Performance Analysis of Fault-Tolerant Multi-Agent Coordination Mechanisms

Riccardo Pinciroli and Catia Trubiani

***Abstract*— Performance evaluation of multi-agent systems (MAS) embraces several challenges due to uncertain operational environments, such as software/hardware failures and unfaithful communications that facilitate the spread of deceptive messages. One way to smooth the impact of reliability and security potential issues in MAS is to enforce different coordination mechanisms among agents (i.e., coordinating multiple agents that need to perform a sequence of actions to maximize a system-level reward), and evaluate their efficiency. In this paper, we propose a tool-based approach, namely COORDINATE, that simulates and compares the performance characteristics of different coordination mechanisms for MAS while considering fault-tolerant and corrupted agents. A Smart Hospital is adopted as illustrative example to show the need of performance-based analysis results pointing out which coordination mechanism is more efficient. Experimental results indicate that the inter-arrival time of tasks to be accomplished, the failure probability of agents, and the ratio of faithful to malicious agents contribute to determining the efficiency of different coordination mechanisms. When varying these parameters in the considered scenarios, the system latency can be reduced up to 4.2× by selecting the optimal coordination mechanism. This way, software developers are informed on the system peculiarities that trigger the switching among such coordination mechanisms for a performance-based optimal solution.**

***Index Terms*— Performance Evaluation, Multi-Agent Systems, Coordination, System Latency.**

I. Introduction

Multi-Agent Systems (MAS) have been recently advocated to accomplish cooperative tasks [1], [2], and the coordination of agents is indeed nontrivial, even more so when they need to maximize a system-level reward. In the literature, it is well-assessed the synergy between MAS and challenges of Cyber-Physical Systems (CPS) since semantic technologies can be used to empower embedded computing and communication capabilities [3], [4]. CPS are increasingly adopted in many industrial environments, such as warehouse logistics and production monitoring [5], [6]. One of the key advantage is that a cyber–physical entity integrates its hardware function with a digital counterpart, hence it acts as a virtual representation for the physical part. From an industrial perspective, this fosters the development of products/processes where agents take autonomous actions in response to their internal state and their perception of the environment [7]. Besides guaranteeing the correctness of the system behavior, there exist important challenges, such as performance, reliability, and security, which must be tackled, especially in an industrial context. For instance, trustworthiness, intended as the capacity of CPS to deal with disruption while guaranteeing a temporary impact on provided services, has been recently outlined in the literature as one of the major challenges [8], [9], mainly due to these difficulties: (i) failures can occur in various unexpected ways due to uncertainties in the operational environment; (ii) adversarial threats (e.g., the opportunistic behavior of agents [10]) are likely to take place in practice [11], and it is not trivial to plan which countermeasures need to be applied for system recovery.

Motivation. With the raising of critical industrial applications subject to stringent quality-of-service requirements, it becomes fundamental to provide support for the performance evaluation of fault-tolerant MAS. In the literature [11] there exist many fault-tolerant control strategies, we are interested to study *cooperative* methodologies where agents work together to achieve the system goal. For instance, let us consider a smart hospital where robots are in charge of delivering medicines to patients within a given time (e.g., 2 minutes from doctors' requests). Robots may show a software/hardware failure while working on their tasks [11]. A failed robot can still communicate and ask for help from other robots to complete its task. This scenario calls for coordination mechanisms that facilitate the spreading of help requests. Robots may also ask for help maliciously (e.g., deception attacks [12] with the intent of affecting the system performance to gain a personal advantage [10]) even if they actually complete their tasks. Note that malicious robots inject deceptive messages (i.e., fake help requests) in the communication [13], but they keep working toward the completion of their tasks. Any robot that receives the malicious request takes a detour and slows down its mission to check if help is really needed. These robots resume their original mission only after ensuring that no help is required. This way, malicious robots gain personal advantage by delivering medicines without delay and faster than misled robots. This scenario implies that it may be more suitable to adopt coordination mechanisms limiting the spread of (fake) help requests. Moreover, in the literature [14] exist *plan synthesis schemes* to handle dependencies among

The authors would like to thank the Editor and the anonymous reviewers for their valuable feedback. We are also grateful to the Computing and Network Service for their support in our experiments on the U-LITE cluster at INFN, LNGS, L'Aquila, Italy. This work has been partially supported by the MUR PRIN project SEDUCE 2017TWRCNB.

R. Pinciroli and C. Trubiani are with the Gran Sasso Science Institute, L'Aquila, Italy (e-mail: name.surname@gssi.it).

agents' actions, i.e., participants have insufficient resources to solve the entire problem and need to cooperate. In our scenario, this means that some robots move medicines from the hospital pharmacy to clinical departments, whereas other robots are in charge of delivering these medicines to the patients. These dependencies on actions motivate the need for (i) evaluating the performance characteristics of MAS and (ii) making software developers aware of the importance of the underlying coordination mechanism as an indicator of the promptness of reaction among agents.

Challenges. The performance evaluation of coordination mechanism for fault-tolerant MAS triggers two main challenges: (i) many system parameters need to be considered, e.g., characteristics of physical space, agent dependencies, workload intensity of application tasks, security and reliability potential issues. Even more the interplay among these parameters makes the performance analysis rather complex; (ii) the complexity of modeling the system behavior with performance engineering techniques (e.g., stochastic Petri nets) that require high expertise to be used efficiently and incur the state space explosion problem, i.e., a solution may not be found in a reasonable time.

Proposed solution. To address the aforementioned challenges, we propose a tool-based approach, namely COORDINATE (COORDINation of fAult Tolerant agEnts), i.e., a *tool* that simulates the behavior of agents starting from the specification of multiple coordination mechanisms. Specifically, we consider three different coordination mechanisms inspired by [15]: (i) *centralized* (CE) – also defined as *internal communication* in [14] – in which agents communicate their failures to a central coordinator that decides which other agent should intervene according to different strategies. For instance, the proximity strategy, i.e., the closest agent, or the importance strategy, i.e., the agent involved in the least important task; (ii) *semi-decentralized* (SD) – also defined as *external communication* in [14] – which selects a subset of agents to be informed of the failure, most likely those occupying spaces adjacent to the occurrence of the failure; (iii) *fully-decentralized* (FD) in which each agent is in charge of verifying the status of the physical space, and it is autonomous in the process of discovering agents to be rescued or uncompleted tasks that still need to be carried out. The goal of COORDINATE is to quantitatively compare the performance characteristics of these coordination mechanisms acting on different space topologies. Our experiments show that both the characteristics of the physical space and reliability/security threats (e.g., failure probabilities, number of malicious agents that do not reveal their intention otherwise the identification of such agents would have a high impact on the system performance [11]) play a key role in identifying the optimal coordination mechanism. Note that the performance evaluation abstracts from the hardware infrastructure, it is intended to be executed at the design time of the software development process. Predictive results represent knowledge for software developers that are informed, with quantitative data, on the performance of different coordination mechanisms. More specifically, we are interested to quantitatively compare different strategies for coordinating multiple fault-tolerant agents (possibly showing dependencies) that need to perform a sequence of actions with the goal of maximizing a system-level reward.

Contributions. In this article, we propose a performance-based evaluation approach for selecting optimal coordination mechanisms while considering reliability and security potential issues. The main contributions of this article are as follows.

- Provide a tool-based approach for the performance analysis of fault-tolerant multi-agent coordination mechanisms, using dependability characteristics (e.g., failure probabilities and number of malicious agents) contributing to the coordination selection.
- Design a tool supporting different pre-defined coordination mechanisms, and flexible to simulate varying scenarios. Software developers can change the values of predefined system parameters and analyze their impact on multiple space topologies.
- Evaluate the proposed tool in terms of workload intensity, scalability, and its applicability to both synthetic and real-world scenarios.

The paper is structured as follows. Section II briefly reviews the related work. Section III describes the proposed tool. Section IV presents the conducted experimentation: research questions, analysis results, and threats to validity. Section V concludes the paper and provides future research directions. The tool and replication data are publicly available [16].

II. Related Work

This paper relates to three main streams of research: coordination among agents, dependability (i.e., reliability and security) issues, and simulation. These three aspects are reviewed in the following. Table I reports the approaches that are more closely related to COORDINATE along with a brief summary of *strengths* and characteristics related to their *scope*, thus to better position our tool-based approach in the field.

Coordination among agents. Many recent studies focus on coordination that is a fundamental aspect of modern systems [17]. For instance, the importance of coordination in MAS and possible algorithms for decision making alternatives are discussed in [18]. Swarm intelligence algorithms are proposed in [19] to enable the autonomous coordination of multi-agent systems. Yun et al. [20] propose a leader-based autonomous coordination mechanism to increase the performance (i.e., data resolution and area coverage) of unmanned aerial vehicles. A control-based approach that allows coordination under intermittent Denial-of-Service (DoS) attacks is presented in [21]. A coordination language that focuses on quantitative aspects (e.g., energy consumption and security) is defined in [22], but it is not complemented with an analysis tool. Task completion time and overall system performance efficiency is evaluated in [23], in particular reward-based task offloading schemes regulate the collaboration among nearby robots. Performance evaluation is conducted through `Veins` [24] and `OMNeT++` [25] simulators tracking vehicles in urban mobility and their communication protocols. Rewards are estimated while increasing vehicle density to determine when network congestion is achieved. A similar problem

TABLE I: Summary of related work - strengths and scope on performance analysis.

	Approach	Strengths	Scope
coordination	[23]	Reward-based task offloading schemes	Tailored for vehicles mobility and their network communications
	[26]	Collision avoidance is a major concern	Focused on automated guided vehicles and acting on their speed
	[28]	Performance modeling of agents' interactions	Subject to the GSPNs state space explosion problem
dependability	[30]	Optimal reliability and ensuring security	Circumscribed to the timing of workflow scheduling algorithms
	[31]	Optimal performance but preserving privacy	Confined to estimate energy storage under privacy requirements
	[32]	Probabilistic operators for stochastic behaviors	Customized to generate counter examples for unsatisfied relations
simulation	ARGoS	Large-scale swarms of (multi-physics) robots	Conceived for analyzing synchronization, flocking, or gripping
	Gazebo	Robots moving in complex environments	Envisaged for robots' path planning and control algorithms
	RST	Versatile to model kinematics and dynamics	Aimed to robots' motion planning and trajectory generation
	COORDINATE	Flexible in the specification of the system design (physical space, application tasks, system agents and their dependencies)	Comparison of three coordination mechanisms. A large number of system design characteristics (e.g., space topology, workload) may affect the system latency.

(i.e., reducing the delivery time prioritizing urgent tasks) is tackled in [26], but the focus is on collision avoidance among vehicles. Microsoft Robotics Developer Studio [27] is adopted as simulator, kinematic equations are aimed to predict the movement of automated guided vehicles with the goal of overcoming the intersections safely, possibly by changing the speed of vehicles. Stochastic models are used in [28] to evaluate the performance of coordination mechanisms, but agents operate in a limited physical space (i.e., three rooms only) and no reliability or security issues are investigated. Generalized Stochastic Petri Nets (GSPNs) [29] are used for performance analysis, but the formalism is known to suffer from state space explosion, and a large physical space may hinder the model-based analysis. Differently from these approaches, this paper investigates the performance behavior of three coordination mechanisms while considering reliability and security issues as first-class concerns.

Dependability issues. Dependability properties of systems are investigated via several approaches, for instance using probabilistic model checking [33] or logic-based specifications in conjunction with a monitoring tools [34]. Reliability and security concerns are jointly analyzed in [30] for scheduling workflows that include timing constraints, but the migration of tasks among agents is not foreseen. The performance evaluation consists of simulating the workflow scheduling algorithms and measuring their execution times, since the overall goal is to decrease the probability of errors caused by transient faults while ensuring security services. Malicious activities are considered in [35] where the communication layer is monitored to guarantee reliability and to avoid the propagation of security threats at the application level. He et al. [12] survey malicious attacks in MAS and report those considered in this paper, i.e., *deception attacks* including false data injection. These attacks are analyzed in Sargolzaei et al. [36] with the goal of detecting their occurrence, however the impact of these attacks on the system performance is not considered. The trade-off between performance and security is analyzed in [31] and privacy mechanisms are considered as overheads that slow down the system. Specifically, performance refers to energy storage that is kept under control, and privacy requirements are embedded in the defined optimization problem. Energy-related constraints are investigated in [37] with DoS attacks limiting sensors to transmit data, and in [38] with convolution neural networks that are proposed to increase the device lifetime by reducing the communication cost (i.e., energy consumption). Non-functional analysis is also carried out in [32] where model-checking techniques support formal verification of UPPAAL models. A set of probabilistic operators is defined to model dynamic and stochastic behaviors, and the verification consists of probabilistic queries that generate counter examples in case of unsatisfied relations. A recent trend consists of building digital twins to support decision making and improve the resilience [39], i.e., machine learning techniques are adopted to look for the best sequence of decisions ensuring resilience in systems subject to uncertainty in their evolution. To the best of our knowledge, there exists no work looking at how the severity of dependability issues (i.e., failure probabilities and number of malicious agents) impacts on different coordination mechanisms among agents, as we do in this paper.

Simulation. There are several tools to simulate robots and their behavior. For instance, ARGoS [40] provides examples of properties applied to swarms of robots, such as synchronization, flocking, or gripping. Gazebo [41] focuses on robots moving in complex indoor and outdoor environments, and it supports path planning and control algorithms; it can be also synchronized with Simulink models through the Robotics System Toolbox (RST) [42] that is versatile in modeling robots' kinematics and dynamics, and it provides motion planning and trajectory generation. To the best of our knowledge, despite the large availability of robotics simulators, none of them integrates coordination mechanisms among agents and provides a performance evaluation in terms of system latency affected by multiple and peculiar system characteristics.

Summary of the literature review. To the best of our knowledge, in the literature most of the work for the performance analysis of coordination mechanisms (see Table I) targets different scopes and relies on specific agents (e.g., vehicles) or communication protocols. Differently from the state-of-the-art, in this paper we investigate the performance characteristics of multiple coordination mechanisms to study their efficiency while varying system characteristics and considering agent dependencies. A key advantage is that our tool (COORDINATE) is flexible in the specification of the system design, hence physical space (e.g., its topology), application tasks (e.g., their priority), and system agents (e.g., their failure probability) can be configured with a certain degree of freedom by system designers. The novelty consists of demonstrating that the sys-

tem performance can be optimized by switching coordination mechanisms on the basis of the application scenario under analysis along with its reliability/security peculiarities.

III. Proposed Tool: COORDINATE

Fig. 1 provides an overview of COORDINATE that is constituted of two main components. First, an analyzer (focus of this paper, see the shaded box) builds the knowledge at design time by simulating the system design with three coordination mechanisms playing in a user-de ned observation time. Fixed/prescribed-time fault tolerant cooperative control for MAS has not received adequate attention and constitutes a promising topic [43]; this strengthens our goal of evaluating the system response time of fault-tolerant and cooperative MAS. Later, at runtime, an evaluation engine takes performance requirements and system monitoring data as input, and returns the optimal coordination mechanism(s) as output. Note that the system monitoring is left aside from this paper contributions, we refer to state-of-the-art approaches [43] for this scope. The evaluation engine fully leverages the design time knowledge that we discuss in the following.

Fig. 1: COORDINATE – Proposed tool for the performance evaluation of coordination mechanisms.

Fig. 2 provides an illustrative scenario to exemplify the description of the proposed tool. We assume that there are ve agents moving into nine rooms to accomplish their application tasks (e.g., carrying medicines). Our scenario includes a Lobby (i.e., the source zone) where agents are instructed on their task, twopatient rooms(i.e., PT1 and PT2), a Nurses Stationfor monitoring all patients, aWaiting Areafor patients' relatives waiting to get information by physicians, and two target zones (i.e., theSurgical Unit and theWaste Managementareas) where agents are supposed to deliver objects and complete their task. A Corridor is represented as multiple inter-connected rooms (see dashed lines in Fig. 2). For simplicity, we x the time required to cross a room, but it can be increased or decreased to cover physical spaces of larger or smaller sizes, respectively.

Fig. 2: Smart Hospital illustrative scenario.

A. System Design

1) Physical Space: Software developers have the possibility to build their own physical space of interest by specifying rooms (through an identi er) and their connections.

Fig. 3 depicts some common space topologies [44] provided by our tool, where (i) the upper part reports the textual description, (ii) the bottom part abstracts the upper speci cation and graphically reproduces the physical space: each node of the graph is a room with its ID (i.e., 0 to 8), whereas edges represent connections.

Mesh. Each node is connected to a subset of neighbour nodes which in turn are connected to other nodes. For instance, the text0: [1, 3] means that the room with ID 0 is connected to two other rooms (i.e., those with ID 1 and 3).

Ring. When considering a ring topology, each room has exactly two connections. For instance, the room with ID 0 is connected to rooms with ID 1 and 8.

Tree. It includes a root node that is connected to a subset of nodes which in turn are connected to other nodes. For instance, in our example the room labeled as 0 is the root node connected to rooms labeled as 1, 3, 5, and 7.

It is worth remarking that these templates represent some feasible structures of the physical space. Software developers can specify their own topology (see Section IV-E) by providing lists of rooms and their connections, the room where tasks are generated (i.e., source), and possible destinations (i.e., targets).

2) Application Tasks: They represent a system goal to be achieved (e.g., delivery of medicines). Each task is regulated by an inter-arrival time that models the frequency with which tasks are generated in the system. Besides, each task is associated with a value denoting its importance (e.g., low, medium, and high) to be accomplished.

3) System Agents: They represent the entities in charge of completing application tasks while moving in the physical space. Each agent is initialized with the option of being mali-

Fig. 3: Templates of space topologies, inspired by [44].

Fig. 4: Sequence diagram for the SD mechanism.

cious and a failure probability that takes into account multiple aspects. We distinguish malicious from faulty agents since they impact the system performance differently. Malicious agents unnecessarily invoke peers and slow down their operation, whereas faulty agents ask for help only when they cannot complete their tasks. We do not explicitly simulate the battery consumption of agents, if they run out of energy a failure is raised. We also assume that failures arise if agents hamper each other when they operate in the physical space.

4) Agent Dependencies: They represent constraints that limit agents from autonomously completing the application tasks and force cooperation among agents. This implies making decisions concerning a global solution plan and distributing the task goals among the agents, possibly operating in different areas of the physical space. We focus on interleaved planning and coordination [14] since this plan synthesis scheme fits with agents efficiently addressing group goals.

B. Coordination Mechanisms

We use the illustrative example shown in Fig. 2 (where agent R1 fails) to describe coordination mechanisms, and we provide sequence diagrams to represent interaction among agents if any. Independently of the coordination mechanism, we assume failed agents take T_{repair} time units on average to be repaired.

1) Fully-Decentralized: This is the most simple mechanism since agents do not communicate with their peers, they autonomously check if there are unaccomplished tasks to be completed. Considering the snapshot of the system in Fig. 2, all agents may enter the RT1 room and decide to swap their task with the uncompleted one if it has greater importance. It is worth noting that for this mechanism, the malicious behavior of agents does not affect the system performance since agents do not receive any deceptive notification, they independently check for unfinished application tasks.

2) Semi-Decentralized: Fig. 4 reports the sequence diagram illustrating the interaction between the agent encountering a failure (A1), and all other agents close to it. A message asking for help (along with the task importance) is broadcasted, and only agents in the same or adjacent room(s) can receive it. Agents compare the communicated importance with the one of the task they are carrying out, and they send a message of being available for rescue only if the importance of the uncompleted task is larger. Free agents have the lowest importance, and they always reply to any rescue request they receive. The agent originating the enquiry for rescue might receive more than one offer, but it sends an acknowledgment to the first offer only, so that all other agents are not interrupted in their activities. Considering the snapshot of the system in Fig. 2, A1 communicates with A2 and A3, but only A2 replies since the importance of its own task is lower than the importance of the uncompleted one. For this coordination mechanism, the malicious behavior of agents affects the system, agents may receive a fake rescue request. To stress this aspect, later in the experimentation, we set the importance of fake uncompleted tasks to a maximum value, so that all agents that receive the request are available for providing help.

Fig. 5: Sequence diagram for the CE mechanism.

3) Centralized: Fig. 5 reports the sequence diagram illustrating the centralized coordination mechanism. The interaction is established between the agent encountering a failure (A1), and a central coordinator (C) that is in charge of deciding which is the most suitable agent to be involved in the rescue. The status check is performed on all agents in the physical space. Four strategies have been implemented for this scope: (i) proximity, i.e., the agent that is closer to the failure; (ii) importance, i.e., the agent showing the lowest importance; (iii) proximity-importance, i.e., the agent with the lowest importance among the closest ones; (iv) importance-proximity, i.e., the closest agent among those with the lowest importance. Considering the example in Fig. 2, A1 contacts C that checks the status of other agents (i.e., A2 to A5). On the basis of the strategies denoted above, C sends a message of acting for rescue to different agents. A2 is contacted whether the proximity is privileged over the importance. C selects A4 or A5 when only the importance is considered, since they are equally eligible due to the condition of being “free”, i.e., no tasks are assigned. In the case of the importance-proximity strategy, C contacts A4 since it is the closest agent to the uncompleted task showing the lowest importance. Similar to SD, the malicious behavior of agents affects the performance of the CE mechanism since the central coordinator may receive fake requests. Note that malicious agents communicate maximum importance when a (fake) rescue request is sent to the central coordinator so that all agents can be selected for rescuing the uncompleted task. Besides, CE builds upon a single component (the coordinator) that, in case of failure, invalidates the rescue operation. Hence, the switch to SD and FD mechanisms becomes even more crucial in this scenario.

Table II summarizes all the parameters that can be set by software developers to analyze their own scenarios. Following

TABLE II: Parameters that software developers can change to de ne and simulate their own system scenarios.

Type	Parameter	Description
Simulation	CoordMech	Coordination mechanism
	T	Observation time
Space	N_{Rooms}	Number of rooms in the space
	Source	Source room
	Targets	Target room(s), source is excluded (-1 means randomly selected)
	Topology	Space topology (room connections)
Tasks	$T_{arrival}$	Avg. time between the creation of two consecutive tasks
	ImpProb	Percentage of tasks with a given importance
	T_{expire}	Avg. time after which a task expires (-1 means tasks do not expire)
Agents	N_{bot}	Number of agents in the space
	$N_{mal}=N_{bot}$	Percentage of malicious agents in the space (N_{mal} N_{bot})
	F_{agent}	Probability of each agent to fail
	T_{load}	Avg. time of agents to start a task
	T_{move}	Avg. time of agents to cross a room
	T_{drop}	Avg. time of agents to nalize a task
	T_{comp}	Avg. time of agents for decision
	T_{comm}	Avg. time of agents to communicate
	T_{repair}	Avg. time to repair an agent
Comm.	F_{msg}	Probability of each message to fail

a survey on decision making in MAS [18], our tool includes time complexity for agents: (i) T_{comp} (computational overhead for decision making), i.e., the amount of time needed to search for a solution; (ii) T_{comm} (communication overhead), i.e., the amount of time needed for the communication. We focus on sequential decision problems with a nite horizon, i.e., decisions need to be made for a nite number of time steps that last up to the accomplishment of the system goal (e.g., the delivery of an object in our scenario). We consider a decentralized learning process, i.e., each agent learns its own policies in parallel to other agents. Our tool also includes the possibility to model the failure probability of communication among agents, i.e., F_{msg} in Table II.

C. Implementation

To enable the system performance analysis, we develop a discrete-event simulator. It is written in Python , and it allows comparing the different coordination strategies. It comes with a library of existing solutions, but developers can easily customize the parameters to build their own application scenarios. Source code is publicly available [16].

IV. Experimentation

In this section, we present experimental results obtained by analyzing synthetic and real-world scenarios where agents are robots whose tasks consist of delivering some items (e.g., medicines) from a source room to a target one. All experimental results (i.e., average, solid line) are shown with the 95% con dence interval (i.e., the shaded area).

Table III shows the value of input parameters used for experiments and set by exploring some literature in this domain [45], [46]. All time parameters follow an exponential distribution. Different values can be used, e.g., the average time of robots to cross a room can refer to ad-hoc requirements. In our scenario, the importance probability of items is set with larger values when referring to items of low/high importance. We consider 50% of generated items showing low importance, 10% medium, and 40% high (see ImpProb in Table III). Underlined values indicate preferred settings unless performance metrics are evaluated as a function of that parameter.

A. Research Questions

The goal of our experimental evaluation is twofold: i) it provides empirical evidence on the impact of different coordination mechanisms when evaluating the performance characteristics of systems subject to reliability and security issues; ii) it shows that our tool can be applied to real-world medium-sized scenarios. We aim to answer three research questions:

- RQ1: What is the impact of workload intensity, malicious robots, and failure probability on the system latency? What is the computation (for decision making) and communication (with probable loss of messages) performance overhead? Motivation: we are interested in evaluating how workload, security, reliability, computation, and communication (with message failures) parameters affect the system performance.
- RQ2: What is the impact of agent dependencies on the system performance? Motivation: we are interested in evaluating to which extent COORDINATE can analyze agent dependencies.
- RQ3: How do workload intensity, number of robots, and rooms in the environment affect the scalability of the proposed tool? Motivation: we are interested in evaluating the scalability of COORDINATE.
- RQ4: Does system design affect the performance of systems deployed in real-world scenarios? Motivation: we are interested in evaluating to which extent COORDINATE can be realistically applied.

To answer these questions, we analyze different scenarios using the proposed tool. COORDINATE is deployed on a cluster hosting a virtual machine with 16 vCPU and 32GB memory. We recall that a replication package is provided for inspection of artifacts and results [16].

B. System Latency

Here we report experiments on the system latency that is de ned as the time interval (i.e., consisting of queuing time and service time) between a user's request of a service and the response of the system [47], and usually upper bounds are de ned as business requirements by stakeholders [48]. In our scenario, the system latency represents the time interval between the generation of an item and its delivery.

Workload intensity. Figs. 6(a)–(c) compare the system latency of the three coordination mechanisms when the average inter-arrival time of items varies from 90 to 300 seconds. As shown in Table III, 25 rooms, 20 robots (of which 18 malicious ones), and a failure probability of 5% are used for these experiments. Fig. 6(a) shows that, with the mesh

(a) Mesh, Latency vs. Workload (b) Ring, Latency vs. Workload (c) Tree, Latency vs. Workload

(d) Mesh, Latency vs. Security (e) Ring, Latency vs. Security (f) Tree, Latency vs. Security

(g) Mesh, Latency vs. Reliability (h) Ring, Latency vs. Reliability (i) Tree, Latency vs. Reliability

Fig. 6: Latency as a function of workload intensity, system security, or system reliability. N_{bot} = 20 robots operate in an environment with N_{rooms} = 25 rooms organized as the mesh, ring, or tree topology. Other system parameters are in Table III.

topology, the inter-arrival time of items affects the three coordination mechanisms in different ways. FD and SD manifest a counter-intuitive result where latency increases when the inter-arrival time is longer, i.e., items are generated less frequently. Checking simulation logs, we observe that a less frequent item generation implies reduced robot mobility (i.e., robots wait in the source room if they have no tasks) and a slower rescue of abandoned objects (i.e., uncompleted tasks). CE has a full knowledge of the system status and a rescue task is assigned to the most suitable robot as soon as an object is dropped. This way, rescue operations are not delayed and the system latency decreases when fewer items are created. Similar considerations hold for the ring topology, see Fig. 6(b). The impact of the load intensity is exacerbated by fewer connections that lead to a larger variation of the system latency. Fig. 6(c) depicts results for the tree topology. Independently of the load intensity, FD is the least efficient mechanism, whereas CE is the best one.

Malicious robots. Figs. 6(d)–(f) depict the system latency when the percentage of malicious robots varies in the analyzed scenario, i.e., N_{Rooms} = 25, N_{bot} = 20, F_{agent} = 5%, and $T_{arrival}$ = 120 seconds. We remind that malicious robots affect only the communication, i.e., they spread fake information to slow down other robots and get a personal advantage. This implies that malicious robots always complete their tasks, hence the system latency does not tend to infinity even when all robots are malicious. Fig. 6(d) shows that, with the mesh topology, CE minimizes the latency independently of the number of malicious robots. This is explained by the strategy adopted by CE that consists of distributing rescue tasks to robots that are not busy, hence optimizing the available agents. FD is not affected by the number of malicious robots since agents do not communicate, and it performs as well as the CE mechanism only when all robots are malicious. The efficiency of CE decreases due to the overhead required to manage a larger number of fake requests. SD is more efficient than FD up to 80% of malicious robots, but for larger values, its performance degrades due to the fake rescue operations required by malicious robots. Differently from CE

TABLE III: Parameter values used to run experiments.

Class	Parameter	Value
Simulation	CoordMech	FD, SD, CE
	T	3 months
Space	N_{Rooms}	4, 9, 16, 25, 36, 49
	Source	0
	Targets	-1
	Topology	Mesh, Ring, Tree
Tasks	$T_{arrival}$ [sec]	90, 120, 150, 180, 210, 240, 270, 300
	ImpProb	50%: low, 10%: medium, 40%: high
	T_{expire}	-1
Agents	N_{bot}	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
	N_{mal} =N_{bot}	0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1
	F_{agent} [%]	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
	T_{load} [sec]	1
	T_{move} [sec]	5
	T_{drop} [sec]	1
	T_{comp} [sec]	0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1
	T_{comm} [sec]	0, .01, .02, .03, .04, . . . , .08, .09, .1
	T_{repair} [sec]	1800
Comm.	F_{msg} [‰]	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

(recall that it tries to assign legit or fake rescue tasks to robots that are not busy) SD allows failed (or malicious) robots to ask for help only to agents that are close (i.e., in adjacent rooms). Fig. 6(e) depicts the ring topology, and the impact of malicious robots is more evident (see the latency magnitude). CE performs well when the number of malicious robots is small, otherwise FD (i.e., no communication) should be preferred. In some cases, SD is also a valuable mechanism since only robots in adjacent rooms may be affected by fake rescue tasks. These results con rm the importance of switching coordination mechanisms for system performance optimality. Fig. 6(f) reports the system latency with the tree topology. In this case CE always outperforms SD and FD. Similarly to Fig. 6(d), we notice that also here the performance of CE and SD is indeed affected by the percentage of malicious robots that entails more robots involved in the rescue of useless tasks, and inevitably delaying the overall delivery of items.

Failure probabilities. Figs. 6(g)–(i) depict the system latency of the three coordination mechanisms when the failure probability of robots (F_{agent}) varies between 1% and 10% [46]. For these experiments, N_{Rooms} = 25, N_{bot} = 20, N_{mal} = 18, and $T_{arrival}$ = 120 seconds. Fig. 6(g) shows that, in the mesh topology, CE is optimal with F_{agent} 7%, FD is optimal for all other values. Fig. 6(h) illustrates the ring topology, and CE is worse than FD and SD when F_{agent} > 4%. Fig. 6(i) shows the performance of the tree topology. In this case, FD is always the least ef cient mechanism, while CE is the best one. For small values of F_{agent}, the system latency of SD is similar to the one of FD, but its performance gets closer to the one of CE when F_{agent} 8%.

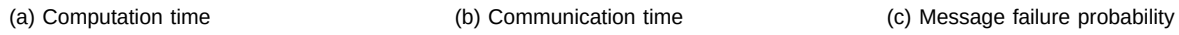
Computation and communication (with probable loss of messages)COORDINATE allows developers to tune other parameters, i.e., T_{comp}, T_{comm}, and F_{msg}. However, their effect on the system latency is negligible due to the different order of magnitude used for mechanical (e.g., movements) and software (e.g., computation/communication) times. For the sake of space, these results are not depicted. To observe the effect of these parameters on the system latency, we considered a different system parameterization (see the replication package as in Table III with only two exceptions:

age [16] for details on numerical values). Results are shown in Fig. 7 for an environment with 16 rooms organized as a ring topology. Fig. 7(a) shows that the computation time (for decision making) does not affect the latency of the CE mechanism since all calculations are made by the coordinator. The latency of systems with FD and SD mechanisms increases with T_{comp} since, in these cases, agents are in charge of determining the next steps. The system latency as a function of communication time is shown in Fig. 7(b). SD and CE are affected by this parameter since agents need to exchange information (with other agents or the coordinator, respectively). No effect is observed when agents act in an independent way, i.e., using the FD mechanism. In this scenario, CE is the optimal mechanism for T_{comm} 0:35 seconds, otherwise FD is preferable. The SD mechanism is still the solution with the longest latency. Fig. 7(c) depicts the effect of lost messages on the system latency. The higher F_{msg}, the more likely a message is lost. While this type of failure has no effect on the FD mechanism that does not leverage communication, it affects the system latency of SD and CE mechanisms. SD works better with large F_{msg}, and this is due to malicious robots not being able to spread fake rescue requests. The system latency deteriorates for large values of F_{msg} when CE is adopted. This is due to the central coordinator that lessens the effect of malicious robots and makes communication crucial for fast rescues.

> RQ1: Workload intensity, malicious robots, and failure probability of robots affect the system latency when adopting different coordination mechanisms. It is crucial to take into consideration the space topology since the observed performance can largely deviate. Computation and communication (with probable loss of messages) time do not affect the performance of the considered system, although they may impact other scenarios.

C. Agent Dependencies

Fig. 8(a) shows the interleaved plan synthesis scheme [14] that we consider in our experimentation. The physical space (N_{rooms} = 25 and N_{bot} = 20) is split into two sub-areas: the first one with $N_{rooms\ 1}$ = 10 (i.e., horizontal-dashed nodes and the white node) and N_{bot1} = ($N_{rooms\ 1}$ N_{bot})=N_{rooms} = 8; the second one with $N_{rooms\ 2}$ = 13 (i.e., shaded nodes and the white node) and N_{bot2} = N_{bot} N_{bot1} = 12. Note that the room represented by the white node (whose ID is 6) is shared by both sub-areas. Robots can move only in the sub-area where they are deployed and are allowed to communicate only with other robots in the same sub-area. All items are generated in room 0 (i.e., top-left node) and may be delivered in any other room. Robots need to cooperate in case an item has to be transported to the second sub-area. For example, an item that should be delivered to the bottom-right node of Fig. 8(a) is first moved from node 0 to node 6 by a robot of the first sub-area. Then, the item is taken by a robot of the second sub-area and brought to its destination. Task and agent parameters are set

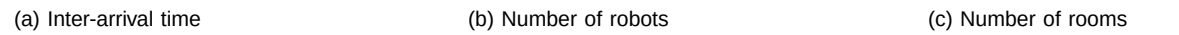


(a) Computation time (b) Communication time (c) Message failure probability

Fig. 7: Latency as a function of computation time, communication time, or message failure probability. $N_{bots} = 20$ robots operate in an environment with $N_{rooms} = 16$ rooms organized as the ring topology.

(a) Plan Synthesis Scheme (b) Latency vs. Workload (c) Latency vs. Security (d) Latency vs. Reliability

Fig. 8: The considered plan synthesis scheme (a) and its latency against workload intensity (b), security (c), and reliability (d). The green-shaded area in (b), (c), and (d) is where the system latency falls when different mechanism combination (i.e., not FD+CE or CE+CE) are used. The 95% con dence interval (still narrow as for Fig. 6) is not depicted for the sake of readability.



(a) Inter-arrival time (b) Number of robots (c) Number of rooms

Fig. 9: Simulation time required to monitor 3 months of activity of a system deployed in a Mesh topology.

$T_{arrival\ 2}$ (i.e., the inter-arrival time of the second sub-area) depends on the number of items delivered to room 6 by robots in the first sub-area;
the number of malicious robots in the two sub-areas is:

$$N_{mal\ 1} = \left\lfloor \frac{N_{mal}}{2} \right\rfloor \text{ and } N_{mal\ 2} = \left\lceil \frac{N_{mal}}{2} \right\rceil.$$

As for all other parameters in Table III, these values are nominal and COORDINATE allows tuning them to better model the system under investigation.

Figs. 8(b)–(d) show the mechanisms that allow the system to minimize its latency when the load intensity, the number of malicious robots, and the failure probability of agents vary. In all these cases, two mechanism combinations (i.e., FD+CE and CE+CE) minimize the latency of the considered system. With light conditions (i.e., long inter-arrival time, a small number of malicious agents, or small agent failure probability), both sub-areas should use a centralized mechanism to coordinate agents (i.e., CE+CE). Otherwise, when the system operates under heavy conditions, it is more efficient to use the fully-decentralized mechanism in the first sub-area, and the centralized mechanism in the second sub-area (i.e., FD+CE). The latency observed using any other mechanism combination falls in the green-shaded area depicted in Figs. 8(b)–(d). Note that Figs. 8(b)–(d) depict the average system latency only; the 95% con dence interval is still observed to be narrow (meaning that our results are accurate, as for predictions in Fig. 6) but it is omitted for the sake of readability.

RQ2: COORDINATE successfully analyzes an interleaved plan synthesis scheme to study the effect of agent dependencies, i.e., robots must collaborate to deliver items. In the considered case, COORDINATE identifies which combination of mechanisms allows optimizing the system latency when agents are deployed and operate in different sub-areas of the physical space.

D. Scalability

To evaluate the scalability of our tool, we focus on the mesh topology since nodes are more interconnected, we refer to the replication package [16] for results of other topologies.

Workload intensity. Fig. 9(a) reports the time required to simulate three months of activities when $90 \leq T_{arrival} \leq 300$ seconds, other parameters are shown in Table III. The three coordination mechanisms behave similarly across the analyzed space topologies. FD requires the lowest simulation time, which is comparable to the time of SD while simulating the CE mechanism takes always the longest time, 800 seconds.

Number of robots. Fig. 9(b) shows the simulation time when the number of robots in the system varies from 10 to 100. The simulation time of FD does not change with N_{bot} since robots do not communicate. The simulation time of SD and CE mechanisms increases with N_{bot} and the largest impact is observed with the CE mechanism, i.e., up to 1.2k seconds.

Number of rooms. Fig. 9(c) depicts the simulation time required to simulate a system with $4 \leq N_{Rooms} \leq 49$. The CE mechanism shows the longest simulation time (i.e., more than 4k seconds), which increases with the number of rooms. This is due to more connections in the topology and, consequently, to more alternatives to reach target areas.

RQ3: The scalability of our tool is influenced by workload intensity, number of robots, and number of rooms in the physical space. Of these parameters, the number of rooms is the one which affects the simulation time the most. The simulation of the CE coordination mechanism takes longer due to many agent interactions.

E. Real-world Topologies

We use our tool to predict the performance of a system deployed at the Ghent University Hospital [49]. The physical space of the considered department (extracted from [49]) is shown in Fig. 10(a), and it is constituted of a hallway, patient rooms, a nursing post, and other service spaces (e.g., storage, kitchen, terrace). A graph with 73 nodes is generated starting from the floor plan of the department: nodes 0–22 describe the hallway, nodes 23–41 are patient rooms, and other nodes are (part of) service spaces. We assume available robots (i.e., $N_{bot} = 20$) pick items up at node 66 (i.e., the elevator hallway) and move them to one of the other nodes (i.e., rooms or hallway). For example, a robot might bring food and bed sheets to a patient's room, medicines and medical equipment to the nursing post (nodes 50–53), or leave a stretcher in front of some rooms, i.e., in the hallway. Parameters used in this scenario (except those related to the physical space) are shown in Table III. We recall that numerical values of these parameters, as well as the topology of the department, can be changed at user discretion.

Results obtained for the considered system and space topology are shown in Figs. 10(b)–(d) when the workload intensity, the percentage of malicious robots, and the robot reliability vary, respectively. Although the system latency magnitude is larger than the one observed for synthetic cases (i.e., mesh, ring, and tree, see Fig. 6), interestingly findings are similar, the performance of considered coordination mechanisms are affected by the parameters used to define application tasks and system agents. This indeed consolidates the importance of switching coordination mechanisms to improve the performance of systems subject to reliability and security issues.

RQ4: Our tool successfully models a system deployed in a real-world environment. Results show that the workload intensity, system security, and reliability affect the performance of real systems. A strategy that adapts the coordination mechanism to system-monitored data is crucial to optimize the system performance.

F. Threats to Validity

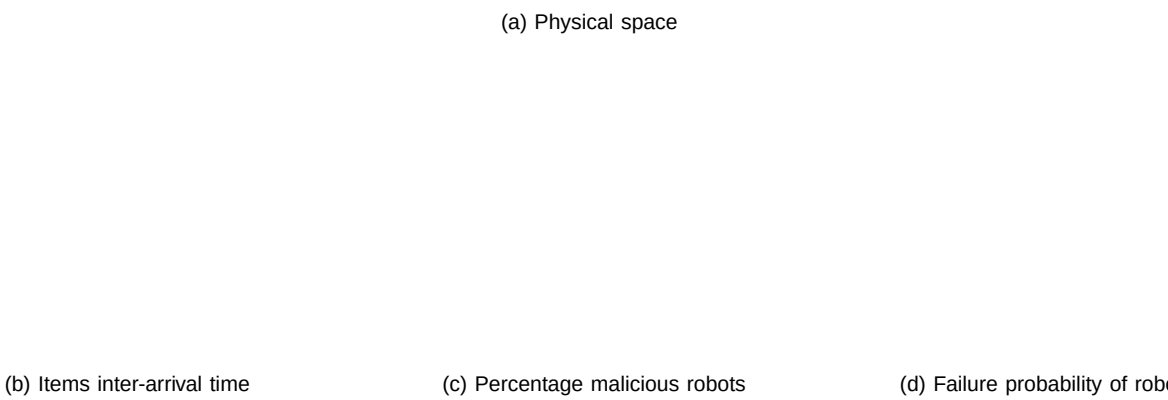
Internal validity. The implementation of the simulator has been manually validated by checking the logs and considering a few scenarios. We are aware that this may represent a threat; to smooth it, we make the code publicly available for inspection and all experiments are fully reproducible.

External validity. Generalization of results is very difficult in this context since our tool has been applied only to medium-sized scenarios. We are aware that conclusions are highly dependent on the settings of parameters, but this is an open issue for all simulation-based approaches. However, we foster predictive results as peculiar and quantitative knowledge for software developers that may want to enable switching of coordination mechanisms on the basis of their own system peculiarities. We plan to involve software developers in the experimentation of further case studies as future work, thus assessing the validity of obtained results.

Construct validity. To keep under control the accuracy of experimental results presented in this section, all performance indicators are provided with their 95% confidence interval. Besides, our interest is not related to absolute values, the goal of our experimentation is to show that coordination mechanisms differently contribute to performance indicators when system peculiarities vary.

V. Conclusion

In this paper, we present a tool-based approach that enables the performance evaluation of three different coordination mechanisms for fault-tolerant MAS. Specifically, we compare such mechanisms quantifying their impact on the system latency while exploring variations of security and reliability



(a) Physical space

(b) Items inter-arrival time

(c) Percentage malicious robots

(d) Failure probability of robots

Fig. 10: Physical space and experimental results for a department of the Ghent University Hospital [49].

issues. We also study the effect of agent dependencies on the performance of MAS. Experimental results con rm that it is crucial to analyze circumstances triggered by security and reliability concerns, and wisely choosing the most suitable (from a performance-based perspective) coordination mechanism. Our tool, namely COORDINATE, identi es the optimal coordination mechanism and improves the system latency up to 4.2 in the considered synthetic scenarios. COORDINATE can be used also to study real-world systems since it scales well with respect to load intensity, number of agents, and number of rooms. In the case of the Ghent University Hospital, we observe average improvements for system latency of,1.5 1.8 , and 2.2 when the load, security, and reliability vary, respectively.

As future work, besides addressing all the limitations discussed as part of threats to validity, we plan: (i) to validate the accuracy of performance results by comparing them with actual measurements from the system implementation; (ii) to use COORDINATE in different domains for investigating its usefulness across diverse applications.

REFERENCES

[1] L. Tian, Y. Hua, X. Dong, J. Lv, and Z. Ren, "Distributed time-varying group formation tracking for multi-agent systems with switching interaction topologies via adaptive control protocols," IEEE Trans. on Industrial Informatics, 2022.

[2] B. Ning, Q.-L. Han, Q. Lu, and J. G. Sanjayan, "Cooperative control of multi-robot systems subject to control gain uncertainty," IEEE Trans. on Industrial Informatics, 2022.

[3] J. Lin, S. Sedigh, and A. Miller, "Modeling cyber-physical systems with semantic agents," in Annual Computer Software and Applications Conf. Workshops (COMPSACW), 2010, pp. 13–18.

[4] L. Zhao and G.-H. Yang, "End to end communication rate-based adaptive fault tolerant control of multi-agent systems under unreliable interconnections," Inf. Sciences, vol. 460-461, pp. 331–345, 2018.

[5] Y. Zhang, Z. Guo, J. Lv, and Y. Liu, "A framework for smart production-logistics systems based on CPS and industrial IoT," IEEE Trans. on Industrial Informatics, vol. 14, no. 9, pp. 4019–4032, 2018.

[6] D. Hästbacka, J. Halme, L. Barna, H. Hoikka, H. Pettinen, M. Laarca, M. Björkbom, H. Mesia, A. Jaatinen, and M. Elo, "Dynamic Edge and Cloud Service Integration for Industrial IoT and Production Monitoring Applications of Industrial Cyber-Physical Systems," IEEE Trans. on Industrial Informatics, vol. 18, no. 1, pp. 498–508, 2022.

[7] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart agents in industrial cyber–physical systems," Proceedings of the IEEE, vol. 104, no. 5, pp. 1086–1101, 2016.

[8] T. Bures, R. Calinescu, and D. Weyns, "Special issue on software engineering for trustworthy cyber-physical systems," Journal of Systems and Software, vol. 178, p. 110972, 2021.

[9] S. Bernardi, U. Gentile, S. Marrone, J. Merseguer, and R. Nardone, "Security modelling and formal veri cation of survivability properties: Application to cyber–physical systems," Journal of Systems and Software, vol. 171, p. 110746, 2021.

[10] N. Garg, D. Grosu, and V. Chaudhary, "Antisocial behavior of agents in scheduling mechanisms," IEEE Trans. on Systems, Man, and Cybernetics, vol. 37, no. 6, pp. 946–954, 2007.

[11] H. Yang, Q.-L. Han, X. Ge, L. Ding, Y. Xu, B. Jiang, and D. Zhou, "Fault-tolerant cooperative control of multiagent systems: A survey of trends and methodologies," IEEE Trans. on Industrial Informatics, vol. 16, no. 1, pp. 4–17, 2019.

[12] W. He, W. Xu, X. Ge, Q. Han, W. Du, and F. Qian, "Secure Control of Multiagent Systems Against Malicious Attacks: A Brief Survey," IEEE Trans. on Industrial Informatics, vol. 18, no. 6, pp. 3595–3608, 2022.

[13] C. Chen, Y. Chen, J. Zhao, K. Zhang, M. Ni, and B. Ren, "Data-Driven Resilient Automatic Generation Control Against False Data Injection Attacks," IEEE Trans. on Industrial Informatics, vol. 17, no. 12, pp. 8092–8101, 2021.

[14] A. Torreño, E. Onaindia, A. Komenda, and M. Stolba, "Cooperative Multi-Agent Planning: A Survey," ACM Computing Surveys, vol. 50, no. 6, pp. 84:1–84:32, 2018.