Organising LTL Monitors over Distributed Systems with a Global Clock

Christian Colombo[1] and Yliès Falcone[2]

[1] Department of Computer Science, University of Malta
christian.colombo@um.edu.mt
[2] Laboratoire d'Informatique de Grenoble, University of Grenoble-Alpes, France
ylies.falcone@ujf-grenoble.fr

Abstract. Users wanting to monitor distributed systems often prefer to abstract away the architecture of the system, allowing them to directly specify correctness properties on the global system behaviour. To support this abstraction, a compilation of the properties would not only involve the typical choice of monitoring algorithm, but also the organisation of submonitors across the component network. Existing approaches, considered in the context of LTL properties over distributed systems with a global clock, include the so-called orchestration and migration approaches. In the orchestration approach, a central monitor receives the events from all subsystems. In the migration approach, LTL formulae transfer themselves across subsystems to gather local information.
We propose a third way of organising submonitors: choreography — where monitors are orgnized as a tree across the distributed system, and each child feeds intermediate results to its parent. We formalise this approach, proving its correctness and worst case performance, and report on an empirical investigation comparing the three approaches on several concerns of decentralised monitoring.

1 Introduction

Due to the end of regular increase of processor speed, more systems are being designed to be decentralised to benefit from more of the multi-core feature of contemporary processors. This change in processors poses a number of challenges in the domain of runtime verification where performance is paramount.

In runtime verification one is interested in synthesizing a monitor to evaluate a stream of events (reflecting the behaviour of a system) according to some correctness properties. When the system consists of several computing units (referred to as components in the sequel), it is desirable to decentralise the monitoring process for several reasons (as seen in [1, 4, 5]). First, it is a solution to benefit from the plurality of computing units of the system if one can design decentralised monitors that are as independent as possible. Second, it avoids introducing a central observation point in the system that presupposes a modification of the system architecture, and it also generally reduces the communication overhead in the system. See [4, 5] for more arguments along this line.

In this paper, we study these questions in the context of monitors synthesized from LTL specifications by considering three approaches, namely orchestration, migration, and choreography, to organise monitors (using terminology from [6]): (i) Orchestration is the setting where a single node carries out all the monitoring processing whilst retrieving information from the rest of the nodes. (ii) Migration is the setting where the monitoring entity transports itself across the network, evolving as it goes along — doing away with the need to transfer lower level (finer-grained) information. (iii) Choreography is the setting where monitors are organised into a network and a protocol is used to enable cooperation between monitors.

Note, there are two important assumptions in our study. First, we assume the existence of a global clock in the system (as in [4]). This assumption is realistic for many critical industrial systems or when the system at hand is composed of several applications executing on the same operating system. Second, we assume that local monitors are attached to the components of the system and that the monitors can directly communicate with each other through some network.

Contributions of this paper. First, we survey the work on LTL monitoring in the context of distributed systems, classifying them under orchestration, choreography, and migration. Second, we introduce choreography-based decentralised monitoring. Third, we propose an algorithm that splits the monitoring of an LTL formula into smaller monitors forming a choreography. Fourth, we empirically compare orchestration, migration (from [4]), and choreography using a benchmark implementation.

Paper Organization. The rest of the paper is organised as follows. Section 2 introduces some background. Sections 3 and 4 recall the orchestration and migration approaches for LTL monitoring, respectively. In Section 5, we introduce the setting of choreography-based decentralised monitoring. Section 6 reports on our empirical evaluation and comparison of the three approaches using a benchmark implementation. Section 7 compares this paper with related work. Finally, Section 8 concludes and proposes future work.

2 Background

In this section, we formally define a distributed system and alphabet, followed by an introduction to the syntax and semantics of LTL.

Distributed systems and alphabet. $\mathbb{N}$ is the set of natural numbers. Let a distributed system be represented by a list of components: $\mathcal{C} = [C_1, C_2, \ldots, C_n]$ for some $n \in \mathbb{N} \setminus \{0\}$, and the alphabet Σ be the set of all events of the components: $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$, where Σ_i is the alphabet of C_i built over a set of local atomic propositions AP_i. We assume that the alphabets and sets of local atomic propositions are pair-wise disjoint[3] and define function $\#$ returning the index of the component related to an event, if it exists: $\# : \Sigma \rightarrow \mathbb{N}$ such that $\#a \stackrel{\text{def}}{=} i$ if $\exists i \in [1; n] : a \in \Sigma_i$ and undefined otherwise. The behavior of each component C_i is represented by a *trace* of events, which for t time steps is encoded as $u_i = u_i(0) \cdot u_i(1) \cdots u_i(t-1)$ with $\forall t' < t : \ u_i(t') \in \Sigma_i$. Finite (resp. infinite) traces over Σ are elements of Σ^* (resp. Σ^ω) and are denoted by $u, u', \ldots$ (resp. $w, w', \ldots$). The set of all traces is $\Sigma^\infty \stackrel{\text{def}}{=} \Sigma^* \cup \Sigma^\omega$. The finite or infinite sequence w^t is the *suffix* of the trace $w \in \Sigma^\infty$, starting at time t, i.e., $w^t = w(t) \cdot w(t+1) \cdots$.

[3] This assumption simplifies the presentation but does not affect the generality of the results.

Linear Temporal Logic. The system's global behaviour, $(u_1, u_2, \ldots, u_n)$ can now be described as a sequence of pair-wise union of the local events in component's traces, each of which at time t is of length $t+1$ i.e., $u = u(0) \cdots u(t)$.

We monitor a system *wrt.* a global specification, expressed as an LTL [9] formula, that does not state anything about its distribution or the system's architecture. LTL formulae can be described using the following grammar:

$$\varphi ::= p \mid (\varphi) \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi,$$

where $p \in AP$. Additionally, we allow the following operators, each of which is defined in terms of the above ones: $\top \stackrel{\text{def}}{=} p \vee \neg p$, $\bot \stackrel{\text{def}}{=} \neg\top$, $\varphi_1 \wedge \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\mathbf{F}\varphi \stackrel{\text{def}}{=} \top\,\mathbf{U}\,\varphi$, and $\mathbf{G}\varphi \stackrel{\text{def}}{=} \neg\,\mathbf{F}\,(\neg\varphi)$.

Definition 1 (LTL semantics [9]). *LTL semantics is defined wrt. infinite traces. Let $w \in \Sigma^\omega$ and $i \in \mathbb{N}$. Satisfaction of an LTL formula by w at time i is defined inductively:*

$$\begin{aligned}
w^i &\models p \Leftrightarrow p \in w(i), \text{ for any } p \in AP\\
w^i &\models \neg\varphi \Leftrightarrow w^i \not\models \varphi\\
w^i &\models \varphi_1 \vee \varphi_2 \Leftrightarrow w^i \models \varphi_1 \vee w^i \models \varphi_2\\
w^i &\models \mathbf{X}\,\varphi \Leftrightarrow w^{i+1} \models \varphi\\
w^i &\models \varphi_1\,\mathbf{U}\,\varphi_2 \Leftrightarrow \exists k \in [i, \infty[\,\cdot\; w^k \models \varphi_2 \wedge \forall l \in [i, k[: w^l \models \varphi_1
\end{aligned}$$

When $w^0 \models \varphi$ holds, we also write $w \models \varphi$.

Several approaches have been proposed for adapting LTL semantics for monitoring purposes (cf. [2]). Here, we follow previous work [4] and consider LTL$_3$ (introduced in [3]).

Definition 2 (LTL$_3$ semantics [3]). *Let $u \in \Sigma^*$, the satisfaction relation of LTL$_3$, $\models_3$: $\Sigma^* \times LTL \to \mathbb{B}_3$, with $\mathbb{B}_3 \stackrel{\text{def}}{=} \{\top, \bot, ?\}$, is defined as*

$$u \models_3 \varphi = \begin{cases} \top & \text{if } \forall w \in \Sigma^\omega : u \cdot w \models \varphi,\\ \bot & \text{if } \forall w \in \Sigma^\omega : u \cdot w \not\models \varphi,\\ ? & \text{otherwise}. \end{cases}$$

3 Orchestration

The idea of orchestration-based monitoring is to use a *central observation point* in the network (see Fig. 1). The central observation point can be introduced as an additional component or it can be a monitor attached to an existing component. In orchestration-based monitoring, at any time t, the central observation point is aware of every event $u_i(t)$ occurring on each component C_i, and has thus the information about the global

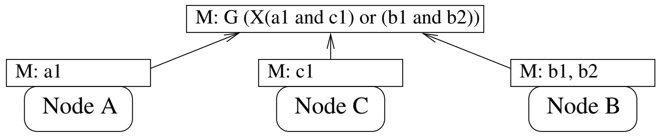

Fig. 1. An example of orchestration architecture

event $u_1(t) \cup \ldots \cup u_n(t)$ occurring in the system. Several protocols can be used by local monitors to communicate events. For instance, local monitors can send their local event at every time instance. Alternatively, the protocol may exploit the presence of a global clock in the system and just signal which propositions are true at any time instance or those whose value has changed. From a theoretical perspective, putting aside the instrumentation and communication, orchestration-based monitoring is not different from typical centralised monitoring.

4 Migration

Migration-based monitoring was introduced in [4]. The idea of migration is to represent (the state of) a monitor as an LTL formula that travels across a network. Upon the reception of a new LTL formula, a component progresses it, i.e., it rewrites the formula given the local observation, so that the resulting formula is the formula that has to hold in the next computation step. Such formula may contain references to past time instants if it has been progressed by components that could not evaluate some parts of it. More precisely, rewriting a formula is done using the so-called *progression*, adapted to the decentralised case, i.e., taking into account the fact that a component has only information about the local propositions it has access to. For example, in Fig. 2 only the valuations of $b1$ and $b2$ would be available for the monitor at component B. For the other propositions whose valuation is not available, an *obligation* is recorded which will have to be satisfied in a future time instant (by looking at the past). In the example, note that $Pa1$ and $Pc1$ refer to the previous values of $a1$ and $c1$ respectively. The rewritten formula is then sent to the most appropriate component — intuitively, the component that has the information about the proposition whose obligation reaches furthest into the past. The recipient component progresses the received formula using its local observation but also using its local history of observations to evaluate the past propositions. After sending a formula, a component is left with nothing to evaluate, unless it receives a formula from another component.

Any verdict found by a component is an actual global verdict. However, since the values of some propositions are known only one or more time instants later, the verdict is typically reached with a delay depending on the size of the network. To keep this delay to a minimum one can initially start monitoring the formula on all components,

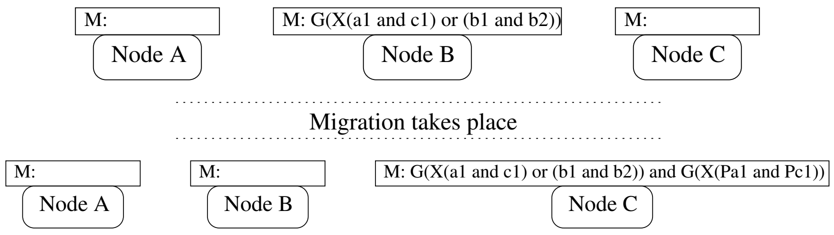
M:
Node A
M: G(X(a1 and c1) or (b1 and b2))
Node B
M:
Node C
Migration takes place
M:
Node A
M:
Node B
M: G(X(a1 and c1) or (b1 and b2)) and G(X(Pa1 and Pc1))
Node C

Fig. 2. An example of migrating architecture

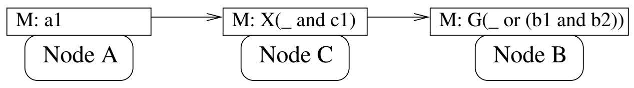

Fig. 3. An example of choreography architecture

enabling different sequences of proposition processing. The downside, however, is that this increases the number of messages as well as the number of progressions.

5 Choreography

Rather than having the whole formula at a single location (whether this is fixed as in orchestration or variable as in migration), choreography breaks down the formula across the network, forming a tree structure where results from subformulae flow up to the parent formula.

5.1 Choreography at an Abstract Level

Figure 3 shows how formula $\mathbf{G}(\mathbf{X}(a1 \wedge c1) \vee b1 \wedge b2)$ is spread across a network of three nodes A, B, and C with sets of local propositions $\{a1\}$, $\{b1, b2\}$, and $\{c1\}$, respectively. Note that each proposition is monitored in what we refer in the following as its native node, i.e., each node is monitoring a subformula that contains reference to either its local atomic propositions or *place holders*. Intuitively, place holders can be understood as three-state propositions that represent the verdict (true, false, or no verdict yet) of a remote subformula being evaluated on another component. Note also that no node is aware of all the propositional values. The progression of a choreographed monitoring network includes the following steps:

1. Progress the subformulae that do not have place holders, and forward the verdicts to their parents.
2. Upon receiving all verdicts for place holders, parent subformulae perform their progression potentially spawning new place holders (e.g., due to the progression of the *Until* operator (defined later)).
3. Verdicts continue to propagate from the leaves to the root of the tree until the root reaches a true or false verdict.

In what follows, we formalise the progression of a choreographed monitoring network, and prove two properties of the proposed choreography: the maximum number of nested place holders and the correctness of the verdict reached.

5.2 Formalizing Choreography

In the rest of this section, we formally define an instantiation of the choreography approach, starting with the distribution of an LTL formula across a network and subsequently showing how interactions take place to reach the verdict for a particular trace. We extend LTL syntax with one modality to support distribution.

Definition 3 (Distributed LTL). *Distributed LTL formulae, in LTL_D, are defined as follows:*

$$\varphi^D ::= \varphi \mid \langle\!| x, y |\!\rangle_\varphi, \textit{ where } x, y \in \mathbb{N} \textit{ and } \varphi \in \textit{LTL}$$

A distributed LTL formula is either an LTL formula or a place holder of the form $\langle\!| x, y |\!\rangle_\varphi$ where natural numbers x, y act as a pointer to a subformula in the LTL network, while the LTL formula is kept as a copy.

Remark 1. The modality related to distribution is only used in our definitions and functions. The end user, i.e., the one writing properties, does not need to be aware of it.

Given a distributed LTL formula, we define a scoring function that returns a natural number representing the desirability of placing the monitor for that LTL formula on some particular component i. To choose where to place a given LTL formula, we choose the one with the highest score.

Definition 4 (Choosing component). *The scoring and choice functions are defined as follows:*

- *The scoring function* $\mathrm{scor}_i : \textit{LTL}_D \to \mathbb{N}$ *is defined as follows (using $\sim$ and $\odot$ to range over unary and binary LTL operators, resp.):*

$$\begin{array}{lll} \mathrm{scor}_i(\varphi) = \textit{match } \varphi \textit{ with} & & \\ \quad \mid \sim\psi \to \mathrm{scor}_i(\psi) & & \mid \psi \odot \psi' \to \mathrm{scor}_i(\psi) + \mathrm{scor}_i(\psi') \\ \quad \mid p \quad \to \begin{cases} 1 \textit{ if } \#p = i \\ 0 \textit{ otherwise} \end{cases} & & \mid \_ \qquad \to 0 \end{array}$$

- *The choice function* $\mathrm{chc} : \textit{LTL}_D \to \mathbb{N}$ *is defined as follows:*

$$\mathrm{chc}(\varphi) \overset{\mathrm{def}}{=} i \textit{ such that } \mathrm{scor}_i(\varphi) = \max(\mathrm{scor}_1(\varphi), \ldots, \mathrm{scor}_n(\varphi))$$

Note that this definition of chc might have several solutions but we leave it up to the implementer to choose any component with a high score, either randomly or through some other strategy.

An important condition for choreography to function correctly is to ensure that for any proposition p, $\mathrm{chc}(p) = \#p$ holds since the value of p can only be resolved at component $\#p$. In what follows we assume this is always the case.

Remark 2. There are several ways of varying the scoring function. The following two are just examples: (i) Vary the weighting of binary operators' operands, e.g., in the case of the *Until* the right subformula is given more weighting than the left; (ii) Giving more weight to a particular component, e.g., to create an orchestration where the whole formula except the remote propositions are on a single component.

Given a list of components making up a system, a monitor network is a corresponding list of monitors (with one monitor per component) where each monitor has certain LTL formulae.

Definition 5 (LTL network). *An LTL network is a function $M : \mathbb{N} \to \mathbb{N} \to \textit{LTL}$ which given a component identifier, returns the component's monitor, which in turn is a function which given the formula identifier, returns the formula.*

We use M, N, O, P to range over the set of networks $\mathcal{M}$. As abbreviations we use M_i to refer to $M(i)$, i.e., the i-th component in network M, and M_i^j to refer to $M_i(j)$, i.e., the j-th formula of the i-th component in M. Moreover, $|M_i| = |\operatorname{dom}(M_i)|$ refers to the size of the domain of M_i, while $M_i^j \mapsto \varphi$ is used as abbreviation for $M \dagger [i \mapsto M_i \cup [(j \mapsto \varphi)]]$ and M_i^* as abbreviation for $M_i^{|M_i|}$, where $\dagger$ is the classical map override operator.[4]

Intuitively, distributing a formula across a network requires two operations: modifying the formula to point to its subparts which are in another part of the network, and inserting the formula with pointers inside the network. The function net defined below handles the latter aspect while the former is handled by distr. In turn distr (through recurs) recursively calls itself on subformulae until it encounters a subpart which belongs to a different component (due to the scoring function). In this case, function net is called once more so that the remote subformula is inserted in the network accordingly. Using function chc, the sub parts of a formula that "choose" a different component from their parent's can be marked as distributed using LTL$_D$ modalities and placed at a different point in the network.

Definition 6 (Generating an LTL network). *Thus, we define function* $\operatorname{net} : \mathcal{M} \times LTL \to \mathcal{M}$*, which given an (initially empty) network, distributes the LTL formula according to the scoring function as follows:*

$$\begin{aligned}\operatorname{net}(M,\varphi) = \ & \textit{let } c = \operatorname{chc}(\varphi) \textit{ in} \\ & \textit{let } M', \varphi' = \operatorname{distr}_c(M,\varphi) \textit{ in } {M'}_c^* \mapsto \varphi'\end{aligned}$$

$$\begin{aligned}\textit{where } \ \operatorname{distr}_i(M,\varphi) = \ & \textit{match } (M,\varphi) \textit{ with} \\ & \mid \sim\psi \quad\ \to \textit{let } N, \psi' = \operatorname{recurs}_i(M,\psi) \textit{ in } N, \sim\psi' \\ & \mid \psi \odot \psi' \to \textit{let } N, \psi'' = \operatorname{recurs}_i(M,\psi) \textit{ in} \\ & \qquad\qquad\ \textit{let } O, \psi''' = \operatorname{recurs}_i(N,\psi') \textit{ in } O, \psi'' \odot \psi''' \\ & \mid \psi \qquad \to M, \psi\end{aligned}$$

$$\textit{and} \quad \operatorname{recurs}_i(M,\varphi) = \textit{let } j = \operatorname{chc}(\varphi) \textit{ in} \begin{cases} \operatorname{distr}_i(M,\varphi) & \textit{if } j = i \\ \operatorname{net}(M,\varphi), \langle\!| j, |M_j| |\!\rangle_\varphi & \textit{otherwise.} \end{cases}$$

Note that, starting with an empty network ($M_E = \{1 \mapsto \{\}, \ldots, n \mapsto \{\}\}$) where n is the number of components), this function returns a tree structure with LTL subformulae linked to their parent. We abbreviate $\operatorname{net}(M_E, \varphi)$ to $\operatorname{net}(\varphi)$. To denote the root of the tree for the network of an LTL formula φ, i.e., the main monitor, we use $\hat{M}$, which is defined as $M_c^{|M_c|-1}$ where $c = \operatorname{chc}(\varphi)$.

Example 1. Consider the scenario of constructing a network for formula $\varphi = a\,\mathbf{U}\,b$ for a decentralised system with two components, A and B (numbered 1 and 2 resp.), with the former having proposition a at its disposal while the latter having proposition b.

Starting with a call to net, we note that $\operatorname{chc}(\varphi)$ may return either 1 or 2 depending on the definition of maximum. In this case, we assume the former and call the distribution function on an empty network: $\operatorname{distr}_1(M_E, \varphi)$. Starting with the basic definitions, the example works out as follows:

[4] For two functions f and g, for any element e, $(f \dagger g)(e)$ is $g(e)$ if $e \in \operatorname{dom}(g)$, $f(e)$ if $e \in \operatorname{dom}(f)$, and *undef* otherwise.

$$\begin{aligned}
N, \varphi' &= \mathrm{recurs}_1(M_E, a) = \mathrm{distr}_1(M_E, a)\\
&= \{1 \mapsto \{\}, 2 \mapsto \{\}\}, a\\
O, \psi' &= \mathrm{recurs}_1(N, b) = \mathrm{net}(N, b), \langle\!|2, 0|\!\rangle_b\\
&= \{1 \mapsto \{0 \mapsto b\}, 2 \mapsto \{\}\}, \langle\!|2, 0|\!\rangle_b\\
\mathrm{distr}_1(M_E, \varphi) &= \{1 \mapsto \{\}, 2 \mapsto \{0 \mapsto b\}\}, a \,\mathbf{U}\, \langle\!|2, 0|\!\rangle_b\\
\mathrm{net}(M_E, \varphi) &= \{1 \mapsto \{0 \mapsto a \,\mathbf{U}\, \langle\!|2, 0|\!\rangle_b\}, 2 \mapsto \{0 \mapsto b\}\}
\end{aligned}$$

At each time step, starting from the main monitor, the network performs one choreographed progression step.

Definition 7 (Choreographed Progression). *Given an LTL network M, the index j of a formula in monitor i, and the current observation σ, the choreographed progression function* $\mathrm{prog}_i : \mathcal{M} \times \mathbb{N} \times \Sigma \to \mathcal{M}$*, returns the resulting LTL network:*

$$\begin{aligned}
\mathrm{prog}_i(M, j, \sigma) = &\ \textit{match } M_i^j \textit{ with}\\
&\mid \top \mid \bot \ \ \to M\\
&\mid p \qquad\ \ \to \begin{cases} M_i^j \mapsto \top \ \textit{if}\ p \in \sigma\\ M_i^j \mapsto \bot \ \textit{otherwise}\end{cases}\\
&\mid \neg\varphi \qquad \to \neg\big(\mathrm{prog}_i(M, j, \sigma)_i^j\big)\\
&\mid \mathbf{X}\,\varphi \qquad \to M_i^j \mapsto \varphi\\
&\mid \varphi \odot \psi \quad \to \textit{let } N = \mathrm{prog}_i(M_i^j \mapsto \varphi, j, \sigma) \textit{ in}\\
&\qquad\qquad\quad\ \ \textit{let } O = \mathrm{prog}_i(N_i^j \mapsto \psi, j, \sigma) \textit{ in}\\
&\qquad\qquad\quad\ \ \textit{let } P, \varphi' = \mathrm{distr}_i(O, \varphi \,\mathbf{U}\, \psi) \textit{ in}\\
&\qquad\qquad\quad\ \ \begin{cases} O_i^j \mapsto N_i^j \vee O_i^j & \textit{when } M_i^j = \varphi \vee \psi\\ P_i^j \mapsto O_i^j \vee (N_i^j \wedge \varphi') & \textit{when } M_i^j = \varphi \,\mathbf{U}\, \psi\end{cases}\\
&\mid \langle\!|x, y|\!\rangle_\varphi \to \textit{let } N = \mathrm{prog}_x(M, y, \sigma) \textit{ in}\\
&\qquad\qquad\quad\ \ \begin{cases} N_i^j \mapsto N_x^y \ \textit{if}\ N_x^y \in \{\top, \bot\}\\ N \qquad\qquad \textit{otherwise}\end{cases}
\end{aligned}$$

Finally, due to the call to distr_i from the progression function, we overload the function to handle distributed LTL formulae by adding the following line enabling the respawning of distributed formulae:

$$\mathrm{distr}_i(M, \langle\!|x, y|\!\rangle_\varphi) \stackrel{\text{def}}{=} \mathrm{net}(M, \varphi), \langle\!|\,\mathrm{chc}(\varphi), |M_{\mathrm{chc}(\varphi)}|\,|\!\rangle_\varphi$$

The progression mechanism in the choreography context is similar to normal LTL. However, due to remote subparts of a formula, the network may change in several parts when progressing a single formula. Thus, when handling LTL operators, subformulae should first be applied one by one on the network, each time operating on the updated network (hence N and O). Slightly more complex is the *Until* case where a fresh copy of any distributed subparts have to be respawned across the network. P handles this by calling the distribution function on the progressed network O.

Example 2. Building upon the previous example, $a \,\mathbf{U}\, b$, assuming a trace $\{a\} \cdot \{b\}$, starting with network $\{1 \mapsto \{0 \mapsto a \,\mathbf{U}\, \langle\!|2, 0|\!\rangle\}, 2 \mapsto \{0 \mapsto b\}\}$, and noting that the main monitor resides at $(1, 0)$, progression would evolve as follows (again starting with the basic definitions):

1. First element of the trace: $\{a\}$

$$\begin{aligned}
N &= \mathrm{prog}_1(\{1 \mapsto \{0 \mapsto a\}, 2 \mapsto \{0 \mapsto b\}\}, 0, \{a\})\\
&= \{1 \mapsto \{0 \mapsto \top\}, 2 \mapsto \{0 \mapsto b\}\}\\
O &= \mathrm{prog}_1(\{1 \mapsto \{0 \mapsto \langle\!|2,0|\!\rangle_b\}, 2 \mapsto \{0 \mapsto b\}\}, 0, \{a\})\\
&= \{1 \mapsto \{0 \mapsto \bot\}, 2 \mapsto \{0 \mapsto \bot\}\}\\
P, \varphi' &= \mathrm{distr}_1(\{1 \mapsto \{0 \mapsto \bot\}, 2 \mapsto \{0 \mapsto \bot\}\}, a\,\mathbf{U}\,\langle\!|2,0|\!\rangle_b)\\
&= \{1 \mapsto \{0 \mapsto \bot\}, 2 \mapsto \{0 \mapsto \bot, 1 \mapsto b\}\}, a\,\mathbf{U}\,\langle\!|2,1|\!\rangle_b\\
\mathrm{prog}_1&(\{1 \mapsto \{0 \mapsto a\,\mathbf{U}\,\langle\!|2,0|\!\rangle_b\}, 2 \mapsto \{0 \mapsto b\}\}, 0, \{a\})\\
&= \{1 \mapsto \{0 \mapsto \bot \vee (\top \wedge a\,\mathbf{U}\,\langle\!|2,1|\!\rangle_b)\}, 2 \mapsto \{0 \mapsto \bot, 1 \mapsto b\}\}
\end{aligned}$$

2. Second element of the trace: $\{b\}$. (Note that the main formula has been simplified using normal LTL simplification rules and unused subformulae garbage collected.)

$$\begin{aligned}
N &= \mathrm{prog}_1(\{1 \mapsto \{0 \mapsto a\}, 2 \mapsto \{1 \mapsto b\}\}, 0, \{b\})\\
&= \{1 \mapsto \{0 \mapsto \bot\}, 2 \mapsto \{1 \mapsto b\}\}\\
O &= \mathrm{prog}_1(\{1 \mapsto \{0 \mapsto \langle\!|2,1|\!\rangle_b\}, 2 \mapsto \{1 \mapsto b\}\}, 0, \{b\})\\
&= \{1 \mapsto \{0 \mapsto \top\}, 2 \mapsto \{1 \mapsto \top\}\}\\
P, \varphi' &= \mathrm{distr}_1(\{1 \mapsto \{0 \mapsto \top\}, 2 \mapsto \{1 \mapsto \top\}\}, a\,\mathbf{U}\,\langle\!|2,1|\!\rangle_b)\\
&= \{1 \mapsto \{0 \mapsto \top\}, 2 \mapsto \{1 \mapsto \top, 2 \mapsto b\}\}, a\,\mathbf{U}\,\langle\!|2,2|\!\rangle_b\\
\mathrm{prog}_1&(\{1 \mapsto \{0 \mapsto a\,\mathbf{U}\,\langle\!|2,1|\!\rangle_b\}, 2 \mapsto \{1 \mapsto b\}\}, 0, \{b\})\\
&= \{1 \mapsto \{0 \mapsto \top \vee (\bot \wedge a\,\mathbf{U}\,\langle\!|2,2|\!\rangle_b)\}, 2 \mapsto \{1 \mapsto \top, 2 \mapsto b\}\}
\end{aligned}$$

Through simplification and garbage collection, the network resolves to $\{1 \mapsto \{0 \mapsto \top\}, 2 \mapsto \{\}\}$, i.e., the main formula is now $\top$, meaning that a verdict has been reached as defined below.

Definition 8 (Decentralised semantics). *The satisfaction relation for choreographed monitors is given according to the verdict reached by the topmost monitor as follows:*

$$u \models_C \varphi \overset{\text{def}}{=} \begin{cases} \top \text{ if } \hat{M} = \top \\ \bot \text{ if } \hat{M} = \bot \\ ? \text{ otherwise} \end{cases}$$

For the purpose of guaranteeing the maximum number of indirections in a choreographed LTL network, we define two depth-measuring functions: one which measures the maximum number of nesting levels in a formula, and another which measures the number of indirections in the network (typically starting from the main formula).

Definition 9 (Depth). *The depth-measuring function* $\mathrm{dpth} : LTL_D \rightarrow \mathbb{N}$ *is defined as:*

$$\begin{aligned}
\mathrm{dpth}(\varphi) = \textit{match } \varphi \textit{ with}&\\
&\mid \sim\psi \quad\;\; \rightarrow 1 + \mathrm{dpth}(\psi)\\
&\mid \psi \odot \psi' \rightarrow 1 + \max(\mathrm{dpth}(\psi), \mathrm{dpth}(\psi'))\\
&\mid \_ \qquad\;\; \rightarrow 1
\end{aligned}$$

The function measuring the depth of nested distribution modalities, taking a network and an x and y pointer to a formula: $\mathrm{dpth}_D : \mathcal{M} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ *is defined as:*

$$\begin{array}{ll}\mathrm{dpth}_D(M,i,j) = \textit{match } M_i^j \textit{ with} & \\ \quad \mid \langle\!|x,y|\!\rangle_\psi & \to 1 + \mathrm{dpth}_D(M,x,y) \\ \quad \mid \sim\psi & \to \mathrm{dpth}_D(M_i^j \mapsto \psi, i, j) \\ \quad \mid \psi \odot \psi' & \to \max(\mathrm{dpth}_D(M_i^j \mapsto \psi, i, j), \mathrm{dpth}_D(M_i^j \mapsto \psi', i, j)) \\ \quad \mid \_ & \to 0 \end{array}$$

Theorem 1 (Maximum nested distributions). *The number of nested distributions in a choreographed LTL formula cannot exceed the number of levels of nesting within a formula:* $\forall\varphi \in \mathit{LTL} : \mathrm{dpth}_D(\mathrm{net}(\varphi)) < \mathrm{dpth}(\varphi)$.

Proof. This follows from the definition of net and by extension distr which at most introduces one place holder ($\langle\!|x,y|\!\rangle_\varphi$) for any particular level and from the definitions of the functions dpth and dpth_D where for any case considered $\mathrm{dpth}_D \leq \mathrm{dpth}$. Furthermore, we note that since a formula must have propositions, true or false at the leafs, then the distribution depth is strictly less than the formula depth.

To aid in the proof of correctness, we define the function $\overline{\mathrm{net}}$ which given a choreography network and a pointer to the main formula, returns the LTL formula being monitored in the network, $\overline{\mathrm{net}} : \mathcal{M} \times \mathbb{N} \times \mathbb{N} \to \mathrm{LTL}$:

$$\begin{array}{ll}\overline{\mathrm{net}}(M,i,j) = \text{match } M_i^j \text{ with} & \\ \quad \mid \sim\psi & \to \sim\big(\overline{\mathrm{net}}(M_i^j \mapsto \psi, i, j)\big) \\ \quad \mid \psi \odot \psi' & \to \big(\overline{\mathrm{net}}(M_i^j \mapsto \psi, i, j)\big) \odot \big(\overline{\mathrm{net}}(M_i^j \mapsto \psi', i, j)\big) \\ \quad \mid \langle\!|x,y|\!\rangle_\varphi & \to \overline{\mathrm{net}}(M,x,y) \\ \quad \mid \psi & \to \psi \end{array}$$

Theorem 2 (Correctness). *The verdict reached by choreographed monitoring is the same as the one reached under normal monitoring* $\models_C = \models_3$.

Proof. In the context of a choreography, the state of the monitor is distributed across the network. By induction on the size of the trace, we show that at every progression step, the state of the monitoring network is equivalent to the formula if monitored centrally.

BC: Initially, if we had to compare the original formula to the distributed formula but "undistributing" it, then they should be equivalent: $\varphi = \overline{\mathrm{net}}(\mathrm{net}(\varphi))$. This follows from the definitions of net and $\overline{\mathrm{net}}$.

IH: After k progressions, the resulting LTL formula is equivalent to the resulting network: ${}_k\varphi = \overline{\mathrm{net}}({}_kM)$ (assuming no simplifications).

IC: Assuming IH, after $k+1$ progressions the resulting formula and network should be semantically equivalent: ${}_{k+1}\varphi = \overline{\mathrm{net}}({}_{k+1}M)$. This follows through a case-by-case analysis of the progression function prog which correspond to the cases of the normal progression function.

6 Evaluation and Discussion

Numerous criteria can be considered for comparing different organisations of LTL monitoring over a network. Below are a number of them which are treated in this study[5]:

[5] We ignore implementation-dependent measurements such as actual overhead of monitors.

Delay: Because of the network organization, it takes some communication steps to propagate intermediate results.

Number and size of messages: Since no component in the network can observe the full behaviour of the system, components have to communicate. Thus, we measure how many messages are required and their size.

Progressions: Different configurations of the monitoring network affect the number of LTL progressions that need to be carried out.

Privacy and security concerns[6]: In certain cases, one might wish to avoid communicating a component's local data across the network. This might be either because of lack of trust between the components themselves or due to an unsecured network.

To compare the three approaches with respect to these criteria, we have carried out two main experiments (whose results are shown in Tables 1 and 2 resp.):

– The first one varies the size of the network, i.e., the number of components, and the number of redirections in the resulting LTL network. This experiment is crucial since the migration approach is sensitive to the size of the network [4] while intuitively we expect the choreography approach to be affected by the depth of the LTL network.
– The second experiment varies the size of the formulae being considered and the pattern of the resulting tree once the formula is distributed. This enabled us to assess the scalability of the approaches and how they react to a different network structures. In particular we considered two kinds of networks: one whose formula is generated purely randomly, and another where we biased the formula generator such that the bottom-most LTL operators always have operands from the same component; essentially emulating networks where the basic subformulae of an LTL formula can be evaluated without communicating.

Some choices needed to be made with respect to the architectural setup of the experiments:

Experiment setup: The setup is based on the tool DecentMon[7] used in a previous study comparing orchestration with migration [4]. For this study we simply extended the tool with a choreography approach[8].

Benchmark generation: For the first experiment, we generated 100 LTL formulae and distributed traces randomly, subsequently tweaking the alphabet to manipulate the number of referenced components and the depth of the resulting LTL network. For the second experiment we could not use the same formulae since one of the variables considered was the size of the formulae. The numbers shown in the tables are thus the average results obtained across the 100 formulae considered in each case.

Communication protocol: Choosing a communication protocol such as communicating only the propositions which are true while assuming that unsent ones are false, makes a significant difference to our results. The chosen protocols were as follows: In the case of orchestration, only the propositions referenced in the formula that hold true are sent. Each sent proposition is considered to be of size one. In the case of migration,

[6] We refrain from going into fault-tolerance issues in this study, leaving it for future work.

[7] `http://decentmonitor.forge.imag.fr`

[8] The new implementation is available at: `http://decentmon3.forge.imag.fr`.

since the whole formula is sent, it is less straightforward to gain quick savings as in the case of propositions. Thus, in this case we measure the size of the formula (one for each proposition and each operator) and use it as the size of the message. In the case of choreography we have two kinds of messages: updates from subformulae to their parent's place holders and redistribution messages. The former kind are similar to those of orchestration but there is also the possibility that the subformula has neither reached true nor false. Thus, if no verdict has been reached, the subformula transmits nothing, otherwise it sends the verdict which counts as one. As for the redistribution messages, recall that each redistribution would have been already communicated during the initial setup of the network. Therefore, we assume that upon redistribution there is no need to resend the formula and we consider its size to be one.

Execution cycles: A major difference between choreography and migration is that the latter could send all the messages in one cycle while in the case of the choreography, since the distribution messages succeed the ones enabling progression, there are two messaging cycles for every time instant. However, the picture is even more complex because the progression within a component may depend on the verdict of others. Thus, while migration (as in [4]) strictly allowed one progression and messaging cycle per system cycle, in our choreography evaluation, we allowed any number of cycles that were necessary for the network to completely process the values in the current system cycle. This makes the choreography approach delay-free (and hence avoids references to the history) but relatively more expensive in terms of the number of cycles and the messages required for each system cycle.

In the following subsections, we discuss the outcome by first comparing choreography with migration, and subsequently comparing choreography to orchestration. We refrain from comparing orchestration to migration as this has already been carried out extensively in [4] and the results in the tables confirm the conclusions.

6.1 Choreography and Migration

We start by comparing the choreography approach to the migration approach by considering each criterion in turn:

Delay: As discussed earlier, since we have opted to allow the monitors to stabilise between each system cycle, we observe no delay for the choreography case. However, had this not been the case, we conjecture that the worst case delay would depend on the depth of the formula network which, as proven in Theorem 1, is less than the depth of the actual LTL formula.

Number and size of messages: A significant difference between choreography and migration is that in migration the whole formula is transmitted over the network while in choreography only when a subformula reaches true or false is the verdict transmitted. This distinction contributes to the significant difference in the size of the messages sent observed in Table 1.

However, the situation is reversed in the case of the frequency of messages. This is mainly because in choreography, not only does the network have to propagate the verdicts, but some progressions require a respawning of some submonitors. For example, consider the case of formula $\varphi \, \mathbf{U} \, \psi$ which is progressed to $\psi' \vee (\varphi' \wedge \varphi \, \mathbf{U} \, \psi)$. First, we note that φ' and ψ' are progressions of their counterparts in the context of the time instance being considered, while copies of the formulae are respawned to be progressed in the following time instance. This means that upon respawning, all remote submonitors have to be respawned accordingly. Naturally, this has to be done using messages, which as shown in Table 1, constitute more than half the total number of messages required.

Although choreography generally obtained better results with respect to the size of messages, the scale starts tipping in favour of migration the bigger the formula is. This is clearly visible in Tables 2 where for bigger formulae the results get closer, with migration surpassing choreography in the third (unbiased) case. The reason behind this is probably that simplification in the choreography context does not work optimally since the simplification function does not have the visibility of the whole network.

As part of the evaluation, we changed the number of components involved in a formula whilst keeping everything constant. Unsurprisingly, changing the number of components did not affect the performance of the choreography approach as much as it affected the performance of the migration approach. Table 1 shows this clearly: the compound size of messages transmitted over nine components is 16 times bigger than that of the three-component experiment. The results for choreography still fluctuated[9] but not clearly in any direction and less than a factor of two in the worst case.

Similarly, keeping everything constant, we altered the alphabet once more, this time keeping the number of components constant but changing the number of indirections required in the choreography, i.e., a deeper tree of monitors. Again, the results in Table 1 confirm the intuition that this change affects the choreography much more than the migration approach. In this case the distinction is somewhat less pronounced. However, if we compare the change from 96.16 to 81.3 in the migration case as opposed to the change from 2.47 to 4.16 in the case of choreography, we note that the percentage change is over four times bigger in the second case (68% as opposed to 15%).

Progressions: The variations in the number of progressions is similar to the number of messages sent/received. The two are linked indirectly in the sense that both the number of messages and progressions increase if the monitoring activity in the network increases. However, we note that this need not be the case, particularly when the number of components is small and monitoring can take place with little communication.

Privacy and security concerns: In general, in both the migration and the choreography approaches no component can view all the proposition values in the network. However, the migration approach is significantly safer in this regard as no proposition values are communicated: only LTL formulae, being less informative to an eavesdropper.

6.2 Choreography and Orchestration

In this subsection, we compare the choreography and the orchestration approaches.

Delay: Since orchestration is a special case of choreography with depth one, the delay of an orchestration is always better or as good as that of a choreography. However,

[9] The reasons for the fluctuations are probably due to the random adaptations of the alphabet to change the number of components a formula is based upon.

[10] The number of distribution messages is included in the previous column. We also note that all choreography messages are of size one and thus these two columns represent the size of the messages too.

Table 1. Same formulae and traces with modified components and distribution depth

Variables		Orchestration		Migration			Choreography		
comps	depth	# msgs	progs	# msgs	|msgs|	progs	# msgs	# distr[10]	progs
3	4	1.3	1.8	0.12	22.10	14.07	4.22	2.90	8.07
5	4			0.21	98.59	55.02	2.18	1.54	5.74
9	4			0.24	353.86	188.06	2.79	1.96	6.25
5	3.15			0.21	96.16	53.98	2.47	1.74	5.98
5	5.83			0.21	81.3	46.43	4.16	2.88	8.05

Table 2. Same formulae and traces with modified components and distribution depth

Variables		Orchestration		Migration			Choreography		
|frml|	bias	# msgs	progs	# msgs	|msgs|	progs	# msgs	# distr	progs
$\sim$2	×	1.97	6.15	1.37	12.05	22.08	3.39	1.19	6.83
	✓	1.93	5.83	0.52	4.80	16.05	0.59	0.18	5.95
$\sim$4	×	21.79	98.08	6.91	108.00	159.93	22.98	14.60	130.36
	✓	28.51	111.09	1.18	23.08	137.77	2.73	1.43	113.72
$\sim$8	×	193.11	833.46	26.67	944.77	1166.72	1041.97	655.42	1635.64
	✓	103.10	334.18	6.58	204.56	433.47	96.71	60.73	592.25
$\sim$16	×	653.20	2259.83	90.15	5828.51	4078.24	4136.77	2680.70	7271.81
	✓	361.54	1372.84	20.69	1802.93	1935.08	589.37	391.60	33981.28

in this study, since any number of monitoring cycles are allowed in between system cycles, neither approach has any delay.

Number and size of messages: Similar to the case of delay, in general (as shown in the empirical results) the number of messages required by an orchestration is less than that required by a choreography. However, this greatly depends on the topology of the tree. For example, having a distributed subformula $b_1 \wedge b_2$, sending updates for the conjunction is generally cheaper than sending updates for b_1 and b_2 separately. This phenomenon is hinted at in Table 1 where the results of the 3.15 depth are worse than those of depth 4 (where in general this should be the opposite). In other words, the performance of choreography is greatly dependent on how much the leaves can propagate their results towards the root of the tree without having to communicate. The hint is then confirmed in Table 2 where we intentionally biased the formula generation algorithm such that propositions from the same component are more likely to appear on the same branch. The results show a significant gain for the choreography approach, performing even better than orchestration for small formulae.

Progressions: Once more, the number of progressions behaves similarly to the number of messages.

Privacy and security concerns: In the case of orchestration, since a single component has visibility of all propositions, a security breach in that component would expose all the system information. On the contrary, generally speaking, no component has the full visibility of the system events in the case of choreography.

Clearly, none of the approaches ticks all the boxes. Rather, these experiments have shed some light as to when it makes more sense to use one approach over another depending on the size of the network, the structure of the LTL formula, the importance of issues such as privacy, frequency/size of messages, etc.

7 Related Work

The idea of splitting the progression of an LTL formula into subparts and propagating the results across a network is somewhat similar to the ideas used in parallel prefix networks [8]. In such networks intermediate results are evaluated in parallel and then combined to achieve the final result more efficiently. Furthermore, this work has two other main sources of inspiration: the work by Bauer and Falcone [4] about monitoring LTL properties in the context of distributed systems having a global clock, and the work by Francalanza et al. [6] which classifies modes of monitoring in the context of distributed systems. We have thus adapted the classification of distributed monitoring showing how orchestration, choreography, and migration can be applied to LTL monitors. We note, however, that we have introduced the global clock assumption which is not present in [6]. Without this assumption, our correctness theorem does not hold due to the loss of the total order between system events. From another point of view, we have classified the approach presented in [4] as a migration approach (using the terminology of [6]) and extended the work by presenting a choreography approach. Furthermore, we have also empirically compared the advantages and disadvantages of the approaches.

As pointed out in [4], decentralised monitoring is related to several techniques. We recall some of them and refer to [4] for a detailed comparison. One of the closest approaches is [10] which proposes to monitor MTTL formulae specifying the safety properties over parallel asynchronous systems. Contrary to [10], our approach considers the full set of (“off-the-shelf") LTL properties, does not assume the existence of a global observation point, and focuses on how to automatically split an LTL formula according to the architecture of the system.

Also, closely related to this paper is a monitoring approach of invariants using knowledge [7]. This approach leverages an apriori model-checking of the system to pre-calculate the states where a violation can be reported by a process acting alone. Both [7] and our approach try to minimize the communication induced by the distributed nature of the system but [7] (i) requires the property to be stable (and considers only invariants) and (ii) uses a Petri net model to compute synchronization points.

8 Conclusions and Future Work

In the context of distributed systems becoming increasingly ubiquitous, further studies are required to understand the variables involved and how these affect the numerous criteria which constitute good monitoring strategies. This would help architects to choose the correct approach depending on the circumstance.

This study shows that while choreography can be advantageous in specific scenarios such as in the case of systems with lots of components and formulae which can be shallowly distributed, generally it requires a significant number of messages and cannot fully exploit the potential of LTL simplification routines. We have noted that a substantial part of the messages required for choreography are in fact messages related to the maintenance of the network, i.e., respawning subparts of a formula. This means that LTL might not be the best candidate when going for a choreography. Contrastingly, non-progression-based monitoring algorithms where the monitors are not constantly modified, might lend themselves better to choreography.

We consider future work in three main directions: First, we would like to investigate how LTL equivalence rules can be used to make the choreography tree shallower. For example distributing $(a_1 \wedge a_2) \wedge ((a_3 \wedge b_1) \wedge b_2)$ might require two hops to reach a verdict while using associativity rules (obtaining $((a_1 \wedge a_2) \wedge a_3) \wedge (b_1 \wedge b_2)$), it can be easily reduced to one. Secondly, it would be interesting to consider the case where for each system cycle, the monitor only performs one cycle too. This introduces a delay for the choreography to reach the verdict and requires a more complex network to manage the dependencies across different time instants. Third, using other notations instead of LTL and/or different monitoring algorithms, particularly ones which are not progression-based, can potentially tip the balance more in favour of choreography approaches.

References

1. Bartocci, E.: Sampling-based decentralized monitoring for networked embedded systems. In: 3rd Int. Work. on Hybrid Autonomous Systems. EPTCS, vol. 124, pp. 85–99 (2013)
2. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Logic and Computation 20(3), 651–674 (2010)
3. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. (TOSEM) 20(4), 14 (2011)
4. Bauer, A.K., Falcone, Y.: Decentralised LTL monitoring. In: 18th Int. Symp. on Formal Methods. LNCS, vol. 7436, pp. 85–100. Springer (2012)
5. Falcone, Y., Cornebize, T., Fernandez, J.C.: Efficient and generalized decentralized monitoring of regular languages. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014: 34th IFIP Int. Conference on Formal Techniques for Distributed Objects, Components and Systems. LNCS, vol. 8461, pp. 66–83. Springer (2014)
6. Francalanza, A., Gauci, A., Pace, G.J.: Distributed system contract monitoring. J. Log. Algebr. Program. 82(5-7), 186–215 (2013)
7. Graf, S., Peled, D., Quinton, S.: Monitoring distributed systems using knowledge. In: Bruni, R., Dingel, J. (eds.) Proc. of the Joint 13th IFIP WG 6.1 Int. Conference and 31st IFIP WG 6.1 Int. Conference. LNCS, vol. 6722, pp. 183–197. Springer (2011)
8. Harris, D.: A taxonomy of parallel prefix networks. In: Signals, Systems and Computers. vol. 2, pp. 2213–2217 (2003)
9. Pnueli, A.: The temporal logic of programs. In: SFCS'77: Proc. of the 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. IEEE Computer Society (1977)
10. Sen, K., Vardhan, A., Agha, G., Rosu, G.: Decentralized runtime analysis of multithreaded applications. In: 20th Parallel and Distributed Processing Symposium (IPDPS). IEEE (2006)