Information Processing Letters 110 (2010) 614–620

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Log′ version vector: Logging version vectors concisely in dynamic replication ☆

Hyun-Gul Roh [a,*], Myeongjae Jeon [b], Euiseong Seo [c], Jinsoo Kim [d], Joonwon Lee [d]

[a] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea
[b] Department of Computer Science, Rice University, Houston, TX, United States
[c] School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea
[d] School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Republic of Korea

ARTICLE INFO

Article history:
Received 1 December 2009
Received in revised form 29 April 2010
Accepted 30 April 2010
Available online 6 May 2010
Communicated by M. Yamashita

Keywords:
Distributed computing
Distributed systems
Version vector
log′ version vector
Replication system

ABSTRACT

In a replication system, version vectors are logged with replicas to detect conflicts among operations. Dynamic replications where replicas are frequently created and destroyed suffer from expensive logging overhead caused by inactive entries of version vectors. Although the rigmarole of pruning vectors can delete inactive entries, the vectors may be incompatible without additional information, which also causes another overhead. This paper proposes a novel version vector called log′ (log-prime) consisting of only three entries. By encoding based on the characteristics of prime numbers, log′ version vectors of fixed size can be logged concisely with no pruning technique at a little sacrifice in accuracy. Simulation studies show that log′ version vectors are accurate enough to detect almost all conflicts in the replication systems where all replicas are fully synchronizing.

1. Introduction

In diverse replication systems [1–5], replicas *operate* version vectors to enumerate distributed operations and *log* a large number of vector values, each of which captures individual operation, in order to track updates of replicas or to detect/resolve conflicts arising from distributed execution. Therefore, many replication systems have been suffering from the *logging overhead* by version vectors [3–5]. The transmission overhead can be reduced by some methods [6,7] primarily devised for the vector clock, which is inherently similar to the version vector in the operating rules and detecting mechanisms [8,9]. However, these methods have no effect on reducing the logging overhead of version vectors that is intrinsically related to the vector sizes. Even though the approaches in [3] and [4] claim to reduce the logging overhead, they are ineffective when vectors are to contain many *inactive entries* in dynamic replications in which replicas are frequently created and destroyed.

Indeed, for the compatibility of vectors, the entries representing destroyed (retired) replicas should be preserved, though they will remain inactive. To reduce the logging overhead however, some research efforts have tried to remove inactive entries through the *vector pruning*, but reckless pruning results in serious side-effects, as explained in [10]. Fig. 1 shows a simple case where an inactive entry is deleted immediately. If vectors are anonymous arrays, they might be corrupted due to the incompatibility of the vectors. To avoid such incompatibility, several systems [11–13] enforce an agreement on the vector pruning usually through distributed consensus protocols, which involve great cost and complexity. Although replicas may manage to update vectors without deleted entries by tagging each vector entry, any two vectors representing different sets of

☆ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0000829)

* Corresponding author.
E-mail address: hgroh@calab.kaist.ac.kr (H.-G. Roh).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2010.04.026

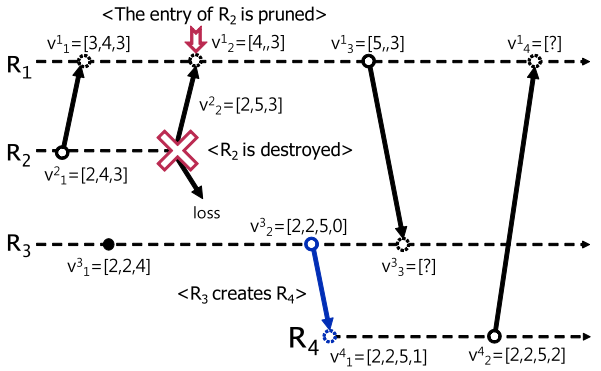

Fig. 1. A case in which a replica is destroyed and created. The entry of the destroyed replica is deleted immediately. If version vectors are in the forms of anonymous arrays, there might be ambiguity to update v_3^3 and v_4^1.

replicas are still incompatible because the pruning makes vectors lose necessary information after all.

To ensure compatibility of pruned vectors, Richard [14] and Wang et al. [15] individually introduced vector pruning techniques that prepare lists of final vector values of the destroyed replicas. These techniques are lightweight because they only require a retiring replica to notify others of its own retirement instead of costly consensus protocols. However, if a replica crashes without explicit notification, it is impossible to prune vectors. Saito [10] suggested a unilateral pruning technique that makes use of a physical clock for each entry. Although this can delete inactive entries without notification after fairly long expiration periods (e.g., a month), maintaining physical clocks incurs considerable overhead.

In fact, all pruning techniques demand additional information, such as entry tags or vector descriptions, which is another source of overhead. Reducing the logging overhead through the pruning techniques is consequently not always valid and practical, and thus vector sizes may grow continuously throughout dynamic replication.

This paper proposes a novel version vector named `log′` (*log-prime*). A `log′` is obtained by taking logarithms on a product of prime numbers which is derived from another novel vector called *prime version vector*. Every `log′` consists of only three fixed-size entries regardless of the number of replicas, thereby requiring no pruning. Meanwhile, we can make a compromise between the data size of `log′` and the accuracy in detecting conflicts. `Log′` satisfies the same property as *the plausible clock*, which was designed to be less powerful and accurate than the vector clock for scalability and efficiency [16]. Unlike the plausible clock, `log′`s cannot be updated for themselves, and are used only for the purpose of logging.

Our simulation studies show various characteristics of `log′` errors, which are controllable with respect to the vector size. In the replication where all replicas synchronize with each other, the error ratio is less than 2% for all pairs of thousands of `log′`s even when only half bits of conventional version vectors are used. Particularly, unlike the plausible clocks yet proposed, no error occurs in characterizing those `log′`s that are generated at around the same time; i.e., *error-free regions* exist. Such error pattern will allow replicas to use `log′`s for resolving conflicts.

Above all, in dynamic replication, the simulation shows `log′`s can reduce the logging overhead considerably with no vector pruning since they waste no space for inactive entries. Thus, we expect that `log′`s can be employed in those dynamic systems, such as multiplayer games and collaborative applications [5], that replicate massive amounts of objects and, at the same time, tolerate incorrect conflict detections to some extent.

2. Prime version vectors

Unlike the conventional version vector [1], we distinguish our version vector into an *operating form* and a *logging form*. This paper assumes that every replica operates a prime version vector as an operating form, whose captured values are transformed into `log′` version vectors, i.e., the logging forms. The purpose of such decoupling is to achieve the concise and accurate `log′`s since the prime version vector can enhance the accuracy of `log′`s in detecting conflicts by supplementing two more entries to the conventional version vector.

For a replica R_i, its prime vector PV_i is represented as $[p_i, \mathtt{L}_i, \vec{v}_i]$, where (1) p_i is a prime number unique to R_i, (2) $\mathtt{L}_i$ is an integer value shared by all replicas, and (3) $\vec{v}_i$ is a vector that has one entry for $\forall p_k \in \mathbb{P}$ such that $\mathbb{P}$ is a set of prime numbers of all the replicas that have/had participated in the replication.[1] For the three operation types generally considered in replication systems, i.e., internal, sync, and rsync operations [1], the operating rules on $PV_i = [p_i, \mathtt{L}_i, \vec{v}_i]$ at R_i are as follows:

– Let `isSent` be an internal boolean variable.

Rule 0) Initial value:
1) $p_i :=$ a prime number unique to R_i,
$\mathtt{L}_i := 0$, $\forall p_k \in \mathbb{P}$: $\vec{v}_i[p_k] := 0$, `isSent` := false;

Rule 1) Before executing an internal operation:
1) Execute **DA**;
2) $\mathtt{L}_i := \mathtt{L}_i + 1$;

Rule 2) Before sending a sync operation:
1) Execute **DA**;
2) $\mathtt{L}_i := \mathtt{L}_i + 1$, `isSent` := true;

Rule 3) Before executing an rsync operation received from R_j with $PV_j = [p_j, \mathtt{L}_j, \vec{v}_j]$:
1) Execute **DA**;
2) $\mathtt{L}_i := \max(\mathtt{L}_i, \mathtt{L}_j) + 1$;

$$\forall p_k \in \mathbb{P}\text{: } \vec{v}_i[p_k] := \begin{cases} \max(\vec{v}_i[p_k], \vec{v}_j[p_k] + 1) & \text{for } p_k = p_j, \\ \max(\vec{v}_i[p_k], \vec{v}_j[p_k]) & \text{otherwise;} \end{cases}$$

(**DA**) Delayed Addition,
if (`isSent` = true) {
  $\vec{v}_i[p_i] := \vec{v}_i[p_i] + 1$;
  `isSent` := false;
}

[1] We assume the entry of a replica R_k is represented by $\vec{v}_i[p_k]$, where p_k is the prime number of R_k instead of an integer index.

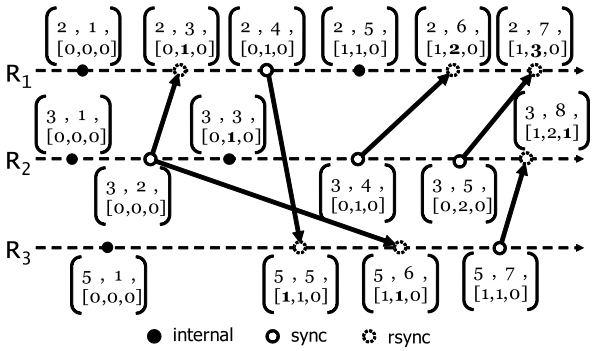

Fig. 2. An example of a replication system wherein three replicas participate using prime version vectors. The prime numbers of R_1, R_2, and R_3 are assumed to be 2, 3, and 5, respectively.

Like a Lamport clock [17], $\mathtt{L}_i$ is always renewed before executing every operation at any replica. Each entry, i.e., $\vec{v}_i[p_k]$, counts only sync operations invoked by R_k, but the addition for the local entry, i.e., $\vec{v}_i[p_i]$, is delayed until the execution of the next operation. According to the rules of the conventional vector [1], when R_i sends a sync operation to R_j, R_i and R_j increase $\vec{v}_i[p_i]$ and $\vec{v}_j[p_j]$, respectively. Due to the delayed addition however, a sync operation allows only a single entry of $\vec{v}_i$ to increase. The sync and its corresponding rsync operations are in dominant relation by Definition 1 because the delayed addition is also applied for a remote sync operation; in R3(2), the maximum between $\vec{v}_i[p_j]$ and $\vec{v}_j[p_j]+1$ (the added value) is taken for $\vec{v}_i[p_j]$. In the meantime, the maximums for other entries are taken as in the rule of the conventional vector [1]. Fig. 2 shows an example of a replication system using prime version vectors.

For two prime vectors $PV_i = [p_i, \mathtt{L}_i, \vec{v}_i]$ and $PV_j = [p_j, \mathtt{L}_j, \vec{v}_j]$, their relations can be defined as follows.

Definition 1 *(Dominant relation ($PV_i \prec PV_j$)).* PV_j dominates PV_i, i.e., $PV_i \prec PV_j$, iff: (1) for $p_i = p_j$, $L_i < L_j$, or (2) for $p_i \neq p_j$, $L_i < L_j$ and $\forall p_k \in \mathbb{P}$: $\vec{v}_i[p_k] \leqslant \vec{v}_j[p_k]$ and $\vec{v}_i[p_i] < \vec{v}_j[p_i]$.

Definition 2 *(Conflict relation ($PV_i \perp PV_j$)).* PV_i and PV_j are in conflict, i.e., $PV_i \perp PV_j$, iff: neither $PV_i \prec PV_j$ nor $PV_j \prec PV_i$.

Prime vectors are able to correctly detect all dominant and conflict relations like the conventional vectors. However, since prime vectors have two more entries than conventional vectors, they might incur insignificant extra overhead. Nevertheless, the prime vector makes $\mathtt{log}'$ version vectors more concise and accurate than the conventional vector because increments in $\vec{v}_i$ are suppressed. The next section explains the reason.

3. Log′ version vectors

Given a prime version vector $PV_i = [p_i, \mathtt{L}_i, \vec{v}_i]$, its $\mathtt{log}'$ version vector is as follows: $LV_i = [p_i, \mathtt{L}_i, \mathtt{LP}_i]$, where

$$\mathtt{LP}_i := \sum_{\forall p_k \in \mathbb{P}} \left(\log_\beta p_k \times \vec{v}_i[p_k]\right) \quad \text{for } \beta > 1.$$

Hence, $\mathtt{LP}_i$ should be a real number. For example, in Fig. 2, the last $\mathtt{log}'$ value of R_2 becomes $[3, 8, \log_\beta 2^1 3^2 5^1]$.

For two $\mathtt{log}'$s $LV_i = [p_i, \mathtt{L}_i, \mathtt{LP}_i]$ and $LV_j = [p_j, \mathtt{L}_j, \mathtt{LP}_j]$, Definitions 1 and 2 are redefined as follows.

Definition 3 *(Dominant relation ($LV_i \prec_l LV_j$)).* LV_j dominates LV_i, i.e., $LV_i \prec_l LV_j$, iff: (1) for $p_i = p_j$, $\mathtt{L}_i < \mathtt{L}_j$, or (2) for $p_i \neq p_j$, $\mathtt{L}_i < \mathtt{L}_j$ and a discriminant $\Delta(LV_i, LV_j) = \beta^\delta \in \mathbb{N}$ for $\delta = \mathtt{LP}_j - \mathtt{LP}_i - \log_\beta p_i$.

Definition 4 *(Conflict relation ($LV_i \perp_l LV_j$)).* LV_i and LV_j are in conflict, i.e., $LV_i \perp_l LV_j$, iff: neither $LV_i \prec_l LV_j$ nor $LV_j \prec_l LV_i$.

In Definition 3, the discriminant is

$$\Delta(LV_i, LV_j) = \beta^\delta = \frac{1}{p_i} \prod_{p_k \in \mathbb{P}}^{\forall p_k \in \mathbb{P}} p_k^{(\vec{v}_j[p_k] - \vec{v}_i[p_k])}.$$

Clearly, if condition (2) of Definition 1 is satisfied, $\Delta = \beta^\delta$ becomes a natural number. However, $\mathtt{LP}_i$ of an irrational number cannot be represented without errors in digital systems, and the discriminant might be misjudged as a result of these errors; note that the errors equally occur to all replicas because they are mathematically deterministic. Hence, the error needs to be exactly predicted and suppressed.

Let $\widetilde{\delta}$ be the value of a real number representation of a positive irrational number δ. We assume that the error between the two, i.e., $\varepsilon = \delta - \widetilde{\delta}$, is small enough to be close to zero. Then, the error of the discriminant is $|E| = |\beta^\delta - \beta^{\widetilde{\delta}}|$, which could be expanded as follows.

For $\dot{\delta} = \min(\delta, \widetilde{\delta})$,

$$\begin{aligned} |E| &= \left|\beta^\delta - \beta^{\widetilde{\delta}}\right| = \left|\beta^{\dot{\delta}+|\varepsilon|} - \beta^{\dot{\delta}}\right| \\ &= \beta^{\dot{\delta}}\left|\beta^{|\varepsilon|} - 1\right| = \beta^{\dot{\delta}}\left|e^{|\varepsilon \ln \beta|} - 1\right|. \end{aligned}$$

If $e^{|\varepsilon \ln \beta|}$ is substituted by Maclaurin series, then

$$|E| = \beta^{\dot{\delta}}\left(|\varepsilon \ln \beta| + \frac{|\varepsilon \ln \beta|^2}{2!} + \frac{|\varepsilon \ln \beta|^3}{3!} + \cdots\right).$$

From this infinite series, the following inequality for $|E|$ is obtained:

$$\begin{aligned} &\beta^{\dot{\delta}}|\varepsilon \ln \beta| < |E| < \beta^{\dot{\delta}}\left(|\varepsilon \ln \beta| + |\varepsilon \ln \beta|^2 + \cdots\right) \\ &\Leftrightarrow\ \beta^{\dot{\delta}}|\varepsilon \ln \beta| < |E| < \beta^{\dot{\delta}} \times \frac{|\varepsilon \ln \beta|}{1 - |\varepsilon \ln \beta|} \quad \left(\because |\varepsilon \ln \beta| < 1\right). \end{aligned}$$

The right term, $\frac{|\varepsilon \ln \beta|}{1-|\varepsilon \ln \beta|} = \frac{1}{|\varepsilon \ln \beta|^{-1}-1} \approx |\varepsilon \ln \beta|$, if $|\varepsilon \ln \beta|$ is small enough. Hence,

$$\begin{aligned} &\beta^{\dot{\delta}}|\varepsilon \ln \beta| < |E| < \beta^{\dot{\delta}}|\varepsilon \ln \beta| \\ &\Leftrightarrow\ |E| \approx \beta^{\dot{\delta}}|\varepsilon \ln \beta| \quad \text{for } \dot{\delta} = \min(\delta, \widetilde{\delta}). \end{aligned} \tag{1}$$

From formula (1), we can say that the following two conditions can reduce the error in the discriminant.

1. $|\varepsilon| = |\delta - \widetilde{\delta}|$ must be small.
2. Since $|E|$ is proportional to $\ln \beta$, smaller β is better.

Based on these conditions, we make the following three suggestions for implementing log′:

S1. Dividing $\mathtt{LP}_i$ into integer and decimal parts.
S2. Fixing the precision of the decimal part.
S3. Letting $\beta = 2$.

Suggestions S1 and S2 make it possible to predict the error of δ and ensure that every $\mathtt{LP}_i$ has the same error bound. We implement the decimal part of $\mathtt{LP}_i$ using **p**-bit raw array, i.e., with **p** precision. The most significant bit of the array denotes 2^{-1}, and the least pth one denotes 2^{-p}. According to [18], since the rounding error from the decimal part is less than or equal to 2^{-p-1}, the error of δ (say ε), i.e., the subtraction of three terms, becomes $|\varepsilon| \leqslant 3 \times 2^{-\mathbf{p}-1}$.

By letting $\beta = 2$ (S3), S1 and S2 also facilitate the calculation of the discriminant. In general, β^{δ} for $\delta \in \mathbb{R}_+$ is calculated by using the relationship $\beta^{\delta} = e^{\delta \ln \beta}$ [19]. If $\beta = 2$ and $\delta = \delta_i + \delta_f$, of which δ_i and δ_f are the integer and decimal part, respectively, then the discriminant $\Delta = 2^{\delta} = 2^{\delta_i + \delta_f} = 2^{\delta_i} \times e^{\delta_f \ln 2}$. After calculating $e^{\delta_f \ln 2}$, the multiplication of 2^{δ_i} can be emulated by shifting $e^{\delta_f \ln 2}$ to the left by δ_i bits. This ensures better performance in determining the discriminant.

The error in $e^{\delta_f \ln 2}$ is also predictable. Assume $\widetilde{\delta_f}$ and $\widetilde{\ln 2}$ are the decimal representations of δ_f and $\ln 2 = 0.6931471\ldots$, respectively. Letting ε_1 and ε_2 be the rounding errors of δ_f and $\ln 2$, respectively, then $|\varepsilon_1| \leqslant 3 \times 2^{-\mathbf{p}-1}$ and $|\varepsilon_2| \leqslant 2^{-\mathbf{p}-1}$ as stated above. Using these, $y = \delta_f \times \ln 2$ can be written as follows.

$$\begin{aligned} y &= \delta_f \times \ln 2 = (\widetilde{\delta_f} + \varepsilon_1)(\widetilde{\ln 2} + \varepsilon_2) \\ &= \widetilde{\delta_f}\widetilde{\ln 2} + \varepsilon_1\widetilde{\ln 2} + \varepsilon_2\widetilde{\delta_f} + \varepsilon_1\varepsilon_2. \end{aligned}$$

If $\widetilde{\delta_f}$ and $\widetilde{\ln 2}$ are used to calculate y, the error of $\widetilde{y} = \widetilde{\delta_f} \times \widetilde{\ln 2}$ would be $\varepsilon_3 = \varepsilon_2\widetilde{\delta_f} + \varepsilon_1\widetilde{\ln 2} + \varepsilon_1\varepsilon_2$. Due to $0 \leqslant \delta_f < 1$, $|\varepsilon_1\varepsilon_2| < |\varepsilon_2\widetilde{\delta_f}| < |\varepsilon_1\widetilde{\ln 2}|$. Hence, $|\varepsilon_3|$ is dominated by the term $\varepsilon_1\widetilde{\ln 2}$ and bounded as $|\varepsilon_3| \leqslant 3 \times 2^{-\mathbf{p}-1} \ln 2 < 2^{-\mathbf{p}+1}$.

From condition (1), the error bound of $e^{\delta_f \ln 2}$ is

$$\begin{aligned} |E| &\approx e^{\dot{\delta_f} \ln 2}|\varepsilon_3| \approx 2^{\dot{\delta_f}}|\varepsilon_3| \\ &< 2^{\dot{\delta_f}} 2^{-\mathbf{p}+1} < 2^{-\mathbf{p}+2} \quad (\because 0 \leqslant \dot{\delta_f} < 1). \end{aligned}$$

Hence, errors can appear in the two least significant bits.

$$1.b_1 b_2 \ldots b_{\mathbf{p}_1-3} \underbrace{b_{\mathbf{p}-2}}_{\text{the bit that should be rounded}} \overbrace{b_{\mathbf{p}-1} b_{\mathbf{p}}}^{\text{the bits that might have errors}}.$$

Only two bits are contaminated, while the fore bits from b_1 to $b_{\mathbf{p}-2}$ are trustworthy. Shifting the decimal point to the right by δ_i bits produces the value corresponding to the discriminant $\Delta = 2^{\delta}$. However, due to the error of the last two bits, the discriminant can be misjudged.

If $LV_i \prec_l LV_j$, $\Delta(LV_i, LV_j) \in \mathbb{N}$; that is, all the bits of the decimal part of Δ must be zeros. Due to the error, however, the trustworthy bits of Δ are all zeros if the error is positive, while they are all ones if the error is negative. To make up the error, we round $e^{\delta_f \ln 2}$ to the nearest on the last trustworthy bit $b_{\mathbf{p}-2}$ so that all the bits of the decimal part can be zeros [18]. This ensures that the discriminant of the dominant relation is always determined correctly.

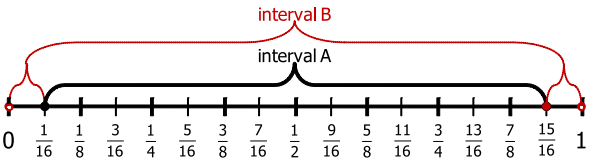

Fig. 3. The intervals of the decimal part of Δ in which the discriminant determination is correct or not, with four trustworthy bits.

If $LV_i \perp_l LV_j$, $\Delta(LV_i, LV_j) \notin \mathbb{N}$; this means that the bits after the decimal point should be neither all zeros nor all ones. However, if the value of the decimal part of Δ is too close to either zero or one, the trustworthy bits might be all zeros or all ones, respectively. This causes a conflict relation to be identified incorrectly as a dominant relation.

As δ increases, the trustworthy bits of the discriminant decrease, thereby causing the accuracy of the determination to decline. If $\delta \geqslant \mathbf{p} - 1$, the discriminant determination is unavailable because no trustworthy bits remain. For example, assuming that $\mathbf{p} = 64$ and $58 < \delta < 59$, only four bits are trustworthy: $64 - 2$ (the bits where an error appears) $- 58$ (δ_i, i.e., the bits that must be shifted to the left) $= 4$. In this case, only when the decimal part of the discriminant for a conflict relation has a value in interval $A = [\frac{1}{16}, \frac{15}{16})$, as shown in Fig. 3, can the discriminant be correctly determined. If it is in interval $B = (0, \frac{1}{16}) \cup [\frac{15}{16}, 1)$, the four trustworthy bits would be either '0.0000' or '0.1111' in binary, resulting in an incorrect determination that the discriminant of the conflict relation is misconceived as a natural number after being rounded on the last trustworthy bit. Nevertheless, we regard this relation as a dominant one because dominant relations frequently happen when two log′s are a long way off.

To this end, log′s have the following property.

$$\begin{aligned} V_i \prec V_j \quad &\Rightarrow \quad LV_i \prec_l LV_j, \\ V_i \perp V_j \quad &\Leftarrow \quad LV_i \perp_l LV_j. \end{aligned}$$

This property is the same as that in the plausible clocks introduced by Torres-Rojas and Ahamad [16], which likewise sacrifice accuracy in detecting causality so as to achieve constant size vectors. Although log′ is superior to typical plausible clocks [16] in terms of both accuracy and logging overhead, this paper does not compare log′ with them because log′s are obtained by the medium of prime version vectors and inherently not designed to be vector clocks but a concise logging form of version vectors. In-

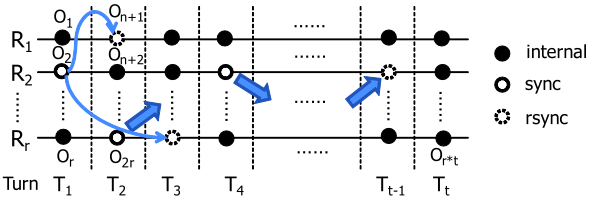

Fig. 4. The design of log′ version vector simulations.

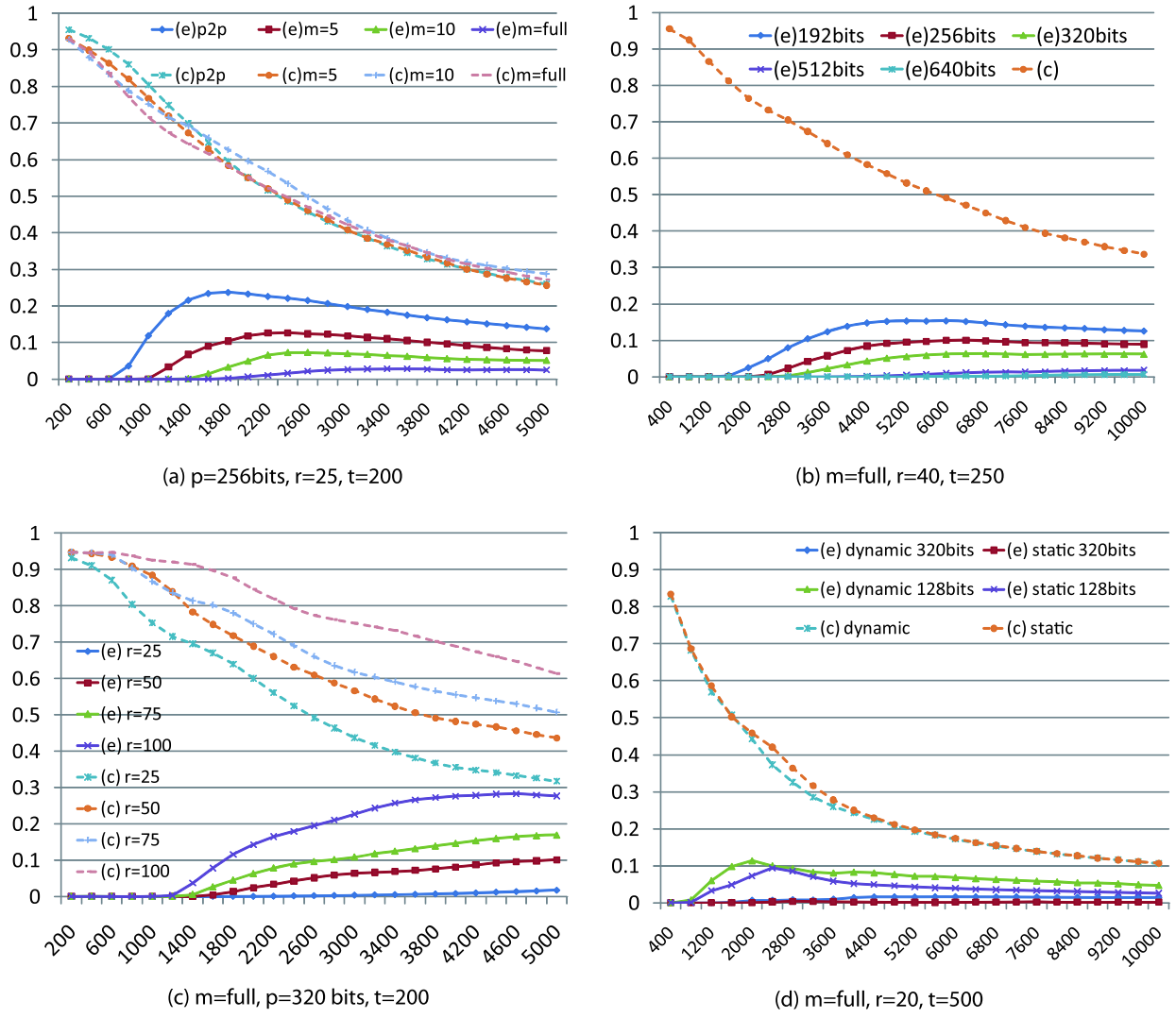

Fig. 5. Accuracy graphs: m = the number of masters, p = precision of decimal part of $\mathtt{LP}_i$, r = number of replicas, t = number of turns.

stead, with respect to various system models, this paper focuses on showing the characteristics of $\mathtt{log}'$ errors in the following section.

4. Simulations

We implement $\mathtt{log}'$ using the MPFR library, which supports a multi-precision floating-point calculation [20]. The synchronization of replicas is simulated to examine the detection accuracy of $\mathtt{log}'$s. The simulations are designed as shown in Fig. 4, in which four parameters are defined as below:

1. $R_1, \ldots, R_r$: r replicas,
2. $T_1, \ldots, T_t$: t turns, at each of which a replica issues either an internal or a sync operation, or receives an rsync operation,
3. $LV_1, \ldots, LV_{r*t}$: $\mathtt{log}'$s of operations $O_1, \ldots, O_{r*t}$ listed in the order as shown in Fig. 4,
4. A sync operation is sent to another random replica within an arbitrary delay (< 10 turns). If a replica is a master, its sync operation is broadcast to all the other replicas.

To show the detection accuracy of $\mathtt{log}'$, we present accuracy graphs wherein every relation from a pair of distinct $\mathtt{log}'$s is compared with its counterpart of conventional version vectors. In Fig. 5, the x-axis represents a set of x $\mathtt{log}'$s, and the y-axis shows the ratios of conflict relations over all possible $\frac{x(x-1)}{2}$ pairs, depicted with the dashed line graphs of legend (c). As evident in the graphs of Fig. 5, the ratio of conflict relations declines as the set size grows. The solid-line graphs of legend (e) represent the error ratios at which relations are incorrectly detected, compared to the detection using conventional version vectors. Note that with $\mathtt{log}'$s, dominant relations are always detected correctly, but conflict relations might not be. Hence, the vertical difference between the two

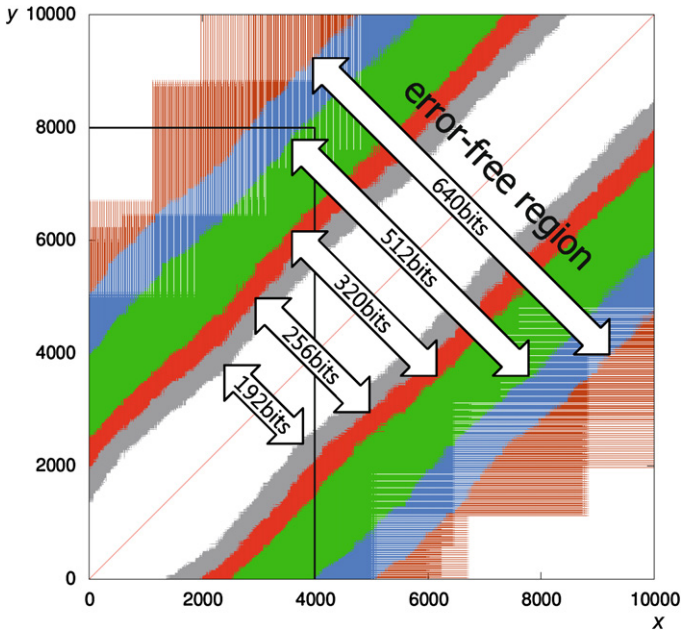

Fig. 6. The error regions of the simulation of Fig. 5(b). If the relation between xth and yth log′s is detected incorrectly, it is colored. The error region of smaller precision includes larger precision's.

classes of graphs reveals the ratio of the conflict relations being detected correctly. Overall, there are sets of log′s, named *error-free* log′ *sets*, of a certain extent (600–4800 log′s) that contain no error despite very high ratios of conflict relations.

Fig. 5(a) shows the accuracy graph with regard to synchronization policies. In many real-world replication systems, replicas synchronize themselves mutually in order to make all replicas consistent. For example, Coda allows several master replicas to broadcast every sync operation [2], and in most collaborative editing systems, every replica broadcasts sync operations for responsiveness [5]. To this end, four synchronization policies are devised with 25 replicas: ($p2p$) every replica synchronizes itself with only one of other replicas, ($m = 5$ and 10) 5 and 10 masters of total 25 replicas broadcast their sync operations, and ($m =$ full) all the replicas broadcast their sync operations. Synchronization makes log′s of different replicas have similar values, for which conflict relations tend to be detected correctly as stated in Section 3; thus, as the number of masters increases, accuracy improves. The rest of our simulations are done on $m =$ full.

Obviously, the precision of the decimal parts affects accuracy, as shown in Fig. 5(b). The accuracy is rapidly enhanced as more bits are used for the decimal parts. For example, for 10000 log′s, the errors are about 9% for $\mathbf{p} = 256$ bits, but less than 2% for $\mathbf{p} = 512$ bits. This result is evident because those log′s that are generated at around the same time have a tendency to be in conflict relation, and *the error-free region*, where all detections are correct, is widened as more bits are used for decimal parts. Fig. 6 shows the region where errors appear for all the pairs of 10000 log′s; such errors are highlighted with different colors. For instance, if some 4000th log′ is in conflict relation with some 0th or 8000th log′s, the relations are correctly detected if 512 bits are used, but might not be for 320 bits.

Fig. 5(c) evaluates the effects of the replica size (from 25 to 100 replicas) on accuracy. To fix the number of log′s, we consider only 5000 log′s of the first 25 replicas under the condition that all the replicas have actively issued log′s. According to the definition of the conflict relation, before two operations are generated at different replicas, if only one of the replicas had received some other operations from other replicas, the version vectors of the two operations must be in conflict relation. For that reason, as the replica size becomes larger, the conflict ratio rises, as shown in Fig. 5(c). Meanwhile, for $r = 25$, the error-free set has about 1800 log′s, but the set sizes are kept up around 1000–1200 log′s for $r = 50$–100. However, as the replica size becomes larger, the error ratio rises, since more operations out of the error-free set are in conflict relation.

Finally, we simulate the effects of membership changes. Amongst $r = 20$ replicas, the replica of the smallest prime number retires every 6 turns, which means the retired replica neither issues nor receives operations any more. The retired replica is replaced with a new one that is assigned a new prime number and a new prime version vector. During 500 turns, a total of 80 replicas newly participate in the replication system, but only 20 replicas are at the same time in the active status that allows a replica to issue and receive operations. If conventional version vectors are used, they will have a hundred entries in the end.

Fig. 5(d) is the accuracy graph, in which 'static' describes the replication with no membership change while 'dynamic' is the result of the replication designed above. For log′s of 128-bit and 320-bit precisions, their conflict ratios are similar, but the error ratios of dynamic replications are slightly higher than those of static repli-

cations because prime numbers being newly assigned increases. Nevertheless, the increase in the error ratios is not grave. On the assumption that entry sizes of log′ $LV_i = [p_i, \mathrm{L}_i, \mathrm{LP}_i]$ are 2 bytes, 6 bytes, and 8 bytes + 320 bits (the integer part + the decimal part), respectively, and that each entry of conventional vectors is a 4 byte-integer, the error ratio of log′s is less than 2% with only about 23% of data overhead of conventional vectors. Obviously, this logging overhead of log′s will be reduced further as the replication ages. Hence, log′s are concise and accurate in dynamic replication.

5. Conclusions

We propose the log′ version vector and its implementation. Log′ is obtained from its operating form, prime version vector designed to enhance the detection accuracy of log′. The primary virtue of log′ is that it requires no vector pruning. Hence, log′ is suitable to dynamic replication of limited membership. The simulation studies show that log′s are logged concisely with few errors in the full synchronization replication.

Errors in detecting conflicts with log′s equally appear to all replicas because they are mathematically deterministic. Therefore, despite these errors, log′s can offer a reasonable compromise for maintaining consistency among replicas. Indeed, an operation's being dominant over some others can be interpreted that it *has directly or indirectly experienced* them. Respecting experience, in general, the effect of dominant operation is preferentially applied to replicas while reconciled effects are taken for conflicting operations. Even if two log′s of conflict relation are misjudged as a dominant relation, the dominant log′ must have gained much more experience than the other. In this regard, incorrect detection with log′s is acceptable enough for some replication systems to adopt log′s.

References

[1] D.S. Parker, G.J. Popek, G. Rudisin, A. Stoughton, B.J. Walker, E. Walton, J.M. Chow, D. Edwards, S. Kiser, C. Kline, Detection of mutual inconsistency in distributed systems, IEEE Transactions on Software Engineering 9 (3) (1983) 240–247.

[2] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, D.C. Steere, Coda: A highly available file system for a distributed workstation environment, IEEE Transactions on Computers 39 (4) (1990) 447–459.

[3] D. Malkhi, D.B. Terry, Concise version vectors in WinFS, Distributed Computing 20 (3) (2007) 209–219.

[4] Y.-W. Huang, P.S. Yu, Lightweight version vectors for pervasive computing devices, in: Proceedings of International Workshop on Parallel Processing (ICPP), IEEE Computer Society, Washington, DC, USA, 2000, p. 43.

[5] C. Sun, X. Jia, Y. Zhang, Y. Yang, D. Chen, Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems, ACM Transactions on Computer–Human Interaction 5 (1) (1998) 63–108.

[6] M. Singhal, A. Kshemkalyani, An efficient implementation of vector clocks, Information Processing Letters 43 (1) (1992) 47–52.

[7] J. Fowler, W. Zwaenepoel, Causal distributed breakpoints, in: Proceedings of International Conference on Distributed Computing Systems (ICDCS), 1990, pp. 134–141.

[8] C. Fidge, Logical time in distributed computing systems, Computer 24 (8) (1991) 28–33.

[9] F. Mattern, Virtual time and global states of distributed systems, in: Proceedings of the International Workshop on Parallel and Distributed Algorithms, Elsevier, 1989, pp. 215–226.

[10] Y. Saito, Unilateral version vector pruning using loosely synchronized clocks, Tech. Rep. HPL-2002-51, HP Labs, Storage Systems Department, 2002.

[11] R.A. Golding, Weak-consistency group communication and membership, Ph.D. thesis, University of California, Santa Cruz, citeseer.ist.psu.edu/golding92weakconsistency.html, 1992.

[12] D. Ratner, P. Reiher, G. Popek, Roam: A scalable replication system for mobile computing, in: International Workshop on Database and Expert Systems Applications, 1999, p. 96.

[13] A. Schiper, K. Birman, P. Stephenson, Lightweight causal and atomic group multicast, ACM Transactions on Computer System 9 (3) (1991) 272–314.

[14] G.G. Richard, Efficient vector time with dynamic process creation and termination, Journal of Parallel and Distributed Computing 55 (1) (1998) 109–120.

[15] X. Wang, J. Mayo, W. Gao, J. Slusser, An efficient implementation of vector clocks in dynamic systems, in: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2006, pp. 593–599.

[16] F.J. Torres-Rojas, M. Ahamad, Plausible clocks: constant size logical clocks for distributed systems, Distributed Computing 12 (4) (1999) 179–195.

[17] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.

[18] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Computing Survey 23 (1) (1991) 5–48.

[19] J.-M. Muller, Elementary Functions: Algorithms and Implementation, Birkhäuser Boston, Inc., Secaucus, NJ, USA, 1997.

[20] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann, MPFR: A multiple-precision binary floating-point library with correct rounding, ACM Transactions on Mathematical Software 33 (2) (2007) 13.