Journal of Intelligent Manufacturing (2019) 30:2999–3013
https://doi.org/10.1007/s10845-019-01478-9

Multi-agent system architectures for collaborative prognostics

Adrià Salvador Palau[1] · **Maharshi Harshadbhai Dhada**[1] · **Ajith Kumar Parlikad**[1]

Received: 26 February 2019 / Accepted: 3 June 2019 / Published online: 12 June 2019

Abstract
This paper provides a methodology to assess the optimal multi-agent architecture for collaborative prognostics in modern fleets of assets. The use of multi-agent systems has been shown to improve the ability to predict equipment failures by enabling machines with communication and collaborative learning capabilities. Different architectures have been postulated for industrial multi-agent systems in general. A rigorous analysis of the implications of their implementation for collaborative prognostics is essential to guide industrial deployment. In this paper, we investigate the cost and reliability implications of using different multi-agent systems architectures for collaborative failure prediction and maintenance optimization in large fleets of industrial assets. Results show that purely distributed architectures are optimal for high-value assets, while hierarchical architectures optimize communication costs for low-value assets. This enables asset managers to design and implement multi-agent systems for predictive maintenance that significantly decrease the whole-life cost of their assets.

Keywords Multi-agent systems · Distributed systems · Prognostics · Asset management · Predictive maintenance · Cost assessment

Introduction

The potential of using computational models to enable real-time machine failure prediction (prognostics) has been known since the 1980's (Buchanan 1986). However, it wasn't until recent advances in sensing and communication technologies that real-time prognostics became possible. Among these advances were cheaper, less power-consuming sensors and improved telecommunications, that allowed for continuous monitoring of machines and led to the emergence of the Internet of Things. In the Internet of Things, a network of connected devices gather and share information about their surroundings (Atzori et al. 2010; McFarlane 2018). In the industrial context, this is referred to as the Industrial Internet of Things (IIoT) (Gilchrist 2016; Li et al. 2018), a paradigm that together with improvements in regression techniques and computing power is set to revolutionize the field of prognostics. In the IIoT, data gathered through sensors embedded in the machines can be leveraged to perform real-time failure detection and prediction for machines in a machine fleet, thus significantly reducing maintenance cost and machine downtime (Li et al. 2018; Ning 2016).

The IIoT enables the use of multi-agent systems as a framework for prognostics and other manufacturing problems (Brennan et al. 2002; Mařík and Lažanskỳ 2007; Monostori et al. 2006). Multi-agent systems (MAS) are systems of independent software elements that can be used to aid humans in the process of taking decisions (Ferber and Weiss 1999). They have been postulated as a suitable framework to deal with the complexity of industrial asset fleets formed by heterogeneous assets (Leitão and Karnouskos 2015). Multi-agent systems have been especially successful in aiding humans to take decisions in complex environments such as traffic management, industrial production, etc. (Wooldridge and Jennings 1995).

The history of multi-agent systems is intrinsically linked to our understanding of the meaning of the word 'agent', the definition of which has been a long-lasting topic of debate (Nwana 1996). This paper conforms to the definition of agents typically used for industrial systems: agents as autonomous, problem-solving, and goal-driven computational entities with social abilities (Leitão and Karnouskos 2015). Multi-agent systems remain one of the most prolific frameworks to manage continuous monitoring systems, and recently they have been postulated as a way of providing assets with a certain degree of agency (Palau et al. 2019b).

✉ Adrià Salvador Palau
as2636@cam.ac.uk

[1] Department of Engineering, Institute for Manufacturing, University of Cambridge, Cambridge CB3 0FS, UK

From this idea, collaborative prognostics has been proposed as a framework in which agents share information with each other in order to improve failure predictions, thus optimizing predictive maintenance (Palau et al. 2019a).

A multi-agent system is defined by its architecture, that determines the structure and topology of its agents. Multi-agent system architectures have been broadly classified into four types: Centralized, Hierarchical, Heterarchical, and Distributed (or peer-to-peer) [see Sallez et al. (2010) and Leitão and Karnouskos (2015)]. In collaborative prognostics, where agents are often linked with individual assets, the optimal architecture will be determined by its influence on the overall cost and reliability of the system. Collaborative prognostics in large fleets of assets comprehends several cost factors: communication, computational, and maintenance. Traditionally, maintenance costs were considered cardinal. With the advent of IIoT technologies, communication and computational costs have become relevant due to the large amount of data processed and transmitted through the internet in continuously monitored fleets.

When applied to predictive maintenance, several of the canonical architectures of multi-agent systems require dramatically increasing the amount of processing and communication within the fleet, as real-time peer-to-peer communication and prognostics are supported. State of the art prognostics use a plethora of machine learning algorithms (Khan and Yairi 2018; Lee et al. 2014), which are often computation and data intensive (Konecnỳ et al. 2016). Therefore, it becomes crucial to quantify how maintenance costs compare to other costs in order to assess the suitability of different MAS architectures.

In this paper, we compare several canonical multi-agent architectures for collaborative prognostics on the basis of different cost balances between communication, maintenance and computation. Concretely, we study the effect of varying asset value and communication costs in the overall cost of the architecture, and we show that different architectures are optimal for different industrial scenarios.

Apart of the cost constraints explicitly dealt with in this paper, the implementation of a multi-agent architecture may be limited by other constraints such as human resources or available capital. This is especially important in the case of SME's, or industries operating in a context of low financial liquidity. This paper does not deal with such managerial details, but they must nonetheless be taken into account beforehand by any Asset Manager wishing to implement the proposed system in practice.

After an Abstract and an Introduction, a further literature review is presented in "Pertaining Literature". This is followed by a description of Collaborative Prognostics, and of the maintenance policy followed by the agents in our implementation. Following this, "Cost analysis" section describes how different architectures are benchmarked through their operational cost, and presents a normalized cost measure. This is followed by a description of the agent typologies used in the system's architectures in a section called "Agent typologies and their failure modes". The multi-agent architectures are presented right after, in a homonymous section. This is followed by a brief description of the implementation of the architectures in a multi-agent system simulation software called Netlogo, and a description of the distributed clustering algorithm used in the implementation of the Distributed architecture. Experiments are described in a section with the same name, and the results obtained from these experiments are described in "Results and discussion" section. A methodology to select a multi-agent architecture is described in "A methodology for architecture evaluation" section. The paper ends with a conclusion, and description of future work.

Pertaining literature

While the formal definition of the term agent varies across the literature, there is at least a consensus over the way MAS function (Weiss 1999; Wooldridge and Jennings 1995; Ferber and Weiss 1999). In multi-agent systems, the overall system goal is subdivided into agent-level goals depending on the knowledge and reasoning skills of the agents within the system. Agents perceive their local environment, and have a partial view of the system by communicating with other agents. It is through this communication that agents collaborate with one another, and make decision to reach the overall system goal. The level of intelligence and relationships among the agents is defined by the designer of the system, or the final user (Brennan et al. 2002; Mařík and Lažanskỳ 2007; Monostori et al. 2006).

Multi-agent system architectures are typically defined in terms of decision-making, and thus vary from being completely distributed (where all agents are at the same decision-making level), to being purely centralized (similar to traditional centralized control systems) (Andreadis et al. 2014). Additional agents like mediators, or brokers, may be present in the system to govern a sub-group of agents, thus generating architectures with an intermediate degree of distribution (Andreadis et al. 2014). In this paper, we focus on four broad classes of MAS architectures: Distributed, Heterarchical, Hierarchical, and Centralized. These four classes correspond to the four classes of decision-making architectures identified in Sallez et al. (2010) and Leitão and Karnouskos (2015).

The Hierarchical, Heterarchical, and Distributed MAS architectures have their origin in traditional control systems, that by the end of the twentieth century evolved to more distributed frameworks (Trentesaux 2009). A flexible decision-making approach was preferred over a rigid one pro-

Table 1 Brief overview of research featuring the use of multi-agent systems in manufacturing industries

Reference	Year	Application	Use of MAS
Duffie and Piper (1986)	1986	Job scheduling	Represent entities of a shop floor using agents to enable dynamic job scheduling
Djurdjanovic et al. (2003)	2003	Prognosis and diagnosis	Agent analyzes the data for diagnosis and prognosis
Wong et al. (2006)	2006	Job scheduling	Agents negotiate and evaluate cost for optimal job scheduling
Tang et al. (2006)	2006	Maintenance planning	Optimize a maintenance model using a reinforcement learning model implemented over a MAS framework
Liu et al. (2007)	2007	Prognostics	Prognostics of shipboard power systems
Xiang and Lee (2008)	2008	Task sequencing	Part and machine agents optimize the task sequencing operation
Fasanotti (2014, 2018)	2014	Maintenance planning	Forecast maintenance needs of geographically distributed assets
Hernández et al. (2014)	2014	Supply chain management	Collaborative learning in supply chain management
Wang et al. (2016)	2016	Smart factory	A coordinator agent decides upon the optimal solution after lower-level agents negotiate
Upasani et al. (2017)	2017	Maintenance planning	Agents representing various departments of a shop floor collaboratively plan a maintenance schedule
Li and Parlikad (2017)	2017	Workload assignment	Coordinator agent continuously monitors the asset agents to assign optimal workload to reduce the overall operations cost
Ghita et al. (2018)	2018	Maintenance planning	Maintainer and Producer agents collaborate to improve prognostics and production and maintenance activities

viding optimal solutions under hard constraints, thus spurring the rise of decentralized architectures (Leitão 2009; Trentesaux 2009).

The earliest type of distributed framework was Hierarchical, where the information flowed from lower levels in the architecture to higher-level agents until a suitable decision-making level was reached. Decisions then flowed in the opposite direction (Leitão 2009). However, the search for a suitable decision maker, and the following computations induced lag, compromising the real-time capabilities of the system. This was solved by allowing decision makers at the same level to coordinate. Such 'heterarchical' frameworks are re-configurable, and have substantially improved short-term optimization (Trentesaux 2009).

The use of MAS as a framework for decision-making and control in manufacturing industries has been proposed by several researchers (Vrba 2013; Shen et al. 2006). Table 1 shows various examples where the use MAS has shown to optimise the operations.

In conclusion, the literature presents ample evidence for the use of MAS as a decision-making framework for varied applications in the manufacturing industry. one such application, collaborative prognostics, is considered in this paper and implemented for different well-known MAS architectures.

Collaborative prognostics

The concept of collaborative prognostics extends the concept of collaborative agents into the field of prognostics and health management. Collaborative agents share information with each other in order to jointly achieve a given objective (Tan 1993; Nwana 1996). In collaborative prognostics, machines (through their agents) behave like social entities, communicating with one another and taking their own decisions. In its core, collaborative prognostics involves formation of clusters of similar machines, and collaboration among machines within these clusters to improve failure prediction and predictive maintenance. This collaboration can either be in the form of exchanging model parameters or condition data (Palau et al. 2019b).

In contrast to conventional fleet-wide prognostics methods that rely on a single computer, collaborative prognostics

is distributed, flexible and occurs in real-time. Moreover, it has been shown that collaborative prognostics is theoretically more cost-effective compared to self-learning [prognostics using the machine's own data (Palau et al. 2019b)], and whole fleet learning under certain conditions (Palau et al. 2019a).

So far, the feasibility of collaborative prognostics has been shown using a modified hierarchical architecture (Bakliwal et al. 2018). This architecture was applied to a scenario in which a simulated fleet of turbofan engines was managed using agents. Each engine was assigned an agent, a "Digital Twin", which in turn were connected to one another via a "Social Platform" agent. Prediction was done initially using sliding-window classification (Bakliwal et al. 2018), later expanded to recurrent neural networks (Palau et al. 2018).

In this paper, collaborative prognostics is implemented using four different canonical multi-agent architectures. In order to evaluate realistic industrial scenarios involving several hundreds of machines communicating with each other, it is important to reduce the complexity of the analysis. In prognostics, a standard way to do this is through a Health Indicator, a synthetic numerical indicator extracted from the asset's sensor values that upon reaching a pre-defined threshold is assumed to signify asset failure [see, for example, Wang et al. (2008) and Yan et al. (2004)].

Similar to Palau et al. (2019b) and Wang et al. (2008), we choose an inverse exponential Health Indicator:

$$\mathrm{HI_i}(t_{li}) = a_i \left(1 - e^{-b_i (t_{fi} - t_{li})}\right) + \epsilon_{0,\sigma}, \tag{1}$$

in this equation, t_{li} is the local time of the asset: the time since the last repair or installation. (a_i, b_i, t_{fi}) are the parameters that define the behaviour of the Health Indicator. b_i is a curvature parameter (the smaller b_i is, the sharper the deterioration). a_i determines the expected value of $\mathrm{HI_i}$ at $t_{li} = 0$. t_{fi} is the average (or expected) time of failure. $\epsilon_{0,\sigma}$ is a random term with standard deviation σ and 0 mean, conforming to a Gaussian distribution. In this paper, a_i is normalised to 1, and thus $\sigma = 1$ represents a level of noise that often reaches 100% of the value of the health indicator. This means that in realistic situations $\sigma < 0.5$. Assets are assumed to have failed when $\mathrm{HI_i} \leq 0$.

In this paper, collaborative prognostics is implemented by sharing Health Indicator data among similar agents in the system. The accumulation of data, if belonging to an asset with similar (a_i, b_i, t_{fi}), increases the accuracy of prediction. On the opposite, if the assets are dissimilar, data sharing decreases it.

Maintenance policy

The agents in the system responsible of prognostics will propose the following predictive maintenance policy to human operators:

- *Predictive maintenance* assets should be preventively repaired when their time since installation or last repair surpasses the predicted time of failure multiplied by a factor, η: $t_{li} > \eta t^e_{fi}$, $\eta < 1$.
- *Corrective maintenance* assets should be correctively repaired immediately upon failure.

In this policy, t^e_{fi} is the estimated time of failure of the asset i in the fleet. Ideally, η can be optimized in real time by assuming that the agent's estimated prognostics parameters approximate the true ones, as the problem reduces to a replacement policy problem [see Jardine and Tsang (2005) and Palau et al. (2019b)]. In this paper, however, we decide to set η into a fixed value of 0.7 in order to satisfy computational constraints and help comparison across experimental cases (experiments showed that the value η was not relevant for the comparison between architectures as long as η was the same across experiments).

Cost analysis

An accurate estimation of the cost of a multi-agent system is crucial to choose between different architectures for a given implementation scenario. The cost, C_T incurred by operating the multi-agent architectures presented in this paper can be divided in three main components: maintenance, communication, and processing (computational) costs,

$$C_\mathrm{T} = C_\mathrm{M} + C_\mathrm{C} + C_\mathrm{P} = N_\mathrm{C}\Gamma + N_\mathrm{P}\gamma + C_\mathrm{C} + C_\mathrm{P}, \tag{2}$$

where C_M is the maintenance cost of the assets, C_C is the communication cost, and C_P is the processing cost. The maintenance cost, C_M is formed by the predictive maintenance cost, γ, and the corrective maintenance cost Γ of one asset. N_C and N_P are the number of times that corrective and predictive actions have been taken at any given time.

In normal conditions, the predictive maintenance cost is a small fraction of the corrective maintenance cost, $\gamma = \alpha\Gamma$ where $\alpha \ll 1$. For this paper corrective maintenance is assumed to correspond to the full replacement of the asset that has failed, which means that its cost can be assumed to be the proportional to the acquisition cost of the asset[1] $\Gamma \propto C_\mathrm{A}$. In this paper 'high value' assets correspond to assets with a high value of Γ, and consequently 'low value' assets correspond to assets with a low value of Γ. Equation (2) can be re-written:

$$C_\mathrm{T} = \Gamma(N_\mathrm{C} + \alpha N_\mathrm{P}) + C_\mathrm{C} + C_\mathrm{P}. \tag{3}$$

[1] Other costs, such as downtime cost, human resources, etc, should be also considered for the exact mathematical dependency.

The precise monetary amount represented by each of these components necessarily depends on the particularities of the system studied. Notwithstanding, it is a safe assumption that individualized communication and processing costs will approximately be the same across different implementation scenarios. Regardless of whether the data comes from a smart phone or a gas turbine, the cost of processing a byte of that data and sending it through the Internet is the same. It is then useful to normalize these costs to the corrective maintenance cost, Γ. Equation (2) then reads:

$$C_t = (N_C + \alpha N_P) + N_{Co} C_c + N_{pro} C_p, \tag{4}$$

where C_c, C_p, and C_t are normalized to the corrective maintenance cost Γ. N_{Co} is the number of fixed-size (a pre-set byte amount) communications between any two agents in the system. N_{pro} is the number of times a fixed computational resource measure (for example, one flop) is used in the system. In practical terms, this means that if in one system the corrective maintenance cost is £10,000, and in another one it is twice that, C_c will be reduced by half in the simulation (as C_c is normalised to Γ). Thus, in this paper, costs are pre-set in three parameters, all normalized to Γ: the fixed-length communication cost C_c, the fixed computational resource cost C_p, and the predictive maintenance cost α.

To further compare across architectures and experimental scenarios, it is important to normalize cost to the time that the system has been operating and to the number of assets present in each experiment. This is needed because experiments with a larger number of assets, and with a longer operation history will generate larger costs. For this purpose, we will use the normalized cost K:

$$K = \frac{1}{TN}\left((N_C + \alpha N_P) + N_{Co} C_c + N_{pro} C_p\right), \tag{5}$$

where N is the number of assets in the system and T is the total time of the simulation (the number of steps since initialization).

Agent typologies and their failure modes

The architectures reviewed in this paper are formed by four elements: Virtual Assets, Digital Twins, Mediator Agents, and a Social Platform. Some of these agents were already described in several publications (Palau et al. 2019a; Bakliwal et al. 2018; Palau et al. 2018, 2019b), and their description here is inspired in the original papers.

The agent's failure modes have been restricted to affect their deliberative and communicative capabilities. The experiments are set up under the assumption that there will be no data loss upon agent failure due to the widespread nature of backup systems in industry.

Virtual asset

Virtual Assets are the lowest-level agents employed in collaborative prognostics. The Virtual Assets' tasks are limited to standardizing the data coming from their corresponding physical assets, and sending that data to upper layers in the architecture. It must be mentioned that because of the rather simple tasks that they perform, Virtual Assets fail to satisfy some widely-accepted definitions of agents [see for example Nwana (1996)]. However, they are critical for the functioning of the system and thus we include them in our analysis.

Virtual Assets act as passive nodes of the architecture, and have no deliberative capabilities. Their data is divided in three main components: a set of sensor-produced features, a set of timed failures or warnings, and a unique identifier. Virtual Assets are formed by two building blocks: a Standardizer, dedicated to standardize the data coming from their assigned assets, and a Communications Manager, that controls the communications with the upper layers of the architecture.

Failure Virtual Asset's failure corresponds to the severance of communications between a deteriorating asset and the rest of the architecture, and thus the halt of prognostics for this particular asset.

Digital twin

Digital Twins are smart agents with prognostics, communication, and data preprocessing capabilities. When Digital Twins are employed, each physical asset in the industrial system is assigned its individual Digital Twin.

Digital Twins are composed of three building blocks: an Analytics Engine, a Data Repository, and a Communications Manager. The Analytics engine computes prognostics and the maintenance policy, the Data Repository manages the data available to the Twin, and the Communications Manager controls the communication between the Digital Twin and other elements of the architecture. This includes the capability of independently choosing other Twins to collaborate with.

Failure The failure of a Digital Twin implies (1) that its communication with other agents is severed, (2) that the system stops providing maintenance recommendations for the physical asset assigned to the faulty Digital Twin, and (3) that the Digital Twin cannot perform any computation.

Mediator agents

Mediator Agents are intermediate agents able to perform prognostics and determine the maintenance policy for groups of assets. They are also able to receive data from the Virtual Assets, and send data to upper layers of the architecture.

Table 2 Brief description of the roles different agents play in each of the architectures

Agent	Centralized	Hierarchical	Heterarchical	Distributed
Social network platform	Clustering	Clustering	Clustering	NA
	Predictive maintenance	Mediator control		
Mediator	NA	Predictive maintenance	NA	NA
Digital twin	NA	NA	Predictive maintenance	Distributed clustering
			Peer to peer communications	Peer to peer communications
				Predictive maintenance
Virtual asset	Standardize the incoming data and pass it on to the higher levels			

Mediator Agents can communicate with each other through the Social Platform.

Mediator agents are composed by the same building blocks as Digital Twins. However, their Analytic engine and Communications Manager do not give them the capacity of choosing which agents to communicate with, as their communications are managed by the Social Platform.

Failure The failure of a Mediator Agent implies (1) that its communication with other agents is severed, (2) that the system stops providing maintenance recommendations for the physical assets assigned to the Mediator Agent and (3) that the Mediator Agent stops using any computing power.

Social platform

The Social Platform is the agent serving as a central node in the Centralized, Hierarchical and Heterarchical architectures. The main task of the Social Platform is to run algorithms leveraging information originating from the whole fleet. These algorithms can be aimed at (1) forming clusters of collaborating assets, (2) retrieving and plotting enterprise-level information, or (3) calculating prognostics and making maintenance decisions. Note that each of these tasks are optional and depend on the architecture in which the Social Platform is embedded (see Table 2).

The Social Platform uses data received from agents in lower layers of the hierarchy in order to form clusters of collaborating assets. In the case of a Hierarchical architecture, the Social Platform acts also as a communication channel between the lower agents of the architecture.

The Platform is formed by three building blocks: a Data Repository, containing clustering information, and the results of the algorithms run in the platform, an Analytics Engine where algorithms are computed, and a Communication Manager, that controls communication with lower-level agents.

Failure The failure of the Social Platform implies the severance of all communications and all computations managed by it. Additionally, in the Centralized architecture, failure of the Social Platform implies the halt of all maintenance recommendations.

Multi-agent system architectures

In this section, we describe the architectures analyzed in this paper. These architectures have been chosen because of their prominence in industrial systems. In here, we describe them within the context of collaborative prognostics, more general descriptions can be found in Brennan et al. (2002), Mařík and Lažanskỳ (2007), Monostori et al. (2006), Andreadis et al. (2014) and Leitão and Karnouskos (2015). Table 2 summarizes the role that each of the components presented the previous section play in each architecture.

Centralized

The Centralized architecture is the simplest case considered in this paper. It consists of a Social Platform with full control over the decision-making of the system, and a set of Virtual Assets that limit themselves to sending data to the Social Platform. The Social Platform computes the clusters of similar assets, and then uses the data from the assets belonging to these clusters to generate maintenance recommendations (see Fig. 1).

A Centralized architecture can technically be argued to not be a multi-agent architecture, as the only agent that really takes decisions and outputs predictions is the Social Platform. Nevertheless, we decide to test it against other architectures because of its importance and widespread use in industrial applications.

Hierarchical

A Hierarchical architecture is defined as an architecture in which intermediate agents provide most of the decision-making in the system, while lower-level agents are left to perform simpler tasks. In our case, these intermediate agents are Mediator Agents. Mediator agents are assigned groups

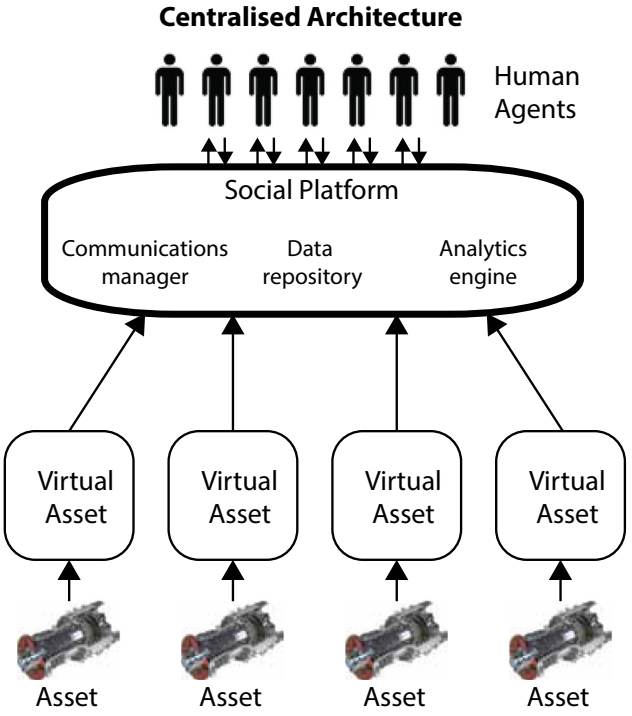

Fig. 1 Block diagram of the Centralized architecture. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The thicker block, pertaining to the Social Platform, indicates the element of the architecture performing prognostics

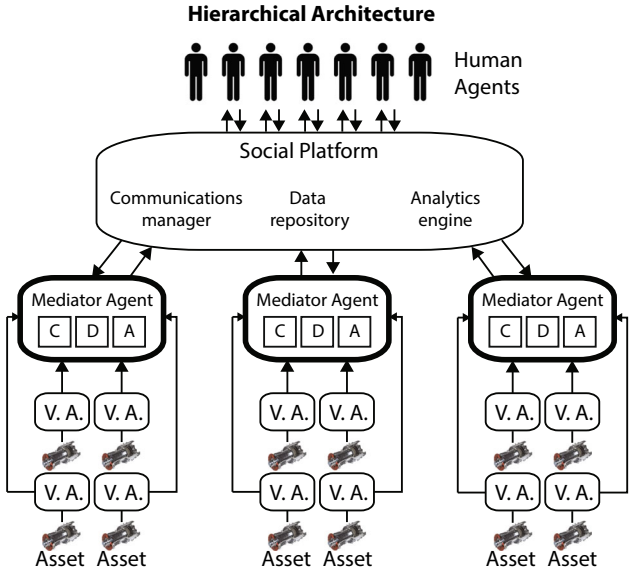

Fig. 2 Block diagram of the Hierarchical architecture. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The thicker blocks, pertaining to the Mediator Agents, indicate the element of the architecture performing prognostics for the assets in the industrial fleet. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine of the Mediator Agent

of Virtual Assets for which they perform prognostics, and schedule maintenance actions (see Fig. 2).

The Social platform is hierarchically superior to the Mediator Agents and in fact assigns them to groups of similar assets. The Social platform can also create or delete Mediator Agents (since the number and membership of clusters may vary over time), and has full control of the communications of the system.

Heterarchical

A Heterarchical architecture differs from the Hierarchical case presented in last paragraph in that it allows for peer-to-peer communication between the Digital Twins. Concretely, in our implementation of this architecture, Digital Twins perform prognostics, take maintenance decisions, and communicate with each other.

The Social Platform, at a higher level in the architecture, decides which Digital Twins will communicate with each other through its clustering algorithm, and serves as a communication link with human operators (see Fig. 3).

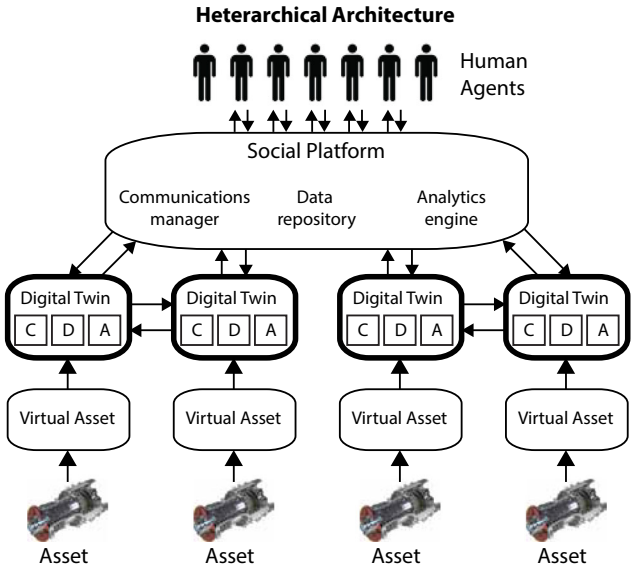

Fig. 3 Block diagram of the Heterarchical architecture. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The thicker blocks, pertaining to the Digital Twins, indicate the element of the architecture performing prognostics for the assets in the industrial fleet. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine of the Digital twin

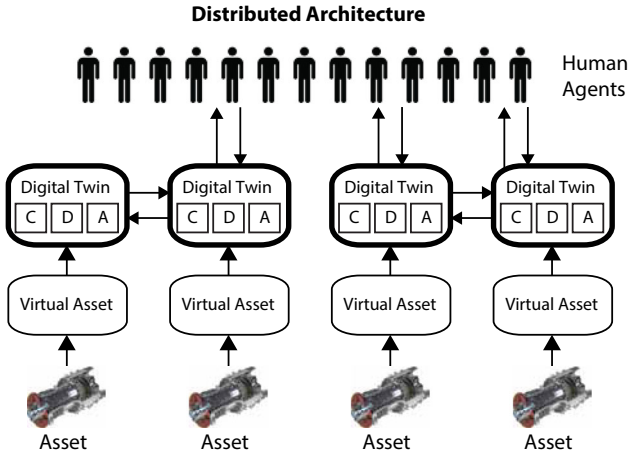

Fig. 4 Block diagram of the Distributed architecture. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The thicker blocks, pertaining to the Digital Twins, indicate the element of the architecture performing prognostics for the assets in the industrial fleet. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine of the Digital twin

Distributed

A Distributed architecture is one in which all its agents are in the same level of the hierarchy, and have the ability to take independent decisions without the supervision of a higher-level agent (see Fig. 4). In this architecture, communication consists of peer-to-peer connections between Digital Twins (i.e. the twins are all connected to one another without any central agent or mediators present).

As in the architectures described previously, similar assets are clustered together for collaborative prognostics, and the Digital Twins within the same cluster collaborate with one another. There is, however, an important difference: the clustering algorithm implemented here has to be a distributed clustering algorithm, unlike the previous architectures, where the Social Platform performs this task. The distributed k-mean clustering algorithm implemented here is similar to the one presented in Qin et al. (2017), and is detailed in "Distributed clustering algorithm" section.

Implementation in Netlogo

The architectures explained above were analysed in terms of their cost components. For this analysis to be done, a set of experiments were performed on Netlogo, with a Python extension. Netlogo is a MAS simulator, which has been used to simulate emergent behaviour of complex systems ranging from herd of sheep, to human behaviour during an emergency (Tisue and Wilensky 2004). Netlogo allows its agents to run Python scripts in the backend, through its official Python extension.[2]

All the architecture types described above were simulated using the same strategy: Netlogo simulated the behaviour of the agents (i.e. initiating the fleet of assets, connecting similar agents together, computing agent failures, etc), and prognostics/clustering algorithms were implemented using Python scripts.

The same approach used in Palau et al. (2019b) for prognostics is followed in our simulations here. However, instead of Matlab's `lsqnonlin` used in Palau et al. (2019b), the least squares fit from Python's Scikit learn (Pedregosa et al. 2011) library was used to fit the a_i, b_i, and t_{fi} values in the Eq. (1). To ensure that the system scaled well with the number of assets, this fit was limited to the last 400 data-points available to the agent.

Agents decided on which other agents to collaborate with by checking whether their corresponding assets belonged to the same asset cluster (information conveyed by the Social Platform). The clusters of assets were formed using the k-means clustering Algorithm (Hartigan and Wong 1979) implemented on Python via the Scikit learn library. This algorithm had as an input the parameters obtained from the non-linear fit: a_i, b_i, and t_{fi}. This excludes the case of the Distributed architecture, in which the distributed clustering algorithm described in the next section was used. To measure the computation cost, we measure the processor's time using python's 'Time' module while the program is run. Thus, N_{pro} is simply the total processing time used by the Python scripts of each architecture.

Distributed clustering algorithm

The clustering algorithm implemented for the distributed architecture differs from the centralized clustering algorithm used for the rest of architectures. The algorithm used here distributes the computation steps across the nodes, as there is no central agent left to compute clustering.

The goal of this algorithm is to form 'k' clusters of assets. 'k' here equals the number of different types of assets in the fleet. To initialise the clustering centroids, first, a random agent in the fleet is chosen as the first centroid. This agent records the distances of the remaining agents from their corresponding closest centroid. 'closeness' of the agent from the centroid is calculated using the history of the past health indices, and the maximum time before failure recorded. The farthest agent is then assigned as a new centroid. The process of generating new centroids continues until we have a total 'k' centroids, each representing its own cluster of assets. This way, centroids are initiated as far away from each other as

[2] https://github.com/NetLogo/Python-Extension.

possible, which is also the rationale behind the distributed k++ means algorithm (Qin et al. 2017).

Once all centroids have been assigned, each agent computes the distance to the centroids identified above, and assigns itself to the cluster corresponding to the closest centroid. As the simulation time progresses, the availability of Health Indicator data increases. Since in our simulations we use a normally distributed noise term [see Eq. (1)], the average difference of the health indicator per time step for similar assets approaches zero. The similarity of the assets therefore becomes more and more apparent with every passing time step, and the clusters eventually converge. This is the only step different from the distributed k-means clustering presented in Qin et al. (2017), where the authors rely on average-consensus to update the clusters. We instead update the clusters based on increasing data availability with the time steps. Algorithm 1 describes our distributed algorithm in pseudo-code (run every time step).

```
Select one random agent from the fleet;
while number of centroids <k do
    for the agent selected above do
        Calculate the distances between the agents and the
        centroids;
        Record the distances of the agents from their closest
        centroid;
        Append the farthest agent to the list of centroids;
    end
end
These centroids represent the clusters;
for every other agent in the system do
    Calculate the distances from each centroid;
    Assign self to the cluster represented by the closest centroid;
end
```

Algorithm 1: The distributed k-means clustering algorithm implemented in this paper

Experiments

Two sets of experiments were designed for this paper: one in which agent failure was not considered, and another one in which agents were made to fail at different layers of the architecture.

In the first set of experiments, a large fleet of assets is simulated to undergo deterioration such as determined by Eq. (1), and prognostics is performed such as described in "Implementation in netlogo". A separated experiment is performed for each type of architecture described in this paper, and prognostics, clustering and maintenance recommendations are executed such as described in the multi-agent system architectures section.

Table 3 Parameters used in the experiments

Var.	Definition	Value (s)
η	Prev. maintenance factor	0.7
N	Number of assets	500
k	Number of clusters	4
σ	Noise standard deviation	[0–0.5]

The second set of experiments is essentially a replica of the first set of experiments in which agents are allowed to randomly fail at all layers of the architecture with a probability of 1/50 each time step. Agent failure is defined in "Agent typologies and their failure modes" section, and typically implies a reduction on communication and processing cost and an increase of asset failures due to the halt of prognostic capabilities (and maintenance recommendations). In our experiments, the duration of an agent failure is randomly assigned between 1 and 50 time-steps.

In both cases, the experiments were set to be as extensive as possible within our computational constraints. Simulating several hundred assets with real-time prognostics and diagnostics capabilities is a computationally demanding task, and thus we restricted the number of simulated assets to 500. The experiments were run eight times and then averaged over to compensate for any effect that the variation of initial parameters could have on the results. Table 3 includes a detailed description of the parameters used whilst running the experiments.

In our experiments, costs are accounted as a series of additive contributions, adding up to a total normalised cost [see Eq. (5)]. Each of these contributions (N_{P}, N_{C}, N_{Co}, N_{pro}) are recorded independently. This allows us to explore all the parameter space of $(\alpha, \Gamma, C_p, C_c)$ with a single experiment per each architecture type, as the dynamics of the simulation are independent of the cost parameters. Therefore, experiments are run with $(\alpha, \Gamma, C_p, C_c) = (1, 1, 1, 1)$, and generalised by multiplying (N_{P}, N_{C}, N_{Co}, N_{pro}) by $(\alpha, 1, C_p, C_c)$ across the parameter range of interest (normalised to Γ).

The parameters determining each asset's Health Indicator in the population, (a_i, b_i, t_{fi}) are chosen such that four distinct classes of assets are present in the experiment. These four classes of assets are chosen randomly within the following pre-set limits $(a_i, b_i, t_{fi}) = ((0, 1), (0, 0.1), (0, 100))$. Assets in the fleet are then randomly assigned to belong to one of the four classes. The Health Indicator of these assets is generated during the experiments using Eq. (1). Experiments are run until $T = 400$ to make sure[3] that even in the case of very long failure times t_{fi}, the asset fleet is able to record multiple failures for each asset.

[3] Four-hundred time-steps.

Results and discussion

The number of times that each cost component contributes to the total cost of the experiment, determined by the quadruplet $(N_P, N_C, N_{Co}, N_{pro})$ is shown in Tables 4 and 5 (in Appendix). These tables translate into specific cost components once these components are weighted by cost weights $(\alpha, 1, C_p, C_c)$. If there is no mention of the contrary, in the results presented here we choose $\alpha = \frac{1}{100}$, which means that the predictive maintenance cost is one hundred times cheaper than the corrective maintenance cost. Additionally, $C_p = 20C_c$ is chosen, as this ensures a significant contribution of the processing costs (note that $N_{pro} \ll N_{Co}$). Due to the linearity of the cost equations, these assumptions have no effect on the generalisation of the observed trends.

The following phenomena are clearly observed in Fig. 5:

1. If agent failures are considered, Distributed and Heterarchical architectures are optimal for high-value assets (or low communication costs). Compare the dashed lines in Fig. 5. In the Centralised and Hierarchical case, the failure of a Mediator Agent or the Social Platform leads to a halt of predictive maintenance operations for hundreds of assets in the system, which then causes a dramatic increase of corrective maintenance actions (for example, for $\sigma = 0.1$ the difference between corrective maintenance actions in the Centralised and Heterarchical architectures nearly doubles (see Tables 4, 5 in Appendix). In the Distributed and Heterarchical cases, maintenance recommendations are produced by the Digital Twins, and no agent failure has the potential to compromise maintenance for hundreds of assets.
2. Higher costs for low-value assets across the board: the normalized cost per asset and time step[4] in architectures containing low-value assets is orders of magnitude higher than for the case of high-value assets (the cost for all architectures increases with $\frac{1}{C_A}$ in Fig. 5). This means that real-time MAS implementations for prognostics are more cost-effective the more expensive the replacement cost of the assets is, assuming that prognostics complexity remains constant.
3. When there are no agent failures, Centralized and Hierarchical architectures are generally cheaper this is expected, as these two architectures are also the ones featuring less communication and computation costs. Only for very noisy experiments, for very low communication costs or very high asset values this becomes false, when maintenance costs dominate over the rest of the costs of the system.

[4] Recall that this cost has been normalised to the corrective maintenance cost Γ, proportional to the asset value C_A.

Table 4 Table showing results for the case of no agent failure

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Cent.	[6045, 0, 200,000, 598]	[6266, 21, 200,000, 626]	[5969, 1056, 200,000, 632]	[4423, 4028, 199,996, 638]	[2672, 10,201, 199,986, 653]	[1792, 16,321, 199,970, 401]
Hier.	[6030, 0, 201,099, 651]	[6505, 32, 201,099, 660]	[5853, 971, 201,099, 704]	[4628, 3698, 201,099, 696]	[2923, 9144, 201,099, 715]	[1714, 16,477, 201,099, 483]
Dist.	[6000, 0, 12,618,174, 349]	[6249, 27, 13,498,609, 360]	[5363, 1305, 16,589,558, 343]	[3497, 4988, 17,319,831, 335]	[2233, 10,595, 17,421,179, 335]	[1379, 16,462, 17,959,496, 317]
Hete.	[5962, 0, 12,687,189, 4111]	[6232, 31, 12,754,968, 5110]	[5428, 1320, 12,703,401, 5980]	[4041, 4215, 13,107,021, 6471]	[2358, 9746, 12,952,249, 6812]	[1414, 15,560, 13,260,666, 5414]

The lists present in every table position correspond to the quadruplets $[N_P, N_C, N_{Co}, N_{pro}]$. These values have been rounded to the closest integer from the average of eight experiments

Table 5 Table showing results for agent failure

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Cent.	[3399, 1657, 92,718, 424]	[3143, 2070, 86,102, 400]	[2585, 3013, 88,106, 408]	[1750, 5730, 86,550, 421]	[1167, 10,172, 89,391, 432]	[769, 16,204, 84,349, 350]
Hier.	[3486, 1674, 77,877, 330]	[3625, 1721, 81,468, 371]	[2175, 2861, 60,803, 304]	[1619, 5045, 68,676, 321]	[1053, 11,082, 74,663, 379]	[711, 16,886, 74,001, 276]
Dist.	[4537, 918, 7,366,053, 228]	[4484, 1108, 6,989,799, 223]	[3526, 2405, 6,646,838, 220]	[2342, 5282, 7,419,548, 215]	[1488, 10,189, 8,322,924, 215]	[919, 17,021, 8,293,278, 167]
Hete.	[4504, 924, 5,988,868, 2587]	[4482, 1080, 5,786,474, 3028]	[3738, 2179, 5,918,000, 3501]	[2334, 5431, 5,797,278, 4099]	[1452, 10,220, 5,851,001, 4300]	[885, 16,856, 5,984,762, 3553]

The lists present in every table position correspond to the quadruplets [N_{P}, N_{C}, N_{Co}, N_{pro}]. These values have been rounded to the closest integer from the average of eight experiments

4. When there are no agent failures, cost differences between architectures minimize as asset value increases: note the convergence of solid lines for low values of $\frac{1}{C_{\mathrm{A}}}$ in Fig. 5. This is due to the fact that in high-value assets maintenance costs dominate over communication and computing costs. If no agent failures are included in the experiments, predictive maintenance in the different architectures has a very similar level of accuracy, and the overall cost is essentially the same.
5. A high communication cost limit exists: if communication costs are high enough (or asset value low enough), agent failures (dashed lines) actually mean lower operational costs. The explanation for this is simple: agent failure increases operational cost through more unwanted corrective maintenance actions, but decreases it by halting computation and communication actions. If the communication costs are high enough, agent failure then leads to a less costly architecture.

Another interesting factor to study from the experimental results is the dependence with the amount of noise present in the Health Indicator, σ. Figure 5 shows that the difference between architectures in the case of no agent failures reduces as the communication cost decreases (and costs are dominated by the maintenance component). To check if cost difference also decreases the more noisy the system is, we measure the normalised index of dispersion across different values of σ,

$$D_{\sigma}^{\mathrm{norm}} = \frac{1}{\max_{\sigma}\left(D_{\sigma}\right)} \frac{\mathrm{Var}\left(K_{\mathrm{cent}}^{\sigma}, K_{\mathrm{hier}}^{\sigma}, K_{\mathrm{dist}}^{\sigma}, K_{\mathrm{hete}}^{\sigma}\right)}{\mathrm{Mean}\left(K_{\mathrm{cent}}^{\sigma}, K_{\mathrm{hier}}^{\sigma}, K_{\mathrm{dist}}^{\sigma}, K_{\mathrm{hete}}^{\sigma}\right)}. \quad (6)$$

The reason why we normalise D_{σ} to its maximum is to be able to show the dependency with σ across different values of C_{c} in the same figure (we already know from Fig. 5 that the absolute value of dispersion decreases with C_{c}).

From Fig. 6, one observes that cost differences between architectures decrease as noise increases: this is a direct effect of an increase of un-predicted failures as the data becomes more noisy, which makes maintenance costs dominate (see Tables 4, 5). This tendency reverts only for unrealistically high communication costs, or very low asset values (note that a communication cost of 0.2 corresponds to 20% of the cost of replacing the asset, and that these cost are for a fix number of bytes sent through the network). This reversion is given by the difference in clustering between the distributed architecture and the rest of architectures: the more noise there is in the system, the more different cluster results are produced (see Tables 6, 7). Normally, this does not affect the index of dispersion because maintenance costs dominate at large values of σ, but for high enough communication costs, this effect is observed.

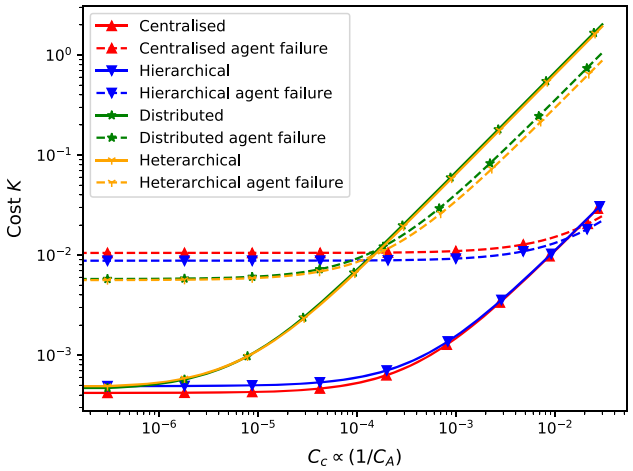

Fig. 5 Normalised cost K for each of the studied architectures, for the case of agent failure (dashed) and no agent failures (solid lines) with respect to the normalised communication cost C_c. The horizontal axis can be interpreted both as the increase of the communication cost given a constant asset value, or as the decrease of asset value given a constant communication cost ($1/C_A$, high-value assets to the left of the chart). The data here is plotted for $\sigma = 0.1$, $\alpha = 1/100$, and $C_p = 20C_c$

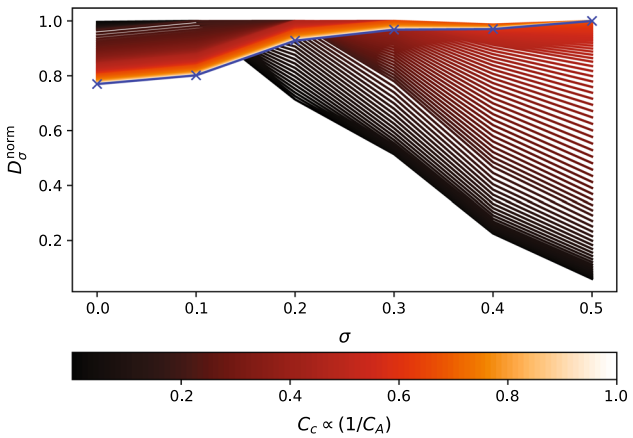

Fig. 6 Normalised index of dispersion D_σ^{norm} across the studied architectures, for the case of no agent failures with respect to the amount of noise in the system σ. Lines are colored depending on the normalised communication cost C_c. The blue line with markers represents the high communication cost/low asset value limit. This can be interpreted both as the increase of the communication cost given a constant asset value, or as the decrease of asset value given a constant communication cost ($1/C_A$). The data here is plotted for $\sigma = 0.1$, $\alpha = 1/100$, and $C_p = 20C_c$ (Color figure online)

A methodology for architecture evaluation

The approach followed in this paper can be used as a foundational methodology to assess the optimality of a given multi-agent architecture in a real industrial application of collaborative prognostics. In this case, an asset manager should take the following steps:

1. Determine the predictive maintenance γ, and corrective maintenance Γ costs of the fleet's assets.
2. Determine the approximate cost of processing and sending through a given unit of data (for example a byte), and encode it in the multi-agent simulation of the system through C_c and C_p.
3. Estimate the accuracy of real-time prognostics, and encode it in Eq. (1) through its stochastic term.
4. Determine the number of assets N present in the asset fleet.
5. Choose a maintenance policy, and encode it in the agent's decision-making process.
6. Determine the probability of agent failure, and the maximum time of agent downtime, encode it in the simulation as described in this paper.
7. Test the different architectures described here with the real cost parameters of the assets, and compare the total cost incurred by them.
8. Choose the best suitable architecture from the simulation outputs.

Conclusion and future work

This paper is a study of the cost consequences of implementing different multi-agent system architectures for collaborative prognostics, a new prognostics approach based on collaboration between agents that represent different assets in the fleet. In this paper, four architectures are analysed, featuring different levels of distribution: Centralized, Hierarchical, Heterarchical, and Distributed.

The main conclusion drawn from this study is that decentralized architectures are not always cost-efficient for the purpose of collaborative prognostics. If the assets in the system have a low value, communication and computing costs become relevant, and more centralized architectures become the best option. However, when the value of the assets is high enough, the implementation of distributed architectures can be justified. In this case, the value of the assets is much larger than communication and computing costs, and the benefits of distributed architectures can be leveraged.

This difference between architectures becomes especially relevant when agent failure is included in the experiments. In this case, architectures where prognostics and maintenance planning is highly dependent of few agents are especially susceptible to agent failure. This, in practice, means that when agent failures are considered, distributed architectures become more competitive.

A secondary conclusion is that multi-agent based collaborative prognostics architectures are more cost efficient in general the more expensive the assets of the system are. This is a common-sense result: there is no point in enhancing very low value assets with IoT technologies, as the cost of

Table 6 Table showing clustering purity results at t = 400 for the case of no agent failure

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Centralised	1.0	0.996	0.972	0.954	0.961	0.922
Hierarchical	1.0	0.995	0.982	0.946	0.909	0.946
Distributed	1.0	0.949	0.842	0.749	0.719	0.734
Heterarchical	1.0	1.000	0.982	0.917	0.948	0.928

These values have been averaged over eight experiments

Table 7 Table showing clustering purity results at t = 400 for the case of agent failure

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Centralised	0.995	0.998	0.999	0.994	0.974	0.934
Hierarchical	0.784	0.801	0.689	0.666	0.748	0.782
Distributed	0.660	0.679	0.655	0.679	0.615	0.608
Heterarchical	0.763	0.769	0.775	0.734	0.740	0.758

These values have been averaged over eight experiments

these technologies outweigh by far the savings of a predictive maintenance policy.

With regards to future work, there are some parametric dependencies that have not been explicitly studied in this paper. Perhaps the most important is the number of clusters k (groups of different assets) in the fleet, which in our experiments has been limited to four. In the Distributed and Hierarchical architectures, communication costs will be proportional to the square of the size of each cluster: $C_c \propto \left(\frac{N}{k}\right)^2$ (assuming that each cluster has a similar size). This means that if k is kept constant but N is increased, the cost of Distributed and Hierarchical architectures will increase at a higher rate than the cost of their Centralised and Heterarchical counterparts. Studying the optimality of different architectures with respect to the heterogeneity of the fleet (given by k), would thus give place to a potentially interesting research study.

Another parametric dependency that has been omitted (kept constant) in the experiments is the dependence of the cost of the architectures with the probability of agent failure, and its duration. In our experiments, the first parameter is kept constant and the second is sampled from a pre-determined probability distribution. We purposely chose both parameters to be relatively high, to compensate for the fact that we assumed that there were no costs associated to repairing agent failure. Further research should focus on exploring this dependency, and placing it within realistic industrial parameters.

Finally, in the experiments presented here, the maintenance threshold η is kept constant. Although this is a reasonable assumption for the purpose of this paper, comparing across architectures, studying the effect of optimizing η in real-time would yield a potentially interesting research study.

Acknowledgements The project that has generated these results has been supported by a "la Caixa" Fellowship (ID 100010434), with code LCF/BQ/EU17/11590049. This research was partly supported by Siemens Industrial Turbomachinery UK. This research was also partly supported by the Next Generation Converged Digital Infrastructure project (EP/R004935/1) funded by the Engineering and Physical Sciences Research Council and BT. The server used to perform the experiments in this paper was funded by the Centre for Digital Built Britain.

Appendix

Pseudocode description of the agents

In this section, we include in pseudo-code the tasks performed by each of the agents in the architecture. In the experiments presented in this paper, these tasks were programmed and performed using the multi-agent simulation software Netlogo, with a python extension. The code shown is performed in parallel at each time step of the simulation by each agent. In our simulations, the Virtual Assets and the Digital Twins share a single agent when present at the same time. The particularities of the transfer of data between Digital Twins varies depending on the architecture used. For example, in the distributed architecture, Digital Twins receive data directly from other Digital Twins, and in Heterarchical architectures, they do so from the Social Platform.

Extended experimental results

In this section, we include the tables including all quadruplets (N_P, N_C, N_{Co}, N_{pro}) used to obtain the results presented in

Virtual Asset

```
if HI_i ≥ 0 then
    Set HI_i = HI_i(t_li);
    Set t_li = t_li + 1;
    if agent-fault is False then
        Update agent connections;
        Send HI_i to a higher-level agent;
    end
end
if HI_i < 0 then
    Set fault True;
    Set HI_i = 0;
    Set t_li = 0;
end
```

Algorithm 2: Pseudocode of the Virtual Asset

Digital Twin

```
Receive HI_i from the Virtual Asset;
Receive data from other Digital Twins;
Fit data using python's least_squares algorithm;
if distributed is True then
    execute algorithm 1;
end
if fault is False then
    Set t^e_fi from fit; if t_li > η t^e_fi then
        Preventively maintain;
    end
end
if fault is True then
    Correctively maintain;
end
Send data to other Digital Twins;
Calculate computation time;
```

Algorithm 3: Pseudocode of the Digital Twin

Mediator Agent

```
Receive HI_i from the Virtual Assets;
Fit data using python's least_squares algorithm;
for Assets connected to the agent do
    if fault is False then
        Set t^e_fi from fit; if t_li > η t^e_fi then
            Preventively maintain;
        end
    end
    if fault is True then
        Correctively maintain;
    end
end
Calculate computation time;
```

Algorithm 4: Pseudocode of the Mediator Agent

"Results and discussion" section. We also include a table showing the purity of the clustering algorithms for each architecture and standard deviation [see Manning et al. (2008) for a description of purity].

Social Platform

```
Receive data from the Digital Twins or Mediator Agents
(depending on architecture);
if centralised is False then
    compute k-means clustering;
    send data to the pertinent clusters;
end
if centralised is True then
    for Assets assigned to each cluster do
        Fit data using python's least_squares algorithm;
        if fault is False then
            Set t^e_fi from fit; if t_li > η t^e_fi then
                Preventively maintain;
            end
        end
        if fault is True then
            Correctively maintain;
        end
    end
end
Compute purity and cost metrics;
Calculate computation time;
```

Algorithm 5: Pseudocode of the Social Platform

References

Andreadis, G., Klazoglou, P., Niotaki, K., & Bouzakis, K. D. (2014). Classification and review of multi-agents systems in the manufacturing section. *Procedia Engineering*, *69*, 282–290.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805.

Bakliwal, K., Dhada, M. H., Palau, A. S., Parlikad, A. K., & Lad, B. K. (2018). A multi agent system architecture to implement collaborative learning for social industrial assets. *IFAC-Papers OnLine*, *51*(11), 1237–1242.

Brennan, R. W., Fletcher, M., & Norrie, D. H. (2002). An agent-based approach to reconfiguration of real-time distributed control systems. *IEEE Transactions on Robotics and Automation*, *18*(4), 444–451.

Buchanan, B. G. (1986). Expert systems: Working systems and the research literature. *Expert Systems*, *3*(1), 32–50.

Djurdjanovic, D., Lee, J., & Ni, J. (2003). Watchdog agent—An infotronics-based prognostics approach for product performance degradation assessment and prediction. *Advanced Engineering Informatics*, *17*(3–4), 109–125.

Duffie, N. A., & Piper, R. S. (1986). Nonhierarchical control of manufacturing systems. *Journal of Manufacturing Systems*, *5*(2), 141.

Fasanotti, L. (2014). A distributed intelligent maintenance system based on artificial immune approach and multi-agent systems. In *2014 12th IEEE international conference on industrial informatics (INDIN)* (pp. 783–786). IEEE.

Fasanotti, L. (2018). An artificial immune intelligent maintenance system for distributed industrial environments. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *232*(4), 401–414.

Ferber, J., & Weiss, G. (1999). *Multi-agent systems: An introduction to distributed artificial intelligence* (Vol. 1). Reading: Addison-Wesley.

Ghita et al. (2018). Scheduling of production and maintenance activities using multi-agent systems. In *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)* (pp. 508–515).

Gilchrist, A. (2016). *Industry 4.0: The industrial Internet of Things*. New York: Apress.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society Series C (Applied Statistics)*, *28*(1), 100–108.

Hernández, J. E., Lyons, A. C., Mula, J., Poler, R., & Ismail, H. (2014). Supporting the collaborative decision-making process in an automotive supply chain with a multi-agent system. *Production Planning & Control*, *25*(8), 662–678.

Jardine, A. K., & Tsang, A. H. (2005). *Maintenance, replacement, and reliability: Theory and applications*. Boca Raton: CRC Press.

Khan, S., & Yairi, T. (2018). A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, *107*, 241–265.

Konecnỳ, J., McMahan, H.B., Ramage, D., & Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. arXiv:1610.02527.

Lee, J., Wu, F., Zhao, W., Ghaffari, M., Liao, L., & Siegel, D. (2014). Prognostics and health management design for rotary machinery systems—Reviews, methodology and applications. *Mechanical Systems and Signal Processing*, *42*(1–2), 314–334.

Leitão, P. (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, *22*(7), 979–991.

Leitão, P., & Karnouskos, S. (2015). *Industrial agents: Emerging applications of software agents in industry*. Burlington: Morgan Kaufmann.

Li, H., & Parlikad, A. K. (2017). Study of dynamic workload assignment strategies on production performance. *IFAC-Papers OnLine*, *50*(1), 13710–13715.

Li, H., Palau, A. S., & Parlikad, A. K. (2018). A social network of collaborating industrial assets. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *232*(4), 389–400.

Liu, L., Logan, K. P., Cartes, D. A., & Srivastava, S. K. (2007). Fault detection, diagnostics, and prognostics: Software agent solutions. *IEEE Transactions on Vehicular Technology*, *56*(4), 1613–1622.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. New York, NY: Cambridge University Press.

Mařík, V., & Lažanskỳ, J. (2007). Industrial applications of agent technologies. *Control Engineering Practice*, *15*(11), 1364–1380.

McFarlane, D. (2018). *Industrial internet of things: Applying IOT in the industrial context*.

Monostori, L., Váncza, J., & Kumara, S. R. (2006). Agent-based systems for manufacturing. *CIRP Annals-Manufacturing Technology*, *55*(2), 697–720.

Ning. (2016). A cloud based framework of prognostics and health management for manufacturing industry. In *2016 IEEE international conference on prognostics and health management (ICPHM)* (pp. 1–5). IEEE.

Nwana, H. S. (1996). Software agents: An overview. *The knowledge engineering review*, *11*(3), 205–244.

Palau, A. S., Bakliwal, K., Dhada, M. H., Pearce, T., & Parlikad, A. K. (2018). Recurrent neural networks for real-time distributed collaborative prognostics. In *2018 IEEE international conference on prognostics and health management (ICPHM)* (pp 1–8). IEEE.

Palau, A. S., Dhada, M., Bakliwal, K., Kumar Parlikad, A. (2019a). An Industrial Multi Agent System for real-time distributed collaborative prognostics. *Engineering Applications of Artificial Intelligence* (Under review).

Palau, A. S., Liang, Z., Lütgehetmann, D., & Parlikad, A. K. (2019). Collaborative prognostics in social asset networks. *Future Generation Computer Systems*, *92*, 987–995.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Qin, J., Fu, W., Gao, H., & Zheng, W. X. (2017). Distributed *k*-means algorithm and fuzzy *c*-means algorithm for sensor networks based on multiagent consensus theory. *IEEE Transactions on Cybernetics*, *47*(3), 772–783.

Sallez, Y., Berger, T., Raileanu, S., Chaabane, S., & Trentesaux, D. (2010). Semi-heterarchical control of FMS: From theory to application. *Engineering Applications of Artificial Intelligence*, *23*(8), 1314–1326.

Shen, W., Hao, Q., Yoon, H. J., & Norrie, D. H. (2006). Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, *20*(4), 415–431.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning* (pp. 330–337).

Tang, L., Kacprzynski, G. J., Bock, J. R., & Begin, M. (2006). An intelligent agent-based self-evolving maintenance and operations reasoning system. In *2006 IEEE aerospace conference* (pp. 12–pp). IEEE.

Tisue, S., & Wilensky, U. (2004). Netlogo: Design and implementation of a multi-agent modeling environment. *Proceedings of Agent*, *2004*, 7–9.

Trentesaux, D. (2009). Distributed control of production systems. *Engineering Applications of Artificial Intelligence*, *22*(7), 971–978.

Upasani, K., Bakshi, M., Pandhare, V., & Lad, B. K. (2017). Distributed maintenance planning in manufacturing industries. *Computers & Industrial Engineering*, *108*, 1–14.

Vrba, P. (2013). Review of industrial applications of multi-agent technologies. In *Service Orientation in holonic and multi agent manufacturing and robotics* (pp. 327–338). Berlin: Springer.

Wang, S., Wan, J., Zhang, D., Li, D., & Zhang, C. (2016). Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, *101*, 158–168.

Wang, T., Yu, J., Siegel, D., & Lee, J. (2008). A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In *International conference on prognostics and health management, 2008. PHM 2008* (pp. 1–6). IEEE.

Weiss, G. (1999). *Multiagent systems: A modern approach to distributed artificial intelligence*. Cambridge: MIT Press.

Wong, T., Leung, C., Mak, K. L., & Fung, R. Y. (2006). Dynamic shopfloor scheduling in multi-agent manufacturing systems. *Expert Systems with Applications*, *31*(3), 486–494.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(2), 115–152.

Xiang, W., & Lee, H. P. (2008). Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Engineering Applications of Artificial Intelligence*, *21*(1), 73–85.

Yan, J., Koc, M., & Lee, J. (2004). A prognostic algorithm for machine performance assessment and its application. *Production Planning & Control*, *15*(8), 796–801.