Case Study on Certifying Distributed Algorithms: Reducing Intrusiveness

Samira Akili, Kim Völlinger

Case Study on Certifying Distributed Algorithms: Reducing Intrusiveness

S. Akili and K. Völlinger

Humboldt University of Berlin, Germany
akilisam@cms.hu-berlin.de voellinger@hu-berlin.de

Abstract. Certifying distributed algorithms (CDAs) are a runtime verification method for distributed systems. A CDA computes additionally a *witness* to an input-output pair – a correctness argument for the pair. The witness is verified at runtime by a distributed *checker* algorithm. In this paper, we apply CDAs to an industrial case study of collaborative transport robots serving machines in a factory. In particular, we present a certifying variant of a distributed bidding algorithm executed by the robots to assign transport jobs amongst each other. Furthermore, we introduce overlays in order to organize the communication of the distributed checker, and compare them regarding their intrusiveness.

1 Introduction

We consider certifying distributed algorithms (CDAs) – a runtime verification method for distributed systems. A CDA computes a *witness* w additionally to an input-output pair (i, o) such that if a *witness predicate* holds for the triple (i, o, w), the pair (i, o) is correct. A *distributable* witness predicate states a property in the system by stating properties for each component, and hence can be decided by a *distributed* checker algorithm at runtime. As an example, consider a distributed algorithm where the components of a network decide if the network graph itself is bipartite. In the case of a non-bipartite network graph, an odd cycle in the graph is a witness since an odd cycle is not bipartite itself. The witness predicate states that an odd cycle exists in a network for which the distributed algorithm outputs that its non-bipartite. In [7] a distributable witness predicate for the example is described. In the typical setup of runtime verification, a system is instrumented to compute outputs for a monitor deciding if a given property holds. Analogously, a CDA is instrumented to compute a witness for the checker deciding if an input-output pair is correct. In this paper, we investigate a case study of transport robots serving machines in a factory [1]. Since the robots execute distributed algorithms to achieve collaborative goals, they can be classified as a multi-agent system. We apply CDAs to verify a distributed bidding algorithm used to assign transport jobs at runtime. Moreover, we consider overlays (i.e. communication topologies imposed on the components of the system) for the distributed checker, and compare them regarding their intrusiveness (i.e. the degree to which runtime verification affects the system).

Related Work. Certifying *sequential* algorithms are established [5] but there is little work on certifying *distributed* algorithms [10,8,7,9]. CDAs can be classified as a distributed and choreographed monitoring approach since the checker is a distributed algorithm, and as a synchronous monitoring approach since the system waits for the checker to accept [2]. Overlay networks are a well established research strand offering sophisticated solutions for various applications [3]. However, to our knowledge, there is no approach of using overlays to reduce intrusivenes for runtime verification.

2 Preliminaries: Certifying Distributed Algorithms

We model the communication topology of a distributed system as a connected undirected graph $G = (V, E)$: a vertex represents a component, an edge a communication channel. A *distributed algorithm*, running on a distributed system, consists of a sub-algorithm for each component such that all components together solve one problem [4]. The input i is distributed such that each component $v \in V$ has a sub-input i_v with $i = \cup_{v \in V} i_v$; analogously for the output. A CDA computes a *witness* w additionally to its input-output pair (i, o) such that if a predicate – the *witness predicate* – holds for the triple (i, o, w), the pair (i, o) is correct [10]. We call a predicate that is defined over a component's sub-input, sub-output and sub-witness a *local predicate*. A predicate Γ is *universally distributable* with a local predicate γ if for all triples (i, o, w) holds: $\forall v \in V : \gamma(i_v, o_v, w_v) \longrightarrow \Gamma(i, o, w)$, and *existentially distributable* if: $\exists v \in V : \gamma(i_v, o_v, w_v) \longrightarrow \Gamma(i, o, w)$. A predicate is *distributable* if one of the former applies, or if it is implied by conjuncted and/or disjuncted universally/existentially distributable predicates [7]. The witness predicate has to be distributable such that it can be decided by a distributed *checker* algorithm at runtime. The *sub-checker* of component v decides all local predicates over (i_v, o_v, w_v). Using a spanning tree, the sub-checkers aggregate the evaluated local predicates upwards and combine them by logical conjunction or disjunction depending on whether the according predicate is universally or existentially distributable; the root decides the witness predicate by combining the evaluated distributable predicates [9]. Hence, if the distributed checker accepts, the distributed input-output pair (i, o) is correct. The user of a CDA does not have to trust the actual algorithm but the checker which is simpler for a well-chosen witness. Using the framework proposed in [8,9] an implemented checker can be verified.

3 Case Study: Certifying Distributed Bidding

We conduct a case study on a fleet of collaborative transport robots serving machines in a factory, provided by InSystems [1]. In particular, we investigate distributed bidding which is executed whenever a machine signals that it needs to be served. The robots communicate via a wireless network by sending broadcast or unicast messages.

Specification. W.l.o.g. let $ID = \{1, ...n\}$ be the set of the robots' unique identifiers. We refer to a robot with ID $k \in ID$ as robot k. For a robot k, the sub-input is its ID ($i_k := k$) and the sub-output is its winner-tuple ($o_k := (winnerID_k, winnerBid_k)$). The correctness of a distributed bidding is specified by the following postconditions: all robots agree on the winner *(agreement)*, the winner exists *(existence)*, and the bid of the winner is the maximum of all bids *(maximum)*. InSystems provides different variants for distributed bidding. However, we treat the algorithm as a black box and ground its certifying variant on the specification.

In the following, we give a certifying variant of distributed bidding by introducing a witness, a witness predicate and distributed checker algorithm. Moreover, we compare different overlays organizing the communication of the distributed checker regarding their intrusiveness.

Distributed Witness. The sub-witness of robot k is its own bid and a set containing the sub-outputs of the other robots. Hence, $w_k = (bid_k, \{o_l | l \in ID \text{ and } l \neq k\}$. The sub-witnesses are computed during bidding by bookkeeping; no additional computation is necessary.

Local Predicates. Let γ_{agree}, γ_{exist} and γ_{max} be local predicates over robot k's sub-input i_k, sub-output o_k, and sub-witness w_k. The predicate γ_{agree} holds iff $o_k = o_l$ for all $k \neq l \in ID$, i.e. if k's winner-tuple equals the winner-tuples of all other robots. The predicate γ_{exist} holds iff $k = winnerID_k$, i.e. if k chose itself as a winner. The predicate γ_{max} holds iff $bid_k \leq winnerBid_k$, i.e. if $k's$ bid is less than or equal to the bid of its chosen winner.

Distributable Predicates. Let Γ_{agree}, Γ_{exist}, Γ_{max} be predicates over the distributed input i, output o and witness w stating the three properties of the specification, e.g. if Γ_{agree} holds *agreement* is ensured. We forego a formalization. The three predicates are distributable with the introduced local predicates. The predicate Γ_{agree} is universally distributable with γ_{agree} since for all triples (i, o, w) holds: $\forall k \in ID, \gamma_{agree}(i_k, o_k, w_k) \longrightarrow \Gamma_{agree}(i, o, w)$. The predicate Γ_{exist} is distributable with γ_{agree} and γ_{exist} since for all triples (i, o, w) holds: $(\exists k \in ID, \gamma_{exist}(i_k, o_k, w_k) \wedge \Gamma_{agree}(i, o, w)) \longrightarrow \Gamma_{exist}(i, o, w)$. The predicate Γ_{agree} ensures that there is exactly one winner. The predicate Γ_{max} is distributable with γ_{agree} and γ_{max} since for all triples (i, o, w) holds: $(\forall k \in ID, \gamma_{exist}(i_k, o_k, w_k) \wedge \Gamma_{agree}(i, o, w)) \longrightarrow \Gamma_{max}(i, o, w)$. The predicate Γ_{agree} ensures that each robot compares its bid with the same winner-bid.

Witness Predicate. A logical conjunction of the predicates Γ_{agree}, Γ_{exist} and Γ_{max} is a witness predicate for the specification of distributed bidding.

Distributed Checker. The sub-checker of each robot runs as a separate process on the robot, and sub-checkers communicate with each other using the robots' IDs. The sub-checker of a robot k executes the following tasks:

(1) collecting the winner-tuples for its robot's sub-witness w_k, and deciding the local predicates γ_{agree}, γ_{exist} and γ_{max} on the triple (i_k, o_k, w_k),
(2) participating in deciding the distributable predicates Γ_{agree}, Γ_{exist} and Γ_{max} on the triple (i, o, w),
(3) and participating in deciding the witness predicate on the triple (i, o, w).

Note that for an arbitrary (connected) overlay, it is sufficient to consider the winner-tuples of neighbors in the overlay for task (1) since agreement is ensured by transitivity over neighborhoods. Hence, for task (1), a sub-checker collects the winner-tuples of neighboring robots. As the chosen overlay determines the number of neighbors, it affects the intrusiveness of the tasks. We investigate the tasks in more detail for each overlay at the end of this Section.

Criteria for Intrusiveness. Intrusiveness denotes the degree to which runtime verification affects the original system [2]. We evaluate intrusiveness by the message overhead, runtime and local computation time of the distributed checker. We measure message overhead as the number of received messages to reflect the processing overhead a message inflicts, e.g. a broadcast message is counted once per receiving component. As usual for asynchronous systems, we measure runtime by assuming that a message is delivered in one time unit [6]. Local computation time denotes the sequential computation time of a robot. In distributed algorithm analysis, local computation time is neglected when reasonably low but pointed out if a component performs a "global" computation (i.e. in our case, if the local computation depends on the number of robots) [6]. As message overhead, runtime and local computation of the checker delay the system and take resources of the robots, we consider these measurements to be reasonable criteria for intrusiveness.

Communication of Sub-Checkers. We investigate three topologies to organize the communication of the distributed checker: the original system without an overlay (complete graph), and two overlays, a star tree and a balanced binary tree. For each topology, we evaluate the intrusiveness of the tasks (1)-(3). The results are summed up in the table in Fig. 1. We denote if the number of sub-checkers having a certain local computation time is constant or linear in the number of components; e.g. $\Theta(n)_1$ denotes that a constant number of sub-checkers has the local computation time $\Theta(n)$, and $\Theta(n)_n$ that the number of sub-checkers having $\Theta(n)$ is linear in the number of components. Moreover, we denote if some sub-checkers have nothing to do with a 0 instead of $\Theta(1)$ to point out how fairly work is distributed between the sub-checkers. For the overlays, the first row of local computation is root's (one of the sub-checkers) effort with the exception of task (2) for the binary tree where it is the effort of all non-leave sub-checkers. Note that the complexity classes of task (1) depend on the particular local predicate, while the complexity classes for the tasks (2) and (3) are the same for each distributable witness predicate.

	Complete Graph			Star			Balanced Binary Tree		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
Local Computation	$\Theta(n)_n$	$\Theta(n)_n$	$\Theta(1)_n$	0_1	$\Theta(n)_1$	$\Theta(1)_1$	0_1	$\Theta(1)_n$	$\Theta(1)_1$
	-	-	-	$\Theta(1)_n$	0_n	0_n	$\Theta(1)_n$	0_n	0_n
Message Overhead	$\Theta(n^2)$	$\Theta(n^2)$	-	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Runtime	$\Theta(1)$	$\Theta(1)$	-	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$

Fig. 1. The intrusiveness of the tasks (1)-(3) for each topology.

Complete Graph. For task (1), each sub-checker broadcasts the winner-tuple of its robot and subsequently compares its robot's winner-tuple with all other tuples to decide the predicate γ_{agree}. Hence, local computation time is linear in the number of robots for each sub-checker. For task (2), each sub-checker broadcasts a triple with its evaluated local predicates, and decides the distributable predicates with the received triples. Note that by comparing its robot's winner-tuple with all other tuples, each sub-checker already decides the predicate Γ_{agree} by deciding γ_{agree} in task (1) since a robots sub-witness equals the distributed witness in this case. However to decide the distributable predicates for the maximum and existence property communication is still needed. For task (3), each sub-checker logically conjuncts the three evaluated distributable predicates.

Star Tree. For task (1), root broadcasts its winner-tuple and the other sub-checkers compare their winner-tuple with it. For task (2), each sub-checker sends the triple of its evaluated local predicates to root. As root decides the distributable predicates, root's local computation is linear in the number of robots. For task (3), root decides the witness predicate and informs the other sub-checkers by a broadcast.

Balanced Binary Tree. For task (1), each non-leaf sub-checker sends its winner-tuple to its children, and each child compares its winner-tuple with the winner-tuple of its parent. For task (2), starting by the leaves, each sub-checker gets the triple of the evaluated local predicates from its children and combines it with its own triple. The root holds a triple of the evaluated distributable predicates. Hence, the runtime is the tree's depth. For task (3), root decides the witness predicate and informs all others using the tree.

Comparison. The complete graph and star have the lowest runtime. However, regarding message overhead and local computation, the complete graph performs the worst. In the star, only root computes a global computation, while in the binary tree no global computation occurs. We conclude that the complete graph is not suitable to organize the communication of the sub-checkers, while the star and binary tree can be both justified. They reflect a trade-off between runtime and local computation time which respectively depend on the depth and the branching factor of a tree. A star is extreme in branching and therefore minimizes runtime. A chain would be extreme in depth. However, we chose a binary tree for comparison since its runtime is sub-linear while local computation time is still constant. A balanced tree additionally restricts the depth. Hence, the branching factor should be optimized according to the requirements of the system.

4 Discussion

We applied CDAs to an industrial case study [1]. Particularly, we presented a certifying variant of distributed bidding to verify it at runtime. Moreover, we introduced overlays to organize the communication of the sub-checkers, and compared them regarding their intrusiveness. We concluded that an overlay with

a tree topology improves a quadratic message overhead to a linear one, and that by adjusting the branching factor, runtime and local computation time can be balanced out. Our results can be generalized to obtain a generic method to verify agreement at runtime (e.g. to be reused for consensus problems) using overlays.

Future Work. Note that for a universally distributable witness predicate, the distributed checker could stop after task (1) if a sub-checker raises an alarm when detecting that the according local predicate is not satisfied. When choosing an overlay, as many checkers as possible should be able to raise an alarm. We reflected that idea e.g. for the binary tree by letting the children check agreement with their parent. If parents check agreement with their children, leaves (about half of the components) cannot raise an alarm. For an existentially distributable witness predicate, a time out could be used: if no sub-checker decides that a local predicate holds before a time out is reached, the checkers conclude that the predicate does not hold. However, this could lead to false negatives. Another criteria for an overlay could be robustness against message loss, e.g. by choosing neighbors in the overlay according to the physical neighbors. Another direction is to consider overlays that can be efficiently updated in case of system dynamics.

References

1. proANT Transport Robots . `http://www.insystems.de/en/produkte/proant-transport-roboter/`
2. Francalanza, A., Pérez, J.A., Sánchez, C.: Runtime Verification for Decentralised and Distributed Systems. In: Lectures on Runtime Verification, pp. 176–210. Springer (2018)
3. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials 7(2), 72–93 (2005)
4. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
5. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying Algorithms. Computer Science Review 5, 119–161 (2011)
6. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000)
7. Völlinger, K.: Verifying the Output of a Distributed Algorithm Using Certification. In: International Conference on Runtime Verification. pp. 424–430. Springer (2017)
8. Völlinger, K., Akili, S.: Verifying a Class of Certifying Distributed Programs. In: NASA Formal Methods Symposium. pp. 373–388. Springer (2017)
9. Völlinger, K., Akili, S.: On a Verification Framework for Certifying Distributed Algorithms: Distributed Checking and Consistency. In: International Conference on Formal Techniques for Distributed Objects, Components, and Systems. pp. 161–180. Springer (2018)
10. Völlinger, K., Reisig, W.: Certification of Distributed Algorithms Solving Problems with Optimal substructure. In: Software Engineering and Formal Methods, pp. 190–195. Springer (2015)