Using Evolution Graphs for Describing Topology-Aware Prediction Models in Large Clusters

Matei Popovici

POLITEHNICA University of Bucharest
Splaiul Independentei nr. 313, Bucharest, Romania, Postal Code 060042
matei.popovici@cs.pub.ro

Abstract. We present and formally investigate a modelling method suitable for describing events and time-dependent properties and for performing possibly complex reasoning tasks regarding the evolution of dynamic domains. Our proposal consists of a distinguished data structure called evolution graph, and a logical language ($L_{\mathcal{H}}$) used for identifying temporal patterns in evolution graphs. First, we define and study the complexity of the model checking problem for our language. We then investigate the relation between our language and the well-known Computation Tree Logic (CTL), both in terms of complexity and expressive power. Finally, we apply our method for solving a well-known problem from High Performance Computing (HPC): the extraction of topology information from event logs produced by supercomputers.

Keywords: temporal knowledge representation, temporal logic, high-performance computing.

1 Introduction

The world of high performance computing (HPC) is preparing for the exascale era, and according to recent studies [10,16], 20% or more of the computing capacity in a large system is wasted due to failures and recoveries. An alternative approach to classical fault tolerance that might optimize this process is failure avoidance, where the occurrence of a fault is predicted and preventive measures are taken. For this, monitoring systems require a reliable prediction method to give information on what will be generated by the system and at what location. To the best of our knowledge, all other log analysis methods in the literature [23,14,18,11,12] propose theoretical models and algorithms for detecting, predicting or characterizing events, without studying the impact of their methods on large-scale HPC systems such as Blue Gene.

In order to develop a sound and robust prediction system and to fully characterize its performance, a suitable modelling method is required. The success of such a method relies on: (i) the ability to describe events and time-dependent properties of systems with possibly non-deterministic behaviour, and (ii) to perform complex reasoning tasks about the temporal relations between such events and properties.

M. Fisher et al. (Eds.): CLIMA XIII 2012, LNAI 7486, pp. 94–109, 2012.

In this paper, we introduce a new modelling method for reasoning about time-dependent properties. Based on it, we extract the topology of large-scale machines after investigating the log files generated by such systems. In our approach, a domain's history is recorded by a distinguished structure called *evolution graph*. It consists of: (i) **action nodes** which model instantaneous stimuli that occur at fixed moments of time, and affect the current state of the domain, (ii) **hypernodes** which capture (discrete) moments of time. The set of hypernodes can be seen as a partitioning of the set of action nodes such that all action nodes belonging to the same hypernode are simultaneous; (iii) **quality edges** which model time-dependent properties which span action nodes. For a given quality edge $q = (a, b)$, we interpret the action node a as the stimulus responsible for *creating* (or introducing) the quality q. Similarly, b is seen as the action node that ceases (destroys) q.

Evolution graphs capture the dynamics of a given domain. In order to provide with a domain-dependent semantics, action nodes and quality edges are labelled with first-order predicates. For a more detailed description of evolution graphs, as well an extended set of examples, we direct the reader to [20].

An example of an evolution graph, capturing the behaviour of (a small part of) a HPC system, can be seen in Figure 1. For simplicity, we have omitted labelling action nodes. Here, two nodes n and m sharing the same rack, experience network failures at different moments of time. This is modelled by the qualities *Net_fail(n)* and *Net_fail(m)*. In the time-slot when both these qualities do not exist, a *network communication* binary quality describes the successful communication between network nodes n and m.

HPC systems consist of a high number of nodes that are usually placed in a hierarchical architecture. For example, in BlueGene systems, nodes are gathered into midplanes and multiple midplanes form a rack. Certain errors in the system, such as networking faults, affect multiple nodes depending on their relative position within the architecture. In another study [11], it has been observed that propagation paths for different error types follow closely the way components are connected in the system. For example if a fan breaks, all nodes sharing the same rack will be affected. The topology of a system is usually not known in advance. This forces failure prediction algorithms to rely on heuristics for tracking the locations of the failure's effects in the system. In our experiments, we use the logs generated by the Blue Gene/L system. This system is one of the few large-scale machines that offer a detailed view of its topology. This information is useful in having a better understanding of the prediction topologies we obtain with our method. Also, the Blue Gene systems are widely used machines in HPC and are representative for today's large-scale systems. For more details on the system-architecture see [1].

In this paper, we focus on the computational properties of the language $L_{\mathcal{H}}$, used for reasoning about evolution graphs, and on the implementation of a topology extraction system. In Section 2, we formally introduce evolution graphs, and in Section 3 we describe the language $L_{\mathcal{H}}$ and its semantics. In Section 4 we introduce the CTL language, and use it for proving some complexity results as

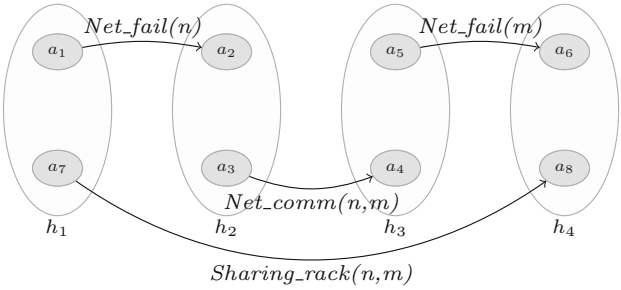

Fig. 1. A simple evolution graph

well as for gaining a better understanding on why $L_{\mathcal{H}}$ is more suitable for topology extraction. In Section 6 we further explore the relationship between $L_{\mathcal{H}}$ and temporal logics in general, as well as look at other similar approaches. Finally, in Section 5, we use a fragment of $L_{\mathcal{H}}$ for the implementation of a topology extraction system for fault propagation in HPC large-scale systems, that optimizes the prediction model presented in [2].

2 The Evolution Graph

Definition 1 (evolution graph). *An evolution graph is defined as a structure* $\mathcal{H} = \langle H, A, \mathcal{T}, E \rangle$*, where:*

- H *is a set of hypernodes, and* A *is a set of nodes;*
- $\mathcal{T} : A \to H$ *is an onto (surjective) function, sending each node to the hypernode it belongs to;*
- $E \subseteq A^2$ *is a directed edge relation with the following restrictions: (i) for any* $a \in A$*, there is* at most *one* $b \in A$ *such that* $(a, b) \in E$ *or* $(b, a) \in E$*. (i.e. any node* creates *or* destroys *a unique edge) and (ii)* $(a, b) \in E \implies \mathcal{T}(a) \neq \mathcal{T}(b)$*;*

Let $\sigma = \{R_1, \ldots, R_n\}$ be a vocabulary. The elements R_i are relation symbols each having a certain arity designated by $arity(R_i)$. In general, we require that σ contains two types of relation symbols: **quality labels** and **action labels** ($\sigma = \sigma_Q \cup \sigma_A$).

Definition 2 (Label structure). *A labeling structure is a* $\sigma_Q \cup \sigma_A$*-structure* $\mathcal{A} = \langle I, R_1^I, \ldots, R_n^I \rangle$*, where:*

- I *is a set of individuals (the structure's universe);*
- *for each* $R_k \in \sigma_Q$ *and* $\overline{i} \in R_k^I$*, the pair* $\langle R_k, \overline{i} \rangle$ *is a* ***quality label****;*
- *for each* $R_k \in \sigma_A$ *and* $\overline{i} \in R_k^I$*, the pair* $\langle R_k, \overline{i} \rangle$ *is an* ***action label****;*

We designate a quality or an action label by $R_k^I(\overline{i})$, with R_k from the appropriate vocabulary. We make a slight abuse of notation and use R^I to designate the

appropriate set of quality or action labels, viewed as pairs $\langle R, \overline{i} \rangle$, instead of a set of tuples $\overline{i}$.

A *path* in an evolution graph $\mathcal{H}$ is a finite sequence $\lambda = a_1, a_2, \ldots, a_n$ of nodes from $\mathcal{H}$ such that, for any two consecutive nodes a_i and a_{i+1}, either of the following is true: (i) $E(a_i, a_{i+1})$ (there is an edge between a_i and a_{i+1}) or (ii) $\mathcal{T}(a_i) = \mathcal{T}(a_{i+1})$ (there exists a hypernode that contains both a_i and a_{i+1}). Intuitively, each path is a temporally ordered sequence of events and properties. An example can be seen in Fig. 1, where $\lambda = a_1, a_2, a_3, a_4, a_5, a_6$ is such a path. An evolution graph $\mathcal{H}$ is *cycle-free*, if there is no pair of nodes $a, b \in A$, having a path from a to b, and one from b to a i.e. $\mathcal{H}$ contains no cycles. In the following, we discuss cycle-free evolution graphs only.

Definition 3 (labelled evolution graph). *An $\mathcal{A}$-labelled evolution graph is a structure $\mathcal{H}_\mathcal{A} = \langle \mathcal{H}, \mathcal{L}_A, \mathcal{L}_Q \rangle$ consisting of a evolution graph and two* total *labelling functions:*

- $\mathcal{L}_Q : E \to \cup_{R \in \sigma_Q} R^I$*; $\mathcal{L}_Q$ maps each edge to a quality label;*
- $\mathcal{L}_A : A \to \cup_{R \in \sigma_A} R^I$ *maps each node to an action label;*

We refer to labeled edges and nodes as **quality edges** and **action nodes**, respectively.

3 The Language $L_\mathcal{H}$

$L_\mathcal{H}$ is defined over a labelling structure $\mathcal{A}$ and expresses temporal relations between quality edges and action nodes from $\mathcal{A}$-labelled evolution graphs. $L_\mathcal{H}$ contains two types of formulae: *Q-formulae* and *A-formulae*. The former evaluate to quality edges, and the latter to action nodes. Each action node a and quality edge (a, b) designates a moment of time $\mathcal{T}(a)$ and an interval $[\mathcal{T}(a), \mathcal{T}(b)]$, respectively. Given two quality edges $q = (a, b)$, $q' = (a', b')$, and action nodes c, d we introduce the following temporal operators, which abbreviate the traditional interval relationships introduced by Allen [22].

- (before): q **b** q' iff b occurs before a';
- (just before): q **jb** q' iff a occurs before a' but b does not occur before a';
- (starts same): q **s** q' iff a coincides with a';
- (ends same): q **e** q' iff b coincides with b';
- (meets): q **m** q' iff b coincides with a';
- (included): q **i** q' iff a occurs after a' and b occurs before b';
- $c \succ d$ iff c occurs before d;
- $c \equiv d$ iff c and d share the same hypernode;
- (creates/created by): c $\mathbf{c}_A$ q and q $\mathbf{c}_Q$ c iff $c = a$;
- (destroys/destroyed by): c $\mathbf{d}_A$ q and q $\mathbf{d}_Q$ c iff $c = b$;

For instance q **b** q' is true if q's interval occurs *before* q''s, and they do not overlap. An example of this relation can be found in Fig. 1, between qualities *Net_fail(n)* and *Net_fail(m)*. q **jb** q' is true if q's interval occurs *before* q''s and

they overlap. We say q is *just before* q'. In the same figure *Sharing_rack(n,m)* occurs just before *Net_fail(m)*. **s** and **e** abbreviate *starts at the same time with* and *ends at the same time with*, respectively. **m** abbreviates *meets*, thus designating qualities that end at the same time other qualities start. **i** abbreviates inclusion. **c** and **d** refer to the creation and destruction of qualities, respectively. The inverse relations are defined in a similar manner. Notice that $\succ$ is only a *partial* order over action nodes, since if, for instance, $\mathcal{H}$ contains disconnected components, one could say nothing about the temporal order between elements from distinct components.

Let $\mathbb{V}ars$ be a set of variables, $\mathcal{A}$ be a labelling structure over vocabulary $\sigma = \sigma_Q \cup \sigma_A$, and $\mathcal{H}_\mathcal{A}$ be a $\mathcal{A}$-labelled evolution graph. We designate by $\propto$ any connective for qualities in $\mathfrak{C}_Q = \{$ **b** , **jb** , **s** , **e** , **m** , **i** $\}$, by $\triangleright$ any connective for actions in $\mathfrak{C}_A = \{\succ, \equiv\}$.

Definition 4 ($L_\mathcal{H}$ syntax). *Let $\mathbb{V}ars$ be a set of variables, $\mathcal{A}$ be a $\sigma_Q \cup \sigma_A$-structure and $\mathcal{H}_\mathcal{A}$ be a $\mathcal{A}$-labelled evolution graph. Also, let $X \in \{Q, A\}$, and $\dagger_Q \in \mathfrak{C}_Q$ and $\dagger_A \in \mathfrak{C}_A$.*

The syntax of a X-formula is recursively defined with respect to $\mathcal{A}$, as follows:

1. *if $R \in \sigma_X$ with $arity(R) = n$ and $\overline{x} \in \mathbb{V}ars^n$, then $R(\overline{x})$ is an **atomic** X-formula (or an atom).*
2. *if ϕ is a X-formula then (ϕ) is also a X-formula;*
3. *if ϕ, ψ are X-formulae then $\phi \dagger_X \psi$ and $\phi\neg \dagger_X \psi$ are also X-formulae. We call $\dagger_X$ a **positive** connective and $\neg\dagger_X$ a **negative/negated** connective.*
4. *Let $R \in \sigma_X$, ϕ, ψ, ω be X-formulae and $\dagger_X$ designate either a positive or negated connective. If ϕ has any of the following forms: (i) $\phi = R(\overline{x})$, (ii) $\phi = R(\overline{x}) \dagger_X \psi$ or (iii) $\phi = (R(\overline{x}) \dagger \omega) \dagger_X \psi$, then it is R-**compatible**. Moreover, if ϕ and ψ are both R-compatible, then $\phi \wedge \psi$ and $\phi \vee \psi$ are X-formulae, and R-compatible as well.*

Negation in $L_\mathcal{H}$ has a restricted use. For instance:

$$(a)\ \text{Net\_fail(x)}\ \neg\mathbf{b}\ \text{Net\_comm(x,y)} \qquad (b)\ \neg\text{Net\_fail(x)} \tag{1}$$

the formula from Equation 1 (a) is well-formed, however Equation 1 (b) is not well-formed. The intuition is the following: we are interested in identifying classes of properties of a certain type (i.e. being labelled with a quality label of a distinct relational symbol in σ_Q). For instance, Equation 1 (a) characterizes the set of properties *of the type Net_fail*, such that they do not occur before some qualities *of the type Net_comm*. If we choose to interpret ¬Net_fail(x) as the complement of the edge relation E from $\mathcal{H}$, with respect to all edges labelled Net_fail(i) (with $i \in I$), then $L_\mathcal{H}$-formulae could characterize edges with arbitrary labels.

The restrictions for the usage of the traditional connectives $\wedge$ and $\vee$ have the same motivation as in the case of negation, i.e. to preserve *property types*.

$L_\mathcal{H}$ formulae are evaluated over *paths* from labelled evolution graphs $\mathcal{H}$. We assume there is no prior information regarding the temporal order of hypernodes. In the absence of such information, a partial ordering of hypernodes can be

inferred only by inspecting sequences of quality edges. We lift this assumption in Section 5, where we consider evolution graphs with temporally ordered hypernodes. The evaluation of $L_{\mathcal{H}}$-formulae is defined by the mappings $\|\cdot\|^Q_{\mathcal{H}} : L_{\mathcal{H}} \to 2^E$ and $\|\cdot\|^A_{\mathcal{H}} : L_{\mathcal{H}} \to 2^A$. If ϕ is a Q or A formula, then $\|\phi\|^Q_{\mathcal{H}}$ and $\|\psi\|^A_{\mathcal{H}}$ are the set of quality edges or action nodes satisfying ϕ and ψ, respectively. Each temporal connective $\propto \in \mathfrak{C}_Q$ requires the existence (or absence) of paths between nodes belonging to the corresponding qualities. For instance, in Fig. 1, given $q = (a_1, a_2)$ and $q' = (a_5, a_6)$, for q **b** q' to be true, there must be (at least) one path between a_2 and a_5. Such a path exists: $\lambda = a_2, a_3, a_4, a_5$. Inclusion (**i**) requires two paths. In the same figure (a_3, a_4) **i** (a_7, a_8) is true because there is a path from a_1 to a_3, and one from a_4 to a_6. We further note that the temporal ordering of hypernodes (e.g. $h_1, \ldots h_4$) is not generally known in advance, but can be deduced based on the existing quality edges (in our example, the *Net_fail* and *Net_comm* quality edges).

Given two quality edges q, q' and a temporal connective $\propto$, we write $\lambda_\propto(q, q')$ to refer to the path conditions required by $\propto$ between q and q'. These are straightforward from the introduction of temporal operators from Section 3. We use a similar notation $(\lambda_{\triangleright}(a, a'))$, for path conditions between actions.

Definition 5 (Semantics). *The semantics of Q and A-formulae are defined as follows. Let $\overline{i}$ be a set of individuals from the universe of a labelling structure $\mathcal{A}$, a, a' denote action nodes, and q, q' denote quality edges from a $\mathcal{A}$-labelled evolution graph $\mathcal{H}_\mathcal{A}$. The tuple $\overline{\alpha}$ and $X \in \{Q, A\}$ are used in the following way: whenever $X = Q$, $\overline{\alpha}$ designates a quality edge, and whenever $X = A$, $\overline{\alpha}$ designates an action node. Finally, by $\phi_{[\overline{x}\setminus\overline{i}]}$, we refer to the formula obtained from ϕ by replacing all variables from $\overline{x}$ with individuals from $\overline{i}$.*

1. $\|\phi(\overline{x})\|^X_{\mathcal{H}} = \bigcup_{\overline{i}}\{\overline{\alpha} \in \|\phi_{[\overline{x}\setminus\overline{i}]}\|^X_{\mathcal{H}}\}$;
2. $\|R(\overline{i})\|^X_{\mathcal{H}} = \{\overline{\alpha} : \mathcal{L}_X(\overline{\alpha}) = R(\overline{i})\}$;
3. $\|(\phi) \propto \psi\|^Q_{\mathcal{H}} = \{q \in \|\phi\|^Q_{\mathcal{H}} : \exists q' \in \|\psi\|^Q_{\mathcal{H}} \text{ and } \lambda_\propto(q, q')\}$
4. $\|(\phi) \triangleright \psi\|^A_{\mathcal{H}} = \{a \in \|\phi\|^A_{\mathcal{H}} : \exists a' \in \|\psi\|^A_{\mathcal{H}} \text{ and } \lambda_{\triangleright}(a, a')\}$
5. $\|\phi \neg\dagger \psi\|^X_{\mathcal{H}} = \|\phi\|^X_{\mathcal{H}} \setminus \|\phi \dagger \psi\|^X_{\mathcal{H}}$;
6. $\|\phi \dagger (\psi)\|^X_{\mathcal{H}} = \|\phi \dagger \psi\|^X_{\mathcal{H}}$
7. $\|R(\overline{i}) \dagger \psi\|^X_{\mathcal{H}} = \|(R(\overline{i})) \dagger \psi\|^X_{\mathcal{H}}$;
8. $\|\phi \vee \psi\|^X_{\mathcal{H}} = \|\phi\|^X_{\mathcal{H}} \cup \|\psi\|^X_{\mathcal{H}}$;
9. $\|\phi \wedge \psi\|^X_{\mathcal{H}} = \|\phi\|^X_{\mathcal{H}} \cap \|\psi\|^X_{\mathcal{H}}$;
10. $\|\phi\ \boldsymbol{c}_Q\ \psi\|^Q_{\mathcal{H}} = \{(a, b) \in \|\phi\|^Q_{\mathcal{H}} : a \in \|\psi\|^A_{\mathcal{H}}\}$
11. $\|\phi\ \boldsymbol{d}_Q\ \psi\|^Q_{\mathcal{H}} = \{(a, b) \in \|\phi\|^Q_{\mathcal{H}} : b \in \|\psi\|^A_{\mathcal{H}}\}$
12. $\|\phi\ \boldsymbol{c}_A\ \psi\|^A_{\mathcal{H}} = \{a \in \|\phi\|^A_{\mathcal{H}} : \exists (a, b) \in \|\psi\|^Q_{\mathcal{H}}\}$
13. $\|\phi\ \boldsymbol{d}_A\ \psi\|^A_{\mathcal{H}} = \{b \in \|\phi\|^A_{\mathcal{H}} : \exists (a, b) \in \|\psi\|^Q_{\mathcal{H}}\}$

Rule 1 states that evaluating a formula with parameters reduces to the evaluation of formulae obtained by all possible substitutions of variables from $\mathbb{V}ars$ with individuals from the labelling structure universe, I. Notice that in rules 2 to 9 the evaluation order of formulae matters. The parentheses and negation enforce the evaluation of the left-side formula. In the absence of parentheses, formulae

evaluate from right to left. For instance, in the evolution graph from Fig. 1, let us designate $Q_1 = \text{Net\_fail}(n)$, $Q_2 = \text{Net\_comm}(n, m)$ and $Q_3 = \text{Net\_fail}(m)$. Then, we have that $\|(Q_1 \ \mathbf{b} \ Q_2) \ \mathbf{m} \ Q_3\|^Q_{\mathcal{H}} = \{(a, b)\}$ while $\|Q_1 \ \mathbf{b} \ Q_2 \ \mathbf{m} \ Q_3\|^Q_{\mathcal{H}} = \emptyset$.

Also, negation should not be seen as interpreting inverse relations. Consider the evolution graph from Fig. 1, but with the quality edge (a_1, a_2) removed. Then, $\|Q_3 \ \neg\mathbf{b} \ Q_1\|^Q_{\mathcal{H}} = \{(a_5, a_6)\}$, whereas Q_3 *after* Q_1 would evaluate to the empty set. The main difference is that negation requires the **absence** of a relation, whereas an inverse relation requires the **presence** of an opposite relation.

Definition 6 (Model checking). *Let $\mathcal{H}$ be a $\mathcal{A}$-labelled evolution graph, and ϕ an X-formula in $L_{\mathcal{H}}$ ($X \in \{Q, A\}$). We write $\mathcal{H}, \overline{\alpha} \models_X \phi$ iff $\overline{\alpha} \in \|\phi\|^X_{\mathcal{H}}$. The problem whether $\mathcal{H}, \overline{\alpha} \models_X \phi$ is the X-model checking problem for $L_{\mathcal{H}}$.*

4 $L_{\mathcal{H}}$ and Computation Tree Logic

The path conditions described in the semantics of $L_{\mathcal{H}}$ can also be expressed in the well-known temporal logic CTL (Computation Tree Logic). CTL is strictly less expressive than $L_{\mathcal{H}}$, since it doesn't allow first-order predicates. Nevertheless, the language $L^*_{\mathcal{H}}$, obtained by restricting the labelling structure from $L_{\mathcal{H}}$ to propositional symbols, can be embedded into CTL, over a particular type of labelled transition systems. This result has a twofold utility: (i) it provides a means of comparing the efficiency of $L^*_{\mathcal{H}}$ and CTL, and (ii) it is useful for obtaining complexity results for both $L^*_{\mathcal{H}}$ and the full $L_{\mathcal{H}}$.

In the following, we give a brief introduction to the syntax and semantics of CTL. For a detailed description see [24].

Let $\mathcal{P}$ be a set of propositional symbols and $\mathcal{L}$ be a finite set of labels. Given $p \in \mathcal{P}$, $a \in \mathcal{L}$, the language CTL consists of formulae generated by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \mathsf{E}\langle a\rangle\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}(\varphi \ \mathcal{U} \ \varphi) \tag{2}$$

The other traditional connectives are introduced as abbreviations. For instance: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathsf{E}\langle a\rangle\varphi \equiv \neg\mathsf{E}\langle a\rangle\neg\varphi$.

Traditionally, CTL formulae are interpreted over *Kripke structures*. Here, we shall use *labelled transition systems* (LTS) instead. For details on evaluating CTL formulae over LTSs, see [8].

A LTS is a structure $\mathcal{M} = \langle S, (R_i)_{i\in\mathcal{L}}, \pi\rangle$ here S is a set of states, each $R_i \subseteq S \times S$ is an i-labelled transition relation, one for each label $i \in \mathcal{L}$, and $\pi : S \to 2^{\mathcal{P}}$ is an *interpretation function* associating for each state s a set $\pi(s)$ consisting of the propositional symbols that hold in s. The size of a LTS $\mathcal{M}$ is the sum of the state-space and the number of transitions of each type: $|\mathcal{M}| = \sum_{s\in S} |\pi(s)| + \sum_{i\in\mathcal{L}} |R_i|$.

A *path* in a LTS is a possibly infinite sequence $\theta = s_1, s_2, \ldots s_n, \ldots$ of states such that, for any s_i, s_{i+1} there is some transition from s_i to s_{i+1}. We use $\theta[i]$ to denote the i-th state in the sequence θ and $\theta[i, \infty]$ to denote the subpath of θ starting from i: $\theta[i]\theta[i+1]\ldots$. For a state $s \in \mathcal{S}$ in some LTS $\mathcal{M}$, we write $\Theta_{\mathcal{M}}(s)$ to designate the set of paths in $\mathcal{M}$ that start in s.

Given a LTS $\mathcal{M} = \langle S, (R_i)_{i\in\mathcal{L}}, \pi\rangle$, a state $s \in \mathcal{S}$, and an interpretation function π, the semantics for CTL is defined as follows:

- $\mathcal{M}, s \models_{\text{CTL}} p$ iff $q \in \pi(s)$;
- $\mathcal{M}, s \models_{\text{CTL}} \neg\varphi$ iff $\mathcal{M}, s \not\models_{\text{CTL}} \varphi$;
- $\mathcal{M}, s \models_{\text{CTL}} \varphi_2 \wedge \varphi_2$ iff $\mathcal{M}, s \models_{\text{CTL}} \varphi_1$ and $\mathcal{M}, s \models_{\text{CTL}} \varphi_2$
- $\mathcal{M}, s \models_{\text{CTL}} \mathsf{E}\langle a\rangle\varphi$ iff there is a path $\theta \in \Theta_{\mathcal{M}}(s)$ such that $R_a(\theta[0], \theta[1])$ and $\mathcal{M}, \theta[1] \models_{\text{CTL}} \varphi$;
- $\mathcal{M}, s \models_{\text{CTL}} \mathsf{E}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ iff there is a path $\theta \in \Theta_{\mathcal{M}}(s)$ such that $\mathcal{M}, \theta[i] \models_{\text{CTL}} \varphi_2$, for some $i \geq 0$ and $\mathcal{M}, \theta[j] \models_{\text{CTL}} \varphi_1$ for all j such that $0 \leq j < i$;

We also introduce some ad-hoc notations. Given a set of labels L:

- $\mathsf{E}\langle L\rangle\phi \equiv \bigvee_{i\in L} \mathsf{E}\langle i\rangle\phi$;
- $\mathsf{E}(\varphi_1 \,\mathcal{U}_L\, \varphi_2) \equiv \mathsf{E}((\varphi_1 \wedge \langle L\rangle\top) \,\mathcal{U}\, (\varphi_1 \wedge \langle L\rangle\varphi_2))$;

For instance $\mathsf{E}\langle a, b\rangle p$ is true in those states s that have access to states s' where p is true, via either a or b-transitions, and $(\neg q)\,\mathcal{U}_{\{a,b\}}\, p$ is true if there is a path consisting of either a or b transitions on which q is false in each state (at least) until p becomes true.

Proposition 1 (CTL model checking [7]). *Given a LTS $\mathcal{M}$, a state s and a CTL formula φ, checking whether $\mathcal{M}, s \models_{CTL} \varphi$ is* PTIME*-complete.*

4.1 The Language $L^*_{\mathcal{H}}$ and CTL

In this section, we describe a fragment of $L_{\mathcal{H}}$ having as labels propositional symbols only, and show that this fragment can be embedded in CTL. Let $\mathcal{A}^* = \langle I, R^I_1, \ldots, R^I_n\rangle$ be any labelling structure over a vocabulary with *unary* relation symbols, where $I = \{i_1, i_2, \ldots, i_n\}$ and for each $1 \leq k \leq n$ we have $R^I_k = \{i_k\}$. Under these restrictions, $\mathcal{A}^*$ becomes nothing more than a set of propositional symbols: $\mathcal{A}^* = \{p_k \equiv R^I_k(i_k) : 1 \leq k \leq n\}$. Let $L^*_{\mathcal{H}}$ denote the subset of the language $L_{\mathcal{H}}$, that is built over $\mathcal{A}^*$, and whose formulae contain no variables.

Starting from an $\mathcal{A}^*$-labelled evolution graph $\mathcal{H}$ we build a LTS $\mathcal{M}^{\mathcal{H}}$ in the following way: (i) the set of labels is $\mathcal{L} = \{h, c, d, c^{-1}, d^{-1}\}$, the set of propositional symbols is $\mathcal{P} = \mathcal{A}^*$ (ii) for each action node $a \in A$ and for each quality edge $q \in E$, we build states s_a and s_q, respectively. The interpretation function π is built as follows: if $\mathcal{L}_Q(q) = p_k$, then $p_k \in \pi(s_q)$ and if $\mathcal{L}_A(a) = p_k$, then $p_k \in \pi(s_a)$. Thus, quality edges and action nodes are transformed into states, and their labels become propositional symbols that hold in these states. (iii) transitions are built in the following way: for any two simultaneous action nodes ($\mathcal{T}(a) = \mathcal{T}(b)$), we build a h-transition between the corresponding states: $R_h(s_a, s_b)$. By this construction simultaneous action nodes form h-labelled cliques in $\mathcal{M}^{\mathcal{H}}$. For any quality edge $q = (a, b) \in E$, we build $R_c(s_a, s_q)$, $R_{c^{-1}}(s_q, s_a)$, $R_d(s_q, s_b)$ and $R_{d^{-1}}(s_b, s_q)$. Thus, c-labelled transitions bind action states to the quality states they introduce, and d-labelled transitions bind quality states to the action states that destroy them. c^{-1} and d^{-1}-labelled transitions make the inverse bindings. The entire construction of $\mathcal{M}^{\mathcal{H}}$ can be done in deterministic polynomial time, with respect to the size of $\mathcal{H}$.

Definition 7 (Embedding $L^*_{\mathcal{H}}$ in CTL). *Let ϕ, ψ be formulae in $L^*_{\mathcal{H}}$. We build equivalent CTL formulae, by applying a transformation procedure $\mathfrak{T} : L^*_{\mathcal{H}} \rightarrow$ CTL, recursively defined as follows:*

1. *for $p_k \in L^*_{\mathcal{H}}$, $\mathfrak{T}(p_k) = p_k$;*
2. *$\mathfrak{T}(\phi \ \boldsymbol{b} \ \psi) = \mathfrak{T}(\phi) \wedge \mathsf{E}(\top \mathcal{U}_{\{c,h,d\}} \mathfrak{T}(\psi))$;*
3. *$\mathfrak{T}(\phi \ \boldsymbol{s} \ \psi) = \mathfrak{T}(\phi) \wedge \mathsf{E}\langle c^{-1}\rangle\mathsf{E}\langle h\rangle\mathsf{E}\langle c\rangle\mathfrak{T}(\psi)$;*
4. *$\mathfrak{T}(\phi \ \boldsymbol{c}_Q \ \psi) = \mathfrak{T}(\phi) \wedge \mathsf{E}\langle c\rangle\mathfrak{T}(\psi)$;*
5. *$\mathfrak{T}(\phi \neg \dagger \psi) = \mathfrak{T}(\phi) \wedge \neg\mathfrak{T}(\phi \dagger \psi)$;*
6. *$\mathfrak{T}((\phi \dagger_1 \phi') \dagger_2 \psi) = \mathfrak{T}(\phi \dagger_1 \phi') \wedge \mathfrak{T}(\phi \dagger_2 \psi)$;*
7. *$\mathfrak{T}(\phi \wedge \psi) = \mathfrak{T}(\phi) \wedge \mathfrak{T}(\psi)$;*
8. *$\mathfrak{T}(\phi \vee \psi) = \mathfrak{T}(\phi) \vee \mathfrak{T}(\psi)$;*

The relation between *action* states and *quality* states can be expressed in CTL using modalities. For instance $\mathsf{E}\langle c\rangle p_k$ is true in all action states that *create* a quality state *labelled* p_k. $\mathsf{E}\langle h\rangle p_k$ is true in all action states which are simultaneous with an action state labelled p_k. The transformation rule 2 gives us a formula that is true in a state where $\mathfrak{T}(\phi)$ is true, and there is a path consisting of *c*,*h* or *d*-labelled edges on which $\mathfrak{T}(\psi)$ will eventually become true.

Intuitively, all *local* relations such as *starts same* or *meets* are described using the *in the next state* operators $\langle a\rangle$ (and other boolean connectives), and those expressing *non-local* relations (i.e. arbitrary path conditions) such as *before* or *includes* are described using the CTL $\mathcal{U}$ (Until).

The translations for other temporal connectives for Q-formulae: $\mathbf{e}$, $\mathbf{jb}$, $\mathbf{m}$, $\mathbf{i}$ and $\mathbf{d}_Q$ and those for A-formulae: $\succ$, $\prec$, $\equiv$ and $\mathbf{d}_A$ are purely technical, and follow the same intuition. Due to limited space, we omit these definitions.

Proposition 2. *Let $\mathcal{H}$ be a $\mathcal{A}^*$-labelled evolution graph, $\phi \in L^*_{\mathcal{H}}$, $q \in E$ and $a \in A$. Then $\mathcal{H}, q \models_Q \phi$ iff $\mathcal{M}_{\mathcal{H}}, s_q \models_{CTL} \mathfrak{T}(\phi)$ and $\mathcal{H}, a \models_A \phi$ iff $\mathcal{M}_{\mathcal{H}}, s_a \models_{CTL} \mathfrak{T}(\phi)$.*

Proof (Sketch): In the following, we discuss Q-formulae only. The case for A-formulae is analogous. The property we prove is:

$$\mathcal{H}, (a, b) \models_Q \phi \iff \mathcal{M}_{\mathcal{H}}, s_{(a,b)} \models_{\text{CTL}} \mathfrak{T}(\phi)$$

The proof is done by structural induction over the construction of formulae ϕ. The basis case is for $\phi = p_k$. Since $\mathfrak{T}(p_k) = p_k$, $\mathcal{H}, q \models_Q p_k \iff \mathcal{M}_{\mathcal{H}}, s_q \models_{\text{CTL}} p_k$ trivially holds, from the construction of $\mathcal{M}_{\mathcal{H}}$. For each of the semantic rules 3...13 described in Definition 5, an induction step is required. Here, we will confine ourselves to rule 2, where $\propto = \mathbf{b}$. The remaining cases can be treated in a similar way.

Assume $\mathcal{H}, (a, b) \models_Q p_k \ \mathbf{b} \ \psi$, for some $p_k \in \mathcal{A}^*$. Therefore: (i) $\mathcal{H}, (a, b) \models_Q p_k$ and (ii) there is a quality edge $(a', b') \in \|\psi\|^Q_{\mathcal{H}}$ and (iii) a path λ, such that λ connects action nodes b and a': $\lambda = c_1, c_2, \ldots, c_n$, where $c_1 = b$ and $c_n = a'$. From (i), (ii) and the induction hypotheses, it follows that $\mathcal{M}_{\mathcal{H}}, s_{(a,b)} \models_{\text{CTL}} p_k$ and $\mathcal{M}_{\mathcal{H}}, s_{(a',b')} \models_{\text{CTL}} \mathfrak{T}(\phi)$. For each c_i, c_{i+1} in λ, we have either $E(c_i, c_{i+1})$

or $\mathcal{T}(c_i) = \mathcal{T}(c_{i+1})$. Therefore, in $\mathcal{M}_{\mathcal{H}}$, between s_{c_i} and $s_{c_{i+1}}$ there is either a quality state s_q such that $R_c(s_{c_i}, s_q)$ and $R_d(s_q, s_{c_{i+1}})$ or there is a h-transition: $R_h(c_i, c_{i+1})$. It immediately follows that, in $\mathcal{M}_{\mathcal{H}}$, there is a path θ from s_b to $s_{a'}$, consisting of c, d, or h-transitions. Since $\mathcal{H}, s_{(a,b)} \models_{\text{CTL}} p_k$ and $\mathcal{H}, s_{(a',b')} \models_{\text{CTL}} \mathfrak{T}(\psi)$, the existence of θ between s_b to $s_{a'}$ makes $\mathcal{H}, s_{(a,b)} \models_{\text{CTL}} p_k \wedge \mathsf{E}(\top\, \mathcal{U}_{\{c,h,d\}}\, \mathfrak{T}(\phi))$ true. Therefore $\mathcal{H}, s_{(a,b)} \models_{\text{CTL}} \mathfrak{T}(p_k\ \mathbf{b}\ \psi)$. The second part of the implication is shown similarly. □

Proposition 3 ($L^*_{\mathcal{H}}$-model checking). *The Q and A-model checking problems for $L^*_{\mathcal{H}}$ are in* PTIME.

Proof (Sketch): Given a evolution graph $\mathcal{H}$, a Q (or A) formula $\phi \in L^*_{\mathcal{H}}$, and $\overline{\alpha} \in E$ (or $\overline{\alpha} \in A$), we build a LTS $\mathcal{M}_{\mathcal{H}}$, and the transformed formula $\mathfrak{T}(\phi)$. The total construction is done in deterministic polynomial time. Finally, we solve $\mathcal{M}_{\mathcal{H}}, s_{\overline{\alpha}} \models_{\text{CTL}} \mathfrak{T}(\phi)$. Proposition 2 guarantees that the answer to the above problem is also an answer for the problem $\mathcal{H}, \overline{\alpha} \models_X \phi$. □

Proposition 3 does not imply completeness for PTIME, nor does it provide with an efficient mechanism for $L^*_{\mathcal{H}}$ model checking. Indeed, the transformation from Section 4.1 provides a solution for $L^*_{\mathcal{H}}$ model checking via CTL model checking, but this approach is not necessarily optimal, since the CTL formulae we model check, might depend in size to the size of the LTS. As we will further see, a more precise bound on $L^*_{\mathcal{H}}$ is unnecessary as the model checking problem for the full $L_{\mathcal{H}}$ is much harder. In order to obtain a hardness result, we use a reduction to the conjunctive query satisfaction problem. For details on conjunctive queries, see [15].

Definition 8 (Conjunctive sentences). *Let σ be an arbitrary vocabulary. A sentence φ over σ is* conjunctive *if it has the following form: $\varphi = \exists \overline{x} \bigwedge_{i=1}^{k} C_k(\overline{y_k})$ where each C_k is in σ and each $\overline{y_k}$ is a tuple consisting of variables from $\overline{x}$ and constant symbols from σ.*

Proposition 4 (Solving conjunctive queries [6]). *Given a σ-structure $\mathcal{S}$ and a conjunctive sentence φ, the decision problem $\mathcal{S} \models \varphi$ is* NP*-complete.*

Proposition 5 (Model checking the full $L_{\mathcal{H}}$). *The Q and A-model checking problems for the full $L_{\mathcal{H}}$ are* NP*-complete.*

Proof (Sketch): Let $\mathcal{H}$ be a $\mathcal{A}$-labelled evolution graph, $\overline{\alpha}$ be an edge (or action) from $\mathcal{H}$, and $\phi(\overline{x})$ be a $L_{\mathcal{H}}$ formula.

Membership: Membership is established by the following procedure: nondeterministically choose a tuple $\overline{i}$, and then solve the problem $\mathcal{H}, \overline{\alpha} \models_X \phi(\overline{i})$, in the following way: replace each occurrence of any $C_k(\overline{i_k})$ in $\mathcal{A}$ with a symbol $p_{C_k(\overline{i_k})}$. Thus, we get a labelling structure $\mathcal{A}^*$ consisting of propositional symbols only. By operating the same replacement in $\mathcal{H}$ and $\phi(\overline{i})$, we get $\mathcal{H}^*$ and ϕ^*, and our problem reduces to the model checking problem for $L^*_{\mathcal{H}}$, which is in PTIME.

Hardness: Hardness is shown for Q-formulae. Let σ be a vocabulary, $\mathcal{S}$ be a σ-structure, and $\varphi = \exists \overline{x} \bigwedge_{k=1}^{n} C_k(\overline{y_k})$ be a conjunctive sentence over σ. We build

$\sigma_Q = \sigma$ and $\sigma_A = \{N\}$ with $arity(N) = 1$. The set of individuals I consists of all elements from the universe of $\mathcal{S}$, and two distinguished elements $\{s, e\}$. The labelling structure $\mathcal{A}$, with the universe I is $\langle \mathcal{S}, N^I \rangle$ where $N^I = \{s, e\}$. The evolution graph is built as follows. First, we create an (initial) hypernode h_0. Assume we have m predicate symbols in $\mathcal{S}$. For each $1 \leq k \leq m$, we do the following: (i) create a new hypernode h_k; (ii) for each tuple $\overline{c} \in C_k^I$, we create two action nodes $a_{\overline{c}}^k$ and $b_{\overline{c}}^k$, as well as a quality edge $(a_{\overline{c}}^k, b_{\overline{c}}^k)$. All action nodes $a_{\overline{c}}^k$ and $b_{\overline{c}}^k$ are in hypernodes h_{k-1} and h_k, respectively ($\mathcal{T}(a_{\overline{c}}^k) = h_{k-1}$ and $\mathcal{T}(b_{\overline{c}}^k) = h_k$). They are labeled as follows: $\mathcal{L}_A(a_{\overline{c}}^k) = N(s)$, $\mathcal{L}_A(b_{\overline{c}}^k) = N(e)$ and $\mathcal{L}_Q(a_{\overline{c}}^k, b_{\overline{c}}^k) = C_k(\overline{c})$. The evolution graph obtained this way is a multi-link *chain*, where all qualities labelled by the same C_k span the same time intervals. Let q be some arbitrary quality edge labelled with a predicate symbol C_1. Now, build a $L_{\mathcal{H}}$-formula $\phi = C_1(\overline{y_1})$ **b** $C_2(\overline{y_2}) \ldots$ **b** $C_n(\overline{y_n})$. It is straightforward that $\mathcal{H}, q \models_Q \phi$ iff $\mathcal{S} \models \varphi$. Hardness for A formulae is shown using a similar chain construction. □

Proposition 5 shows us that, in terms of computational complexity, $L_{\mathcal{H}}$ pays much more for allowing first-order predicates than for exploring the temporal structure it is defined on. Restricting the arity of our predicates also brings no improvement. The proof of Proposition 4 makes this obvious since, even for predicates with arity 2, model checking a conjunctive sentence is NP-complete. It would seem unjustified to explore other fragments of $L_{\mathcal{H}}$ that do allow first-order predicate. As it turns out, there are practical cases where such fragments are interesting.

5 $L_{\mathcal{H}}$ in HPC Systems

In the following, we consider evolution graphs where the set of hypernodes has a total order: $\langle H, < \rangle$. This is motivated by the fact that, in practical applications such as ours, the moment when an action occurs is known in advance. The restriction has no impact on the model checking complexity, since, as seen previously, the source of complexity is in the usage of first-order predicates as labels.

In order to deploy $L_{\mathcal{H}}$ for the extraction of topology information in large-scale systems, we use evolution graphs as a means for the storage of events and event-related properties generated by such systems. For each event recorded in the HPC system log, the following information is given: the moment of occurrence (given as a Unix timestamp), the event type (e.g. network interface card error, memory error, etc.), and the machine that generated the message. An example can be seen in Fig. 2 a. We model the occurrence of an event as an action node, and use the node labelling to encode all event-dependent information. For instance, the record shown in Fig. 2 a, is described by an action node a, such that $\mathcal{L}_A(a) = Event(1244192545, abem5, 1130)$, where 1244192545 indicates the timestamp, $abem5$ indicates the device where the event was signalled and 1130 encodes the event type.

```
1244192545 abem5 1130    1004 1045 20
        a                      b
```

Fig. 2. Log entries and correlations

Hypernodes are built by considering a certain interval δi as time unit. For practical reasons, the δi value used in our experiments was 5 seconds. The entire duration covered by the log is split into intervals of size δi, and a hypernode is created for each such interval. All action nodes having a timestamp falling in some interval of size δi is associated to the corresponding hypernode.

Based on the analysis model presented in [2], a list of correlated event types is built. An example of a simplified correlation record is shown in Fig. 2 b. We model a correlation between two actual events a and b as a quality edge (a, b) labelled $Correlation(1004, 1045, 20)$, where 1004 and 1045 are the correlated event types, and 20 indicates the approximate delay between events.

After the evolution graph is built in the manner described above, $L_{\mathcal{H}}$-model checking is used for extracting correlation patterns and statistical information about the system. An interpreter for $L_{\mathcal{H}}$ was implemented in Java and Jess. The Jess engine was responsible for storing the evolution graph as a knowledge base, and for the pattern matching process involved in $L_{\mathcal{H}}$-model checking. We use $L_{\mathcal{H}}$ to express correlations between events, and the locations where they occur. For instance, the formula from Fig. 3 identifies correlated error events between machines m_1 and m_2. If the set of quality edges satisfying the formula has a considerable size, then it can be assumed that the error event et_1 from m_1 will propagate as et_2 on m_2. By looking at the most frequent occurrences of correlated locations, a set of propagation paths for faults can be built. We call this structure a *propagation topology* and we further use it to get some insight of the behaviour of faults in large HPC systems.

Our results highlighted previous observations regarding error propagation and brought new insights. Firstly, our experiments show that, for fault messages, 98% of the correlations are between locations in the same midplane and rack, the 2% representing unknown locations. For informational messages the percentage is a little lower, around 90%. However, this still confirms our initial finding that messages tend to propagate following the architecture topology of the system. This is an important result since it highlights a possible optimization for the prediction method by distributing its execution independently on each rack.

Secondly, we found that only around 20% of the nodes in the system appear in the extracted topology. This was surprising and it shows that the propagation topology does not follow the exact architecture but only a subset of it. As a result,

$$Correlation(et_1, et_2, x)\ \mathbf{c}_Q\ Event(t, m_1, et_1) \wedge$$
$$Correlation(et_1, et_2, x)\ \mathbf{d}_Q\ Event(t, m_2, et_2)$$

Fig. 3. Inferring correlated locations

whenever attempting to predict failures, it is not always necessary to explore the entire system architecture. Another observation is that informational messages propagate less than faults and on a smaller number of locations. Therefore, these two types tend to behave differently and any prediction system needs to analyse them separately.

We have also investigated the distribution of identified correlations on different locations in the system, depending on how much of the log is analysed. We observed a logarithmic growth of the number of correlations found when analysing different number of months. After 6 months the number of correlations is almost identical with what we found after analysing the entire lifespan of the system, the difference being of only 0.71%. Therefore, 6 months of log data prove sufficient for building accurate prediction topologies.

An application running on a HPC machine usually uses only a subset of computing nodes, so not all node crashes influence its execution. It is important that prediction includes not only the time, but also the location of the next failure in order for current fault tolerance mechanisms to be able to take proactive measures. By extracting topology information, our implementation is able to identify the set of nodes that are potential threats to the application execution.

6 Related Work

Current knowledge representation methods have limitations with respect to modelling dynamic domains. They focus more on providing a static description of a modelled universe and less on capturing time and change. Approaches such as DL (Description Logics) [4] are suitable for describing snapshots of an application domain, but fail in describing its evolution. Although, in many cases, temporal concepts can be embedded, either as modelling primitives (Temporal Extensions of Description Logics (TEDL) [3]), or as pre-defined concepts (see OWLTime [17]), this approach is rather impractical since many TEDL's are either undecidable or have very high complexity bounds.

Temporal logics such as Linear Temporal Logics (LTL) [19], Computation Tree Logic (CTL and CTL*)[7] and the Mu Calculus [9] would seem appropriate for this task. Indeed, the model checking problem is in PTIME for interesting fragments of these logics. However, there is an important difference between Kripke Structures and evolution graphs. The first are *computational structures*[24] that encode all possible states of a deterministic system. Temporal reasoning (i.e. model checking) relies on unfolding the computational structure, and establishing whether certain properties hold. Unlike Kripke Structures, evolution graphs are models of behaviour, and thus can be compared to paths or computation trees. An evolution graph encodes an unique evolution of a system which is not necessarily deterministic. This means that there is no structure able to encode all possible configurations of a system, and which unfolded, can produce an evolution graph. This is the case for error message generation in HPC computing. Also, any system in which actions can occur arbitrarily (but which have foreseeable effects, i.e. the introduction or ceasing of qualities), fall in the same

category. As a result, unlike temporal logics, $L_{\mathcal{H}}$ can only be used to *look into the past*. Based on the domain's history, one can make assumptions about the future evolution of the system, as is the case in HPC error message prediction.

As shown in Section 4.1, the information captured by evolution graphs could be embedded in Kripke Structures, and CTL-model checking could be used to verify temporal relations. However, there is a scalability issue. Computational structures are expected to have limited size. However, the logs used in our analysis had a number of 1,969,710 messages, and future systems are expected to produce much larger logs. Even with symbolic model checking, using such large Kripke Structures can become unfeasible.

Also, using Kripke Structures for storing temporal information in a way different from that shown in Section 4.1 may be inefficient. Conventionally, if a property p holds over a number of n states, all n states must be labelled with p. If there are two such properties in the system, then this will result in $2n$ labelings, whereas in evolution graphs (and in the particular type of Kripke Structures discussed in Section 4.1), such a property is represented as an edge (or a quality state), irrespective of it's lifespan. An exponential growth in the number of states can also be found in other approaches for encoding temporal intervals such as the one presented in [8].

Finally, evolution graphs and $L_{\mathcal{H}}$ provide with more expressiveness in describing the possible properties of the system. This is achieved by using relations for labelling, instead of just sets of propositional symbols.

7 Conclusions

Evolution graphs and $L_{\mathcal{H}}$ provide a straightforward way for representing and reasoning about domains that are constantly changing. While model checking is not as fast as in temporal logics, $L_{\mathcal{H}}$ provides an acceptable trade-off between efficiency and an increased flexibility in describing domain-dependent properties. The $L_{\mathcal{H}}$ connectives offer a simple way for expressing temporal patterns. The same task is not always as easy in temporal logics such as CTL. Also, evolution graphs and $L_{\mathcal{H}}$ prove useful. The $L_{\mathcal{H}}$ model checking mechanism provides with an accurate analysis method for HPC logs that highlights a couple of optimization solutions for fault prediction systems: (i) prediction algorithms can be parallelised, since errors propagate only locally (ii) the entire system architecture is not always relevant for fault prediction and (iii) for the building of a precise propagation topology, 6 months of logs suffice.

Also, we would like to note that the usage of $L_{\mathcal{H}}$ is not limited to the HPC setting. The implementation of our $L_{\mathcal{H}}$-model checker can be naturally extended to a language that allows the specification of time-dependent behaviour in Multi-Agent Systems. Rule-based languages such as Jason [5], SOAR [13], and even CLIPS [21], do not incorporate primitives for expressing temporal relations between entities in an application-dependent domain. As shown in our previous work [20], a language equipped with temporal primitives allows for more flexibility in describing time-dependent domains. Therefore, a temporally-flat knowledge base of an agent can be replaced by an evolution graph. Thus, $L_{\mathcal{H}}$ can

be used to add a temporal dimension to the reasoning process in Multi-Agent Systems.

Acknowledgements. The work has been funded by Project 264207, ERRIC-Empowering Romanian Research on Intelligent Information Technologies/FP7-REGPOT-2010-1 and by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/88/1.5/S/61178.

References

1. Almási, G.S., Bellofatto, R., Brunheroto, J.R., Caşcaval, C., Castaños, J.G., Ceze, L., Crumley, P., Erway, C.C., Gagliano, J., Lieber, D., Martorell, X., Moreira, J.E., Sanomiya, A., Strauss, K.: An Overview of the Blue Gene/L System Software Organization. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 543–555. Springer, Heidelberg (2003)
2. Gainaru, A., Franck Cappello, W.K.: Taming of the shrew: Modeling the normal and faulty behavior of large-scale hpc systems. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), pp. 24–35 (to appear, 2012)
3. Artale, A., Franconi, E.: A survey of temporal extensions of description logics. Annals of Mathematics and Artificial Intelligence 30, 171–210 (2001)
4. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
5. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
6. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC 1977, pp. 77–90. ACM, New York (1977), http://doi.acm.org/10.1145/800105.803397
7. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986)
8. De Nicola, R., Vaandrager, F.: Action versus state based logics for transition systems. In: Proceedings of the LITP Spring School on Theoretical Computer Science on Semantics of Systems of Concurrent Processes, pp. 407–419. Springer-Verlag New York, Inc., New York (1990), http://dl.acm.org/citation.cfm?id=111693.111710
9. Emerson, E.A.: Model checking and the mu-calculus. In: Descriptive Complexity and Finite Models, pp. 185–214 (1996)
10. Capello, F., Geist, A., Gropp, B., Kale, S., Kramer, B., Snir, M.: Toward exascale resilience. International Journal of High Performance Computing Applications 23 (2009)
11. Gainaru, A., Cappello, F., Trausan-Matu, S., Kramer, B.: Event Log Mining Tool for Large Scale HPC Systems. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 52–64. Springer, Heidelberg (2011)

12. Gallet, M., Yigitbasi, N., Javadi, B., Kondo, D., Iosup, A., Epema, D.: A Model for Space-Correlated Failures in Large-Scale Distributed Systems. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Part I. LNCS, vol. 6271, pp. 88–100. Springer, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1887695.1887707
13. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: an architecture for general intelligence. Artif. Intell. 33(1), 1–64 (1987), http://dx.doi.org/10.1016/0004-3702(87)90050-6
14. Lan, Z., Zheng, Z., Li, Y.: Toward automated anomaly identification in large-scale systems. IEEE Trans. on Parallel and Distributed Systems 21(2), 174–187 (2010)
15. Libkin, L.: Elements Of Finite Model Theory. Texts in Theoretical Computer Science. An Eatcs Series. Springer (2004)
16. Oldfield, R.A., Arunagiri, S., Teller, P.J., Seelam, S., Varela, M.R., Riesen, R., Roth, P.C.: Modeling the impact of checkpoints on next-generation systems. In: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies, MSST 2007, pp. 30–46. IEEE Computer Society, Washington, DC (2007)
17. Pan, F.: An Ontology of Time: Representing Complex Temporal Phenomena for the Semantic Web and Natural Language. VDM Verlag, Saarbrucken (2009)
18. Park, Geist, A.: System log pre-processing to improve failure prediction. In: DSN 2009, pp. 572–577 (June 2009)
19. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, SFCS 1977, pp. 46–57. IEEE Computer Society, Washington, DC (1977)
20. Popovici, M., Muraru, M., Agache, A., Giumale, C., Negreanu, L., Dobre, C.: A Modeling Method and Declarative Language for Temporal Reasoning Based on Fluid Qualities. In: Andrews, S., Polovina, S., Hill, R., Akhgar, B. (eds.) ICCS-ConceptStruct 2011. LNCS, vol. 6828, pp. 215–228. Springer, Heidelberg (2011)
21. Riley, G.: NASA Clips: A Tool for Building Expert Systems (June 2006), http://www.ghg.net/clips/CLIPS.html
22. Roşu, G., Bensalem, S.: Allen Linear (Interval) Temporal Logic – Translation to LTL and Monitor Synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 263–277. Springer, Heidelberg (2006)
23. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. ACM Comput. Surv. 42(3), 10:1–10:42 (2010)
24. Schnoebelen, P.: The complexity of temporal logic model checking. In: Proceedings of Advances in Modal Logics AiML 2002. World Scientific (2003)