# Concurrent Knowledge and Logical Clock Abstractions

Ajay D. Kshemkalyani

Dept. of EECS, University of Illinois at Chicago, Chicago, IL 60607-7053, USA
ajayk@eecs.uic.edu

**Abstract.** Vector and matrix clocks are extensively used in asynchronous distributed systems. This paper asks, "how does the clock abstraction generalize?" and casts the problem in terms of concurrent knowledge. To this end, the paper motivates and proposes logical clocks of arbitrary dimensions. It then identifies and explores the conceptual link between such clocks and knowledge. It establishes the necessary and sufficient conditions on the size and dimension of clocks required to declare $k$-level concurrent knowledge about the most recent global facts for which this is possible without using control messages. It then gives algorithms to compute the latest global fact about which a specified level of knowledge is attainable in a given state, and to compute the earliest state in which a specified level of knowledge about a given global fact is attainable.

## 1 Introduction

### 1.1 Motivation

A large number of application areas in asynchronous distributed message-passing systems use vector clocks and matrix clocks. Some example areas that use vector clocks [6,14] are checkpointing, garbage collection, causal memory, maintaining consistency of replicated files, taking efficient snapshots of a system, global time approximation, termination detection, bounded multiwriter construction of shared variables, mutual exclusion, debugging, and defining concurrency measures. Some example areas that use matrix clocks are designing fault-tolerant protocols and distributed database protocols [9,20], including protocols to discard obsolete information in distributed databases [18], and protocols to solve the replicated log and replicated dictionary problems [20].

Vector clocks can be thought of as imparting knowledge to a process: when $V[i] = x$ at process $h$, process $h$ knows that process $i$ has executed at least $x$ events. Matrix clocks give one more level of knowledge: when $M[i, j] = x$ at process $h$, process $h$ knows that process $i$ knows that process $j$ has executed at least $x$ events. Vector and matrix clocks are convenient as they are updated without sending additional messages; knowledge is imparted via the inhibition-free *ambient message passing* that (i) *eliminates control messages* by using piggybacking, and (ii) *diffuses knowledge using* only *computation messages, whenever sent.*

S. Kapoor and S. Prasad (Eds.): FST TCS 2000, LNCS 1974, pp. 489–502, 2000.

This paper asks the question: *"how does this clock abstraction generalize?"* The problem is cast in terms of concurrent knowledge ("everybody knows on consistent cuts"), which is a form of knowledge appropriate for (time-free) asynchronous distributed systems [16] — all the applications mentioned above implicitly use concurrent knowledge that is *not* common knowledge in their clock algorithms, although this has never been formally studied as such.

### 1.2 Background

A distributed system can be modeled by a network $(N, L)$, where $N$ is the set of processes that communicate by message passing over $L$, the set of logical links. We assume an asynchronous distributed (message passing) system, i.e., there is no global clock or shared memory, relative process speeds are independent, and message delivery times are finite but unbounded [2,16]. *Common knowledge*, which has been proposed as a definition of agreement in distributed systems, is defined as follows [8]. A process $i$ that knows a fact $\phi$ is said to have knowledge $K_i(\phi)$, and if "every process in the system knows $\phi$", then the system exhibits knowledge $E^1(\phi) = \bigwedge_{i \in N} K_i(\phi)$. A knowledge level of $E^2(\phi)$ indicates that every process knows $E^1(\phi)$, i.e., $E^2(\phi) = E\ (E^1(\phi))$. Inductively, a hierarchy of levels of knowledge $E^j(\phi)$ $(j > 0)$ gets defined, where $E^{k+1}(\phi) \Longrightarrow E^k(\phi)$. Common knowledge of $\phi$, denoted as $C(\phi)$, is defined as the knowledge $X$ which is the greatest fixed point of $E(\phi \wedge X)$ and is equivalent to $\bigwedge_{j \in Z*} E^j(\phi)$, where $Z*$ is used to denote the set of whole numbers. Common knowledge requires simultaneous action for its achievement and is therefore unattainable in asynchronous distributed systems [5,7,8,17].

Panangaden and Taylor proposed *concurrent common knowledge* (CCK) which is required to be attained simultaneously in *logical time* based on *causality* [13], and is attainable in asynchronous distributed systems [16]. Specifically, CCK can be attained at a "consistent cut" or possible global state [1] in the system execution. To define concurrent common knowledge, [16] first defines $P_i(\phi)$ to represent the statement "there is some consistent global state of the current execution that includes $i$'s local state, in which $\phi$ is true." $E^C(\phi) = \bigwedge_{i \in N} K_i P_i(\phi)$ and is attainable by the processes at a consistent global state. Likewise, higher levels of knowledge $(E^C)^k(\phi)$, for $k > 1$, are attainable by the processes at a consistent global state. Concurrent common knowledge of $\phi$, denoted by $C^C(\phi)$, is defined as the knowledge $X$ which is the greatest fixed point of $E^C(\phi \wedge X)$ and is equivalent to $\bigwedge_{j \in Z*} (E^C)^j(\phi)$. This form of knowledge underlies many existing protocols involving processes reaching agreement about some property of a consistent global state, defined using logical time and causality.

$C^C(\phi) \Longrightarrow (E^C)^j(\phi)(j \in Z*)$. Several applications (see Section 1.1) need only lower levels of such knowledge. Vector clocks [6,14] provide specific knowledge of a global fact/state $\phi$, equivalent to concurrent knowledge $(E^C)^0(\phi)$, in the application domain. However, vector clocks are not sufficient for other applications, for which it is necessary to use matrix clocks. Matrix clocks provide concurrent knowledge $(E^C)^1(\phi)$ about facts $\phi$ in the application domain. Thus, although levels of concurrent knowledge (besides CCK) have not received formal

attention besides [16], they are implicitly used in a wide range of applications. Hence, studying levels of concurrent knowledge is important.

Two important and desirable characteristics of the clock protocols used to achieve $(E^C)^0(\phi)$ and $(E^C)^1(\phi)$ are that they *do not use any control messages* and *they diffuse knowledge on a continual basis*, using piggybacked timestamps on the application messages as and when they are sent (see Section 1.1). These clock protocols are not full-information protocols [5], yet for each of the above applications, they suffice to provide the required degree of concurrent knowledge because the clocks are defined so as to capture the property of interest.

All other known protocols to attain concurrent common knowledge and levels of concurrent knowledge are variants of the global state recording algorithm [1,4,16]. Such global state protocols require (i) $O(min(k \cdot |N|, |L|))$ messages to attain $(E^C)^k(\phi)$ and $O(|L|)$ messages to attain $C^C(\phi)$; (ii) $O(d)$ communication time steps, where $d$ is the network diameter. In the above, the $|L|$ factor in the message complexity can be reduced to $|N|$ if inhibitory protocols are used, at the cost of inhibitory time delays [4]. All protocols based on global state recording require control messages for a one-time knowledge attainment of each fact, and may additionally use freezing/inhibition. Hence, they are not considered further.

In Theorem 1, we show that for the class of facts we consider and which includes the applications listed in Section 1.1, $(E^C)^k(\phi)$ is equivalent to $E^k(\phi)$. So we also refer to $(E^C)^k(\phi)$ as just $E^k(\phi)$.

### 1.3 Objectives

This paper examines the feasibility of and mechanisms for achieving levels of knowledge $E^k(\phi)$ $(k > 0, 1)$ using ambient message passing, i.e.,

1. No control messages can be used. Control information may be piggybacked on computation messages. Also, no freezing/inhibition is allowed.
2. The latest knowledge about the (past) computation should be diffused as much as possible, using only the computation messages, whenever sent.

As justified in Section 1.2, we focus only on clock-based protocols. We now formalize the objectives. The *full-information protocol (FIP)* which attains common knowledge (in a synchronous system) has been defined such that "at each step (after each local event), a process broadcasts (via control messages) to other processes its local state (which captures everything it knows)" [5]. The FIP is very expensive, and does not meet our criteria. We now define a "no control messages" protocol, the *full-information piggybacking protocol (FIPP)*, to be one in which on each computation message of the application, the local state information is piggybacked by the sender. This protocol meets the criteria but is expensive in terms of the information piggybacked. We define the $k$-bounded-information piggybacking protocol to overcome this drawback of the FIPP.

**Definition 1.** *The* k-bounded-information piggybacking protocol (KIPP) *is such that on each computation message of the application, $k$-bounded state information*[1] *is piggybacked by the sender $j$, where $k$-bounded information is in-*

[1] Presumably about some property of interest.

*formation of the form:* $K_j(K_{i_1}(K_{i_2}\ldots(K_{i_k}(\phi))\ldots))$, *where* $i_1, i_2, \ldots i_k \in N$, *for any fact* $\phi$ *on the system state.*

*Facts about the property of interest*, which are a function of the system state, are *represented by the timestamp of that system state* in the applications of Section 1.1. *Similarly, 0- and 1-bounded information about these facts are also represented as timestamps* in these applications. Therefore, we will assume that appropriate timestamps can represent facts relevant to the application, and k-bounded information about them. The type of facts considered in Section 1.1 and which we will consider satisfy *monotonicity*. Informally, $\phi$ is a *monotonic fact* in a run if $\phi$ holds in some global state, and for every later global state, some $\psi$ holds and $\psi \Longrightarrow \phi$ in that later state (see Definition 5). Monotonic facts are also stable. The paper answers the following questions.

*Problem 1.* In a system using the KIPP protocol, what are the necessary and sufficient conditions on the timestamp information required to achieve and declare $E^k(\phi)$, where $\phi$ is the greatest possible monotonic fact (most recent possible system state) about which $E^k(\phi)$ is possible to be declared in the current state?

*Problem 2.* For any global monotonic fact $\phi$ on the system state, what is the earliest global state in which $E^k(\phi)$ is attained using the KIPP protocol?

*Problem 3.* Given a timestamp of a system state, what is the maximum possible monotonic fact $\phi$ (most recent possible system state) about which $E^k(\phi)$ can be declared in the given state in a system using the KIPP protocol?

Section 2 describes the system model and existing clock systems. Section 3 defines monotonic facts. Section 4 proposes $\alpha$-dimensional clocks. Section 5 analyzes the levels of knowledge that can be inferred using $\alpha$-dimensional clocks and answers the above problems. Section 6 concludes. See [12] for the full paper.

## 2 Preliminaries

### 2.1 System Model

We assume an asynchronous distributed system (see the first paragraph of Section 1.2). The notion of the local state of a process is primitive. An event $e$ at process $i$ is denoted $e_i$. An event causes a local state transition. The local history of process $i$, denoted $h_i$, is a possibly infinite sequence of alternating local states (beginning with a distinguished initial state) and events [16]. It is equivalently described by the initial state and the sequence of local events.

Formally, an asynchronous distributed system consists of (i) a network $(N, L)$, (ii) a set $\mathcal{H}_i$ of possible local histories for each process $i$, (iii) a set $\mathcal{A}$ of *asynchronous runs* or *executions*, or *computations*, each of which is a vector of local histories, one per process, and (iv) a set of messages sent in any possible asynchronous run. The system follows the KIPP protocol (Definition 1).

A given run of a distributed system has a poset event structure model as in [13]. Let $(H, \prec)$ represent the set of events $H$ in a system run that are related by the causality relation $\prec$, an irreflexive partial order [13]. $H$ is partitioned into local executions, one per process. Each local execution defines the local history. We assume the initial state of each process is common knowledge.

A *global state* (or *cut*) of run $a$ is a $n$-vector of prefixes of local histories of $a$, one prefix per process. It can be viewed equivalently as the union of the events in prefixes of the local histories of $a$, one prefix per process. A consistent global state (*consistent cut*) is a global state such that if the receipt of a message is recorded, then the sending of that message is also recorded [1]. It can be viewed equivalently as a downward-closed subset of $H$. Let $H^{\perp}$ denote the empty cut.

For a given run, the set of all cuts, $\mathcal{C}uts$, forms a lattice ordered by "$\subset$" (subset); the set of downward-closed cuts is its sublattice [14]. The seq. of states in actual time is a chain in this sublattice. The sublattice is not visible to any process, but gives the possible consistent cuts which could have occurred and are "valid" views of the run. Our results implicitly deal with such a run.

We define $F(Cut)$ to be the set consisting of the latest event at each process in cut $Cut$. $F(Cut)$ denotes the "front" of cut $Cut$.

**Definition 2.** $F(Cut) =_{def} \{e_i \in Cut \mid \forall e_i' \in Cut, e_i' \preceq e_i\}$

Given a cut $Cut$, its projection $F_i(Cut)$ is the element of $F(Cut)$ at process $i$.

Define $\downarrow e$ as $\downarrow e =_{def} \{e' \mid e' \preceq e\}$. The cut $\downarrow e$ has a unique maximal event $e$ and is downward-closed in $(H, \prec)$. As the set of all downward-closed cuts forms a lattice, therefore $\bigcap_{x \in X} \downarrow x$ and $\bigcup_{x \in X} \downarrow x$ , also denoted as $\cap_{\Downarrow} X$ and $\cup_{\Downarrow} X$, resp., are downward-closed cuts for any set of events $X$. These cuts are used to prove Theorems 6 and 7. Based on the definition of $\downarrow e$, we can assert as follows.

**Proposition 1.** $e \in \cap_{\Downarrow} X \iff \forall x \in X,\ e \preceq x$

$\cap_{\Downarrow} X$, the largest set of events that causally precede every $x \in X$, represents the largest execution prefix with the following property: any fact in this execution prefix can be known in the local state of each process after event $x \in X$ [11].

A Kripkean interpretation of knowledge modality requires the identification of an appropriate set of possible worlds – in the system model, the possible worlds are the (consistent) cuts of the set of possible asynchronous runs [7,8]. $(a, c)$ denotes cut $c$ in run $a$. Standard definitions for the modal operators $K_i$ and $P_i$, and for various forms of knowledge are used. The formal semantics are given by the satisfaction relation $\models$ and are the same as in [16]. Proposition 1 can now be expressed in this logic. Assuming that adequate knowledge about local histories is propagated, for any cut $X$, $(a, X) \models E(\cap_{\Downarrow} F(X))$, i.e., all the processes know $\cap_{\Downarrow} F(X)$ after execution of $X$.

### 2.2 Logical Clocks

Logical clocks track causality which determines the extent of the past computation that could possibly be known at any state/event. A clock is a function that

maps cuts in a run to elements in the time domain $\mathcal{T}$. Thus, $Clk : \mathcal{C}uts \mapsto \mathcal{T}$. Clocks provide a quantitative identifier for cuts. For any run, the timestamp of a cut (which is the union of a prefix of the local history of each process), is defined using timestamps of cuts of the form $\downarrow e$. When we say that an event $e$ is assigned a clock value/timestamp, more formally we mean that the cut $\downarrow e$ is assigned that clock value/timestamp. Also, a subscripted timestamp $T_i$ denotes a timestamp of an event at process $i$, and $|N|$ is also denoted as $n$.

Scalar clocks [13], vector clocks [6,14], and matrix clocks [9,18,20] are the only clocks proposed in the literature. A canonical clock updates the local component of the clock by one at each local event. Henceforth, we assume canonical clocks. A canonical vector clock assigns timestamps to an event as follows.

**Definition 3.** $T(e) =_{def} (i \in N)\ T(e)[i] = |\{e_i \mid e_i \preceq e\}|$, *i.e.,* $T(e)[i]$ *is the number of events on process* $i$ *that causally precede or equal* $e$.

For any run, vector clocks of size $n$ track the progress at each process (and are needed to capture concurrency; see discussion on dimension of $(H, \prec)$ [3]). For cut $Cut$, we define its timestamp $T(Cut)$ such that its $i$th component is the $i$th component of the timestamp of event $F_i(Cut)$ [11].

**Definition 4.** $T(Cut) =_{def} (i \in N)\ T(Cut)[i] = T(F_i(Cut))[i]$

The vector timestamp of a cut identifies the number of events at each process in the cut. For any run, there is an isomorphism between $\mathcal{C}uts$ and $\mathcal{T}^1$, the set of canonical vector timestamps such that $(T \in \mathcal{T}^1)\ T[i] \leq |h_i|$ in that run.

**Proposition 2.** *For a run* $(H, \prec)$, $(\mathcal{C}uts, \subset)$ *is isomorphic to* $(\mathcal{T}^1, <)$.

**Lemma 1.** *The timestamp of cut* $\bigcap_{x \in X} \downarrow x$ *, denoted* $T(\cap_{\Downarrow} X)$*, is expressed as a function of the timestamps of the members of* $X$ *as follows.* $(i \in N)\ T(\cap_{\Downarrow} X)[i] = min_{x \in X}(T(x)[i])$.

In Lemma 1, $X$ can be an arbitrary set of events, also termed a nonatomic event [10,11]. Lemma 1 will be shown to have a counterpart Lemma 3 that is based on higher dimensional clocks, and which is used in the proof of Theorem 7.

For any run $(H, \prec)$, observe from Definition 2 that there is a bijection from the set containing each cut $Cut$ to the set containing each front of a cut $F(Cut)$. So, the timestamp of $F(Cut)$ is defined to be the timestamp of $Cut$.

## 3 Monotonic Facts

We now define monotonic facts – such facts capture the relevant properties of the applications in Section 1.1, and it is this class of facts that we consider. Examples of such facts are "computation has progressed at least up to global state *state_vector*", and "all logs upto global state *state_vector* can be discarded". As in the applications in Section 1.1, we assume facts of interest are related by a semantic inclusion relation "$\sqsubseteq$" (if $\phi \sqsubseteq \psi$, then $\psi$ semantically includes $\phi$).

**Definition 5.** *For a given run $a$, any fact $\phi$ is monotonic iff for every cut $c$ at which $(a, c) \models \phi$, and for every cut $c'$ such that $c \prec c'$, there exists some fact $\psi$ such that $(a, c') \models \psi$ and $(a, c') \models (\phi \sqsubseteq \psi)$.*

Monotonic facts are also stable; however, not all stable facts are monotonic.

**Lemma 2.** *For a monotonic fact $\phi$, the following are all stable facts: $\phi$, $K_i(\phi)$, $K_iP_i(\phi)$, $E(\phi)$, and $E^k(\phi)$.*

Let $\psi$ be any of $\phi$, $K_i(\phi)$, $K_iP_i(\phi)$, $E(\phi)$, and $E^k(\phi)$, where $\phi$ is a monotonic fact. When process $m$ receives a message with $\psi$ piggybacked on it from process $j$ at event $e^y_m$ resulting in local state $s^y_m$, we have $s^y_m \models K_mP_mK_j(\psi)$. Using Lemma 2, we can show that "the $P_m$ operator can be safely removed", and hence $s^y_m \models K_mK_j(\psi)$ (and also $s^y_m \models K_m(\psi)$). Similarly, $\bigwedge_i K_iP_i(\psi)$ is equivalent to $\bigwedge_i K_i(\psi)$. Developing this idea further leads to Theorem 1 that allows us to replace concurrent knowledge with the equivalent normal knowledge.

**Theorem 1.** *In a system following the KIPP protocol, the greatest possible monotonic fact $\phi$ about which $(E^C)^k(\phi)$ is possible to be declared in a given state is the greatest possible monotonic fact $\phi'$ about which $E^k(\phi')$ is possible to be declared in the given state.*

As in the applications of Sect. 1.1, we assume that for any run, the set of monotonic facts ordered by $\sqsubseteq$ is a lattice, there is a semantically greatest fact in each state, and that there is an (iso/homo)morphism from $(\mathcal{C}uts, \subset)$ to $(\mathcal{M}, \sqsubseteq)$, where $\mathcal{M}$ is the set that contains the greatest monotonic fact (of interest) at each cut in $(\mathcal{C}uts, \subset)$. Combining this (iso/homo)morphism with Prop. 2 (and restricting to consistent states for semantic integrity) leads to Prop. 3.

**Proposition 3.** *The (semantically greatest) monotonic fact of interest in a global state, whose truth value is a function of that global state, will be uniquely identified by the timestamp of that global state.*

## 4 Clocks of Arbitrary Dimensions

**Definition 6.** *An $\alpha$-dimensional clock $Clk^\alpha$ defines the mapping $Clk^\alpha : \mathcal{C}uts \mapsto (Z*)^{n^\alpha}$ (i.e., $Clk^\alpha$ is an $\alpha$-dimensional array of integers, where each dimension is of size $n$), satisfying the following properties.*

**SP1.** *The local clock component at process $j$, $Clk^\alpha_j[j, j, \ldots, j]$, is common knowledge in the initial system state, i.e., $(a, H^\perp) \models C(Clk^\alpha_j[j, j, \ldots, j])$.*

**SP2.** *The local clock component at process $j$, $Clk^\alpha_j[j, j, \ldots, j]$, must be incremented by a natural number when a computation event occurs at $j$.*

**SP3.** *Any element $Clk^\alpha(e_j)[i_1, i_2, \ldots, i_\alpha]$ is the maximum scalar clock value $\phi_{i_\alpha} = Clk_{i_\alpha}[i_\alpha, i_\alpha, \ldots, i_\alpha]$ at $i_\alpha$ such that $K_j(K_{i_1}(K_{i_2}(K_{i_3}(\ldots K_{i_\alpha}(\phi_{i_\alpha})\ldots))))$.*

**R0.** (Initial state:) $Clk_i^\alpha = \alpha$ dimensional 0-vector

**R1.** (Internal event:) Before process $i$ executes the event, $Clk_i^\alpha[i, i, \ldots, i] = Clk_i^\alpha[i, i, \ldots, i] + d$ $(d > 0)$

**R2.** (Send event:) Before process $i$ executes the event, $Clk_i^\alpha[i, i, \ldots, i] = Clk_i^\alpha[i, i, \ldots, i] + d$ $(d > 0)$. Send message timestamped with $Clk_i^\alpha$.

**R3.** (Receive event:) When process $j$ receives a message with timestamp $T^\alpha$ from process $i$,

1. **for** $\beta = 1$ **to** $\alpha - 1$ **do**
$$\forall q_1 \in N \setminus \{j\}, \forall q_2, q_3, \ldots, q_\beta \in N,$$
$$Clk_j^\alpha[\underbrace{j, \ldots, j}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \ldots, q_\beta}_{\beta \text{ entries}}] =$$
$$max(Clk_j^\alpha[\underbrace{j, \ldots, j}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \ldots, q_\beta}_{\beta \text{ entries}}], T^\alpha[\underbrace{i, \ldots, i}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \ldots, q_\beta}_{\beta \text{ entries}}])$$
2. $\forall q_1 \in N \setminus \{j\}, \forall q_2, \ldots, q_\alpha \in N,$
$$Clk_j^\alpha[q_1, q_2, \ldots, q_\alpha] = max(Clk_j^\alpha[q_1, q_2, \ldots, q_\alpha], T^\alpha[q_1, q_2, \ldots, q_\alpha])$$
3. $Clk_j^\alpha[j, j, \ldots, j] = Clk_j^\alpha[j, j, \ldots, j] + d$ $(d > 0)$
4. Deliver the message.

**Fig. 1.** Protocol to operate $\alpha$-dimension clocks

With canonical clocks, $d = 1$ and $Clk^\alpha(e)[i_1, i_2, \ldots, i_\alpha] = |F_{i_\alpha}(\ldots \downarrow F_{i_3}(\downarrow F_{i_2}(\downarrow F_{i_1}(\downarrow e)))\ldots)|$. The value of $Clk^\alpha$ assigned as a timestamp is denoted $T^\alpha$. $T^\alpha[i]$, also represented as $T^\alpha[i, \cdot]$, is a timestamp of dimension $(\alpha - 1)$ and is derived from $T^\alpha$ by instantiating the first dimension variable $i_1$ by $i$. $T^\alpha(e_p)[i, \cdot]$ is the $(\alpha - 1)$ dimensional timestamp of the most recent event at process $i$, as known to process $p$ after event $e_p$. Moreover, this most recent event at process $i$ has a scalar timestamp $T^\alpha(e_p)[i, i, \ldots, i]$. In terms of knowledge, $T^\alpha(e_p^x)[i, \cdot]$ represents the knowledge $s_p^x \models K_p(K_i K_{i_2} K_{i_3} \ldots K_{i_\alpha}(\phi))$, where only $i_1$ is instantiated by $i$ in $K_p(K_{i_1} K_{i_2} K_{i_3} \ldots K_{i_\alpha}(\phi))$, for all $i_1, i_2, \ldots i_\alpha \in N$. Analogously, $T^\alpha[\underbrace{a, b, \ldots, f}_{\beta \text{ entries}, \beta \geq 0}, \cdot]$ is a timestamp of dimension $(\alpha - \beta)$. $T^\alpha(e_p^x)[a, b, \ldots, f, \cdot]$ represents the knowledge $s_p^x \models K_p(K_a K_b \ldots K_f K_{i_{\beta+1}} K_{i_{\beta+2}} \ldots K_{i_\alpha}(\phi))$, where the first $\beta$ dimension variables $i_1, \ldots, i_\beta$ are instantiated in $K_p(K_{i_1} K_{i_2} K_{i_3} \ldots K_{i_\alpha}(\phi))$, for all $i_1, i_2, \ldots i_\alpha \in N$. When $p = (a = b = \ldots = f)$, $T^\alpha(e_p^x)[a, b, \ldots, f, \cdot]$ is effectively a $(\alpha - \beta)$ dimensional timestamp of $e_p^x$.

**Theorem 2.** *The protocol in Fig. 1 implements the $\alpha$-dimensional clock specification of Definition 6.*

The protocol in Fig. 1 has a space and time complexity of $\Theta(n^\alpha)$. Rules (R3.1 and R3.2) can be simplified using simple observations, as shown in [12]. The size of each clock of dimension $\alpha$ is $n^\alpha$ integers. This clock/timestamp size may be reduced by using information such as the message pattern, logical network topology, and the partial order $(H, \prec)$, using analysis such as in [15,19,20], or by using approximations to the true clock, using schemes such as in [9,20].

The $\alpha$-dimensional timestamp of a cut is defined using the $(\alpha-1)$-dimensional timestamp of the latest event at each process in that cut.

**Definition 7.** $T^{\alpha}(Cut) =_{def} (i \in N)\ T^{\alpha}(Cut)[i,\cdot] = T^{\alpha}(F_i(Cut))[i,\cdot]$

**Lemma 3.** *The timestamp of cut* $\bigcap_{x \in X} \downarrow x$ *, denoted* $T^{\alpha}(\cap_{\Downarrow} X)$*, is expressed as a function of the timestamps of the members of* $X$ *as follows.* $(i \in N)T^{\alpha}(\cap_{\Downarrow} X)[i,\cdot]$ *is the* $(\alpha-1)$*-dimensional timestamp* $T^{\alpha}(x')[i,\cdot]$*, where* $T^{\alpha}(x')[i,i,\ldots,i] = min_{x \in X}(T^{\alpha}(x)[i,i,\ldots,i])$*.*

Lemma 3 gives a way to implement the test for Proposition 1. It will be used in Theorem 7 to identify the maximum computation prefix $\phi$ (cut) about which knowledge $E^k(\phi)$ has been attained at a given cut $Cut$.

Recall that by Proposition 3, the problem of identifying the minimum possible computation prefix (cut) $c$ such that $(a,c) \models E^k(\phi)$ for a given $\phi$ (Problem 2) is equivalent to the problem of identifying the minimum possible computation prefix $c$ such that $(a,c) \models E^k(Cut)$, where $Cut$ is the cut in which $\phi$ is true. Likewise, the problem of identifying the maximum possible fact $\phi$ such that $(a,c) \models E^k(\phi)$ at a given cut $c$ (Problems 1 and 3), is equivalent to the problem of identifying the maximum possible computation prefix $Cut$ such that $(a,c) \models E^k(Cut)$. We now give the main results linking clocks and knowledge.

## 5 Attaining Knowledge Using Clocks

At process $i$, $k$-bounded knowledge (of global facts about a property of interest) is of the form $K_i(K_{i_1}K_{i_2}K_{i_3}\ldots K_{i_k}(\phi))$. The number of unique permutations of the $K_{i_j}$ operators that represent $k$-bounded knowledge is computed as follows. $i_1 \neq i$, and $\forall j \in [2,k]$, $i_j \neq i_{j-1}$. Thus, $\forall j \in [1,k]$, $i_j$ can take one of $n-1$ values, giving $(n-1)^k$ permutations; each permutation denotes a global fact about which $k$-bounded knowledge exists at process $i$. Each global fact is represented by a cut and requires a vector ($n$ integers). Thus, the space for $k$-bounded knowledge at $i$ is $n \cdot (n-1)^k$ integers. The space requirement for all levels of knowledge upto $k$ at process $i$ is $n \cdot \sum_{j=0}^{k}(n-1)^j$ integers.

**Lemma 4.** *Representation of* $k$*-bounded knowledge (of global facts about a property of interest) needs* $n \cdot \sum_{j=0}^{k}(n-1)^j$ *integers.*

From the inequality $n^k <> n \cdot \sum_{j=0}^{k}(n-1)^j < n^{k+1}$, we now get Theorem 3.

**Theorem 3.** *$k$-bounded knowledge (of global facts about a property of interest) cannot be represented by a $k$-dimensional clock system, but can be represented by a $(k+1)$-dimensional clock system.*

By definition, $E^k(\phi) = \bigwedge_i K_i P_i(E^{k-1}(\phi))$, i.e., each process knows $E^{k-1}(\phi)$ along some (consistent) global state. To identify the bounds on space complexity to determine $E^k(\phi)$ knowledge for the latest possible $\phi$ in a system

(1) **Problem Inputs:**
(1a) **array of int** $T^1_\phi$; //vector timestamp of earliest state in which $\phi$ is true
(1b) **int** $k$; //level of knowledge $E^k(\phi)$ to be attained
(2) **Problem Output:**
(2a) **array of int** $TS^1 = Compute\_State(T^1_\phi, k)$.
(2b) //vector timestamp of earliest state in which $E^k(\phi)$ is attained

(3) *function Compute_State*(**array of int** $T^1_\phi$; **int** $k$) **returns** $TS^1$
(4) **for** $lvl = 1$ **to** $k+1$ **do**
(5) $\forall p \in N$ **do**
(6) identify earliest event $e_p \mid T^1(e_p) \geq TS^1$;
(7) $T'^1[p] = T^1(e_p)[p]$;
(8) $\forall p \in N$ **do**
(9) $TS^1[p] = max(T^1(e_1)[p], T^1(e_2)[p], \ldots, T^1(e_n)[p])$;
(10) // $(a, TS^1) \models E^{lvl}(T^1_\phi) \bigwedge \nexists\, TS'^1 \mid (TS'^1 < TS^1 \wedge (a, TS'^1) \models E^{lvl}(T^1_\phi))$
(11) **return**$(TS^1)$.

**Fig. 2.** Given $\phi$, protocol to compute earliest system state in which $E^k(\phi)$ is achievable

following the KIPP protocol (Problem 1), it can be shown that $\forall i_1, i_2, \ldots i_k$, $K_i(K_{i_1}K_{i_2}\ldots, K_{i_k}(\psi_{i_1,i_2,\ldots,i_k}))$ must be available at each process $i$, where $\psi_{i_1,i_2,\ldots,i_k}$ is the max. execution prefix about which the corresponding knowledge is available, i.e., "$i$ knows $i_1$ knows $i_2$ knows $\ldots$ $i_k$ knows $\psi_{i_1,i_2,\ldots,i_k}$". The max. execution prefix $\phi$ about which $E^k(\phi)$ is attained is given by $\bigcap_{i_1,i_2,\ldots,i_k \in N} \psi_{i_1,i_2,\ldots,i_k}$.

**Theorem 4.** *In a system following the KIPP protocol, $k$-bounded knowledge at each process is required to attain and declare $E^k(\phi)$, where $\phi$ is the maximum possible monotonic fact (most recent possible system state) about which $E^k(\phi)$ is possible to be declared in the current state.*

Theorem 5 (= Thms. 3 + 4) and Theorem 6 answer Problems 1 and 2, resp..

**Theorem 5.** *In a system following the KIPP protocol, a $(k+1)$-dim clock system is sufficient but a $k$-dim clock system is not sufficient to attain and declare $E^k(\phi)$, where $\phi$ is the maximum possible monotonic fact (most recent possible system state) about which $E^k(\phi)$ is possible to be declared in the current state.*

**Theorem 6.** *Given a global monotonic fact $\phi$, the earliest global state in which $E^k(\phi)$ is attained in a system following the KIPP protocol is given by the protocol in Fig. 2.*

Given the earliest cut where $\phi$ becomes true, specified by $T^1_\phi$, (which by Proposition 3 captures fact $\phi$), Fig. 2 gives a protocol to determine the earliest global state at which $E^k(\phi)$ can be attained. The protocol is iterative. Function *Compute_State* uses two inputs: (i) $T^1_\phi$, the vector timestamp of the earliest

state in which $\phi$ holds, and (ii) $k$, the level of knowledge $E^k(\phi)$ to be attained. The output is $TS^1$, the vector timestamp of the earliest state in which assertion $E^k(\phi)$ can be made. The protocol is proved correct by showing that the invariant in line (10) holds after each iteration. Note that in each iteration, $T'^1$ (line (7)) identifies a global state that may not be consistent; hence a consistent global state $TS^1$ (line (9)) is computed.

**Complexity:** Time complexity is (# send and receive events in $(H, \prec)$ after the cut at which $\phi$ is defined). Space complexity is that of a vector clock system, and also requires each process to store a trace of the timestamps of its send and receive events beyond the cut at which $\phi$ is defined.

To answer Problem 3, "Given a timestamp $T^{\beta+1}$ of a state, what is the maximum $\phi$ such that $(a, T^{\beta+1}) \models E^{\beta}(\phi)$?" we can apply the function *min* to the $n^{\beta}$ 1-dimensional timestamps of size $n$ in the given $T^{\beta+1}$. This requires $n \cdot n^{\beta}$ comparisons. Theorem 7 gives a solution of $\Theta(\beta \cdot (n^2 + n))$ time complexity.

**Theorem 7.** *Given a timestamp $T^{\beta}$, the maximum possible monotonic fact $\phi$ (most recent possible system state) about which $E^k(\phi)$, where $k \leq \beta - 1$, can be declared at the given state $T^{\beta}$ in a system following the KIPP protocol is given by the protocol in Fig. 3.*

The proof is by construction. Fig. 3 gives a protocol to derive the max. computation prefix $\phi$ about which the processes have knowledge $E^k(\phi)$, given the timestamp $Ob\_T^{\beta}$, where $\beta > k$. *Compute_Phi* has inputs (i) $T^{\alpha}$, the (variable dim.) timestamp of the maximum cut about which knowledge $E^{atn}$ is attained in $Ob\_T^{\beta}$, (ii) $m$, the level of knowledge that is yet to be attained, and (iii) $atn$, the level of knowledge already attained. The output is the timestamp $\phi$ of the max. cut about which $E^k$ knowledge is attained in the given state $Ob\_T^{\beta}$.

*Compute_Phi* is invoked as $Compute\_Phi(Ob\_T^{\beta}, k, 0)$ and is tail-recursive. $T^{\alpha}$ is progressively decreased at each recursion level to add another level of knowledge to what is known of $T^{\alpha}$ at cut $Ob\_T^{\beta}$. So at each additional recursion level, $T^{\alpha}$ therein converges towards $\phi$. Each recursion level behaves as follows.

- Given $T^{\alpha}(Cut)$, the loop in lines (5)-(6) computes the $(\alpha - 1)$-dimensional timestamp of $F_p(Cut)$ which is the latest event of the cut $Cut$ at process $p$, $(p \in N)$. $T^{(\alpha-1)}(F_p(Cut))$ is simply $T^{\alpha}[p, \cdot]$.
- Let $X$ denote the events $F(Cut)$ identified in line (6). The loop in lines (7)-(9) applies Lemma 3 to $X$ to compute $\cap_{\Downarrow} X$. By doing so, it identifies the timestamps $T^{(\alpha-2)}(F_p(\cap_{\Downarrow} X))$ for each process $p$. Then $T^{(\alpha-1)}(\cap_{\Downarrow} X)$ is simply the aggregation of the $n$ timestamps $T^{(\alpha-2)}(F_p(\cap_{\Downarrow} X))$, as shown in line (10). By Proposition 1, $T^{(\alpha-1)}$ is the timestamp of the maximal prefix about which all the processes have knowledge at $X = F(Cut)$ and this can be asserted only at or after $F(Cut)$. Thus, $E(T^{(\alpha-1)})$ holds in the state with timestamp $T^{\alpha}$ in this recursion level and we assert the invariant on line (11).
- The above steps also add a level of knowledge to that at the given initial state $Ob\_T^{\beta}$; we assert this in the invariant on line (13). If this is the desired level of knowledge, then we have the terminating case for the recursion and the value of $T^{(\alpha-1)}$ is returned (lines (14)-(16)), else *Compute_Phi* is recursively

(1) **Problem Inputs:**
(1a) $\beta$-**dim. array of int** $Ob\_T^{\beta}$; // timestamp of observation state
(1b) **int** $k$, where $\beta > k \geq 1$; // level of knowledge to be attained
(2) **Problem Output:**
(2a) $(\beta - k)$ **dim. array of int** $\phi = Compute\_Phi(Ob\_T^{\beta}, k, 0)$.
(2b) //timestamp of maximum possible state such that $(a, Ob\_T^{\beta}) \models E^k(\phi)$

(3) *function* $Compute\_Phi$(**var dim. array of int** $T^{\alpha}$; **int** $m$, $atn$) **returns** $\phi$
(4a) // $T^{\alpha}$ is timestamp of the max. possible state such that $(a, Ob\_T^{\beta}) \models E^{atn}(T^{\alpha})$
(4b) // $m$ is the level of knowledge yet to be attained
(4c) // $atn$ is the level of knowledge already attained. $atn = k - m$.
(5) $\forall p \in N$ **do**
(6) $T_p^{\alpha-1} = T^{\alpha}[p, \cdot]$;
(7) $\forall p \in N$ **do**
(8) let $r$ be such that $T_r^{\alpha-1}[p, p, \ldots, p] = min_{q \in N}(T_q^{\alpha-1}[p, p, \ldots, p])$;
(9) $T_p^{\alpha-2} = T_r^{\alpha-1}[p, \cdot]$;
(10) $T^{\alpha-1} = [T_1^{\alpha-2}, T_2^{\alpha-2}, \ldots, T_n^{\alpha-2}]$;
(11) // $(a, T^{\alpha}) \models E^1(T^{\alpha-1}) \bigwedge \nexists\, T'^{\alpha-1} \mid (T'^{\alpha-1} > T^{\alpha-1} \wedge (a, T^{\alpha}) \models E^1(T'^{\alpha-1}))$
(12) $atn = atn + 1$; $m = m - 1$;
(13)// $(a, Ob\_T^{\beta}) \models E^{atn}(T^{\alpha-1}) \bigwedge \nexists\, T'^{\alpha-1} \mid (T'^{\alpha-1} > T^{\alpha-1} \wedge (a, Ob\_T^{\beta}) \models E^{atn}(T'^{\alpha-1}))$
(14) **if** $m = 0$ **then**
(15) $\phi = T^{\alpha-1}$;
(16) **return**$(\phi)$;
(17) **else**
(18) $\phi = Compute\_Phi(T^{\alpha-1}, m, atn)$;
(19) // $(a, T^{\alpha-1}) \models E^m(\phi) \bigwedge \nexists\, \phi' \mid (\phi' > \phi \wedge (a, T^{\alpha-1}) \models E^m(\phi'))$
(20) // $(a, T^{\alpha}) \models E^{m+1}(\phi) \bigwedge \nexists\, \phi' \mid (\phi' > \phi \wedge (a, T^{\alpha}) \models E^{m+1}(\phi'))$
(21) // $(a, Ob\_T^{\beta}) \models E^{atn+m}(\phi) \bigwedge \nexists\, \phi' \mid (\phi' > \phi \wedge (a, Ob\_T^{\beta}) \models E^{atn+m}(\phi'))$
(22) **return**$(\phi)$.

**Fig. 3.** Protocol to compute latest $\phi$ for which $E^k(\phi)$ holds in a state with timestamp $T^{\beta}$, where $\beta > k$

invoked to determine the greatest $\phi$ that is known at $T^{(\alpha-1)}$ for the remaining $m$ levels of knowledge to be attained (lines (17)-(18)).

The invariants on lines (11,13,19,20,21) are seen to hold. Hence, $\phi$ is the max prefix such that $(a, Ob\_T^{\beta}) \models E^k(\phi)$, derived from the recursive use of Lemma 3.
**Complexity:** The time complexity is $\Theta(k \cdot (n^2 + n))$. The space complexity is that of $\beta$-dimensional clocks, which is $\Theta(n^{\beta})$ integers and meets the tight bound established by Theorem 5. The time complexity is less than the space complexity because information is selectively accessed dynamically.
*Necessary and sufficient conditions required to declare $E^k(\phi)$ using the KIPP:* Lemma 4 and Theorem 4 together give the conditions on the exact size of clocks, whereas Theorem 5 gave the conditions on the dimension of clocks.

## 6 Concluding Remarks

So far, concurrent knowledge has been studied much less than normal knowledge although asynchronous systems are much more prevalent than synchronous ones. This paper made significant contributions to the theory of concurrent common knowledge and proposed logical clock systems of arbitrary dimensions. Specifically, it made the following contributions. (i) It motivated and proposed logical clocks of arbitrary dimensions, and also formalized the KIPP protocol for knowledge transfer used by such clock systems. (ii) It showed that there exists a tight relation between the dimension of logical clocks and the level of concurrent knowledge attainable, and established some complexity bounds. Here it identified and explored an important conceptual link. (iii) It proposed algorithms to compute the latest global fact about which a specified level of knowledge is attainable in a given state, and to compute the earliest state in which a specified level of knowledge about a given global fact is attainable.

**Acknowledgements:** This work was supported by the U.S. National Science Foundation grant CCR-9875617.

## References

1. M. Chandy, L. Lamport, Finding global states of a distributed system, *ACM Transactions on Computer Systems*, 3(1): 63-75, 1985.
2. M. Chandy, J. Misra, How processes learn, *Distributed Computing*, 1: 40-52, 1986.
3. B. Charron-Bost, Concerning the size of clocks in distributed systems, *Information Processing Letters*, 39: 11-16, 1991.
4. C. Critchlow, K. Taylor, The inhibition spectrum and the achievement of causal consistency, *Distributed Computing*, 10(1): 11-27, 1996.
5. R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about Knowledge, MIT Press, 1995.
6. C. Fidge, Timestamps in message-passing systems that preserve partial ordering, *Australian Computer Science Communications*, 10(1): 56-66, Feb. 1988.
7. J. Halpern, R. Fagin, Modeling knowledge and action in distributed systems, *Distributed Computing*, 3(4): 139-179, 1989.
8. J. Halpern, Y. Moses, Knowledge and common knowledge in a distributed environment, *Journal of the ACM*, 37(3): 549-587, 1990.
9. N. Krishnakumar, A. Bernstein, Bounded ignorance in replicated systems, *Proc. ACM Symp. on Principles of Database Systems*, 1991.
10. A. Kshemkalyani, Temporal interactions of intervals in distributed systems, *Journal of Computer and System Sciences*, 52(2): 287-298, Apr. 1996.
11. A. Kshemkalyani, Causality and atomicity in distributed computations, *Distributed Computing*, 11(4): 169-189, Oct. 1998.
12. A. Kshemkalyani, On continuously attaining levels of concurrent knowledge without control messages, Tech. Rep. UIC-EECS-98-6, Univ. Illinois at Chicago, 1998.
13. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, 21(7): 558-565, July 1978.
14. F. Mattern, Virtual time and global states of distributed systems, *Parallel and Distributed Algorithms*, North-Holland, pp 215-226, 1989.

15. S. Meldal, S. Sankar, J. Vera, Exploiting locality in maintaining potential causality, *Proc. 10th ACM Symp. on Principles of Distributed Computing*, 231-239, 1991.
16. P. Panangaden, K. Taylor, Concurrent common knowledge: Defining agreement for asynchronous systems, *Distributed Computing*, 6(2): 73-94, Sept. 1992.
17. R. Parikh, P. Krasucki, Levels of knowledge in distributed computing, *Sadhana Journal*, 17(1): 167-191, 1992.
18. S. Sarin, N. Lynch, Discarding obsolete information in a distributed database system, *IEEE Transactions on Software Engineering*, 13(1): 39-46, 1987.
19. M. Singhal, A. Kshemkalyani, Efficient implementation of vector clocks, *Information Processing Letters*, 43, 47-52, Aug. 1992.
20. G. Wuu, A. Bernstein, Efficient solutions to the replicated log and dictionary problems, *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, 232-242, 1984.