# Monitoring Usage-control Policies in Distributed Systems

David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu
*ETH Zurich, Computer Science Department, Switzerland*

***Abstract*—We have previously presented a monitoring algorithm for compliance checking of policies formalized in an expressive metric first-order temporal logic. We explain here the steps required to go from the original algorithm to a working infrastructure capable of monitoring an existing distributed application producing millions of log entries per day. The main challenge is to correctly and efficiently monitor the trace interleavings obtained by totally ordering actions that happen at the same time. We provide solutions based on formula transformations and monitoring representative traces. We also report, for the first time, on statistics on the performance of our monitor on real-world data, providing evidence of its suitability for nontrivial applications.**

## I. Introduction

Determining whether the usage of sensitive data complies with regulations and policies is a growing concern for companies, administrations, and end users alike. In the context of IT systems, this question amounts to whether one can implement processes that monitor other processes. In previous work [1], [2], we have demonstrated that metric first-order temporal logic (MFOTL) is a good candidate for monitoring data usage to determine policy compliance. In particular, the metric temporal operators allow one to formalize both qualitative and quantitative temporal relationships between actions and, as the logic is first-order, we can also formulate dependencies between the finite but unbounded number of agents and data elements in IT systems. We have given a monitoring algorithm for MFOTL [1] and many usage-control policies can be naturally formulated in the fragment that the monitor handles efficiently [2].

In this paper, we extend our previous work by deploying and evaluating our monitoring approach in a real-world concurrent and distributed setting. This is in contrast to our previous analysis [2], which we carried out in a non-distributed setting where we used log files filled with synthetically generated actions. In the following, we describe our monitoring setup and the challenges we faced. We begin with an abstract description of the systems that we handle.

**System Model.** The types of entities in our systems are *data*, *(data) stores*, *agents*, and *actions*. Data is stored in distributed data stores such as databases and repositories and created, read, modified, propagated, combined, and deleted by actions initiated by agents. Agents are either humans or applications, including database triggers.

Agents always access the data directly from a store and never indirectly from another agent. Whenever an agent wants to use some data, it accesses the appropriate store, uses the data, and discards it afterwards. For subsequent usage, it must access the store again. Before discarding the data, the agent may write it, possibly after processing it in some way, into the same or a different store. In this way, data can propagate between stores. A consequence of this restriction on the interaction between system entities is that the use of data is always observable at the data stores.

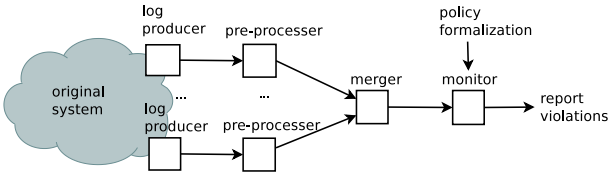


Figure 1. System Extension

Systems are governed by *(usage-control) policies*, which state requirements on the usage of the data. For example, only agents with particular credentials may modify data, or data must be deleted after two years from a given store. Agents may or may not comply with policies.

**Logging and Monitoring.** Given a system that is an instance of the above system model, we must extend it to support logging and monitoring. To determine whether a policy is violated we usually need to relate actions that are carried out in different parts of the system. Moreover, the ordering of actions and the time elapsed between them is important. To relate actions and the times when they happen, we log them locally, annotating each action with a timestamp, and merge these logs after some pre-processing. We then monitor this merged stream of logged actions. These system extensions are depicted in Figure 1.

**Challenges and Contributions.** Individual logs are totally ordered and timestamped using local clocks. However, even assuming clock synchronization [3], we have only a partial order on system actions [4] as multiple actions with the same timestamp may occur in different logs. Our main theoretical challenge is to monitor such a partially ordered set of actions, which is, in general, an intractable problem. In Section III, we identify a subclass of formulas that describe properties that are insensitive to the ordering of actions labeled by the same timestamp and for which it suffices to monitor a particular merging of the logs, namely, the merging that assumes that actions with equal timestamps happen simultaneously. Furthermore, in case the given formula is outside this class we provide means to meaningfully monitor this merge by approximating the described property.

A practical challenge is to deploy adequate logging mechanisms. The mechanisms should be complete in that they log all occurrences of policy-relevant system actions. They should also be accurate in that if an action is logged then it has happened in the system and the corresponding log entry accurately describes the action, e.g, it describes the involved data and the associated timestamp is the actual time when the action happened. Incomplete or inaccurate logging may lead to false positives and false negatives when monitoring the system.

In Section IV, we explain how we handle these practical challenges in our case study. Where possible, we use existing logging mechanisms and extract policy-relevant information from the produced log entries. For system components where no logging was available, we either added logging directly to the components or we extended the components with proxy mechanisms that logged actions. However, proxies have limitations: agents do not necessarily access a store over a proxy and proxies see requested actions but not necessarily all the effects on the involved data. In our case, the interactions could be accurately observed but not for all agents, which led to accurate but incomplete logs.

Summarizing, we see our contributions as follows. We provide solutions for efficiently monitoring partially ordered logs, which is a central problem in monitoring real-time concurrent distributed systems. Moreover, we evaluate the performance of our monitoring approach and demonstrate its effectiveness on a real-world application.

**Organization.** The remainder of this paper is structured as follows. In Section II, we give background on MFOTL and our monitor. In Section III, we show how we handle the interleavings of multiple streams of logged actions from different log producers. In Section IV, we report on our case study. In Section V, we discuss related work and in Section VI, we draw conclusions. Due to space limitations, some details have been omitted. They can be found in the full version of the paper, which is publicly available from the authors' web pages.

## II. Preliminaries

We briefly review metric first-order temporal logic (MFOTL) and describe how we use it to monitor systems.

**Syntax and Semantics.** Let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We will write an interval $I \in \mathbb{I}$ as $[b, b') := \{a \in \mathbb{N} \mid b \leq a < b'\}$, where $b \in \mathbb{N}$, $b' \in \mathbb{N} \cup \{\infty\}$, and $b < b'$. A *signature* $S$ is a tuple $(C, R, \iota)$, where $C$ is a finite set of constant symbols, $R$ is a finite set of predicates disjoint from $C$, and the function $\iota : R \to \mathbb{N}$ associates each predicate $r \in R$ with an arity $\iota(r) \in \mathbb{N}$. In the following, let $S = (C, R, \iota)$ be a signature and $V$ a countably infinite set of variables, assuming $V \cap (C \cup R) = \emptyset$.

*Formulas* over the signature $S$ are given by the grammar

$$\phi ::= t_1 \approx t_2 \mid t_1 \prec t_2 \mid r(t_1, \ldots, t_{\iota(r)}) \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\exists x.\, \phi) \mid (\bullet_I\, \phi) \mid (\bigcirc_I\, \phi) \mid (\phi\ \mathsf{S}_I\ \phi) \mid (\phi\ \mathsf{U}_I\ \phi),$$

$$\begin{array}{lll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t' & \text{iff} & v(t) = v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \prec t' & \text{iff} & v(t) < v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)}) & \text{iff} & (v(t_1), \ldots, v(t_{\iota(r)})) \in r^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\neg\phi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi \vee \psi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\exists x.\, \phi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v[x/d], i) \models \phi, \text{for some } d \in |\bar{\mathcal{D}}| \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bullet_I\, \phi) & \text{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bigcirc_I\, \phi) & \text{iff} & \tau_{i+1} - \tau_i \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi\ \mathsf{S}_I\ \psi) & \text{iff} & \text{for some } j \leq i,\ \tau_i - \tau_j \in I, (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi, \text{for all } k \in [j+1, i+1) \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi\ \mathsf{U}_I\ \psi) & \text{iff} & \text{for some } j \geq i,\ \tau_j - \tau_i \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi,\ \text{for all } k \in [i, j)
\end{array}$$

Figure 2. Semantics of MFOTL

where $t_1, t_2, \ldots$ range over the elements in $V \cup C$, and $r$, $x$, and $I$ range over the elements in $R$, $V$, and $\mathbb{I}$, respectively.

To define MFOTL's semantics, we need the following notions. A *structure* $\mathcal{D}$ over $S$ consists of a domain $|\mathcal{D}| \neq \emptyset$ and interpretations $c^{\mathcal{D}} \in |\mathcal{D}|$ and $r^{\mathcal{D}} \subseteq |\mathcal{D}|^{\iota(r)}$, for each $c \in C$ and $r \in R$. A *temporal structure* over $S$ is a pair $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \ldots)$ is a sequence of structures over $S$ and $\bar{\tau} = (\tau_0, \tau_1, \ldots)$ is a sequence of natural numbers (i.e., timestamps), where:

(1) The sequence $\bar{\tau}$ is monotonically increasing (i.e., $\tau_i \leq \tau_{i+1}$, for all $i \geq 0$) and makes progress (i.e., for every $i \geq 0$, there is some $j > i$ such that $\tau_j > \tau_i$).
(2) $\bar{\mathcal{D}}$ has constant domains, i.e., $|\mathcal{D}_i| = |\mathcal{D}_{i+1}|$, for all $i \geq 0$. We denote the domain by $|\bar{\mathcal{D}}|$ and require that $|\bar{\mathcal{D}}|$ is strict linearly ordered by a relation $<$.
(3) Each constant symbol $c \in C$ has a rigid interpretation, i.e., $c^{\mathcal{D}_i} = c^{\mathcal{D}_{i+1}}$, for all $i \geq 0$. We denote $c$'s interpretation by $c^{\bar{\mathcal{D}}}$.

A *valuation* is a mapping $v : V \to |\bar{\mathcal{D}}|$. We abuse notation by applying a valuation $v$ also to constant symbols $c \in C$, with $v(c) = c^{\bar{\mathcal{D}}}$. For a valuation $v$, a variable $x$, and $d \in |\bar{\mathcal{D}}|$, $v[x/d]$ is the valuation mapping $x$ to $d$ and not altering the other variables' valuation.

The semantics of MFOTL, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$, is given in Figure 2, where $(\bar{\mathcal{D}}, \bar{\tau})$ is a temporal structure over the signature $S$, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \ldots)$, $\bar{\tau} = (\tau_0, \tau_1, \ldots)$, $v$ a valuation, $i \in \mathbb{N}$, and $\phi$ a formula over $S$. Note that the temporal operators are labeled with intervals $I$ and a formula of the form $(\bullet_I\, \phi)$, $(\bigcirc_I\, \phi)$, $(\phi \mathsf{S}_I \psi)$, or $(\phi \mathsf{U}_I \psi)$ is only satisfied in $(\bar{\mathcal{D}}, \bar{\tau})$ at the time point $i$, if it is satisfied within the bounds given by the interval $I$ of the respective temporal operator, which are relative to the current timestamp $\tau_i$.

**Terminology and Notation.** We use standard syntactic sugar such as $\blacksquare_I\, \phi := \neg(\mathit{true} \mathsf{S}_I \neg\phi)$ and $\square_I\, \phi := \neg(\mathit{true} \mathsf{U}_I \neg\phi)$, where $\mathit{true} := \exists x.\, x \approx x$. We also use non-metric operators like $\square\, \phi := \square_{[0,\infty)}\, \phi$. We omit parentheses where possible, e.g., unary operators (temporal and Boolean) bind stronger than binary ones. A formula $\phi$ is *bounded* if the interval $I$ of every temporal operator $\mathsf{U}_I$ occurring in $\phi$ is finite. We use standard terminology like *atomic formula* and *subformula*.

**System Monitoring.** We illustrate our use of MFOTL and our monitoring algorithm [1] for compliance checking by the simple policy stating that reports must be approved

within at most 10 time units before they are published:

$$\Box \forall x.\, publish(x) \rightarrow \blacklozenge_{[0,11)}\, approve(x)\,.$$

We assume that the actions for publishing and approving reports are logged in relations. Specifically, for each time point $i \in \mathbb{N}$, we have the unary relations $PUBLISH_i$ and $APPROVE_i$ such that (1) $x \in PUBLISH_i$ iff report $f$ is published at time point $i$ and (2) $x \in APPROVE_i$ iff report $x$ is approved at time point $i$. Observe that there can be multiple approvals at the same time point for different reports. Furthermore, every time point $i$ has a timestamp $\tau_i \in \mathbb{N}$.

The corresponding temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ of a sequence of logged publishing and approval actions is as follows. The only relational symbols in $\bar{\mathcal{D}}$'s signature are *publish* and *approve*, both of arity 1. The domain of $\bar{\mathcal{D}}$ consists of all reports. The $i$th structure in $\bar{\mathcal{D}}$ is timestamped with $\tau_i$ and contains the relations $PUBLISH_i$ and $APPROVE_i$.

To detect policy violations, our monitor [1] iteratively processes the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ representing the stream of logged actions. This can be done offline or online. At each time point $i$, it outputs the valuations satisfying the negation of the formula $publish(x) \rightarrow \blacklozenge_{[0,11)}\, approve(x)$. Note that we drop the outermost quantifier since we are not only interested in whether the policy is violated but also which data is responsible for the reported violations.

In general, we assume that policies formalized in MFOTL are of the form $\Box\psi$, where $\psi$ is bounded. Since $\psi$ is bounded, the monitor need only take into account a finite prefix of $(\bar{\mathcal{D}}, \bar{\tau})$ when determining the satisfying valuations of $\neg\psi$ at a time point $i$. To effectively determine all these valuations, we also assume here that predicates have finite interpretations in $(\bar{\mathcal{D}}, \bar{\tau})$, i.e., the relation $r^{\mathcal{D}_j}$ is finite, for every predicate $r$ and every $j \in \mathbb{N}$. Furthermore, we require that $\neg\psi$ can be rewritten to a temporal-subformula-domain-independent formula, a generalization of the standard notion of domain-independent database queries [5].

## III. Monitoring Concurrently Logged Actions

In this section, we first prove the intractability of monitoring where logs are produced in a concurrent setting. We then show how to partially overcome this obstacle by monitoring a single log where all actions with equal timestamps are assumed to have happened at the same point in time.

**Log Interleavings.** Intuitively, an interleaving of logs preserves the ordering of the logged actions with respect to their timestamps, but allows for all possible orderings of actions with equal timestamps that are recorded by different log producers. To define this, let $\mathrm{img}(f)$ denote the set $\{y \in Y \mid f(x) = y, \text{ for some } x \in X\}$, for a function $f : X \rightarrow Y$. Furthermore, we assume in this section that all temporal structures have the same signature $(C, R, \iota)$, equal domains, and that constant symbols are equally interpreted. Note that any two temporal structures in which the common constant symbols are equally interpreted can easily be extended so that their extensions fulfill this requirement.

**Definition 1.** *Let $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$, $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$, and $(\bar{\mathcal{D}}, \bar{\tau})$ be temporal structures. $(\bar{\mathcal{D}}, \bar{\tau})$ is an* interleaving *of $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ if there are strictly monotonic functions $f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N}$ with*
*(1)* $\mathrm{img}(f_1) \cup \mathrm{img}(f_2) = \mathbb{N}$,
*(2)* $\mathrm{img}(f_1) \cap \mathrm{img}(f_2) = \emptyset$, *and*
*(3)* $\tau_i^k = \tau_{f_k(i)}$ *and* $r^{\mathcal{D}_i^k} = r^{\mathcal{D}_{f_k(i)}}$, *for all* $k \in \{1,2\}$, $i \in \mathbb{N}$, $r \in R$.
*We denote by $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ the set of all interleavings of the temporal structures $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$.*

Since there are usually multiple interleavings of two temporal structures, we formulate policy violations in terms of a set of temporal structures.

**Definition 2.** *Let* $\mathbf{T}$ *be a set of temporal structures.*
*(1)* $\mathbf{T}$ weakly violates *the formula $\phi$ at time point $i \in \mathbb{N}$ if for some $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ and some valuation $v$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$.*
*(2)* $\mathbf{T}$ strongly violates *the formula $\phi$ at time point $i \in \mathbb{N}$ if for all $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$, there is some valuation $v$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$.*

Unfortunately, even in a propositional setting, determining whether the set of interleavings weakly or strongly violates a formula is intractable.

**Theorem 3.** *Let $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ be temporal structures, $i \in \mathbb{N}$, and $\phi$ a quantifier-free sentence with only Boolean and non-metric past operators that neither contains the equality symbol $\approx$ nor the ordering symbol $\prec$.*
*1. Determining whether the set of interleavings $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ weakly violates $\phi$ at $i$ is NP-complete.*
*2. Determining whether the set of interleavings $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ strongly violates $\phi$ at $i$ is coNP-complete.*

Note that both decision problems are well defined as $\phi$ does not contain future operators. We therefore only need to examine the finite prefixes with length $i + 1$ of the interleavings to determine whether $\phi$ is weakly or strongly violated at the given time point $i$.

**Collapsing Interleaved Logs.** We first give conditions with respect to an arbitrary set of temporal structures for when it suffices to monitor a single temporal structure. We then identify a natural temporal structure for the set of interleavings of two temporal structures, which we use for monitoring.

**Definition 4.** *The temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ is* sufficient *for the formula $\phi$ on the set $\mathbf{T}$ of temporal structures if for all valuations $v$, the following conditions are fulfilled:*
*(C1) If $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \models \phi$ then $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$, for all $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$.*
*(C2) If $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \not\models \phi$ then $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$, for all $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$.*

In the following, the set $\mathbf{T}$ in the above definition will be the set of interleavings of two temporal structures. For the temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$, we will use the so-called collapse:

**Definition 5.** *Let* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(\bar{\mathcal{C}}, \bar{\kappa})$ *be temporal structures.* $(\bar{\mathcal{C}}, \bar{\kappa})$ *is a* collapse *of* $(\bar{\mathcal{D}}, \bar{\tau})$ *if there is a monotonic surjective function* $f : \mathbb{N} \to \mathbb{N}$ *such that*
*(1) if* $\tau_i = \tau_j$ *then* $f(i) = f(j)$*, for all* $i, j \in \mathbb{N}$*,*
*(2)* $\kappa_{f(i)} = \tau_i$*, for all* $i \in \mathbb{N}$*, and*
*(3)* $r^{\mathcal{C}_j} = \bigcup_{i \in f^{-1}(j)} r^{\mathcal{D}_i}$*, for all* $j \in \mathbb{N}$ *and* $r \in R$*.*

Intuitively, the structures of the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ with equal timestamps are collapsed into a single structure. The collapse is uniquely defined and we denote it by $col(\bar{\mathcal{D}}, \bar{\tau})$. Furthermore, the collapses of temporal structures in the set of interleavings of two given temporal structures are all isomorphic.

Before we identify formulas for which the collapse of an interleaving of given temporal structures can be correctly used for monitoring, we give practical reasons that justify its use for monitoring. First, observe that the collapse can be incrementally obtained from an arbitrary interleaving of two given temporal structures. Hence, monitoring the collapse can be done efficiently. Second, note that the actual ordering of actions logged with equal timestamps in a concurrent system cannot be known. Hence, it does not make sense to consider just one arbitrary interleaving. Assuming that equally timestamped actions have happened at the same point in time naturally "hides" the differences between interleavings. Moreover, reasonable policies for a concurrent system should not care about the ordering of equally timestamped actions in case of accurate and precise clocks. In other words, if the collapsed temporal structure is not sufficient for the policy on the set of interleavings, then the policy might not be the intended one for the system. Finally, monitoring the collapsed temporal structure is practically more efficient than monitoring an interleaving. This is because the monitor is invoked less often since time points with equal timestamps are merged to a single one. Hence, the monitor processes the logged actions with equal timestamp in a single invocation.

**Monitoring the Collapse.** Intuitively, collapse-sufficient formulas are formulas that do not yield false positives and false negatives when monitoring the collapse of an interleaving:

**Definition 6.** *Let* $\phi$ *be a formula. For* $k \in \{1, 2\}$*, we say that* $\phi$ *has the property (Ck) if* $(\bar{\mathcal{C}}, \bar{\kappa})$ *fulfills the condition (Ck) in Definition 4 with respect to* $\phi$ *and* $(\bar{\mathcal{D}}, \bar{\tau}) \between (\bar{\mathcal{D}}', \bar{\tau}')$*, for every* $(\bar{\mathcal{D}}, \bar{\tau})$*,* $(\bar{\mathcal{D}}', \bar{\tau}')$*, and* $(\bar{\mathcal{C}}, \bar{\kappa})$*, where* $(\bar{\mathcal{C}}, \bar{\kappa})$ *is the collapse of an interleaving of* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(\bar{\mathcal{D}}', \bar{\tau}')$*. Moreover,* $\phi$ *is* collapse-sufficient *if it has the properties (C1) and (C2).*

Monitoring the collapse of a collapse-sufficient formula is correct with respect to strong violations. Since the formula has property (C2), violations found in $(\bar{\mathcal{C}}, \bar{\kappa})$ imply that the set of interleavings strongly violates the formula. The converse is ensured by the property (C1): if no violation is found in $(\bar{\mathcal{C}}, \bar{\kappa})$ then all interleavings are policy compliant. Furthermore, by monitoring $(\bar{\mathcal{C}}, \bar{\kappa})$ we also detect when the set of interleavings weakly violates the given formula. The reason is that if a formula is strongly violated by the set of interleavings then it also weakly violated, since the set of interleavings is always nonempty.

**Example 7.** *The formula* $\Box \forall x.publish(x) \to \blacklozenge_{[0,11)} approve(x)$ *is not collapse-sufficient. Suppose that a report* $x$ *is published in* $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ *at time point* $i$*, i.e.,* $x \in publish^{\mathcal{D}^1_i}$ *and only approved in* $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *at the equally timestamped time point* $j$*, i.e.,* $x \in approve^{\mathcal{D}^2_j}$ *with* $\tau^2_j = \tau^1_i$*. Then there is an interleaving* $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *where the approval action comes (pointwise) strictly after the publish action. As a result, we cannot handle this formula correctly by monitoring the collapsed temporal structure* $(\bar{\mathcal{C}}, \bar{\kappa})$ *of an interleaving of the given temporal structures* $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ *and* $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$*.*

*A slightly stronger policy can be efficiently monitored. Namely, the policy that requires that an approval action must happen timewise strictly before the publish action, i.e.,* $\Box \forall x.\, publish(x) \to \blacklozenge_{[1,11)} approve(x)$*. This formula is collapse-sufficient. Similarly,* $\Box \forall x.\, publish(x) \to \Diamond_{[0,1)} \blacklozenge_{[0,11)} approve(x)$ *is also collapse-sufficient. It formalizes the slightly weaker policy where publish actions must be timewise but not pointwise previously approved.*

Note that stutter-invariance [6] is a necessary condition for collapse-sufficiency. However, it is not a sufficient condition. For example, the formula $\Box \forall x.\, p(x) \wedge q(x)$ is stuttering-invariant but not collapse-sufficient.

**A Collapse-sufficient Fragment.** In the following, we present a fragment of collapse-sufficient formulas. Our fragment is defined in terms of an algorithm that identifies formulas that have property (C1) or property (C2).

The algorithm labels the atomic subformulas of the given formula and propagates these labels bottom-up to the formula's root using a fixed set of inference rules. The labels represent invariants, which capture the relation between violations found in a collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ at some time point and violations found in its pre-images $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ at a time point with an equal timestamp, where $col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ denotes the set of temporal structures $(\bar{\mathcal{D}}', \bar{\tau}')$ with $col(\bar{\mathcal{D}}', \bar{\tau}') = (\bar{\mathcal{C}}, \bar{\kappa})$. Note that $(\bar{\mathcal{D}}, \bar{\tau}) \between (\bar{\mathcal{D}}', \bar{\tau}') \subsetneq col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$, where $(\bar{\mathcal{C}}, \bar{\kappa})$ is the collapse of an interleaving of the temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$.

The labels and their corresponding invariants are as follows for a formula $\phi$:

($\models \forall$): For all valuations $v$ and all $i \in \mathbb{N}$, if $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$ then for every $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ and every $j \in \mathbb{N}$ with $\kappa_i = \tau_j$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \phi$.

($\models \exists$): For all valuations $v$ and all $i \in \mathbb{N}$, if $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$ then for every $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$, there is some $j \in \mathbb{N}$ with $\kappa_i = \tau_j$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \phi$.

($\not\models \forall$): For all valuations $v$ and all $i \in \mathbb{N}$, if $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \not\models \phi$ then for every $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ and every $j \in \mathbb{N}$ with $\kappa_i = \tau_j$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \not\models \phi$.

$$\frac{}{t \approx t' : (\models \forall)} \qquad \frac{}{t \approx t' : (\not\models \forall)}$$

$$\frac{}{r(t_1, \ldots, t_{\iota(r)}) : (\models \exists)} \qquad \frac{}{r(t_1, \ldots, t_{\iota(r)}) : (\not\models \forall)}$$

$$\frac{\psi : (\models \forall)}{\psi : (\models \exists)} \qquad \frac{\psi : (\not\models \forall)}{\psi : (\not\models \exists)}$$

$$\frac{\psi : (\models \forall)}{\blacklozenge_I \psi : (\models \forall)} \qquad \frac{\psi : (\models \exists)}{\blacklozenge_I \psi : (\models \exists)} \qquad \frac{\psi : (\models \exists)}{\blacklozenge_I \psi : (\models \forall)}\ 0 \notin I \qquad \frac{\psi : (\not\models \forall)}{\blacklozenge_I \psi : (\not\models \forall)}$$

$$\frac{\psi : (\models \exists)}{\blacklozenge_I \lozenge_J \psi : (\models \forall)}\ 0 \in I \cap J$$

Figure 3. Selection of Inference Rules

($\not\models \exists$): For all valuations $v$ and all $i \in \mathbb{N}$, if $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \not\models \phi$ then for every $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$, there is some $j \in \mathbb{N}$ with $\kappa_i = \tau_j$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \not\models \phi$.

The first symbol ($\models$ or $\not\models$) in a label states whether the formula is satisfied in the collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$. The second symbol ($\forall$ or $\exists$) states whether the formula is satisfied at some equally timestamped time point or at all equally timestamped time points in all temporal structures $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$.

Due to space limitations, Figure 3 shows only some of our inference rules. All rules can be found in the full paper, where we also prove their soundness.

First, consider the rules in Figure 3 for atomic formulas. An atomic formula $t \approx t'$ depends only on the valuation and therefore can be labeled ($\models \forall$) and ($\not\models \forall$). An atomic formula of the form $r(t_1, \ldots, t_{\iota(r)})$ can be labeled ($\models \exists$) and ($\not\models \forall$). We only explain the labeling ($\models \exists$). The explanation for the label ($\not\models \forall$) is analogous. The interpretation of a predicate in a collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ at a time point $i$ is the union of the predicate's interpretations at all time points $j$ in a temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ for which $\tau_j$ equals $\kappa_i$. Therefore, if $\bar{a} \in r^{\mathcal{C}_i}$ then $\bar{a} \in r^{\mathcal{D}_j}$, for some $j \in \mathbb{N}$ with $\tau_j = \kappa_i$. Note that $\bar{a} \in r^{\mathcal{D}_j}$ does not necessarily hold for all these $j$s; hence, we cannot label $r(t_1, \ldots, t_{\iota(r)})$ with ($\not\models \forall$).

The next two rules in Figure 3 express that the invariants corresponding to the labels ($\models \forall$) and ($\not\models \forall$) imply the invariants corresponding to ($\models \exists$) and ($\not\models \exists$), respectively.

Next, we consider the inference rules for the temporal operator $\blacklozenge_I$. We first justify the inference rule that allows us to propagate the label ($\models \forall$) from $\psi$ to $\blacklozenge_I \psi$. If $\blacklozenge_I \psi$ is satisfied in the collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ at time point $i$ then $\psi$ is satisfied at some previous time point $j \leq i$ in $(\bar{\mathcal{C}}, \bar{\kappa})$ with $\kappa_i - \kappa_j \in I$. Because $\psi$ is labeled with ($\models \forall$), all time points with timestamp $\kappa_j$ in the temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ also satisfy $\psi$, and hence, all time points with timestamp $\kappa_i$ satisfy $\blacklozenge_I \psi$ in $(\bar{\mathcal{D}}, \bar{\tau})$. When $\psi$ is labeled with ($\models \exists$), possibly only a single time point $k$ in $(\bar{\mathcal{D}}, \bar{\tau})$ with $\tau_k = \kappa_j$ satisfies $\psi$. If $0 \in I$ then $\blacklozenge_I \psi$ might not be satisfied at time points before $k$, even if these time points have the timestamp $\kappa_i$. So, we can label $\blacklozenge_I$ with ($\models \exists$) but not with ($\models \forall$). However, if $0 \notin I$ then $\psi$ is satisfied in $(\bar{\mathcal{C}}, \bar{\kappa})$ at a time point $j$ with the timestamp $\kappa_j < \kappa_i$. Hence $\blacklozenge_I \psi$ is satisfied in $(\bar{\mathcal{D}}, \bar{\tau})$ at all time points with the timestamp $\kappa_i$. This allows us to label $\blacklozenge_I \psi$ with ($\models \forall$). Finally, consider the rule where $\psi$ is labeled ($\not\models \forall$). If $\blacklozenge_I \psi$ is violated in the collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ at timestamp $\kappa_i$ then $\psi$ is violated at all previous points in the temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ that satisfy the metric constraints given by $I$. But then $\blacklozenge_I \psi$ is also violated in $(\bar{\mathcal{D}}, \bar{\tau})$ at all time points with the timestamp $\kappa_i$. Hence we can label $\blacklozenge_I \psi$ with ($\not\models \forall$).

We can try to label a formula solely based on inference rules that involve only a single Boolean or temporal operator. However, with more specialized inference rules like the one for $\blacklozenge_I \lozenge_J \psi$ given in Figure 3, we are more likely to succeed in propagating labels to the root of the formula. Intuitively, with the nesting of the operators $\blacklozenge_I$ and $\lozenge_J$, and when $0 \in I \cap J$, the ordering of equally timestamped time points becomes irrelevant since from a given time point, we can freely choose any of these time points that satisfy the metric constraints given by the intervals $I$ and $J$. Hence, a labeling ($\models \exists$) for $\psi$ allows us to label $\blacklozenge_I \lozenge_J \psi$ with ($\models \forall$).

Finally, we remark that there are no inference rules for the temporal operators $\bullet_I$ and $\circ_I$ because these operators inherently rely on the relative ordering of the structures in a temporal structure.

Based on the labels at the root of the formula, we can determine if the formula has the property (C1) or the property (C2). The conclusions we can draw are stated in the following lemma, which follows from the soundness of the inference rules.

**Lemma 8.** *Let $\phi$ be a formula.*
1. *If $\phi$ can be labeled by ($\models \forall$) then $\phi$ has property (C1).*
2. *If $\phi$ can be labeled by ($\not\models \forall$) then $\phi$ has property (C2).*
3. *If $\phi$ can be labeled by ($\models \exists$) then $\lozenge \phi$ has property (C1).*
4. *If $\phi$ can be labeled by ($\not\models \exists$) then $\square \phi$ has property (C2).*

Based on this lemma, we obtain the following theorem.

**Theorem 9.** *If the formula $\phi$ can be labeled by ($\models \forall$) and ($\not\models \forall$) then it is collapse-sufficient. Moreover, we can determine in linear time in the formula's length whether $\phi$ can be labeled by ($\models \forall$), ($\models \exists$), ($\not\models \forall$), and ($\not\models \exists$).*

Note that formulas of the form $\square \psi$ are already collapse-sufficient if $\psi$ can be labeled by ($\not\models \exists$) and $\square \psi$ can be labeled by ($\models \forall$). Even if only one of these labellings can be derived, monitoring $\square \psi$ on the collapsed temporal structure of an interleaving is still useful. For example, if $\psi$ is labeled by ($\not\models \exists$) then violations that are found on the collapsed temporal structure relate to strong violations on the set of interleavings. However, we might miss some violations.

**Example 10.** *We illustrate our algorithm and its inference rules by applying it to the formula $\square \forall x.\, publish(x) \rightarrow \blacklozenge_{[0,11)}\, approve(x)$. We first remove some syntactic sugar and obtain the formula $\square \forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$. We start by labeling the atomic subformulas. Both $publish(x)$ and $approve(x)$ are labeled with ($\models \exists$) and ($\not\models \forall$). According*

*to the inference rules for the temporal operator $\blacklozenge_I$ we label $\blacklozenge_{[0,11)}\, approve(x)$ with $(\models \exists)$ and $(\not\models \forall)$. We cannot label it with $(\models \forall)$ since the interval contains 0. Moreover, the subformula $\neg publish(x)$ is labeled with $(\not\models \exists)$ and $(\models \forall)$. The subformulas $\neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ and $\forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ are labeled $(\models \exists)$ and $(\not\models \exists)$. We conclude that the formula $\Box \forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ has the property (C2). It does not have the property (C1), as shown in Example 7.*

*The formula $\Box \forall x.\, publish(x) \rightarrow \blacklozenge_{[1,11)}\, approve(x)$ has both properties (C1) and (C2). The labeling starts similarly but $\blacklozenge_{[1,11)}\, approve(x)$ is additionally labeled with $(\models \forall)$ since the interval of the temporal operator does not contain 0. This label propagates to the root of the formula. We conclude that $\Box \forall x.\, \neg publish(x) \vee \blacklozenge_{[1,11)}\, approve(x)$ also has property (C1).*

**Policy Approximation.** In Example 7, we have seen that we can obtain collapse-sufficient policies by strengthening or weakening the original policy. In the following, we present a systematic approach along these lines by over-approximating and under-approximating policies.

Let $\phi$ be a formula in positive normal form. We obtain the *weakened* formula $\phi^w$ by replacing each atomic subformula $r(t_1, \ldots, t_{\iota(r)})$ that occurs positively in $\phi$ by $\blacklozenge_I \Diamond_{I'} r(t_1, \ldots, t_{\iota(r)})$, for some intervals $I$ and $I'$ with $0 \in I \cap I'$. Analogously, in the *strengthened* formula $\phi^s$, we replace each negative occurrence of an atomic subformula $r(t_1, \ldots, t_{\iota(r)})$ by $\blacklozenge_I \Diamond_{I'} r(t_1, \ldots, t_{\iota(r)})$.

**Theorem 11.** *Let $\phi^w$ and $\phi^s$ be weakened and strengthened formulas of the formula $\phi$ in positive normal form. The formulas $\phi \rightarrow \phi^w$ and $\phi^s \rightarrow \phi$ are valid. Moreover,*
*1. if $\phi^s$ is collapse-sufficient then $\phi$ has property (C1), and*
*2. if $\phi^w$ is collapse-sufficient then $\phi$ has property (C2).*

Weakened and strengthened formulas are more likely to be collapse-sufficient, since their subformulas of the form $\blacklozenge_I \Diamond_{I'} r(t_1, \ldots, t_{\iota(r)})$ can be labeled with $(\models \forall)$, while $r(t_1, \ldots, t_{\iota(r)})$ can only be labeled with the weaker label $(\models \exists)$. Simultaneously weakening and strengthening always results in a collapse-sufficient formula. However, the resulting formula does not necessarily relate to the original formula.

Finally, note that by inserting the temporal operators $\blacklozenge_{[0,1)}$ and $\Diamond_{[0,1)}$ around positively occurring atomic subformulas, the ordering of equally timestamped actions becomes irrelevant. This is desirable in systems where the clocks used to timestamp the actions are synchronized but too coarse-grained. Taking this idea further, by putting temporal operators $\blacklozenge_{[0,b)}$ and $\Diamond_{[0,b)}$ around these subformulas with $b \geq 1$, we take into account that the timestamps in a temporal structure are inaccurate and might differ from their actual value by the threshold $b$—a situation that occurs in practice.

## IV. Practical Experience

In this section, we describe the implementation of our monitoring approach within Nokia's Data-collection Campaign [7], which is a real-world application with realistic usage-control policies. Furthermore, we report on the monitor's performance and our findings.

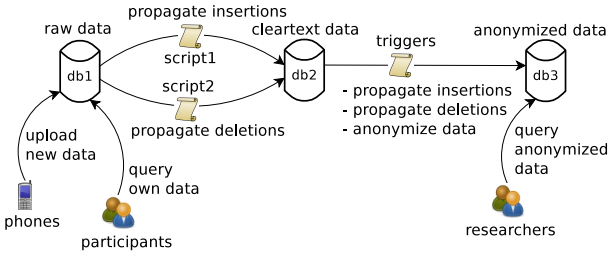


Figure 4. Nokia's Data-collection Campaign

Table I. Policy Formalizations in MFOTL

| policy | MFOTL formalization |
|---|---|
| *delete* | $\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db2}, data) \rightarrow user \approx \mathsf{script2}$ |
| *ins-1-2* | $\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db1}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,1s)}\, \Diamond_{[0,30h]}\, \exists user'.\, insert(user', \mathsf{db2}, data) \vee delete(user', \mathsf{db1}, data)$ |
| *ins-2-3* | $\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db2}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,1s)}\, \Diamond_{[0,60s)}\, \exists user'.\, insert(user', \mathsf{db3}, data)$ |
| *del-1-2* | $\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db1}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $(\blacklozenge_{[0,1s)}\, \Diamond_{[0,30h)}\, \exists user'.\, delete(user', \mathsf{db2}, data)) \vee$ $((\Diamond_{[0,1s)}\, \blacklozenge_{[0,30h)}\, \exists user'.\, insert(user', \mathsf{db1}, data)) \wedge$ $(\blacksquare_{[0,30h)}\, \Box_{[0,30h)}\, \neg \exists user'.\, insert(user', \mathsf{db2}, data)))$ |

**Scenario.** The campaign,[1] which was launched in 2009, collects contextual information from cell phones of about 180 participants. This sensitive data includes phone locations, call and SMS information, and the like. The data collected by a participant's phone is propagated into the databases db1, db2, and db3. The phones use WLAN to periodically upload their data to database db1. Every night, the synchronization script script1 copies the data from db1 to db2. Furthermore, triggers running on db2 anonymize and copy the data to db3, where researchers can access and analyze the anonymized data. The participants can access and delete their own data using a web interface to db1. Deletions are propagated to all databases: from db1 to db2 by the synchronization script script2, which also runs every night, and from db2 to db3 by database triggers. Figure 4 summarizes the various usages of data in the campaign.

Within the campaign, data is organized by records and can easily be identified. When uploading data from a phone into db1, a unique identifier is generated for each record. This identifier together with an identifier of the participant who contributed the data is attached to the record.

**Policies.** The collected data is subject to various policies in order to protect the participants' privacy. For example, there are access control rules and policies governing the process of propagating the data between databases. In particular, any insertion or deletion of data in db1 must be propagated to db2 within 30 hours, and from db2 to db3 within 1 minute. Furthermore, only the latest version of the synchronization scripts may be used and the scripts may not run longer than 6 hours. Finally, access to the databases is restricted to selected user accounts and the account script1 may be used only while the script script1 is running.

[1] See http://research.nokia.com/page/11367 for details.

Due to space constraints, we present just a few representative policies in Table I. Details about all the 14 policies are given in the full paper. The predicates *insert* and *delete* correspond to the equally-named database commands. The arguments of these predicates are the agent that initiated the action, the name of the database where the action was carried out, and an identifier of the involved data.

Note that all policy formalizations in Table I are collapse-sufficient. However, some policies have slightly weaker or stronger variants that are not collapse-sufficient. For example, we obtained *ins-2-3* from the policy "all data inserted into db2 must also be inserted into db3 within 60 seconds" by weakening the formula $\Box \forall users.\, \forall data.\, insert(user, \mathsf{db2}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow \Diamond_{[0,60s)}\, \exists user'.\, insert(user', \mathsf{db3}, data)$. Intuitively, *ins-2-3* is the policy formalization that we actually intended: we do not want to distinguish the relative ordering of the insertions into db2 and db3 when they are logged with the same timestamp. This is because the 1 second timestamp granularity that is used may not be fine-granular enough: the database triggers may be activated within milliseconds.

**Logging Mechanisms.** We extended the data-collection setup with mechanisms to log policy-relevant actions. We installed logging mechanisms for the three databases, the script script1, and the SVN repository, assuming synchronized clocks for timestamping. We now discuss details of these logging mechanisms.

As logs for the database db1 were not available, we implemented a proxy to inspect interactions of participants and phones with db1. The proxy logs what data is inserted and deleted. To observe the insertion of new data, we monitor the network traffic when the phone uploads data. For deletions, we use a custom front-end that logs the requests for deleting data. For practical reasons, we could deploy these mechanisms only for 2 out of the 180 participants. Hence, we have only partial logging for db1, which only affects 2 out of the 14 policies.

The databases db2 and db3 reside physically on a single PostgreSQL server, which logs the SQL queries. We extract relevant actions from these PostgreSQL logs. The main challenge is to determine what data is processed in a query since only the query itself is logged. Fortunately, most relevant queries are made by automated scripts or database triggers and contain enough information to determine what data is used. For example, an insert or delete query initiated by a synchronization script includes the identifier of the used data record. Hence, a simple syntactic analysis of these queries suffices to log the relevant actions in sufficient detail. When the analysis failed to extract the data, we identified the data with the constant unknown.

**Evaluation.** To evaluate the performance of our monitor on different data sets, we split the logs into smaller files, where each file corresponds to roughly 24 hours of log entries. Table II provides details about the collapsed

Table II. Log Statistics

| log | # time points | total # actions | # *insert* actions | | | # other actions |
|---|---|---|---|---|---|---|
| | | | db1 | db2 | db3 | |
| 1 | 29,672 | 1,462,700 | 82,486 | 678,840 | 678,840 | 22,534 |
| 2 | 10,870 | 969,520 | 23,828 | 472,369 | 472,369 | 954 |
| 3 | 6,601 | 1,019,428 | 33,229 | 492,411 | 492,411 | 1377 |
| 4 | 20,330 | 962,766 | 12,918 | 468,844 | 468,844 | 11,298 |
| 5 | 8,114 | 687,402 | 7,067 | 339,674 | 339,647 | 12,160 |
| 6 | 9,218 | 630,287 | 4,207 | 311,882 | 311,835 | 1,366 |
| 7 | 7,327 | 554,733 | 3,251 | 275,208 | 275,199 | 1,014 |
| 8 | 86,892 | 936,249 | 47,786 | 400,490 | 400,475 | 87,498 |
| 9 | 86,764 | 986,249 | 30,118 | 434,268 | 434,259 | 87,604 |

temporal structures corresponding to these logs. Observe that the number of *insert* actions is significantly larger than the number of other actions. None of the log files used contains more than 100 *delete* actions. Table III shows the monitor's running times and memory usage for each policy and log file. For the experiments, we used a desktop computer with a 1150 MHz AMD Phenom 9600B Quad-Core CPU.

Monitoring invariants like the policy *delete* is fast: the monitor needed no more than 10 seconds for a 24-hours log file. More complex policies involving temporal operators with large time windows, take more time to monitor. For example, for the policy *ins-1-2*, the monitor took more than 4 hours in some cases. The policy *del-1-2* with an even larger time window, however, could be quickly monitored. The reason here is that the log files used contain only few *delete* actions. Although we monitored the logs offline, the running times indicate that an online monitoring approach is possible, since the running times are less than the time period covered by the logs. The memory requirements are also modest. For the policies *delete* and *ins-2-3*, the monitor does not require more than 10 MB of RAM. For *ins-1-2* and *del-1-2*, the monitor used under 400 MB of RAM, which is also acceptable due to the large time windows.

**Findings.** The monitor reported the following policy violations. First, some static access control policies like *delete* were violated. These violations were due to testing, debugging, and other improvement activities going on while the system was running. Second, an earlier version of one of the synchronization scripts contained a bug, which was not detected in previous tests. Only a subset of the insertions were propagated between the databases. Third, while the campaign was running, the infrastructure was migrated to another server. After the migration, the deployment of the scripts was delayed, which caused policy violations.

Overall, the main reason for these violations is that we monitored an experimental system still under development. In this case study the monitor proved to be a powerful debugging tool. For commercial systems, it can detect policy violations thereby protecting the users' privacy and increasing users' trust in using the systems. Our findings also show that policy monitoring makes sense even in systems where users are honest and interested in honoring the policies.

## V. Related Work

The usage-control architecture described by Pretschner et al. [8] and the $\text{UCON}_{\text{ABC}}$ architecture of Park and

Table III. Monitor Performance — Running Times / Memory Usage

| policy | log 1 | log 2 | log 3 | log 4 | log 5 | log 6 | log 7 | log 8 | log 9 |
|---|---|---|---|---|---|---|---|---|---|
| *delete* | 10 s / 4 MB | 7 s / 4 MB | 7 s / 4 MB | 6 s / 4 MB | 5 s / 4 MB | 4 s / 4 MB | 4 s / 4 MB | 6 s / 4 MB | 6 s / 4 MB |
| *ins-1-2* | 231 m / 161 MB | 44 m / 103 MB | 67 m / 107 MB | 24 m / 102 MB | 9 m / 71 MB | 5 m / 65 MB | 3 m / 57 MB | 73 m / 115 MB | 48 m / 111 MB |
| *ins-2-3* | 9 m / 8 MB | 3 m / 7 MB | 5 m / 8 MB | 4 m / 8 MB | 2 m / 8 MB | 2 m / 7 MB | 1 m / 7 MB | 2 m / 8 MB | 1 m / 6 MB |
| *del-1-2* | 24 s / 176 MB | 16 s / 139 MB | 13 s / 87 MB | 11 s / 79 MB | 8 s / 58 MB | 7 s / 53 MB | 12 s / 111 MB | 21 s / 184 MB | 11 s / 102 MB |

Sandhu [9] both utilize monitoring techniques. However, the two architectures are only conceptual and have neither been deployed nor evaluated in a real-word setting.

Goodloe and Pike [10] recently surveyed the state of the art for monitoring distributed systems. We restrict ourselves here to the most related work. Bauer et al. [11] examine a setting where actions are totally ordered and system requirements are given in a propositional linear-time temporal logic. Both assumptions are too restrictive in our setting. However, their monitoring architecture additionally includes a component that analyzes the cause of a failure, which is fed back into the system. Genon et al. [12] present a monitoring algorithm for propositional LTL, where events are partially ordered. They use symbolic exploration methods to cope with the interleavings of events. It is unclear how their algorithm extends to a first-order setting. Moreover, in our approach, we consider formulas in a richer logic for which monitoring a single trace is sufficient. In contrast to these works and ours, Sen et al. [13] present a distributed monitoring approach, where multiple monitors are implemented locally and communicate with each other. These monitors are generated from a propositional past-time linear-time distributed temporal logic. A potential bottleneck is the monitors' communication overhead.

Finally, research on checking temporal integrity constrains [14], [15] of stored data and temporal triggers [16] in databases is related to our monitoring algorithm [1]. In fact, our monitoring algorithm extends Chomicki's monitor [14] by handling bounded future operators. These temporal operators are extremely useful for formalizing usage-control policies, which usually contain obligations. We are not aware of any implementation and experimental evaluation of Chomicki's monitoring algorithm.

## VI. Conclusion

We theoretically and practically tackled the problem of monitoring the usage of data in concurrent distributed systems. We provided means to efficiently monitor concurrently generated logs. We also deployed and evaluated a monitoring architecture in a real-world application, Nokia's Data-collection Campaign. Our case study demonstrates the feasibility and benefits of monitoring the usage of sensitive data.

As future work we plan to develop monitoring techniques for more complex systems with more agents, actions, and databases. The challenges will be to handle less accurate and less complete logging, and to provide monitoring algorithms that scale up from millions to billions of log entries per day. Our future work also includes developing monitoring techniques that can also be used for policy enforcement, i.e., preventing policy violations.

**Acknowledgments.** This work was supported by the Nokia Research Center, Switzerland. The authors thank Imad Aad, Debmalya Biswas, Olivier Bornet, Olivier Dousse, Juha Laurila, and Valtteri Niemi for valuable input.

## References

[1] D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann, "Runtime monitoring of metric first-order temporal properties," in *28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2008, pp. 49–60.

[2] D. Basin, F. Klaedtke, and S. Müller, "Monitoring security policies with metric first-order temporal logic," in *15th ACM Symposium on Access Control Models and Technologies*, 2010, pp. 23–34.

[3] A. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[4] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[5] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.

[6] L. Lamport, "What good is temporal logic?" in *IFIP 9th World Computer Congress*, ser. Information Processing, vol. 83, 1983, pp. 657–668.

[7] I. Aad and V. Niemi, "NRC data collection campaign and the privacy by design principles," in *International Workshop on Sensing for App Phones*, 2010.

[8] A. Pretschner, M. Hilty, and D. Basin, "Distributed usage control," *Commun. ACM*, vol. 49, no. 9, pp. 39–44, 2006.

[9] J. Park and R. Sandhu, "The $\text{UCON}_{\text{ABC}}$ usage control model," *ACM Trans. Inform. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, 2004.

[10] A. Goodloe and L. Pike, "Monitoring distributed real-time systems: A survey and future directions," NASA Langley Research Center, Tech. Rep. NASA/CR-2010-216724, 2010.

[11] A. Bauer, M. Leucker, and C. Schallhart, "Model-based runtime analysis of distributed reactive systems," in *2006 Australian Software Engineering Conference*, 2006.

[12] A. Genon, T. Massart, and C. Meuter, "Monitoring distributed controllers: When an efficient LTL algorithm on sequences is needed to model-check traces," in *14th International Symposium on Formal Methods*, ser. Lect. Notes Comput. Sci., vol. 4085, 2006, pp. 557–572.

[13] K. Sen, A. Vardhan, G. Agha, and G. Roşu, "Efficient decentralized monitoring of safety in distributed systems," in *26th International Conference on Software Engineering*, 2004, pp. 418–427.

[14] J. Chomicki, "Efficient checking of temporal integrity constraints using bounded history encoding," *ACM Trans. Database Syst.*, vol. 20, no. 2, pp. 149–186, 1995.

[15] U. Lipeck and G. Saake, "Monitoring dynamic integrity constraints based on temporal logic," *Inform. Syst.*, vol. 12, no. 3, pp. 255–269, 1987.

[16] P. Sistla and O. Wolfson, "Temporal triggers in active databases," *IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 3, pp. 471–486, 1995.