Trans. Parallel Distrib. Syst. **32**(4), 772–785 (2021). https://doi.org/10.1109/TPDS.2020.3032293
16. Ramabaja, L.: The bloom clock. CoRR (2019). http://arxiv.org/abs/1905.13064
17. Schwarz, R., Mattern, F.: Detecting causal relationships in distributed computations: in search of the holy grail. Distrib. Comput. **7**(3), 149–174 (1994). https://doi.org/10.1007/BF02277859
18. Singhal, M., Kshemkalyani, A.D.: An efficient implementation of vector clocks. Inf. Process. Lett. **43**(1), 47–52 (1992). https://doi.org/10.1016/0020-0190(92)90028-T
19. Tarkoma, S., Rothenberg, C.E., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. IEEE Commun. Surv. Tutor. **14**(1), 131–155 (2012). https://doi.org/10.1109/SURV.2011.031611.00024
20. Torres-Rojas, F.J., Ahamad, M.: Plausible clocks: constant size logical clocks for distributed systems. Distrib. Comput. **12**(4), 179–195 (1999). https://doi.org/10.1007/s004460050065