Analysis of Bounds on Hybrid Vector Clocks*

Sorrachai Yingchareonthawornchai[1], Sandeep Kulkarni[2], and Murat Demirbas[3]

1 **Michigan State University, MI, USA**
`yingchar@cse.msu.edu`
2 **Michigan State University, MI, USA**
`sandeep@cse.msu.edu`
3 **University at Buffalo, SUNY, NY, USA**
`demirbas@buffalo.edu`

Abstract

Hybrid vector clocks (HVC) implement vector clocks (VC) in a space-efficient manner by exploiting the availability of loosely-synchronized physical clocks at each node. In this paper, we develop a model for determining the bounds on the size of HVC. Our model uses four parameters, ϵ: uncertainty window, δ: minimum message delay, α: communication frequency and n: number of nodes in the system. We derive the size of HVC in terms of a differential equation, and show that the size predicted by our model is almost identical to the results obtained by simulation. We also identify closed form solutions that provide tight lower and upper bounds for useful special cases.

Our model and simulations show the HVC size is a sigmoid function with respect to increasing ϵ; it has a slow start but it grows exponentially after a phase transition. We present equations to identify the phase transition point and show that for many practical applications and deployment environments, the size of HVC remains only as a couple entries and substantially less than n. We also find that, in a model with random unicast message transmissions, increasing n actually helps for reducing HVC size.

1998 ACM Subject Classification C.2.4 Distributed Systems, distributed databases, D.1.3 Concurrent Programming, D.4.2. Distributed memories, D.4.3 Distributed file systems

Keywords and phrases Vector Clocks, Physical Clocks, Large Scale Systems

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2015.34

1 Introduction

Work on theory of distributed systems abstract away from the wall-clock/physical-clock time and use the notion of logical clocks for ordering events in *asynchronous* distributed systems [12, 10, 13]. The causality relationship captured by these logical clocks, called happened-before (**hb**), is defined based on passing of information, rather than passing of time.[1] Lamport's logical clocks [12] (LC) prescribe a total order on the events: $A \underline{hb} B \implies lc.A < lc.B$ but vice a versa is not necessarily true. Vector clocks [10, 13] (VC) prescribe a partial order on the events: $A \underline{hb} B \iff vc.A < vc.B$ and $A \underline{co} B \iff (\neg(vc.A < vc.B) \wedge \neg(vc.B < vc.A)$. Using LC or VC, it is not possible to query events in relation to physical time. Moreover, for capturing **hb**, LC and VC assume that all communication occur

* This work is supported by NSF CNS 1329807, NSF CNS 1318678, NSF XPS 1533870, and XPS 1533802.
[1] Event A **hb** event B, if A and B are on the same node and A comes earlier than B, or A is a send event and B is the corresponding receive event, or this is defined transitively based on the previous two.


in the present system and there are no backchannels. This assumption is obsolete for today's integrated, loosely-coupled system of systems. Finally, the space requirement of VC is shown to be $\Theta(n)$ [3], the number of nodes in the system, and is prohibitive.

Practice of distributed systems, on the other hand, employ loosely synchronized clocks, mostly using NTP [15]. Unfortunately, there are fundamental limits to clock synchronization and perfect synchronization is unachievable due to the nature of distributed systems: messaging with uncertain latency, clock skew among processors, and NTP glitches [15]. Even using atomic clocks, as in Google TrueTime [5], it is hard to reduce ϵ, the uncertainty of the clock synchronization, to less than a couple milliseconds. This requires that operations/transactions wait out these ϵ uncertainties, which takes its toll on the performance.

Recently, we introduced a third option, hybrid clocks [6, 11]. Hybrid clocks combine the best of logical and physical clocks; hybrid clocks are immune to their disadvantages while providing their benefits. Hybrid clocks are loosely synchronized using NTP, yet they also provide provable comparison conditions as in LC or VC within ϵ uncertainty. Hybrid clocks also address the backchannel communication issue by introducing the notion of $\epsilon - \underline{hb}$ that captures the intuition that if event B happened far later than event A, then event A can affect event B due to out-of-bound communication. If events A and B are close, then the causality relation is taken into account to identify whether A can affect B.

Our hybrid clocks come in two flavors: hybrid logical clocks (HLC) [11] and hybrid vector clocks (HVC) [6]. HLC satisfy the logical clock comparison condition as in LC [12]. HLC finds applications in multiversion distributed database systems [4] and enable efficient querying of consistent snapshots for read transactions, while ensuring commits of write transactions do not get delayed despite the uncertainties in NTP clock synchronization [11]. HVC satisfy the vector clock comparison condition as in VC [10, 13], and can serve in applications that HLC become inadequate. In contrast to HLC that can provide a single consistent snapshot for a given time, HVC is able to provide all possible/potential consistent snapshots for that given time. As such, HVC finds applications in debugging for concurrency race conditions of safety critical distributed systems and in causal delivery of messages to distributed system nodes.

HVC reduces the overhead of causality tracking in VC by utilizing the fact that the clocks are reasonably synchronized. When ϵ is infinity, HVC behaves more like VC used for causality tracking in asynchronous distributed systems. When ϵ is very small, HVC behaves more like a scalar physical synchronized clock, but also combines the benefits of causality tracking in uncertainty intervals. Although the worst case size for HVC is $\Theta(n)$, we observe that if j does not hear (directly or transitively) from k within ϵ time then $hvc.j[k]$ need not be explicitly maintained. In that case, we still infer implicitly that $hvc.j[k]$ equals $hvc.j[j] - \epsilon$, because $hvc.j[k]$ can never be less than $hvc.j[j] - \epsilon$ thanks to the clock synchronization assumption. Therefore, in practice the size of $hvc.j$ would only depend on the number of nodes that communicated with j within the last ϵ time and provided a fresh timestamp that is higher than $hvc.j[j] - \epsilon$. In other words, by using temporal slicing, HVC can circumvent the Charron-Bost result [3] and can potentially scale the VC benefits to many thousands of processes by still maintaining small HVC at each process.

Contributions of this paper. But how effective are HVC for reducing the size of VC? What bounds should we expect on the number of entries in HVC for a given ϵ? Determining these bounds on HVC would help developers to budget the size of the messages the nodes send, the size of the memory to maintain at the nodes, and the scalability and performance of their system. In this paper, we derive and identify these bounds.

To this end, we develop an analytical model that uses four parameters, ϵ: uncertainty window, δ: minimum message delay, α: message rate, and n: number of nodes in the system.

We derive the size of HVC in terms of a differential equation, and show that the size predicted is almost identical to the results obtained by simulation experiments. We also identify closed form solutions that provide tight lower and upper bounds for useful special cases.

Our model and simulations show the HVC size is a sigmoid function with respect to increasing ϵ; it has a slow start but it grows exponentially after a critical phase transition. Before the phase transition threshold, HVC maintains couple entries per node, however when a threshold is crossed, a node not only gets entries added to its clock from direct interaction but also indirect transfer from another processes HVC, and this makes the HVC entries blow up. We present equations to identify this transition point. Specifically, for the common case of $\alpha * \delta < 1$, we derive this threshold as $(\frac{1}{\alpha} + \delta)(\ln((2 - \sqrt{3})(n - 1)))$.

Using this equation, we describe how to avoid/delay the threshold point. If an application developer reduces α, the phase transition is delayed, and small HVC sizes are still achievable for a given ϵ and δ. Moreover, while in VC the size increases directly with n, we find that in HVC, surprisingly, the increase of n, in fact, benefits in reducing the size of HVC. Using a model with random unicast message transmissions, for larger n, the probability of indirect HVC entry addition/transfer reduces slightly, and hence larger n, in fact delays the phase transition to large HVC sizes.

We show in our discussion section that for most practical applications and deployment environments, the size of HVC remains only as a couple entries and substantially less than n. Yet, when it is needed HVC expands on demand to allow more entries to capture causality both ways in the ϵ uncertainty slices.

Outline of the rest of the paper. After presenting the preliminaries in Section 2, we present our analytical solutions in Section 3, and solutions for useful special cases in Section 4. We present evaluation results in simulation to show how well the analytical models capture the HVC bounds in Section 6. We discuss practical implications of our findings in Section 7, related work in Section 8, and conclude in Section 9.

2 System Model

We use n to denote the number of processes in the system. Although processes can be added dynamically, we assume that each of them has a distinct identifier. Each process j is associated with a physical clock $pt.j$. We assume that clock synchronization algorithm such as NTP [15] is used to provide a reasonable but imperfect clock synchronization to the processes. For ease of presentation, we assume the existence of an *absolute time*: this time is not accessible to processes themselves, and it is used only for the presentation and proofs associated with our algorithm. Specifically, we assume that at any given time the difference between any two clocks at processes, $pt.j$ and $pt.k$, is bounded by $\epsilon, \epsilon \geq 0$.

Processes communicate via messages. We make no assumptions such as FIFO ordering or bounded delivery time. In other words, messages could be delivered out of order. They could also be delivered a long time after they are sent. We assume that there is a *minimum message delay* δ_{min} (as computed by the absolute global time) before message is delivered.

In our analytical model to compute the size of HVC at any process, we assume that at each absolute time tick, each process sends a message to some other process (selected randomly) with probability α. We permit messages to be delivered as early as possible, we allow a process to receive multiple messages simultaneously.

Let $s_j(t)$ be a random variable representing size of active HVC of process j at time t. Thus, our goal is to identify an expected average active HVC size $\Psi(t) = E[\sum_j s_j(t)/n]$, and

$\psi(t) = \Psi(t)/n$. We aim to find an analytical solution to $\psi(t)$ given four parameters ϵ, δ, α, and n.

2.1 Unconstrained and Constrained Time Models

To develop this analytical solution, we develop two models: 1) an unconstrained model where we compute the size of HVC by assuming that $\epsilon = \infty$, and 2) a constrained model that considers the value of ϵ. Without loss of generality, we focus on one sender process, say j. Our goal is to identify the number of processes that maintain he clock of this process at a given time t. In turn, this enables us to find the expected size of each HVC entry. To make this analysis simpler to understand, we introduce the notion of a color –red or green– for each process. The color of process k is red at time t iff k is maintaining the clock of process j at time t. In other words, *color.k* is red iff the knowledge that k has about the clock of j is more than that provided by clock synchronization. Clearly, in the initial state $t = 0$, j is red and all other processes are green.

Model 1: unconstrained time model. Given the notion of color maintained by each process, we can observe that if a red process sends a message to a green process, then the green process learns information about the clock of j. In other words, it makes the recipient red. Messages sent by green process can be ignored since they do not provide non-trivial information about the clock of j.

In this model, let $Y(t)$ denote number of red processes at time t. Note that $n - Y(t)$ is number of green processes at time t. Also, let $y(t) = Y(t)/n$ be the fraction of red process at time t. We aim to analytically compute $y(t)$ given δ, α, n for $\epsilon \to \infty$.

Model 1 captures the case where $\epsilon = \infty$. The reason we consider this model is due to an important result (shown in Theorem 13) that demonstrates that the value of $Y(\epsilon)$ can be used to compute the number of red processes in the ϵ-constrained model (discussed next) that utilizes the actual value of ϵ in the given system.

Model 2: ϵ-constrained time model. To capture the effect of the hybrid model where ϵ has a finite value (and hence, a red process will turn green if it does not hear recent clock information of process j), we define τ-message as a message that is originated by the initial red process j at time τ. τ-message triggers green process to be red if $\tau + \epsilon \leq t$. Otherwise, even if the green process receives information about the clock of j, this information is still beyond the uncertainty interval. Let $Y_\epsilon(t)$ be number of red processes of Model 2 at time t. We aim to compute an analytical solution to $y_\epsilon(t) = Y_\epsilon(t)/n$ for given ϵ, δ, α, and n.

3 Analytical Solutions

Given that ϵ-constrained time model can be answered by unconstrained time model as shown in Theorem 13, this means analytical solution to unconstrained time model implies the solution to our system model.

Based on the definition of *color.k*, in the initial state, *color.j* is red and *color.k* is green for any $k \neq j$. It follows that at time $t = [0..\delta]$, j is the only red process as message sent by j has not been received by anyone. When a green process receives a message it turns red and stays red forever. Let $Y(t)$ be number of red processes at time t. Note that number of green processes at time t is then $n - Y(t)$. Since message delay for every message is δ, $Y(t)$ depends upon $Y(t - \delta)$, i.e., the number of processes that were red δ time before.

Our first result in this context, given in Lemma 1, captures the number of messages delivered at time t to green processes.

▶ **Lemma 1.** *The expected number of messages delivered to green processes at time t is $\alpha Y(t-\delta)(1-Y(t)/n)$*

Proof. The expected number of red messages delivered at time t is $\alpha Y(t-\delta)$ since each red process in $Y(t-\delta)$ has α probability to send a unicast message. At time t, the probability of a message getting delivered to green is the fraction of green process at time t, $1-Y(t)/n$ assuming that each process has equally likely change to receive such message. The result follows immediately by linearity of expectation. ◀

Although Lemma 1, counts the number of red messages sent to green processes, it overcounts the processes that can become red, as one green process may receive multiple messages. To analyze the number of processes that turn red, we observe that this problem can be viewed as throwing a number of balls (i.e., messages sent by red processes) into a set of bins (i.e., the green processes) to identify the expected number of non-empty bins (i.e., processes that receive at least one ball and therefore turn red). In this context, we use Lemma 2.

▶ **Lemma 2.** *Consider occupancy problem where there are A balls and B bins. All balls are thrown to random bins. Expected number of non-empty bins is $B(1-(1-1/B)^A)$.*

Proof. Fix one bin. Probability of the bin being empty is $(1-1/B)^A$ since all balls must miss this bin. By linearity of expectation, expected number of empty bins is $B(1-1/B)^A$. Hence, expected number of non-empty bins is B minus number of empty bins. ◀

Now, we can compute the change of red processes at time t by applying Lemma 2 using $A=\alpha Y(t-\delta)(1-Y(t)/n)$ (from Lemma 1) and $B=n-Y(t)$ since B is number of green process at time t. Hence, $\frac{dY(t)}{dt}$ is $(n-Y(t))(1-(1-\frac{1}{(n-Y(t))})^{\alpha Y(t-\delta)(1-Y(t)/n)})$

We can simplify the expression by using the fact that $\lim_{n\to\infty}(1+x/n)^n=e^x$. We adjust some terms in Equation above and let $x=\frac{-1}{(1-Y(t)/n)}$, we obtain

$(n-Y(t))(1-(1-\frac{1}{n(1-Y(t)/n)})^{\alpha\frac{n}{n}Y(t-\delta)(1-Y(t)/n)})=(n-Y(t))(1-e^{\frac{-\alpha Y(t-\delta)}{n}})$

Since $y(t)=Y(t)/n$, we get $\frac{dy(t)}{dt}=(1-y(t))(1-e^{-\alpha y(t-\delta)})$

Finally, based on the initial values, we have $y(t)=1/n$ for $t<\delta$. And, since we can consider each process j independently, which means the expectation does not change. Thus, we have the following Theorem.

▶ **Theorem 3.** *The expected average size of hvc per process of $\psi(t)$ satisfies the following delay differential equation.*

$$\frac{d\psi}{dt}=(1-\psi(t))(1-e^{-\alpha\psi(t-\delta)})$$

where initial condition is $\psi(t)=1/n$ for $t<\delta$.

From this point on, we use $\psi(t)$ (random variable of fraction of average size of *hvc*) and $y(t)$ (random variable of fraction of red processes) interchangably since they have same expactation value.

4 Explicit Solutions for Special Cases

Theorem 3 provides a mechanism to compute the size of *hvc*. Since the differential equation in Theorem 3 cannot be solved explicitly, one must utilize numerical tools, such as MATLAB and Mathematica, to obtain the size of *hvc* from that equation.

However, closed form solutions —that can be computed with a basic calculator— may be more desirable since they can offer a quick insight into the size of *hvc*.

In this section, we provide closed form solutions for some special cases. Specifically, when α is arbitrarily small, we obtain an explicit solution to Theorem 3 given that $\alpha * \delta$ is small. If $\alpha * \delta$ is not necessarily small, we derive an yet explicit solution up to $\epsilon \leq 3\delta$ for arbitrary δ. Using simplification technique, we can obtain upper and lower bounds solution to Theorem 3 if α is not necessarily small. Based on our evaluation, the value of $\alpha * \delta < 1$ is sufficient to obtain accurate closed form solutions. Otherwise, $\epsilon \leq 3\delta$ can capture almost all value of y.

These bounds are fairly tight as shown in the simulation results in Section 6. The problem for computing closed form solution where $\delta > 0$ and $\epsilon > 3\delta$ is currently open.

4.1 Explicit Solution for Arbitrarily Small α and $\alpha * \delta$

We put two main simplifications to obtain explicit solutions. First, we assume that α is small (typically, $\alpha < 0.1$) so that we have good approximation of $1 - e^{-\alpha y(t-\delta)}$ using Taylor's series expansion. The expansion is $\alpha y - \alpha^2 y^2/2 + O(\alpha^3 y^3)$. If α is small, this expansion is approximately αy. Hence, the differential equation in Theorem 3 becomes $\frac{dy}{dt} = \alpha(1 - y(t))y(t - \delta)$. Second simplification is suppose $\alpha * \delta$ that is arbitrarily small. We have the following Lemma.

▶ **Lemma 4.** *if $\alpha\delta > 0$ is arbitrarily small, then $y(t) = (1 + \alpha\delta)y(t - \delta)$*

Proof. We can approximate the change of $y(t)$ over δ period of time in the past. That is, the change $\frac{y(t)-y(t-\delta)}{\delta}$ is approximately $\frac{dy}{dt}$. Based on expression above, the change is roughly $\alpha(1 - y(t))y(t - \delta)$. Therefore, $y(t) = y(t - \delta) + \alpha\delta(1 - y(t))y(t - \delta)$. The result follows from that the product $\alpha\delta$ is approaching zero. ◀

Using Lemma above, we reduce delay differential equation to ordinary differential equation as in the following. The differential equation is elementary to be solved by standard ordinary differential equation procedure.

▶ **Theorem 5.** *For the case where $\alpha * \delta > 0$ is arbitrarily small, the change of y over time is $\frac{dy}{dt} = \frac{\alpha}{1+\alpha\delta}(1 - y)y$ with initial condition $y(0) = 1/n$. Further, the explicit solution to the differential equation is*

$$y(t) = \frac{1}{1 + (n - 1)e^{-\alpha t/(1+\alpha\delta)}}.$$

4.2 Phase Transition

The result for Theorem 5 implies that the graph of y is essentially a logistic function (or Sigmoid function). One important characteristic of this function is it has slow start in the initial state and then the function grows exponentially after a phase transition. In this section, we discover such transition in terms of δ, α and n. We define phase transition point ϵ_p as the earliest point where the change of slope is maximum. In particular, we show the following result.

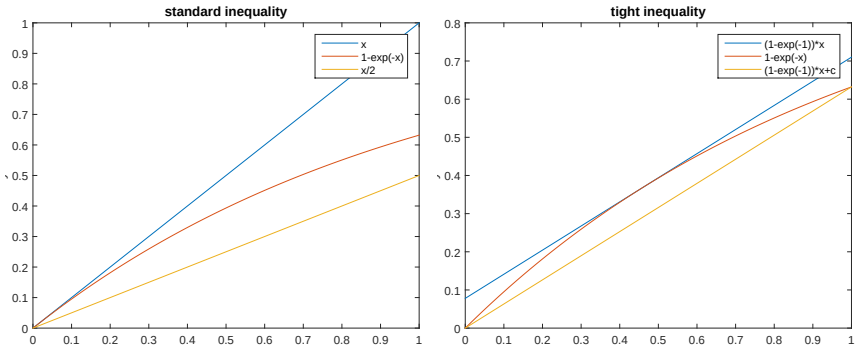

Figure 1 Standard inequality and tight inequality.

▶ **Theorem 6.** *The phase transition ϵ_p for Theorem 5 is* $(\frac{1}{\alpha}+\delta)(\ln((2-\sqrt{3})(n-1)))$.

Proof. The slope of $y(t)$ is $y'(t)$. The change of slope is $y''(t)$. The maximum of change of slope is when $y^{(3)}(t)=0$. We get the result by finding third order derivative of y. Then, we set y''' to 0. Suppose the function is in the following form: $\frac{dy}{dt}=a(1-y)y$. We apply derivative twice from $\frac{dy}{dt}$ to obtain the third order derivative of y. By simple differentiation, we have

$$y'''(t)=\frac{a^3(n-1)e^{at}(-4(n-1)e^{at}+e^{2at}+(n-1)^2)}{(e^{at}+n-1)^4}\,.$$

When $y'''(t)=0$, we obtain quadratic equation in the form of e^{at}:

$$e^{2at}-4(n-1)e^{at}+(n-1)^2=0\,.$$

Solving quadratic quation, we obtain $e^{at}=(n-1)(2\pm\sqrt{3})$. We select the earlier time by definition of phase transition. Then, $t=(1/a)(\ln((n-1)(2-\sqrt{3}))$. The result follows when we substitute $a=\frac{\alpha}{1+\alpha\delta}$. ◀

4.3 Explicit Solution for $t < 3\delta$

If α is not necessarily small, we can obtain bounds in terms of upper and lower bounds. The technique is to simplify the function so that differential equation is easily solvable. Since the equation in Theorem 3 involves e^x, in Lemma 7, we first identify a tight bound on the value of e^x when x is in the range $[0..1]$.[2]

▶ **Lemma 7.** *For $x,\alpha\in[0,1]$, this inequality holds*

$$(1-e^{-\alpha})x\le 1-e^{-\alpha x}\le(1-e^{-\alpha})x+\xi$$

where

$$\xi=1-(\frac{1-e^{-\alpha}}{\alpha})(1+\ln(\frac{\alpha}{1-e^{-\alpha}}))\,.$$

[2] **Remark.** The standard inequality identity regarding e^x is that $1-e^{-x}\le x$ for any real number x, and $x/2\le 1-e^{-x}$ for some small range x. We considered using these upper and lower bounds in subsequent results. However, these bounds are not tight when $x\in[0..1]$ as shown in Figure 1, which is the case in Theorem 3. This is the reason we use Lemma 7 in subsequent computation.

Proof. We only need to find slope and y-intercept for two lines. The lower bound is easily attainable by considering two points $(0, 0)$ and $(1, 1 - e^{-\alpha})$. For the upper line, we know that the slope must be equal to the lower line, which is $1 - e^{-\alpha}$. We want the upper line to touch exactly one point above the function $1 - e^{-\alpha x}$ in some point $x \in [0, 1]$. The only remaining part is to find y-intercept. First, we find a point of the function $1 - e^{-\alpha x}$ such that the line passing it has slope of $1 - e^{-\alpha x}$. Using basic derivative and solve for x we get $x = \frac{1}{\alpha} \ln(\frac{\alpha}{1-e^{-\alpha}})$

Substituting x in the function $1 - e^{-\alpha x}$ yields $1 - (\frac{1-e^{-\alpha}}{\alpha})$. Finally, we find y-intercept of upper line given slope of $1 - e^{-\alpha}$.

$$
\begin{aligned}
y &= mx + c \\
1 - (\frac{1 - e^{-\alpha}}{\alpha}) &= \frac{(1 - e^{-\alpha})}{\alpha}(\ln(\frac{\alpha}{1 - e^{-\alpha}})) + c \\
c &= 1 - (\frac{1 - e^{-\alpha}}{\alpha})(1 + \ln(\frac{\alpha}{1 - e^{-\alpha}}))
\end{aligned}
$$

◀

Subsequently, we use the upper and lower bounds identified in Lemma 7 in Theorem 3 for the case where δ is arbitrary but $t \leq 3\delta$. In other words, this allows us to capture how the size of *hvc* grows in the first 3δ time. This gives us another explicit function if $\alpha * \delta > 1$ and is evaluated in the Simulation section. Note that the bound in Lemma 7 is quite tight as we can see the result presented in Section 6.3.

▶ **Theorem 8.** *The solution $\psi(t)$ to Theorem 3 is bounded by the following time condition.*

- *For $t \in [\delta, 2\delta]$,*

  $$\psi(t) = 1 - ke^{-\alpha t/n}$$

  where $k = (1 - 1/n)e^{\alpha\delta/n}$.
- *For $t \in [2\delta, 3\delta]$,*

  $$1 - k_\ell H(t) \leq \psi(t) \leq 1 - k_u H(t) e^{\xi(t-\delta)}$$

  where $H(t) = e^{(1-e^{-\alpha})(kne^{-\alpha(t-\delta)/n}/\alpha + t - \delta)}$ and

  $$
  \begin{aligned}
  k_\ell &= (1 - (1 - ke^{-2\delta\alpha/n}))e^{(1-e^{-\alpha})(\frac{kn}{\alpha}e^{-\delta\alpha/n} + \delta)} \\
  k_u &= k_\ell e^{\delta\xi} \\
  \xi &= 1 - (\frac{1 - e^{-\alpha}}{\alpha})(1 + \ln(\frac{\alpha}{1 - e^{-\alpha}}))
  \end{aligned}
  $$

Proof. For $t \in [\delta, 2\delta]$, we can model as a sequence of single unicast message from the past $t - \delta$ and quantify the change accordingly. During $t \in [\delta, 2\delta]$, there is at most one message delivered because during $t - \delta$ there is only one green process, i.e., process j. Therefore, at any time the change of y depends only current y and one message with probability α. The expected change of fraction of Y over time is given a simple differential equation: $\frac{dy}{dt} = \frac{\alpha}{n}(1 - y)$ with initial condition $y(\delta) = 1/n$. Solving ordinary differential equation is an easy exercise.

▶ **Lemma 9.** *The solution to differential equation: $\frac{dy}{dt} = \frac{\alpha}{n}(1 - y)$ with initial condition $y(\delta) = 1/n$ is $y_1(t) = 1 - ke^{-\alpha t/n}$ where $k = (1 - 1/n)e^{\alpha\delta/n}$.*

For $t \in [2\delta, 3\delta]$, we replace the term $1 - e^{-\alpha y}$ with corresponding lower and upper bounds in Lemma 7. Consider the delay differential equation in Theorem 3 $\frac{dy}{dt} = (1-y)(1-e^{-\alpha y(t-\delta)})$

During $t \in [2\delta, 3\delta]$, the function $y(t-\delta)$ becomes $y_1(t-\delta) = 1 - ke^{-\alpha t/n}$ By Lemma 9. We use the tight lower and upper bounds as in Lemma 7 to obtain the equation:

$$(1-y)Ay \leq \frac{dy}{dt} \leq (1-y)(Ay+\xi)$$

where $A = (1-e^{-\alpha})$. Then, we instantiate value of $y = y_1 = 1 - ke^{-\alpha(t-\delta)/n}$ to obtain the following inequality:

$$(1-y)((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})) \leq \frac{dy}{dt} \leq (1-y)((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})+\xi)\,.$$

From this expression, we can consider only the equation $\frac{dy}{dt} = (1-y)((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})+\xi)$ as the lower bound term follows immediately from upper bound result when instantiating $b=0$ of Lemma 10, which is easily verified.

▶ **Lemma 10.** *We have the following integral results*

$$\int ((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})+b)dt = (1-e^{-\alpha})(\frac{kn}{\alpha}e^{-\alpha(t-\delta)/n}+(t-\delta))+b(t-\delta)+C\,.$$

The remaining part is to use the result from Lemma 10 to solve ordinary differential equation and find a constant term with initial condition $y(2\delta) = 1 - ke^{-2\delta\alpha/n}$.

$$\begin{aligned} \frac{dy}{dt} &= (1-y)((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})+\xi) \\ \int(\frac{1}{1-y})dy &= \int((1-e^{-\alpha})(1-ke^{-\alpha(t-\delta)/n})+b)dt \\ y &= 1-k_0e^{-((1-e^{-\alpha})(\frac{kn}{\alpha}e^{-\alpha(t-\delta)/n}+t-\delta)+b(t-\delta))} \end{aligned}$$

The results follow since k_0 can be solved with initial condition $y(2\delta) = 1 - ke^{-2\delta\alpha/n}$. This completes the proof. ◀

5 Reduction of ϵ-Constrained Time Model to Unconstrained Time Model

In this section, we show that unconstrained and ϵ-constrained time model are closely related. In particular, in Theorem 13, we show that ϵ-constrained time model can be solved by the solution for the unconstrained time model.

We first describe the basic idea behind Theorem 13. Initially, the process j is the only one red process. After some time, the number of red processes increases since process j sends message to some other processes and other processes that carry active information about j also send this entry, i.e., red processes help disseminate red messages. At the same time, if a process does not hear a message that contains newer information (directly or indirectly) about process j then in ϵ-constrained time model, this process should turn green. Therefore, at any time, the change of number of red processes is due to (1) green processes turning red, and (2) red processes turning green. We show that the number of red processes remains unchanged after some period of time. That is, the increase due to (1) is equal to the decrease due to (2), i.e., it reaches an equilibrium point.

To prove our result about ϵ-constrained time model (i.e., Model 2 in Section 2), we put different time labels on color. A process is τ-red if it receives τ-message directly or transitively, i.e., a message that is originated from process j at time τ. A process is red at time t if and only if it is τ-red for some $\tau \in [t-\epsilon, t]$.

Let $r_\tau(t)$ be a set of τ-red processes at time t. Based on definition of $r_\tau(t)$, we can compute the cardinality of $r_\tau(t)$.

▶ **Lemma 11.** *The expected number of τ-red processes is given by*

$$E[|r_\tau(t)|] = \begin{cases} 0 & \text{if } t \leq \tau \text{ or } t > \tau + \epsilon \\ y(t-\tau) & \text{otherwise} \end{cases}$$

Proof. If $t \leq \tau$ or $t > \tau + \epsilon$, it is either τ-message non-existent or expired. Otherwise, at time τ, process j sends first τ-message. This time is the initial condition of $y(t)$ which is $y(0)$. Thereafter, the number of τ-red process is equivalent to that of unconstrained time model since the τ-message is not expired until $t > \tau + \epsilon$. Hence, the result follows. ◀

▶ **Corollary 12.** *The following equation holds $|r_\tau(t)| = |r_{\tau+1}(t+1)|$ with high probability.*

Proof. The expression $E[|r_\tau(t)|] = E[|r_{\tau+1}(t+1)|]$ holds by simply substituting t as $t+1$ and τ as $\tau+1$ in the Lemma 11. ◀

Using these two results, we show that the fraction of the red processes in the ϵ-constrained time model can be derived by using the unconstrained time model as follows:

▶ **Theorem 13.** *Let $y(t), y_\epsilon(t)$ be fraction of red process at time t from model 1 and 2 respectively. $y_\epsilon(t)$ can be computed by the following expression.*

$$y_\epsilon(t) = \begin{cases} y(t) & \text{if } t \leq \epsilon \\ y(\epsilon) & \text{otherwise} \end{cases}$$

Proof. Define $R(t)$ as a set of red processes at time t. This is basically a union of τ-red processes at time t for $t - \epsilon \leq \tau \leq t - 1$. That is, $R(t) = \bigcup_{i=t-\epsilon}^{t-1} r_i(t)$. We note that $r_{\leq 0}(t) = \emptyset$ by definition. Hence, $R(i)$ for $0 \leq i \leq \epsilon - 1$ collects more term until $i \geq \epsilon$, which follows terms from definition of $R(t)$.

We show that expectation of $E[|R(t)|] = E[|R(t+1)|]$ for $t \geq \epsilon$. By definition, observe that $R(t+1) = \bigcup_{i=t-\epsilon+1}^{t} r_i(t+1) = \bigcup_{i=t-\epsilon}^{t-1} r_{i+1}(t+1)$. Now, we can compare $R(t)$ with $R(t+1)$ term by term. That is, we can compare $r_i(t)$ from $R(t)$ with $r_{i+1}(t+1)$ from $R(t+1)$. By Corollary 12, we know that cardinality of both terms are equal for $t - \epsilon \leq i \leq t - 1$. That last thing to show is that stochastic process gives us equality. Consider the following random process, there are n coupons. We can draw coupon ϵ trials by the following rule. For i-th trial, we draw $r_i(t)$ distinct number of coupons randomly. Let X be a random variable representing number of distinct coupons collected for ϵ trials. In this situation, $R(t)$ and $R(t+1)$ both represent X. Therefore, the expectation of two random variables must be equal because $R(t)$ and $R(t+1)$ are random variables of identical stochastic process.

Hence, $E[|R(t+1)|] = E[|R(t)|]$ for $t \geq \epsilon$. That is, $y_\epsilon(t) = y(\epsilon)$ for $t \geq \epsilon$. ◀

This result implies that we can use t and ϵ interchangeably since the *hvc* size of ϵ-constrained time model reaches equilibrium point after $t \geq \epsilon$.

6 Simulation Results

In this section, we evaluate our analytical model by comparing to simulation results. Since the analytical results in this paper are captured by Theorems 3–13, we perform simulation experiments to validate them.

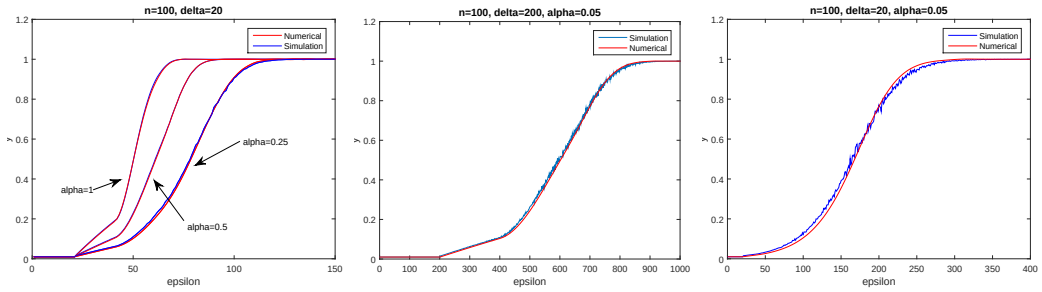

Figure 2 Simulation vs. Numerical Results from Theorem 3.

Simulation Setup. We implement according to our model in various configurations. For the purpose of experiments, we simulate distributed processes with central absolute time in one machine. We simulate sending event by adding a new message to priority queue of destination process with on arrival time $t + \delta$ in future. At each absolute time, each process checks if its inbox has messages with deliver time less than or equal to t. If so, we perform receive events. We repeat until no such message exists in the inbox. Each process has an access to physical time with ϵ-uncertainty interval guarantee. While we simulate sending and receiving events using central absolute time, the absolute time is oblivious to the processes. For purpose of reproducibility, all source codes for simulation are available at `http://www.cse.msu.edu/~yingchar/hvc.html`. All parameters are configurable.

6.1 Analytical vs. Simulation Results (Validation of Theorem 3)

In this case, we compare the analytical model from Theorem 3 with simulation results. For analytical solution, we use standard numerical solver *dde*23 in MATLAB. For experiments, we run for sufficiently long time so that the active clock (i.e., number of *hvc* entries) is stabilized. In particular, we plot the result for ϵ from various value of ϵ. For each ϵ, we run simulation for t up to 2000 and calculate the average starting from $t = \epsilon$ since we start from a state where a process knows only its own clock, we omit the initial clock values where the size of the clock is small.

The results are shown in Figure 2. We overlay numerical solution and simulation results. In Figure 2 (left), we set $n = 100, \delta = 20$ and run for three different values of $\alpha = 1$, 0.5 and 0.25 respectively. We overlay numerical solution and simulation results. In Figure 2 (mid) and (right), we set $n = 100, \alpha = 0.05$ and different values of $\delta = 200$ and 20, respectively. Note that the middle figure has $\alpha * \delta = 10$ where as the right figure has $\alpha * \delta = 1$. We notice the difference of $\alpha * \delta$ and its effect to characteristic of the plot. When $\alpha * \delta$ is small as suggested by Theorem 5, the graph looks like sigmoid function. This shows our analytical model in Theorem 3 gives us an exact plot with simlation results with minimal error. We notice slight perturbation for small value of α. This is due to discontinuity of the discrete events.

From these results, we corroborate that the relation predicated in Theorem 3 is valid and tight.

Given that we have an exact analytical model, we now consider the bound we have for closed form approximation results. From now on, we use the numerical solution as a baseline.

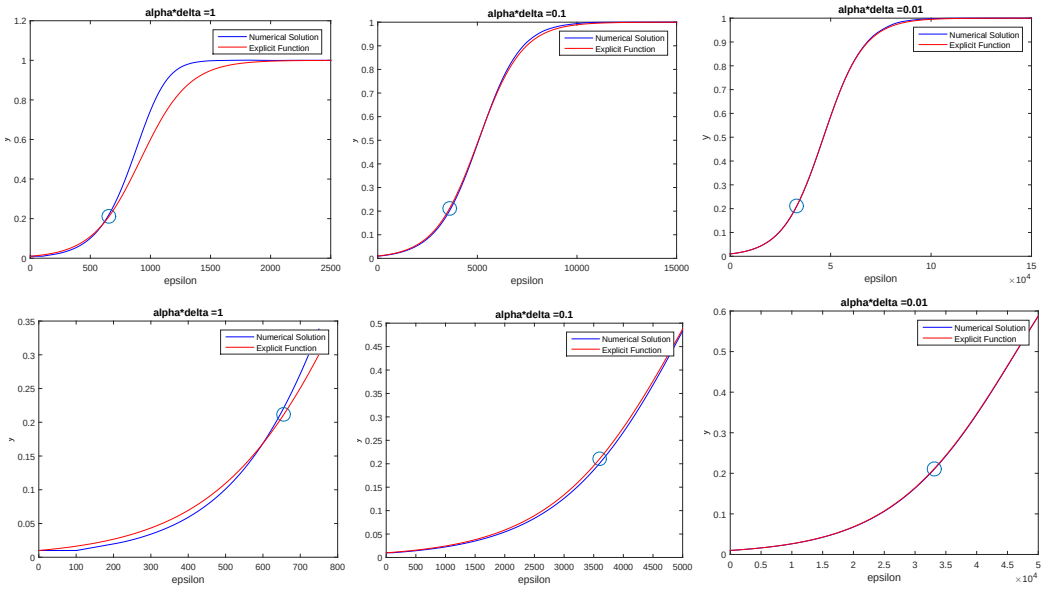

Figure 3 We compare explicit function with numerical solution. The circle is the phase transition ϵ_p obtained by Theorem 6. The bottom figures are zoomed version of the corresponding upper ones.

6.2 Explicit Form vs. Numerical Solutions (Validation of Theorem 5, 6 and 8)

Theorem 5 gives an explicit function when $\delta * \alpha$ is arbitrarily small. How small does it need to be is a subject of this section. In Figure 3, we fix $\delta = 100$ and vary α for $\delta * \alpha = 1, 0.1$,and 0.01, respectively. This shows that the explicit function in Theorem 5 is identical to numerical solution when $\alpha * \delta$ is small. Typical value is $\alpha * \delta < 1$. Note that phase transition $\epsilon_p = (\alpha^{-1} + \delta)(\ln((2 - \sqrt{3})(n - 1)))$ is shown in circle. The bottom figures are the zoomed-in version of corresponding top ones.

If $\alpha * \delta$ is large, the hvc size is typically big during $t < 3\delta$. We obtain the upper and lower bound close forms using a technique called method of steps in delay differential equation. We evaluate the result accordingly.

By Theorem 8, we have exact solution for $t \leq 2\delta$ and approximate closed form for $t \in [2\delta, 3\delta]$. We simulate in various configurations. The result is shown in Figure 4. According to these experiments, Theorem 8 gives us an exact bound during $t \in [\delta, 2\delta]$ and reasonable upper and lower bound during $t \in [2\delta, 3\delta]$. Note that when α is small, the approximation converges to exact as shown in Figure 4 (mid and right).

6.3 Unconstrained vs. ϵ-constrained Time (Validation of Theorem 13)

We evaluate relationship between $y(t)$ (from the unconstrained time model) and $y_\epsilon(t)$ (from ϵ-constrained time model). Theorem 13 implies that $y_\epsilon(t) = y(t)$ for $t \geq \epsilon$. Specifically, in Figure 5, we simulate the programs for 100 processes with $\alpha = 0.25$ and $\delta = 10$. The result show that $y_x(t)$ is almost same as $y(t)$ when $t \leq x$. And, $y_x(t)$ is almost same as $y(x)$ when $t > x$ for $x = 30, 60$. This conforms to the prediction in Theorem 13.

In addition, we plot the distribution of sizes of hvc of all processes at each time. In Figure 5 (right), we plot box distribution which is based on normal distribution. The middle point represents average value at time t. The thick area represents area within a standard

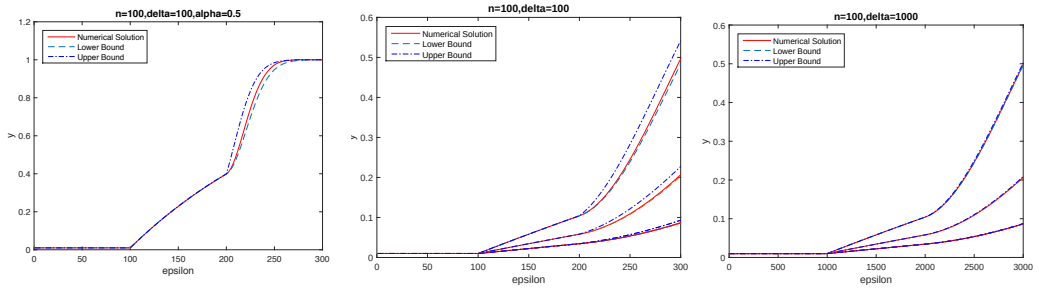

Figure 4 Numerical solution vs. closed form for $t \leq 3\delta$. Note we use $\alpha = 0.1, 0.05$ and 0.025 for middle figure. For right figure, we use $\alpha = 0.01, 0.005$, and 0.0025, respectively.

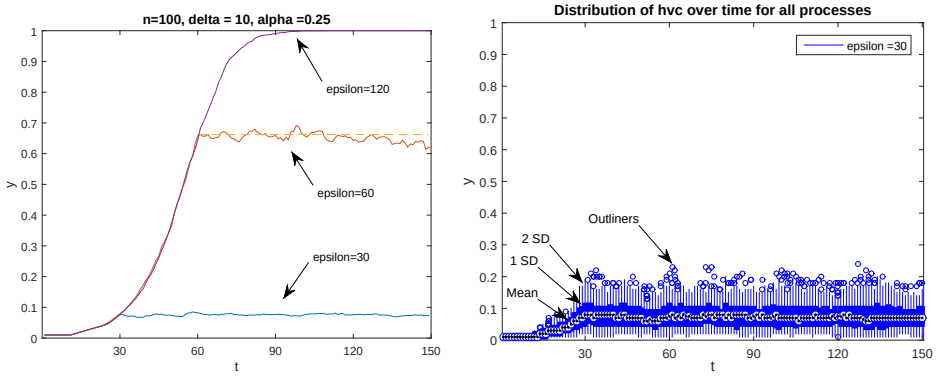

Figure 5 Validation of Theorem 13. (Left) We plot three graphs of $n = 100, \delta = 10$, and $\alpha = 0.25$. Each graph uses different value of ϵ. Note that after $t = \epsilon$, the function is stabilized. (Right) We plot actual distribution in terms of box plot for each time t.

deviation. The thin area represents twice standard deviation. The above dots are outliners. During before phase transition, we can expect a distribution around $y(\epsilon)$.

From these results, we find that the relation predicated in Theorem 13 is valid.

7 Practical considerations for HVC sizes and the phase transition

Our analytical derivations and simulation experiments point to a phase transition on HVC size. Here we use typical values for δ and α from datacenter environments, and determine the phase transition threshold. We show that ϵ achieved using NTP is much less than this phase transition threshold, so for practical distributed systems and modern deployments, the HVC sizes will remain very small and significantly less than n.

For convenience, we calculate phase transition ϵ^* in terms of seconds rather than unit time (clock tick) as follows. Let r be resolution of the time protocol, e.g., NTP has a theoretical resolution of 2^{-32} seconds (233 picoseconds) [15]. The unit of r is seconds per clock tick. Define f as messages frequency in terms of number of messages per second. Also, let d be minimum messages delay in terms of seconds. It is easy to see that $\alpha = fr$ and $\delta = \frac{d}{r}$. We assume that α is small. This is true when the time protocol has sufficient resolution.

[2] Technically, number of messages are equivalent to number of clock ticks that trigger the sending events. Hence, the message rate α means proportion of such clock ticks over all clock ticks in the long run.

If $\alpha * \delta << 1$, or equivalently $f * d << 1$, we can apply these values into our phase transition formula in Theorem 6 to obtain $\epsilon_p = \frac{(f^{-1}+d)}{r} \ln((2-\sqrt{3})(n-1))$. Note that ϵ_p has unit of clock ticks. We can convert to seconds by multiplying by r. Therefore, $\epsilon^* = (f^{-1}+d)\ln((2-\sqrt{3})(n-1))$. In other words, ϵ^* is proportional to $f^{-1}+d$, for fixed value of n.

For example, consider the following configuration. A small value for d is 1 millisecond. To determine a value for α, communication frequency, we will consider chatty nodes that send 100 messages a second to other nodes in a distributed system of n nodes. Most practical applications in fact use orders of magnitude lower α, since reducing message communication rates can improve efficiency of distributed systems. Techniques for reducing α include aggregation and batching of messages before sending a message.

Since $\alpha * \delta << 1$, we can use Theorem 6 to calculate the phase transition threshold ϵ^*. When we substitue above values for d and f with $n = 100$ in the formula, we get $(100^{-1} + 10^{-3})\ln((2-\sqrt{3})(99)) = 0.036$ seconds.

In this situation, it is possible to get ϵ less than 10ms using NTP synchronized clocks, which is less than the phase transition for above configuration. Our simulation results show that this corresponds to an average 1.1 *hvc* size for 100 nodes. That is each *hvc* clock maintains only few entries most of the time, yet when it is needed *hvc* expands on demand to allow more entries to capture causality both ways in the ϵ uncertainty slices. Therefore, we can see that for various deployments of practical distributed applications, our HVC component will avoid the phase transition and achieve very small *hvc* sizes. Moreover, using less chatty nodes, hence a smaller α communication frequency, would also lead to larger ϵ^*.

Finally, $\epsilon_p = (\frac{1}{\alpha} + \delta)(\ln((2-\sqrt{3})(n-1)))$ also tells us that the HVC sizes scale very well with respect to n, the number of nodes in the network. As we can see from the above equation increase in n will increase ϵ_p, and will delay the critical phase transition of HVC sizes. In the traditional VC, the size of the VC increase directly with n. In HVC, surprisingly, the increase of n, benefits in delaying the phase transition and reducing the size of HVC. The intuition behind this is that, the critical phase transition occurs when a process not only gets entries added to its clock from direct interaction but also from indirect transfer with another processes' HVC entries. This indirect hearing and addition makes the HVC entries blow up. For larger n, the probability of such indirect addition reduces slightly, and hence larger n, delays the phase transition to large HVC sizes.

7.1 Extensions to the model

Unicast is the predominant communication pattern in cloud computing systems. Sending a message to many recipients is often implemented in terms of multiple unicast messages. Our modeling however failed to capture the incast problem that occurs back when several nodes send message back to the same node. Due to large fan-in/fan-outs in some cloud computing, and especially web services systems, incast problem may occur. When many nodes may be sending back to the same node at the same time, this may grow the number of HVC entries at the recipient beyond what the model predicts.

In addition, our model uses a single worst case ϵ to denote clock synchronization uncertainty in the system. However, in a large scale distributed system, there will some nodes that are more tightly synchronized with real time, versus some nodes that are poorly synchronized with real time. It is possible to go finer grain tracking of ϵ and record and use per-node ϵ. We leave this as future work to explore.

8 Related work

Logical clock (LC) [12] was proposed in 1978 by Lamport as a way of timestamping and ordering events in an asynchronous distributed system. In 1988, the vector clock (VC) [10, 13] was proposed to maintain a vectorized version of LC. VC maintains a vector at each node which tracks the knowledge this node has about the logical clocks of other nodes. While LC finds one consistent snapshot (that with same LC values at all nodes involved), VC finds all possible consistent snapshots, which is useful for debugging applications. Unfortunately, the space requirement of VC is on the order of nodes in the system, and is prohibitive, and it stays prohibitive with optimizations [9, 18, 14] that reduce the size of VC. Resettable vector clocks (RVC) generalizes the notion of VC, and provides a bounded-space and fault-tolerant implementation of VC applications. To this end, RVC identifies an interface contract under which the RVC implementation can be substituted for VC in the client applications, without affecting the client's correctness. The number of entries in RVC is still n, the number of nodes in the system.

A version vector (VV) [16] is a version of VC that is customized for summarizing the set of updates applied to a replica in a replicated database system. While VC is employed for establishing a partial order among a set of events occurring in the nodes, VV is employed for establishing a partial order among the replicas in the distributed system. VV generally uses less number of entries (typical replica sizes are 3 or 5) and offers more opportunities for bounding the size of each entry [1]. As another approach to reducing VV sizes, a unilateral VV pruning algorithm is introduced using loosely synchronized clocks [17]. That algorithm assumes synchronous networking: it demands that each event be delivered to all live nodes and processed by them within a fixed period [17].

The clocks discussed above have been adopted by many cloud computing systems. Dynamo [19] adopts version vectors for causality tracking of updates to the replicas. Orbe [7] uses dependency matrix along with physical clocks to obtain causal consistency. In the worst case, both these solutions require large timestamps. Cassandra uses physical time and LWW-rule for updating replicas. Spanner [5] employs TrueTime (TT) to order distributed transactions at global scale, and facilitate read snapshots across the distributed database. TT relies on a well engineered tight clock synchronization available at all nodes thanks to GPS clocks and atomic clocks made available at each cluster. In order to ensure $e \underline{hb} f \Rightarrow tt.e < tt.f$ and provide consistent snapshots, Spanner requires waiting-out uncertainty intervals of TT at the transaction commit time which restricts throughput on writes. In contrast, HVC and HLC does not require waiting out the clock uncertainty, since they are able to record causality relations within this uncertainty interval using the VC and LC update rules.

A recent work [2] surveys the use of clocks in cloud computing and investigates how the logical and physical clock concepts are applied in the context of developing distributed data store systems for the cloud and review the choice of clocks in relation to consistency/performance tradeoffs.

An alternate approach for ordering events is to establish explicit relation between events. This approach is exemplified in the Kronos system [8], where each event of interest is registered with the Kronos service, and the application explicitly identifies events that are of interest from causality perspective. This allows one to capture causality that is application-dependent at the increased cost of searching the event dependency relation graph.

9 Conclusion

We presented an analytical model to compute the size of HVC. This analytical model had four parameters: ϵ (window of uncertainty), δ (minimum message delay), α (message rate) and n (size of the network). We presented a differential equation whose solution provides the estimated size of HVC. We also identified closed form solutions for some special cases. We used simulation results to validate the analytical model. In particular, we showed that the results predicated by the analytical model are identical to the simulation results. Moreover, the upper and lower bounds computed by the closed form solutions are also very close to the simulation results. Hence, they can be used to predict the size of HVC in the given system setting. We also showed that many deployments of practical distributed applications will avoid the phase transition easily and achieve very small HVC sizes, significantly less than n, the number of nodes in the system.

References

1 J. Almeida, P. Almeida S. Paulo, and C. Baquero. Bounded version vectors. *Distributed Computing: 18th International Conference, DISC 2004*, pages 102–116, 2004.

2 M. Bravo, N. Diegues, J. Zeng, P. Romano, and L. Rodrigues. On the use of clocks to enforce consistency in the cloud. *IEEE Data Eng. Bull*, 38(1):18–31, 2015.

3 B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Inf. Process. Lett.*, 39(1):11–16, 1991.

4 Cockroachdb: A scalable, transactional, geo-replicated data store. http://cockroachdb.org/.

5 J. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, JJ. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. *Proceedings of OSDI*, 2012.

6 M. Demirbas and S. Kulkarni. Beyond truetime: Using augmentedtime for improving google spanner. *LADIS'13: 7th Workshop on Large-Scale Distributed Systems and Middleware*, 2013.

7 J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC'13, pages 11:1–11:14, New York, NY, USA, 2013. ACM. `doi:10.1145/2523616.2523628`.

8 R. Escriva, A. Dubey, B. Wong, and E.G. Sirer. Kronos: The design and implementation of an event ordering service. *EuroSys*, 2014.

9 M. Ahamad F. J. Torres-Rojas. Plausible clocks: Constant size logical clocks for distributed systems. *Proceedings of WDAG*, pages 71–88, 1996.

10 J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10(1):56–66, Feb 1988.

11 S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone. Logical physical clocks. In *Principles of Distributed Systems*, pages 17–32. Springer, 2014.

12 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

13 F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

14 S. Meldal, S. Sankar, and J. Vera. Exploiting locality in maintaining potential causality. In *Proceedings of Principles of Distributed Computing (PODC)*, pages 231–239, 1991. doi:10.1145/112600.112620.

15 D. Mills. A brief history of ntp time: Memoirs of an internet timekeeper. *ACM SIGCOMM Computer Communication Review*, 33(2):9–21, 2003.

16 D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, SE-9(3):240–247, May 1983. doi:10.1109/TSE.1983.236733.

17 Y. Saito. Unilateral version vector pruning using loosely synchronized clocks. Technical report, HP Labs, 2002.

18 M. Singhal and A. Kshemkalyani. An efficient implementation of vector clocks. *Information Processing Letters*, 43:47–52, 1992.

19 W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.