An Industrial Multi Agent System for real-time distributed collaborative prognostics

Adrià Salvador Palau[a,*], Maharshi Harshadbhai Dhada[a], Kshitij Bakliwal[b], Ajith Kumar Parlikad[a]

[a]*Department of Engineering, Institute for Manufacturing, University of Cambridge, Cambridge, CB3 0FS, UK*
[b]*Indian Institute of Technology (IIT) Indore, Khandwa Road, Simrol, Indore 453552 India*

Abstract

Despite increasing interest, real-time prognostics (failure prediction) is still not widespread in industry due to the difficulties of existing systems to adapt to the dynamic and heterogeneous properties of real asset fleets. In order to address this, we present an Industrial Multi Agent System for real-time distributed collaborative prognostics. Our system fulfils all six core properties of Advanced Multi Agent Systems: Distribution, Flexibility, Adaptability, Scalability, Leanness, and Resilience. Experimental examples of each are provided for the case of prognostics using the C-MAPPS engine degradation data set, and data from a fleet of industrial gas turbines. Prognostics are performed using the Weibull Time To Event - Recurrent Neural Network algorithm. Collaboration is achieved by sharing information between agents in the system. We conclude that distributed collaborative prognostics is especially pertinent for systems with presence of sensor faults, limited computing capabilities or significant fleet heterogeneity.

Keywords: Multi Agent Systems; Distributed Systems; Recurrent Neural Networks; Prognostics; Networks; Asset Management.

1. Introduction

Manufacturing companies are increasingly focused on servitization: instead of selling assets, they sell the services that these assets provide [1]. In this new business model, clients may not own the assets any more, but instead pay for the right of using them. An example of this are bicycle sharing services, in which the bicycle is owned by the operator and clients pay for the right of using it for a period of time [2].

Servitisation makes companies responsible for the maintenance and upkeep of the assets, and hence have to bear the associated costs. This is providing

*Corresponding author
Email address: as2636@cam.ac.uk (Adrià Salvador Palau)

manufacturers the impetus to develop more reliable, smarter products with reduced whole-life costs [3]. At the same time, the steady drop in the price of Internet of Things (IoT) technologies has resulted in the widespread deployment of sensors in high-value assets, making real-time diagnostics (failure detection and classification) the state of the art in a wide range of industries [4]. These technological advances have resulted in a significant reduction of inspection costs and have helped furthering our understanding of the behaviour of asset fleets. Ultimately, this has decreased maintenance costs through the development of more sophisticated preventive maintenance strategies [5, 6].

The popularization of servitization, combined with the improvements in sensor-based diagnostics mentioned above, has resulted in a growing interest in real-time prognostics: prediction of failures. Albeit not yet at the adoption level of diagnostics, real-time prognostics applications exist for assets governed by well understood physical processes or assets with a very high intrinsic value such as wind turbines and semiconductor devices [5, 7, 8].

Achieving widespread deployment of real-time prognostics largely depends on the implementation of scalable, resilient and dynamic systems able to deal with the heterogeneity present in industrial asset fleets. For the case of industrial applications, Multi Agent Systems [9] and holonic systems [10] have been postulated to fulfil these properties. Multi Agent Systems are distributed systems of independent agents that cooperate or compete to achieve a certain objective [11]. In this paper, we pay special attention to Advanced Multi Agent Systems, and present a system of this kind especially designed for distributed real-time prognostics.

Described in 2015 by Rainer Unland, Advanced Multi Agent Systems can be seen as the archetype of Multi Agent Systems and are defined as Distributed, Flexible, Adaptable, Scalable and Lean [9]. In comparison, the state-of-the-art is that often prognostics implementations avoid considering a Multi Agent architecture, choosing instead to perform prognostics with a single piece of centralised software.

The development and deployment of distributed real-time asset management systems may have been hampered by the fact that most prognostics scenarios considered by researchers are small enough with regards to its data and asset population that a distributed approach can be deemed unnecessary. Despite this, heuristic-based Multi Agent Systems aimed to solve a variety of tasks within asset health management have been proposed and tested for different scenarios (see, for example [12, 13, 14]). For the specific case of distributed real-time prognostics, the examples that have been presented do not yet exhibit all the properties of Advanced Multi Agent Systems (see Appendix 6). Concepts such as industrial [15], social [16], and other intelligence-enabled assets have been proposed but seldom demonstrated besides for the case of some simple simulation scenarios [17].

In prognostics, the shift towards distributed systems has been driven by the observation by a few researchers that larger heterogeneous populations often require a subset of differently-trained models to conform to the population heterogeneity. A good example of this is the medical field, where the realisation

that different population segments reacted differently to the same treatments led to the development of personalized medicine. This has led to differences in sensory allocation [18], and in some cases even medication choice [19] for different population segments. In the same way, distribution allows to tailor prognostics models to each particular machine, instead of using a single model to manage the whole asset population.

This new approach towards handling heterogeneous asset fleets has been named independently by two groups of researchers as "collaborative prognostics". These are Shuai Hang's group from the university of Washington [18, 20, 21] and us [17]. Both focus on the aforementioned considerations of population heterogeneity but define collaboration differently. Hang et al. define collaboration as a different machine learning approach to develop prognostic models within a population with no consideration to multi agent deployment or real-time prognostics, while we see it as the result of providing assets with a certain degree of agency, similar to the concept of intelligent and cooperative agents described in literature [22, 23]. In order to avoid confusion, we refer to our approach as distributed collaborative prognostics to emphasize its multi agent nature. In this paper, collaboration is restricted to the concept of sharing data (experiences) between the agents of the system (as, for example, done in [24]). This is one of the simplest typologies of cooperation, but it suffices to exhibit all the desired properties of Multi-Agent Systems.

We present an Advanced Multi Agent System design and implementation for real-time distributed prognostics. This means a system that is Distributed, Flexible, Adaptable, Scalable and Lean. In order to achieve many of the characteristics of Advanced Multi Agent Systems we rely on the concept of collaborative prognostics: the ability of the agents to share information with each other to improve their prognostics capability. Architecturally, we build on our previous work [25, 16] and propose a modified hierarchical architecture. In the proposed implementation, each asset is associated with a Digital Twin, an agent that processes asset data and performs prognostics. This Digital Twin is also able to connect to the root node of the hierarchical architecture, an agent known as the Social Platform. This agent collects data from other Digital Twins and enables inter-asset communication. The prognostics algorithm implemented in the Digital Twins is based on Weibull Time To Event Recurrent Neural Networks (WTTE-RNN) [26]. The WTTE-RNN approach combines survival analysis theory with Recurrent Neural Networks in order to train a Weibull probability distribution of the remaining time to event. The algorithm is chosen for its simplicity and versatility, and because it is fit for the purpose of discrete event prediction.

This paper commences by describing the system proposed as an Advanced Multi Agent System (Section 2). The description of the system is divided in three parts: a brief discussion on Advanced Multi Agent Systems (Section 2.1), a description of the employed architecture (Section 2.2), and a section describing the methodology to implement collaborative prognostics (Section 2.3). After the system description, its implementation for the case of the C-MAPPS turbine degradation data set is presented in detail in Section 3. Firstly, the process of

data preparation is discussed (Section. 3.1), including a detailed description of the experiments run to benchmark this implementation against the properties of Advanced Multi Agent Systems (Section 3.1.3). This is followed by a description of the employed algorithms (Section 3.2). The section concludes with a presentation and discussion of the results obtained with the C-MAPPS data set (Section 3.3) where the system's conformance to the properties of Advanced Multi Agent Systems is proven (in Section 3.3.5, data from a real industrial scenario is also used). The paper concludes with a discussion and a summary of future work proposed by the authors (Section 4).

2. System Description

2.1. Advanced Multi Agent Systems

The ideal properties of industrial Multi Agent Systems have been postulated by several researchers [10, 9]. Although differences exist, there seems to be a certain degree of consensus, summarized by Unland's definition of *Advanced Multi Agent Systems* [9]. In order to avoid redundancy, we decide to closely follow his definition. With this, an *Advanced Multi Agent System* is one that fulfil the following characteristics (adapted from [9]):

1. Distributed: the system is designed to exhibit a decentralised structure. Decision making, control, and optimisation are all performed locally by the agents.
2. Flexibility: the system can adapt continuously to changes in the environment and the agents.
3. Adaptability/System learning: due to the learning properties of the agents, operation plans are continuously updated and refined according to environmental changes and human requirements.
4. Scalability: due to their decentralised nature, Advanced Multi Agent Systems are easily scalable. A new device can be added to the network without any need of a change in the central configuration of the system.
5. Leanness: ideally, Advanced Multi Agent Systems are formed by many equally designed agents. In practice, the number of sub-categories of agents must be limited as low as possible in order to keep the system scalable and lean. Differences between these categories of agents should be kept as small as possible in order to obtain a swarm-like [27] population.
6. Resilience: due to the distributed and lean nature of the system, an agent fault never provokes a whole-system fault.

2.2. Architecture

The architecture used in this paper is a modified hierarchical architecture with three layers: Virtual Assets, Digital Twins, and a Social Platform (see Fig. 1) [9, 25]. In the sections that follow, we describe how we have designed the Virtual Assets, Digital Twins and Social Platform in order to adapt them to perform distributed collaborative prognostics in an Advanced Multi Agent

System. The original architecture, designed to facilitate a more general case of distributed collaborative learning, is described in more detail in [25]. To describe our agent types, we have used H. S. Nwana's widely used classification [28].

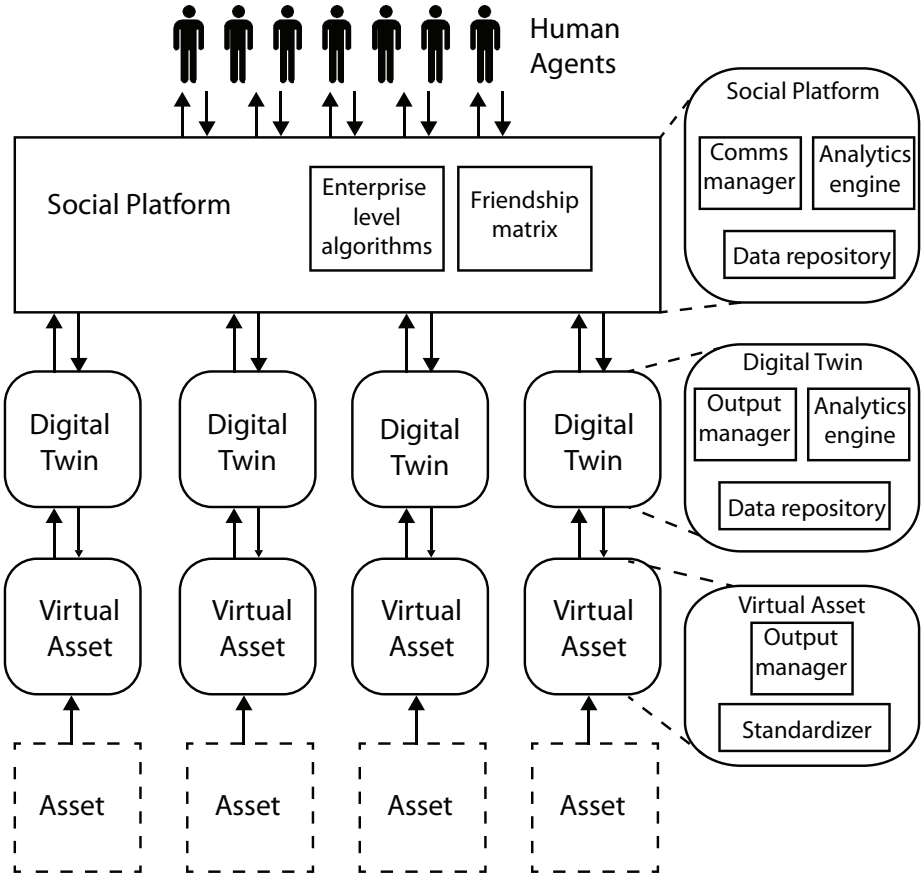

Figure 1: Block diagram of the architecture used in this paper. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The architecture is extracted from [25]. The thickness and size of the arrows indicate the size of the data exchange.

2.2.1. Virtual Asset

As the lowest-level blocks of the system's architecture, the Virtual Assets are designed to standardize the data coming from their corresponding assets. This enables the Digital Twins and the Social Platform to conform to a lean design, with nearly no heterogeneity across industrial scenarios, despite any variability in the physical assets themselves. Another function of these architectural elements can be to simulate assets operating in real time using an existing prognostics data set [25, 29]. In this case, the Virtual Assets load the data from

a data file, convert it into the system's standard format and send it at fixed time intervals to their assigned Digital Twins. Virtual Assets can be classified as Autonomous, Static (always assigned to the same Asset) and Reactive (see [28]). The data originating from the Virtual Assets is divided in three main components: a set of features (sensor time-series), a set of timed events (related to asset failures or warnings), and an asset identifier. This identifier must be designed so that it contains all the information related to the asset not directly included in the feature and event data (for example, machine make, country of deployment, etc). It is assumed that the format and content of these components will be standardised for widespread industrial implementation.

The Virtual Assets are formed by two building blocks: a Standardizer, dedicated to standardize the data coming from the Assets, and an Output manager, that controls the communications with their assigned Digital Twin (see Fig. 1).

2.2.2. Digital Twin

Each Virtual Asset is assigned a Digital Twin, an agent consisting of three building blocks: a Data Repository, an Output Manager, and an Analytics Engine. The Analytics Engine of the Digital Twin is responsible for performing prognostics, data preprocessing, and analysing the events received from the Virtual Asset to discern between failure events and other events unrelated to prognostics. The Output Manager is responsible for managing the communication between the Digital Twin and the other components of the system's architecture. Finally, the Data Repository stores and manages the data generated by the Analytics engine and the Output Manager. This data is divided in five main sets: the three sets of data generated by the Virtual Assets mentioned in the previous paragraph, a set of variables defined by the Social Platform, and a set of variables generated by the prognostics algorithm.

It is important to mention that collaborative prognostics will involve data sharing between assets. Therefore, the data used in the Analytics Engine of the Digital Twins consists also of data coming from other collaborating Digital Twins. Note that for real-world implementation, there is no presumption regarding the location of the data repository. This could be in a single central location or distributed. In fact there is no similar presumption regarding the physical location of the Digital Twin. The twin might reside within the asset, or all the Digital Twins of a fleet might be co-located.

Collaboration allows to classify Digital Twins as Smart Agents. Because each Digital twin is always connected to the same Virtual Asset, they can be also be technically classified as Static agents according to the classification presented in [28]. Their ability to react to their environment makes them also Reactive Agents (they do not conform to the deliberative thinking paradigm).

Only a subset of the data generated by the prognostics algorithm is kept permanently in the Twin's Data Repository: the prognostics model at each time step, and the model predictions. Some of the data stored in the Twin's repository originates in the Social Platform: the events for which predictions must be computed, the similarity distances between the twin's asset and each other asset in his groups of collaborating assets, and the features to be used in

the prognostics algorithm. Some of these variables, for example the events that the prognostics algorithm will predict, are human defined requirements.

2.2.3. Social Platform

As the node of the system's modified hierarchical architecture, the Social Platform is responsible for enabling and regulating communication between Digital Twins, and to pass down human requirements to the rest of the agents. Additionally, the Social Platform runs enterprise-level algorithms. These algorithms are aimed at (1) forming groups of collaborating assets, and (2) retrieving and plotting enterprise-level information. These properties make it a Static, Reactive, and Interface agent according to the classification published in [28].

The Social Platform uses features, event information and asset identifiers from the Digital Twins in order to form groups of collaborating assets. To form these groups, the Social Platform runs (in parallel, and in real-time) a clustering algorithm. The identities of the assets corresponding to each group are saved in a matrix, featuring also distances between assets, that can be used to evaluate and weight inter-asset collaboration. This is known as the *Friendship Matrix*. These distances can be calculated as per definition of the asset manager, from simple euclidean distances including only continuous attributes, or from heterogeneous distances including also discrete asset attributes such as the asset identifier.

The Social Platform is formed by three building blocks: a Data Repository, containing the Friendship Matrix, and the results of the enterprise-level algorithms, a Communication Manager, that controls communication with the Digital Twins, and an Analytics Engine responsible for computing enterprise-level algorithms (see Fig. 1).

The Social Platform regulates communication as follows: the Digital Twins can send messages to the Platform at any time, these messages are taken in and, if they contain new information, are stored within the Social Platform's Data Repository. The Social Platform decides which Digital Twins share data with each-other at any time by reading from the Friendship Matrix. Upon this decision, it sends information to the Digital Twins in the fleet from other Digital Twins that belong to the same cluster (see Fig. 7, and Section 3.2.1).

2.3. Real-time collaborative prognostics

This section is dedicated to describe real-time collaborative prognostics and its implementation in an Advanced Multi Agent System. Distributed collaborative prognostics relies on clustering groups of similar assets in real time according to their similarities, and enabling data sharing among these asset groups in order to improve prediction accuracy. In the section that follows, we describe these algorithms and in the subsequent section we describe how collaboration is enabled within the framework of the proposed multi agent system.

2.3.1. Prognostics

In order to perform real-time prognostics, we choose to implement a machine learning approach known as Weibull Time To Event - Recurrent Neural Net-

works (WTTE-RNN) proposed by Egil Martinsson [26]. This approach has the benefit that it is designed to solve multivariate time to event prediction problems using both censored and uncensored data. For the sake of completeness we provide a brief description here. The reader is referred to [26] for full details.

Recurrent Neural Networks are chosen for prognostics because they are designed to handle patterns containing all the characteristics typically encountered in industrial failure data: non-linearity, noise, and time-dependency. In theory, RNN's are Turing complete and thus can learn complex temporal patterns [30]. The caveats of using Recurrent Neural Networks are that they require more computational resources than other regression methods, and that they have a tendency towards overfitting. In this paper, the first problem is addressed in Section 3.3.4, and over-fitting is controlled for through early-stopping and a network with few free parameters.

The WTTE-RNN approach combines techniques from survival analysis and Recurrent Neural Networks. In WTTE-RNN, a bespoke log-likelihood loss function is used to train a Recurrent Neural Network to provide the two parameters of a Weibull probability distribution of the Time To Event for a vector of multi-sensor feature data. The proposed log-likelihood function to be maximized by the Recurrent Neural Network is:

$$\log(\mathcal{L}) = \sum_{n=1}^{N}\sum_{t=0}^{T_n} u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right] + (1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right], \tag{1}$$

where u_t^n indicates whether the observation at time t is censored (the real failure time, $u_t^n = 0$ has not yet been observed). The first term in the right hand side of the equation is $u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right]$, which simply means: *in case that the real time to failure ($u_t^n = 1$, uncensored) has been observed, maximise the probability of the predicted time to failure Y_t^n being equal to the real time to failure y_t^n given the known values of the sensor value time series before time t, $x_{0:t}^n$.* The second term, $(1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right]$ means: *if the real time to failure ($u_t^n = 0$, censored) has not been observed, maximise instead the probability of the predicted time to failure Y_t^n being bigger than time left until the time at which we know that there has been no failure yet (y_t^n).* The summations $\sum_{n=1}^{N}\sum_{t=0}^{T_n}$ account for the summation over all the recorded failure trajectories (N) and over all the time-steps of each trajectory (T_n). A trajectory is defined as the set of sensor time-series contained in between each of the recurrent failures of an asset. In order to clarify what we mean with that, we have included a sketch of the matrix fed to the Recurrent Neural Network (see Fig. 2)

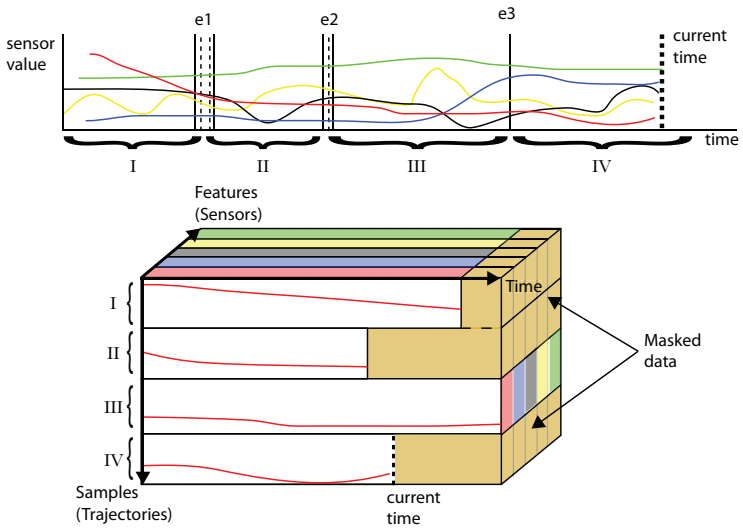

Figure 2: Sketch showing the training data matrix fed to the asset's Recurrent Neural Network for the case of *no collaboration*. For collaboration, additional trajectories from different assets in the fleet will be added to the training data matrix. Masked data refers to a fill-in number used to let the algorithm know that the values attached are not to be taken in account when training the Recurrent Neural Network.

The probabilities appearing in Eq. (1) can be obtained by means of survival analysis, in essence for the discrete case it can be shown that:

$$\log(\mathcal{L}) = u \log\left(e^{d(t)} - 1\right) - \Lambda(t+1). \tag{2}$$

Where $\Lambda(t)$ is known as the cumulative hazard function and $d(t) = \Lambda(t+1) - \Lambda(t)$ is the step cumulative hazard function. $\Lambda(t)$ is defined as the integral of the hazard function ($\lambda(t)$):

$$\Lambda(t) = \int_0^t \lambda(w)dw; \tag{3}$$

$$\lambda(t) = \lim_{\epsilon \to 0} \frac{Pr(t < T \leq t + \epsilon | T > t)}{\epsilon}.$$

Where T is a positive random variable. If one assumes that T follows a Weibull distribution[1], the cumulative hazard is:

$$\Lambda(t) = \left(\frac{t}{\alpha}\right)^{\beta}. \tag{4}$$

Where α is the scale parameter and β is the shape parameter. Combining Eqs. (2) and (4) the discrete log-likelihood (added over all trajectories and all

[1]With the following parametrization: $f(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} \exp\left[-\left(\frac{t}{\alpha}\right)^{\beta}\right]$.

time-steps, and using the concept of Recurrent Cumulative Hazard Function as shown in [26]) can be shown to be:

$$\log(\mathcal{L}_d) = \sum_{n=1}^{N}\sum_{t=0}^{T_n}\left(u_t^n \log\left\{\exp\left[\left(\frac{y_t^n+1}{\alpha_t^n}\right)^{\beta_t^n} - \left(\frac{y_t^n}{\alpha_t^n}\right)^{\beta_t^n}\right] - 1\right\} - \left(\frac{y_t^n+1}{\alpha_t^n}\right)^{\beta_t^n}\right). \tag{5}$$

Where α_t^n, β_t^n are the parameters of the Weibull distribution and y_t^n is the time to event or failure at each time-step t and trajectory n. Note that the left term will appear when there is no censoring, and for $\beta_t^n \to \infty$, the Weibull distribution corresponds to the Dirac delta function at $t = \alpha_t^n$. This is the expected behaviour as the ideal prediction is a probability distribution centred at the real time to event with zero variance.

The unconstrained optimization problem to be solved by the Recurrent Neural Network can be then summarized in finding the weights w that maximize $\log(\mathcal{L}_d)$. A comprehensive description can be found in the original source [26].

The dependency of the shape of a Weibull distribution (and more specifically its variance) with its defining parameters, α and β, make solving this optimization problem often numerically difficult. Eq. (5) features some opportunities for numerical instabilities: negative values of the logarithm's argument and exploding gradients being the most common. Reducing numerical instabilities was specially relevant in order to perform the experiments presented in this paper in which all agents operated using the same computer. Due to this, the computational resources assigned to each agent were limited and failure to converge to a stable solution compromised their operability. To solve this we clipped the norm of the gradients to a maximum value, and we followed the suggestions in [26].

1. Setting up a maximum for allowed β_t^n. If not capped by a superior limit, the optimization algorithm has a tendency to drive β_t^n to very large values, as this corresponds to a nearly perfect prediction. This tends to cause exploding gradients or over-fitting. An effect of bounding β_t^n is that close-to-event predictions tend to converge to low values of beta, as this is the most effective way that the optimization algorithm has to reduce the variance of the predicted distribution for low values of α_t^n.
2. Initialization of α_t^n, and β_t^n. The values at which the parameters of the Weibull distribution are initialized have a big influence on numerical stability and on convergence to a desirable solution. In this paper, we follow the suggestion by [26] and initialize our parameters as a *geometric initialization*. This corresponds to $\beta_t^n = 1$, and $\alpha_t^n = -\frac{1}{\log\left(1-\frac{1}{1+\bar{y}_i}\right)}$, where $\bar{y}_i$ is the mean time to failure at $t = 0$, $\bar{y}_i = \frac{1}{n}\sum y_0^n$.

To perform prognostics, each Digital Twin in the asset fleet implements a Recurrent Neural Network training algorithm in order to maximise Eq. (5), and predict further failures. While prognostics are performed in real time, the Recurrent Neural Network can be trained at larger time intervals T_{T} in order to reduce computational costs. This is known as the Digital Twin training time.

2.3.2. Collaboration

In this work, inter-agent collaboration is defined as the sharing of information between agents assigned to similar assets. This data is then used to train asset-specific prognostics algorithms. In order to allow collaboration, the assets in the fleet must be clustered according to a method that incorporates an inter-asset difference metric d_{ij}, where i and j are indexes of the assets i and j within the fleet. For non-trivial cases, a measure of clustering tendency should be calculated first in order to assess whether the asset population has any grouping structure [31]. Measures such as the multidimensional Cox-Lewis and Hopkins statistics [32, 33] are good examples of metrics allowing for efficient estimation of the clustering tendency. If this tendency is found to be strong, the number of clusters in the fleet should be determined by a combination of expert input and existing methods such as the Minimum Description Length, Aikaike Information Criterion, Bayes Information Criterion etc. Other methods such as the silhouette score [34] or Density-based spatial clustering of applications with noise (DBSCAN) can also be considered for the same purpose [35].

A crucial characteristic of distributed collaborative prognostics is that it is updated in real time, thus adapting to changes in the system's assets. In practice, this means that the number of clusters, and the assets belonging to each cluster are updated through the system's life-cycle. Once the clusters have been determined, each Digital Twin obtains feature and event data from other Digital Twins in the fleet through communication with the Social Platform. This data is then used in the training of the Recurrent Neural Network to improve its accuracy. The data can be incorporated in the neural network's training either weighting the trajectories according to the inter-asset difference metric d_{ij}, or directly without modification (see Fig. 2). In this paper, we implement the latter.

3. Implementation in the C-MAPPS data set

3.1. Data preparation

In order to implement the proposed approach, we use the Turbofan Engine Degradation Simulation Data Set [36] (from now on, referred as C-MAPPS data set, in reference to the software used to generate it). C-MAPPS is a MATLAB-based software designed to simulate the behaviour of a commercial turbofan engine. The advantage of C-MAPPS is that it allows for control of environmental, operational and failure parameters. This is done by varying parameters such as the altitude where the engine is operating (between 0 and 40000 ft), the Mach number (constricted to sub-sonic flight), and the temperature at sea level (comprehending all range of temperatures in which such an engine is expected to operate). The turbofan engines simulated by C-MAPPS are formed by several interdependent sub-systems, including regulators, limiters and control systems. These limiters resemble the mechanisms typically present in industrial machinery, that prevent machines from exceeding pre-set tolerances. In C-MAPPS, there are limiters for the core speed, the engine-pressure ratio, for the High

Pressure Turbine (HPT) exit temperature, and for the static temperature at the High-Pressure Compressor (HPC). A comprehensive description and a diagram of the turbines simulated by C-MAPPS can be found in [36]

The parameters of the simulation include a set of health-parameter inputs that are designed to simulate deterioration and fault. This can be done for any of the engine's rotating components. The C-MAPPS data set used in this paper includes degradation in the HPC and Fan modules. For each failure, the data set includes time-series variables that feature parameters such as sub-system temperatures, fan and core speeds, engine pressure ratio, bleed enthalpy, etc. A detailed list can be recovered in [36].

The framework proposed in this paper is specially relevant for machines that experience recursive failures. Instead, the C-MAPPS data set features trajectories to failure unique to every machine in the data set. In order to use this data for our approach, we must treat several trajectories to failure as if they correspond to the same asset. To do so, we cluster the turbines according to their operational setting and to their failure modes. This heterogeneity contained in the data set is especially conducive to test whether the proposed system fulfils the properties introduced in Section 2.1, because it can be used to simulate machines operating in varying environments and failure types. The C-MAPPS data set only includes a general description of the characteristics of each simulated failure trajectory, and therefore we had to carefully separate the many trajectories included in the data set. We describe how we did this in the following Sections 3.1.1 and 3.1.2.

3.1.1. Classification of failure type

The C-MAPPS data set used in this paper includes two different kinds of failures: one related to High Pressure Compressor (HPC) degradation and another to fan degradation. Although the data set does not explicitly indicate which assets experience each kind of failure, it is easy to classify them when the machines are operating in sea level conditions (that is for the FD001 and FD003 training sub-sets of C-MAPPS). In order to do so, we realised that the values of sensor 7 had a very strong positive tendency for the case of fan degradation failures and a negative tendency for the case of High Pressure Compressor degradation. Therefore, by simply applying a rolling average[2] of the sensor's data together with a majority rule we were able to classify trajectories in these data sets according to their failure type with 100% accuracy (see Fig. 3).

[2]With a window size of 40.

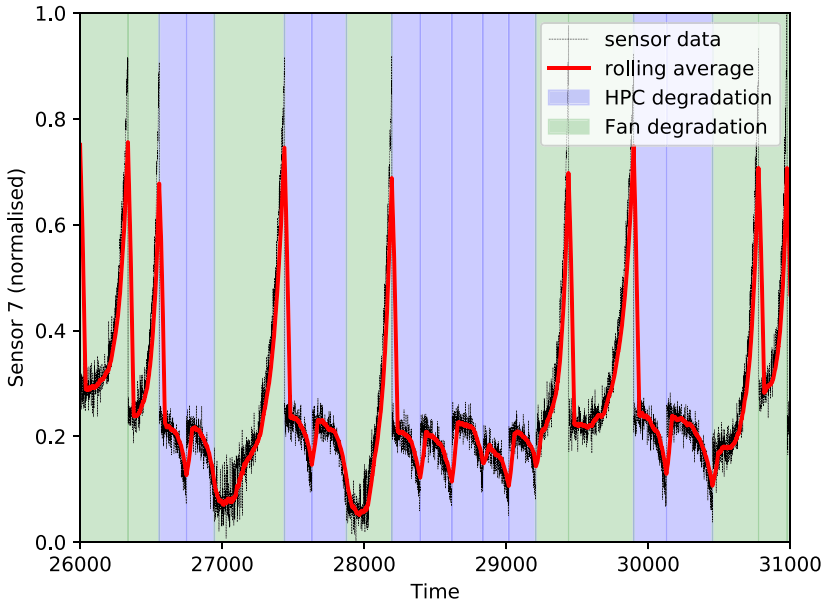

Figure 3: A subset of trajectories to failure extracted from the FD003 data set, featuring raw values of the s7 sensor (black line), its rolling average (red line), and their classification as High Pressure Compressor (HPC) degradation or fan degradation according to a simple majority rule. Majority of positive values on the first derivate of the rolling average correspon to fan degradation, majority of negative values correspond to HPC degradation.

3.1.2. Classification of operational settings

In order to check that the proposed system fulfils the properties of Advanced Multi Agent Systems, we need to test it against varying operational settings. It is therefore useful to find which of the six operational settings included in the data set each asset goes through in order to design experiments in which an asset is made to change operational settings in between recurrent failures. Luckily C-MAPPS trajectories allow for easy classification in six major operational settings (see Fig. 4). After analysing the data, there is a subset of assets that remain static in the same operational condition while other vary their operational setting across each degradation trajectory. In practice, this means that there are two types of assets in the data set: those that remain at sea level, and those that vary their operational level along their life-cycle (see Fig. 5).

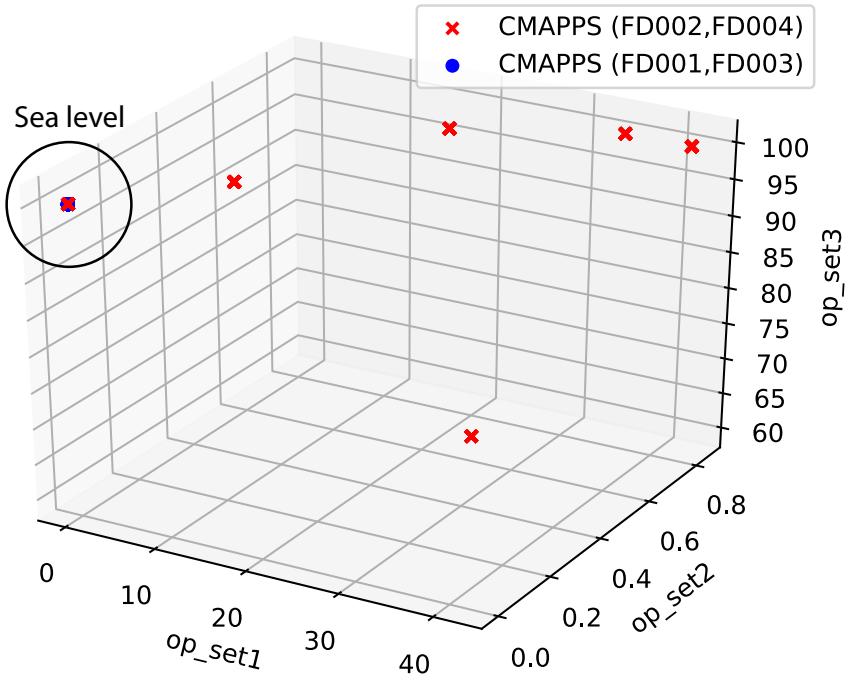

Figure 4: Six major operational settings of the C-MAPPS data set, colored according to the sub-sets of C-MAPPS. Barely visible because of their overlap, all data sets are shown to contain units that operate at certain point in sea level. FD001 and FD003 (blue) have only units that operate in Sea level and no units operating in any other opeational setting. Once controlling for the unit's lifecycles one observes that all units belonging to FD002 and FD004 (red) found to operate in sea level also operate in at least one of the other settings (see Fig. 5).

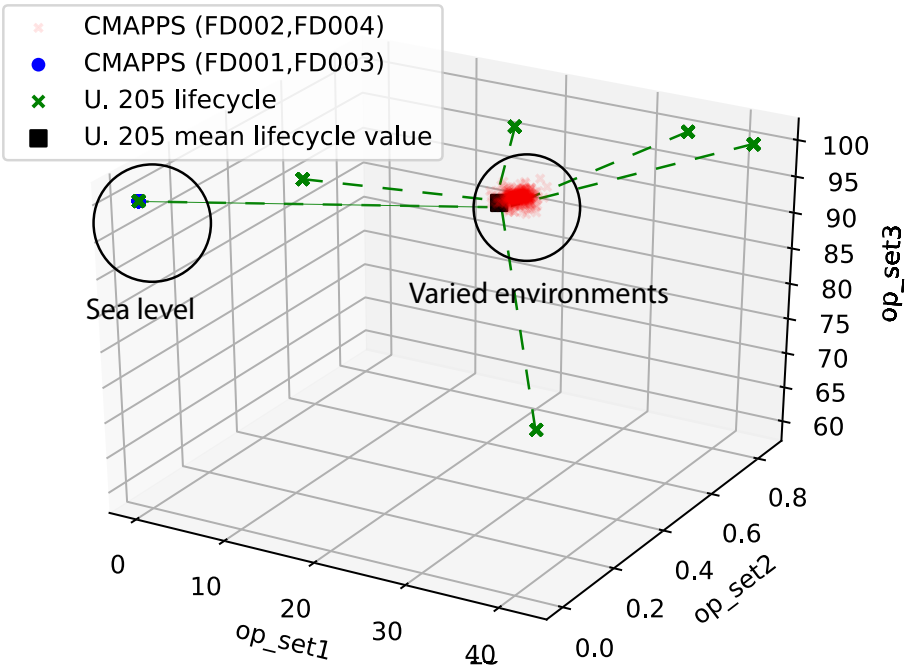

Figure 5: Mean operational settings in the C-MAPPS data set. Note how FD001 and FD003 (blue circle) operate at all time in sea level conditions, while the rest of units vary their operational setting accross their lifecycles (red cross). To show how the averages are constructed, a sample unit belonging to FD002 is plotted across its lifecycle (green crosses) where it transitions from sea level to other operational settings. The filled black square shows the mean of its operational settings.

3.1.3. Experiments

In this Section, we describe the experiments performed to test the system against the properties of Advanced Multi Agent Systems, presented in Section. 2.1.

- *Distribution*: distribution is a characteristic of the system by design. Concretely, this is achieved by running independent python scripts that share information with each other using the websocket protocol [37]. Thus, the agents of the architecture are executed by default as a distributed system, regardless of them being hosted within a single or several computers. In practice, this means that all the experiments presented in this paper are a test of the distributed property of the system. As an initial proof of concept, we present an experiment with two fleets of different assets that showcases the standard behaviour of the system under stable asset and environmental properties. To do so, we only use trajectories belonging to turbines operating at sea level, and divide them in 10 sets of 20 trajectories, each set representing a virtual machine, linked to a Virtual Asset. Clustering in the data set is made possible by the fact that some of these

machines include failures due to fan degradation, and some other feature failures due to HPC degradation (see Fig. 6).

- *Flexibility/Adaptability/System learning*: an important property of Advanced Multi Agent Systems is their capacity to continuously adapt to changes in the environment and in the agents. In order to test whether our system fulfils these characteristics, we construct our data set to represent two experimental scenarios: a system with assets with changing asset properties (failure types), and a system with assets with changing operational settings (for example, an asset that starts operating at sea-level and then moves to higher altitudes). On top of that, the real-time nature of the prognostics algorithm means that the system is adaptable by default to changes in the sensor readings as the agent's models are continuously updated.

- *Scalability*: to test the system against this property, we expand the C-MAPPS data set with data from the PHM08 prognostics data challenge, expanding the size of our asset fleet to 45 assets undergoing recurrent failures in several different operational conditions.

- *Resilience*: in order to investigate the resilience of the system, we provoke faults both in the agents and in their data quality and observe whether these faults are propagated across the asset fleet. We test for both system and sensor faults: (1) we abruptly stop one of the agent's processing without properly closing its socket connection with the platform (as if for example the corresponding asset would have suffered a catastrophic failure), and (2) we corrupt the data corresponding to one of the assets with some typical sensor reading errors: constant (or flat reading) errors, short (or outlier) errors, and drift. Scaling and noise errors are skipped because the Recurrent Neural Network is supposed to deal with the first automatically, and noisy data is already incorporated in C-MAPPS [38, 39, 40].

- *Leanness*: to investigate the leanness of the proposed system, we describe which changes are required in order to apply it to a different prognostics case, concretely for the fleet of industrial gas turbines of a major international manufacturer. As for the case of Distribution, Leanness is a design property of the architecture of the system.

3.2. Real-time distributed collaborative prognostics algorithm

In this section, we present our algorithm written in pseudocode. In practice, the implementation has been done using Python. Concretely, we use Keras [41], Tensorflow [42] and the wtte-rnn Python package [43] for the machine learning implementation. We have chosen this over Multi Agent software frameworks to ensure its deployability in real industrial systems and to conform to state-of-the-art machine learning practices that commonly use the aforementioned libraries. We have used the websocket library to enable communication in between agents

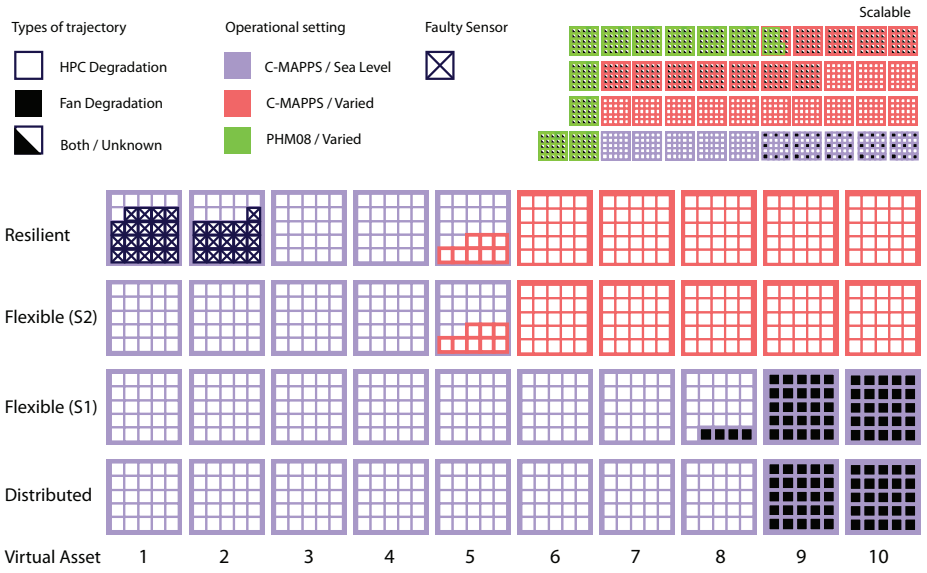

Figure 6: Sketch of the re-structuring to the C-MAPPS and PHM08 data sets in order to conform to the different experimental scenarios described in this section. Each experiment involves 10 Virtual Assets. Each cluster of squares represents an asset going through 20 recursive failures. Each different background colour represents a different operational setting. Each square represents a failure trajectory. Filled squares represent a machine failing due to fan degradation, and empty squares due to High Pressure Compressor degradation. Half-filled squares represent trajectories with an unknown failure type. Squares with a black cross represent trajectories where a sensor failure has been induced.

together with the asyncio library and Threads to ensure asynchronism and parallelism. The algorithm is designed so that it can be run both in a single terminal (through a bespoke bash script) and in a physically distributed way. In any case, the memory spaces of each agent are always separated.

To reduce bugs stemming from the loss of coupling between each agent's internal clock, the algorithm is designed such that different communication tasks are allocated to different parallel loops. In essence, send and receive loops are separated and use different ports. Each Virtual Asset - Digital Twin couple uses two ports, and two more ports are used to connect all the Digital Twins to the platform, acting as server. Digital Twins are programmed to act as websocket clients.

In a real prognostics scenario, the time period of the asset's recurrent failures will be larger than the training time of the agent's machine learning algorithms. However, when running the proposed system using Virtual Assets, the frequency of sensor reading is not limited by physical constrains, and the agent's communication loops are able to process the entire data set extremely fast. This does not allow the relatively slower deep learning prognostics algorithm to complete its training in time in between the occurrence of failures. In order to address this, we have incorporated the analytics engine of the Digital Twin in the same loop in which it receives data from the Virtual Asset instead of running it in a parallel thread. This allows to stop the reading of the Virtual Asset's data until the prognostics algorithm in the Digital Twin is complete, therefore replicating the behaviour of the system in a real industrial scenario. Such a modification is not needed in the Social Platform, where enterprise level algorithms are run in real time and in parallel with the communication loops.

With regards to the data flow, a system of inset Python dictionaries is used. Agent data increases in size and complexity as it travels up through the hierarchy of the Multi Agent System. In the Virtual Asset, the main data component is the data of the machine i recorded in the time interval Δt (data$(i, \Delta t)$). This component is then sent to the Digital Twin together with the machine's id and is concatenated in a dictionary (Dict$(i, \Delta t)$) that contains the data of the machine until the time $t = \sum \Delta t$ where t is the current lifetime of the Digital Twin. By means of its analytics engine, the Digital Twin produces another dictionary, (PreDict$(i, \Delta t)$), that contains prognostics data (event predictions and recorded accuracies). These two higher order dictionaries are then sent to the Social Platform where they are saved together with the dictionaries coming from all the other Digital Twins in a dictionary of the Social Platform called 'machine data'. Together with this dictionary, in the Social Platform there are two other dictionaries: 'glob data' (global data) and 'control variables'. 'glob data' contains variables such as the Friendship Machine (the output of the clustering algorithm) and the accumulation of prognostics scores reported from the Digital Twins. 'control variables' contains the variables needed for the normal functioning of the Social platform such as human inputs and information about the number of agents active in the system.

In our implementation, the trajectories shared among groups of similar assets are not weighted according to their clustering distances, but directly incorpo-

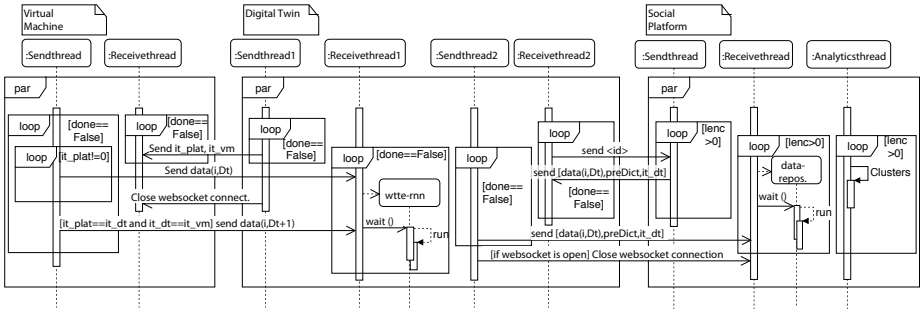

Figure 7: UML diagram of the system's algorithms, including the major tasks performed by each block of the architecture (see Fig. 1). Black arrows are used to signify communication using the websocket protocol. The variable len_{co} indicates the amount of machines connected to the Social Platform.

rated. In addition, censored trajectories are not included in the training of the Recurrent Neural Network. This has been done to enable faster computation, as it reduces the number of trajectories included in the training matrix.

Agents in Multi Agent Systems typically communicate using an Agent Communication Language [44]. In our system, this can be achieved through pykqml [45], a python library that allows writing messages following the Knowledge Query and Manipulation Language (KQML) [46]. Concretely, in our system agents convert the Python's dictionaries into a KQMLList and then send it through the websocket protocol.

3.2.1. Clustering

Among the desired properties of the proposed system are leanness and adaptability. Thus, we use standard clustering methods already available in literature. Finding a clustering algorithm able to simultaneously process environmental and failure type changes proved to be challenging. This was largely due to the small number of assets in the experiments compared with the number of sensors representing each asset, also known as curse of dimensionality [47]. We found that DBSCAN, K-Means clustering and Hierarchical clustering all provided good insights. We chose DBSCAN because (1) it was much more accurate in predicting the number of clusters, (2) it transferred well across experiments, and (3) scalable extensions of this clustering algorithm, able to handle large population scenarios, are available [48]. The clustering algorithm used in our experiments is shown in Algorithm 1. For our experiments, we used a standard euclidean distance. However, distances based on the Manhattan and fractional distance metrics should be also considered in scenarios where the number of assets is of the order of the number of features representing each asset, as they have shown to produce better results for high dimensional data [49].

Algorithm 1 Clustering algorithm

1: collect un-censored trajectories from each asset
2: **for** last N trajectories **do** take the mean for each feature across all time-steps
3: **end for**
4: append the means in a matrix containing all asset's in the fleet
5: normalise across the matrix using sklearn's MinMax scaler
6: calculate euclidean distances between the assets of the population
7: retrieve the maximum distance D
8: compute the clusters using Python's DBSCAN with epsilon=$5D$ divided by the number of assets in the fleet and min samples equal to one.

3.3. Results and discussion

3.3.1. Initial parameters

In all cases, we run the algorithm shown in Fig. 7 with the following initial parameters:

- Each asset starts with six trajectories to event as previous experience. The training time of the Digital Twins, T_{T} is set to the time at which a Digital Twin records a new failure.
- The Recurrent Neural Network has a LSTM architecture of $26 \times 24 \times 10 \times 2$ being the layer with 24 neurons the only recurrent layer. We use tanh as activation function for all layers except for the last two-neurons custom output layer. No Dropout, or regularisation are used.
- The Adam optimizer [50] is used with learning rate 0.01 and norms clipped at 10 to avoid exploding gradients (see Section. 2.3.1 for a further discussion on stability).
- The maximum number of epochs for the neural network is 400 except for the experiment where scalability is tested. Early stopping is implemented through the validation loss to limit over fitting. The patience is 50 and the minimum delta is 0.05. The split is 5/6 training 1/6 validation.
- For the clustering algorithm, N=2 (the last two uncensored trajectories are considered).

In order to asses prognostics accuracy, we use the mean square error of the predicted times to failure. The analysis thread of the Social Platform is programmed to average over fleet scores in real time and plot them. However, for the results presented here we use bespoke algorithms that import the results of an experiment from the agent's databases. The agents are programmed so that if running as an experiment they periodically save their variable space in the server memory, so that these results can be accessed at any time. Experiments were performed on DIAL's high performance center 'optimusprime'.

3.3.2. Distributed

In order to test the Distributed property of the system we perform the experiment proposed in Section. 3.1.3. Assets were quickly clustered in two fleets of different assets according to their failure type, and the Digital Twins were able to estimate their failure times in real time and with a good degree of accuracy.

To assess the validity of the idea of real-time distributed collaborative prognostics, we compared the results of a fleet of assets sharing data only with similar assets versus a fleet of assets randomly sharing data. From the results, the collaborative strategy is found to outperform the non-collaborative (random) strategy. The clustering algorithm performed by the Social Platform manages to cluster assets properly for every time step, with the exception of assets number 9 and 10 that are clustered properly 80% of the times (see Figs. 8 and 9)

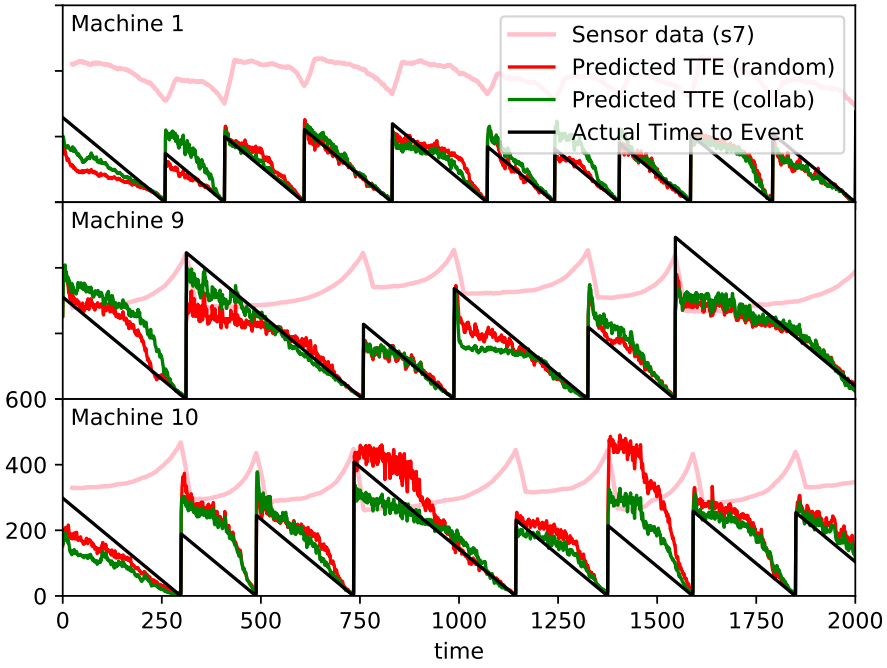

Figure 8: Output from three of the Digital Twins in the asset fleet featuring predicted times to failure for an experiment enabling collaborative learning (green) and another one allocating each asset's friends randomly (red). Overlayed, scaled measurements of the seventh sensor of their corresponding assets (pink). Note how especially for the Digital Twins corresponding to assets enduring fan degradation failures (assets 9 and 10), collaborative predictions often outperform non-collaborative predictions significantly (true time to failure shown in black).

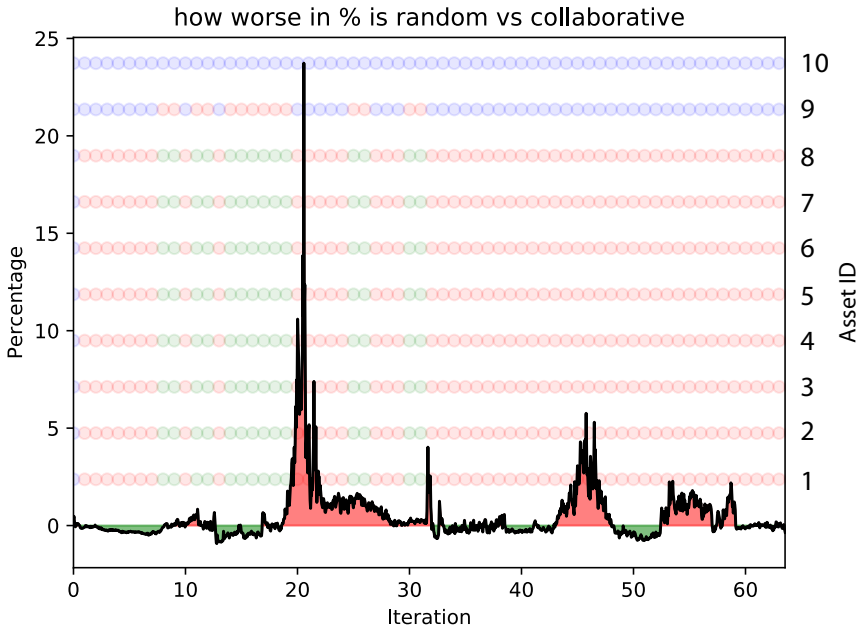

Figure 9: Difference in percentage in accuracy scores between random and collaborative learning in the asset fleet. Shaded in red, timesteps when random allocation underperforms collaborative learning. Overlayed, clusters to which each machine of the asset fleet were assigned as the experiment advanced. Note how assets 9 and 10 were properly clustered together for most of the experiment, as they are the only ones featuring fan degradation failures only.

3.3.3. Flexible/Adaptable/System Learning

Two scenarios were devised to test the system flexibility and adaptability, and its ability to learn in real time; one in which the failure type of some of the fleet's assets changed across time, and another one in which their operational setting was set to vary (see Section. 3.1.3).

- *Scenario 1*: in this scenario, one of the assets that formerly only featured HPC degradation failures starts experiencing fan degradation failures towards the end of the experiment. The system reacts as follows: (1) the platform clusters this asset with the other assets in the fleet that share the same failure type (Fig. 10), (2) the Digital Twin incorporates this knowledge in its forthcoming predictions (Fig. 11).

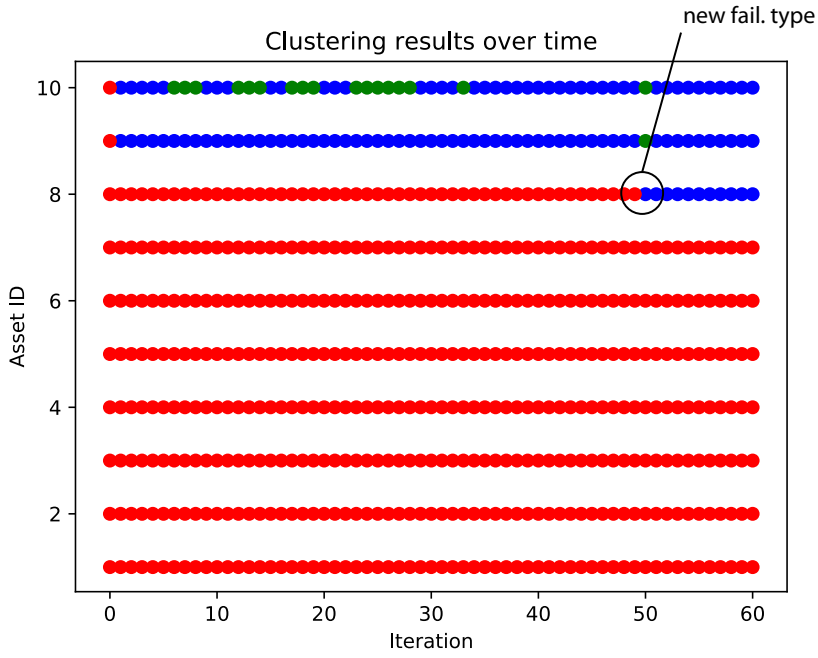

Figure 10: Output from the Social Platform for Scenario 1. The cluster to which each asset belongs at each timestep is indicated by its colour. Note shortly after asset number 8 starts experiencing a new kind of failures, it is clustered together with its peers. The clustering algorithm works well with the exception of a couple of short-term glitches.

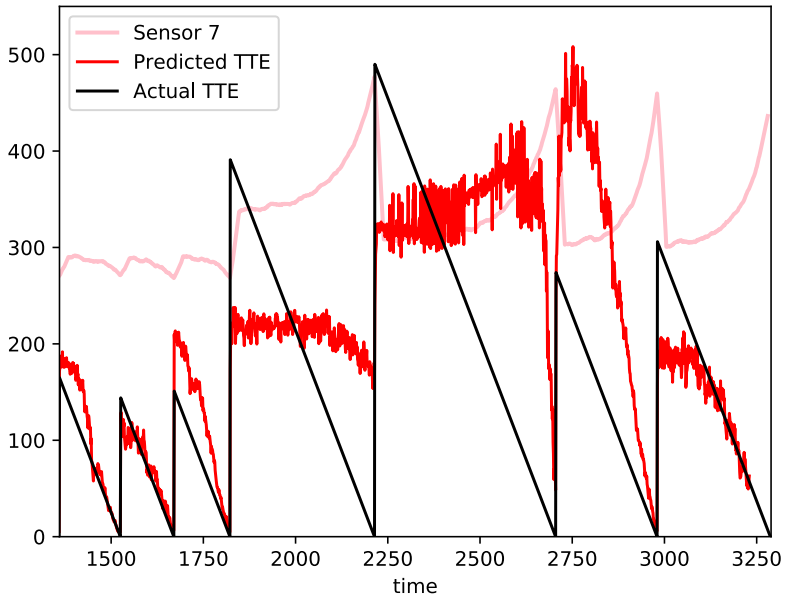

Figure 11: Output from the Digital Twin of Asset nr. 8 in Scenario 1. Note how the twin is able to learn the new failure type thanks to the data from its peers even if its own data consists of just a couple of trajectories.

- *Scenario 2*: in this scenario, the failure type of the assets in the fleet is kept constant (data from FD001/2 contains solely fan degradation failures). However, the operational setting of one of the assets vary moving from sea level to a varied environmental condition. The system rapidly reacts to these changes and clusters this asset with the other assets operating under similar conditions (see Figs. 12 and 13).

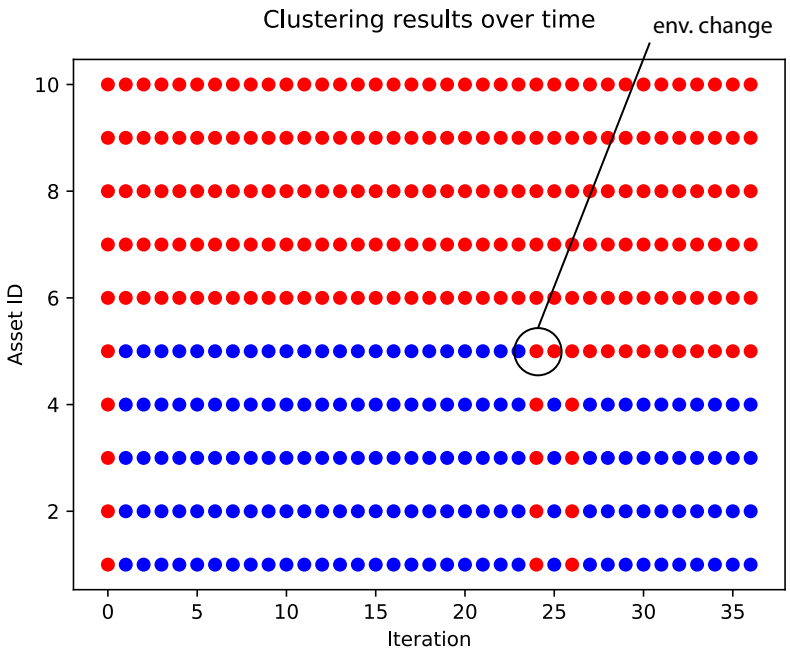

Figure 12: Output from the system's platform data for Scenario 2. The cluster to which each asset belongs at each timestep is indicated by its colour. Note that inmediately after asset number 5 changes its environment, it is clustered together with the rest of the fleet.

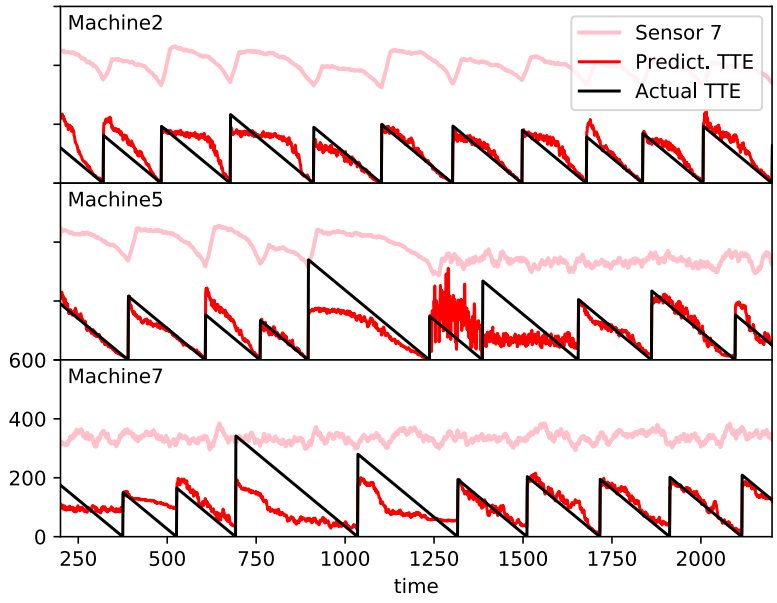

Figure 13: Output from the Digital Twin of asset nr. 5 in Scenario 2. Note how inmediately after switching environment, the asset is able to learn from its new environment thanks to the data from its peers even if its own data is scarce.

3.3.4. Scalable

To test the system scalability, we designed an experiment in which we computed as many agents as it was possible without repeating any of the agent's data and whilst yielding enough trajectories per asset to represent an interesting machine learning scenario. Here, simulations are run in a central server while in reality they could also be run in a physically distributed way. Combining the C-MAPPS data set with the PHM08 prognostics data set, one obtains 926 independent trajectories to failure that once divided in sets of 20 end up yielding a total of 46 assets (see Fig. 6). In Fig. 14 and 15, one can observe how the system was able to process all assets properly. Scalability, in theory, should be ensured by the websocket protocol, where a websocket server is able to service hundreds of thousands of clients without significant lag. However, when running experiments in a central server as we do in this paper, the computational cost of distributed Recurrent Neural Networks must be taken into account. In our case, we set the maximum number of epochs of the Recurrent Neural Network to 100.

In a real industrial scenario, the real-time functionality of the system is guaranteed by the fact that the training time of the Recurrent Neural Network is much lower than the time in between failures or events of interest of a typical

industrial machine. This applies even if modest computational resources are assumed (see [29] for an extended discussion).

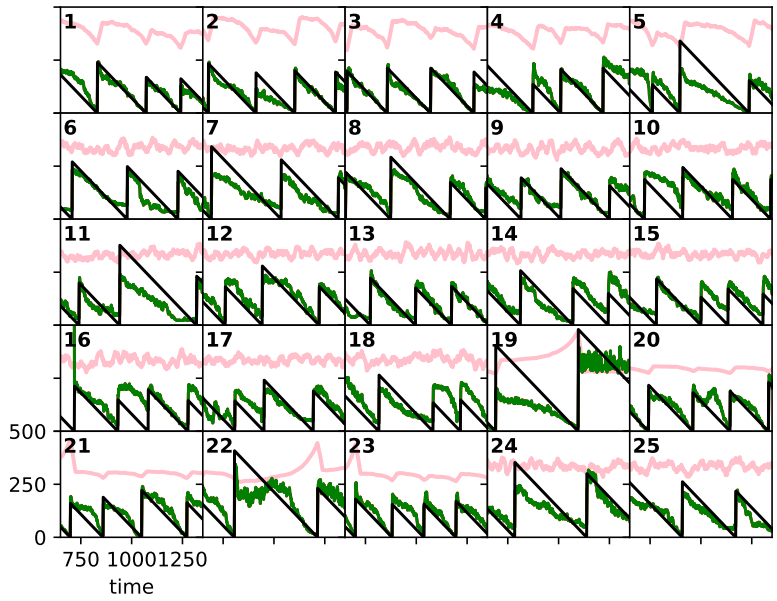

Figure 14: Output from 25 of the 46 assets in the data set with a number indicating the identity of each asset. The pink line shows the scaled value of sensor 7, the green line shows the predicted time to event and the black line shows the real time to event.

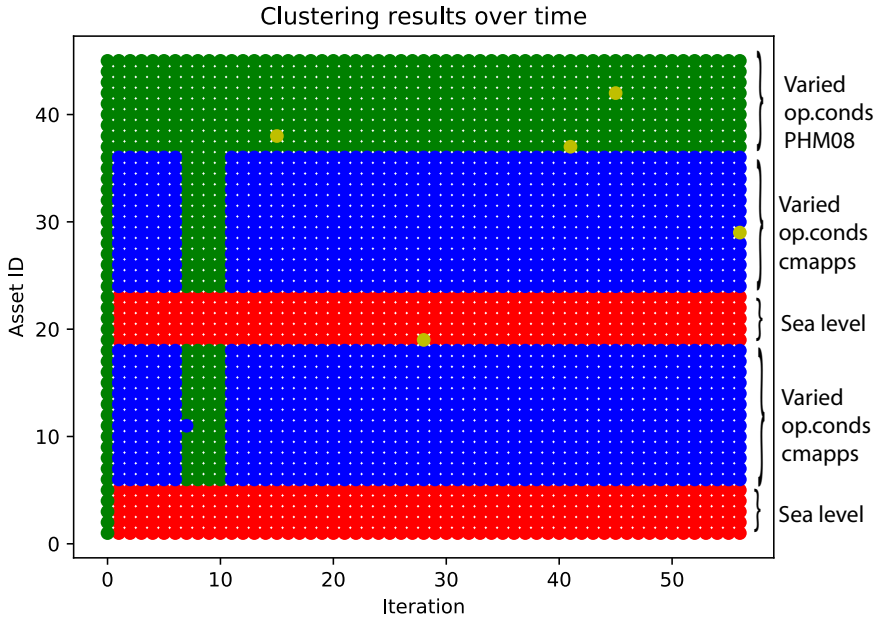

Figure 15: Clusters formed for the assets in the data set. Note how clustering seems to be strongly dominated by the environmental condition, and how the five machines corresponding to FD001 and FD003 (sea level) are clustered together. Environmental conditions cause the algorithm to cluster together machines corresponding to the C-MAPPS and PHM08 data sets for a few time steps.

3.3.5. Lean

In order to test the system for leanness, a different prognostics scenario was implemented to test how much of the architecture and code had to be changed for the system to be able to process it. In this section, we summarize the changes performed in the code and we present a first version of our results for this experimental case. Concretely, we focused on predicting recurrent electronic control unit faults in the gas turbines of a major international manufacturer.

As expected from the design of the architecture, the system was able to operate by just modifying the code belonging to the Virtual Asset. This had to be done in order to process the data, which came in a different format than for the case of C-MAPPS.

Compared to the C-MAPPS case, the Virtual Asset agent required about 140 additional lines of code, programmed to execute the following tasks:

1. Basic import, sampling and standardization of gas turbine data.
2. Segmentation of gas turbine data according to different kinds of events.
3. Adaptability to new scenarios: some machines did not record any electronic control unit fault. This, in practice, meant that until data was

shared from other turbines, their Digital Twins could not train their prognostics algorithm.

No fine tuning was performed for the gas turbine prognostics and clustering algorithms, and the same configuration as in C-MAPPS was used. Figs. 16 and 17 show the initial results.

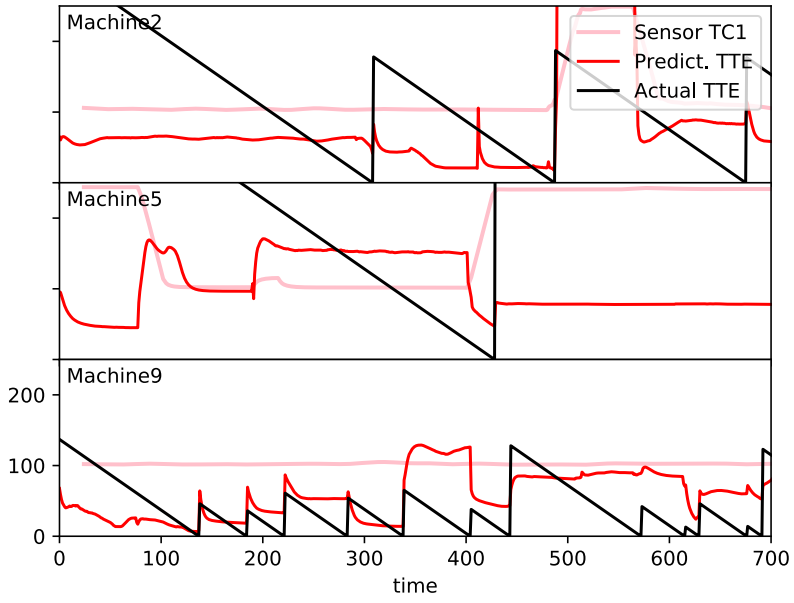

Figure 16: Output from the Digital Twins of asset nr. 2, 5 and 9 in a real industrial scenario: note how despite no tuning is done in the training of the Recurrent Neural Network, the Digital Twins are able to output some rudimentary predictions for electronic control unit faults

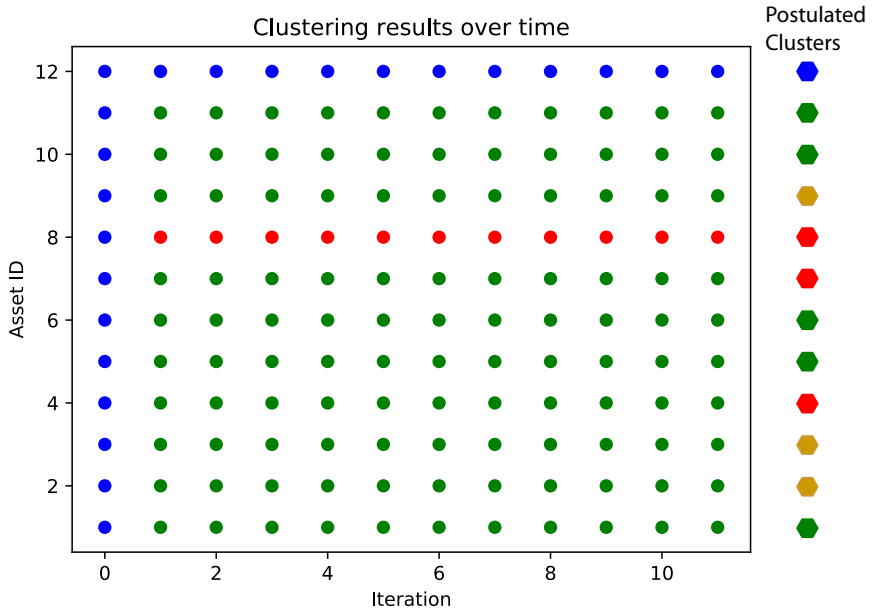

Figure 17: Clusters formed for the assets in the data set for the real industrial experiment. From our qualitative knowledge of the asset fleet, we postulate what we think should be the clustering structure. Note how the predicted structure does not differ too much from the one that we finally obtain.

3.3.6. Resilient

Resilience was tested as described in Section. 3.1.3. The system was benchmarked against several system-fault scenarios, all of them tested for Scenario 2 of Section. 3.3.3. Faults were introduced in sensors 2, 5, 7, 9, 11, and 14 in assets 1 and 2.

1. Flat reading fault: assets were made to experience flat reading faults. Asset 1 experienced its fault from the beginning of its operational cycle, and asset 2 experienced it at about half-life. Fig. 18 shows how the assets were clustered away from the rest of the fleet, and avoided contaminating the rest of predictions. Despite flat reading faults, the fleet was still able to replicate the flexible properties of Scenario 2.

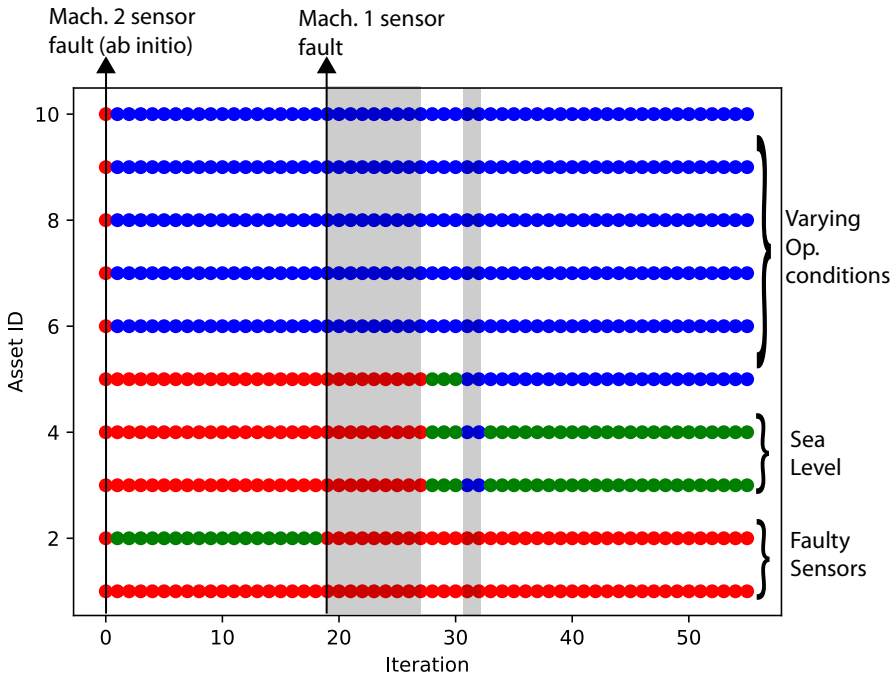

Figure 18: Clusters formed for the assets in the data set for the case of flat reading sensor faults. Note how assets containing sensor faults are automatically separated from the rest of the fleet, and how the normal fleet clustering is not affected. The areas shaded in gray indicate the areas where the clustering algorithm failed to cluster some of the assets properly.

2. Outliers: regular outliers of 5000000 points were introduced every 30th time step. The Social Platform was able to cluster assets with outliers together properly for most of the time while maintaining the rest of the clusters known to the fleet (see Fig. 19). Despite not using a quartile-based scaler specially designed to remove outliers, the prognostics algorithm operated well in their presence, generally returning similar predictions to the case with no outliers. This could be caused by the outlier's periodic nature, likely to have been learnt by the Recurrent Neural Network.

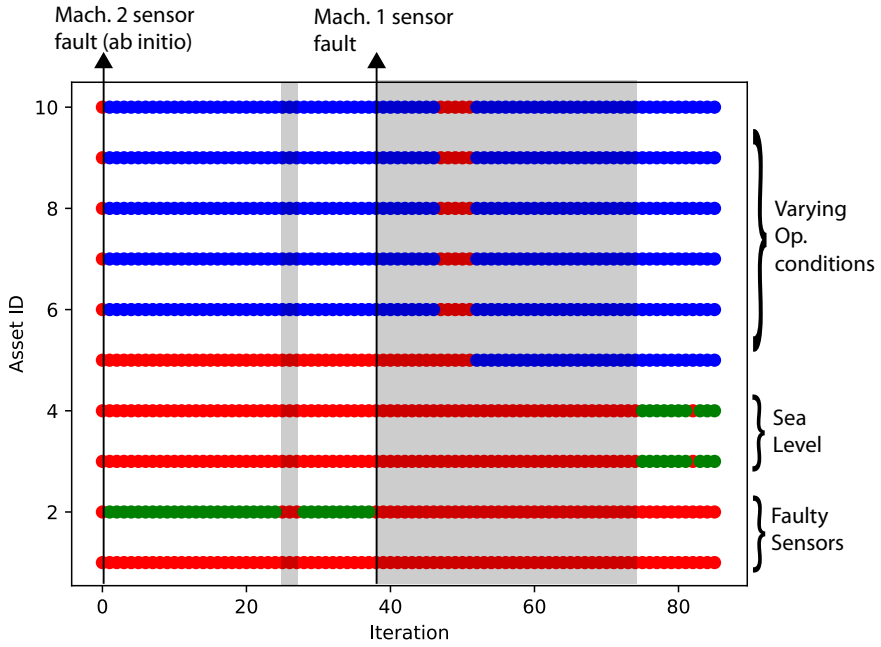

Figure 19: Clusters formed for the assets in the data set for the case of assets with recurrent outliers. Note how assets containing sensor faults are automatically separated from the rest of the fleet, and how the normal fleet clustering is not affected. The areas shaded in gray indicate the areas where the clustering algorithm failed to cluster some of the assets properly.

3. Drift: a positive exponential drift with a cap at very large numbers was introduced in the trajectories of the aforementioned assets. An exponential drift was chosen because it represents the extreme case where the sensors end up malfunctioning completely. The Social Platform reacted clustering the assets presenting faulty sensors together and separating them from the rest of the fleet (see Fig. 20). Other kinds of drift were also implemented (linear, and quadratic), and the ability of the system to react to these depended on the metric used in the clustering algorithm and the drift magnitude.

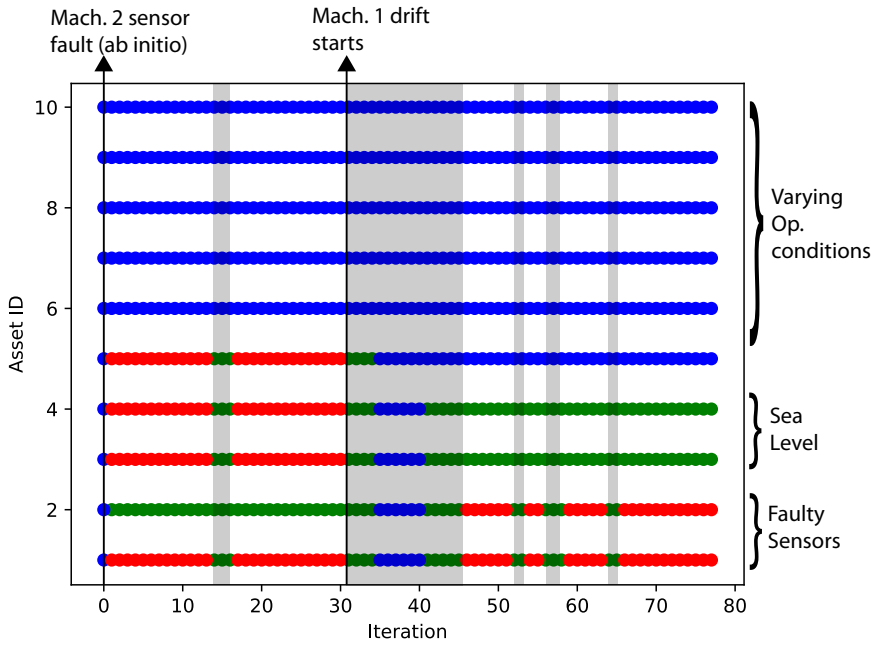

Figure 20: Clusters formed for the assets in the data set for the case of assets with exponential drift. Note how assets containing sensor faults are automatically separated from the rest of the fleet, and how the normal fleet clustering is not affected. The areas shaded in gray indicate the areas where the clustering algorithm failed to cluster some of the assets properly.

Additionally, to test the system against complete agent failure, agents were terminated at different levels of the hierarchy by killing their corresponding Python processes. Terminating a Digital Twin or Virtual Asset resulted in the loss of the ability of the system to output predictions for their corresponding asset, but did not affect the behaviour of the other agents in the system. The Social Platform automatically kept the last data received from the corresponding Digital Twin and repeatedly used that data for clustering and data sharing. Terminating the Social Platform had a different effect: the Digital Twins are programmed to await platform input to continue with their prognostics algorithm thus stopping the platform stops the system altogether.

4. Conclusion and future work

We present and rigorously test an Industrial Multi Agent System for real-time distributed fleet prognostics. This system can adapt to the characteristics of real asset fleets that make of prognostics a challenging problem: sensor faults, asset heterogeneity, change of environmental and asset conditions, challenging data preparation etc. We conclude that the presented system clearly shows the benefits long postulated to be inherent to these systems, which can be summarised as the ability of operating in dynamic and heterogeneous scenarios.

Additionally, we test the approach known as distributed collaborative prognostics, that is, the idea that by sharing data among similar assets more accurate predictions can be achieved. We observe that this is the case especially when asset fleets exhibit a clear clustering tendency. However, we also observe that due to the flexibility of Recurrent Neural Networks, including data from different assets in the Digital Twin's training is less damaging to the prediction accuracy than it may be expected (for a fleet of 10 assets, the difference in accuracy peaks at 25%). Distributed collaborative learning is found to be specifically pertinent in scenarios that fulfil any of the following properties:

- Presence of sensor faults.
- Limited processing capabilities.
- Large fleet heterogeneity.

Which is the case of most continuously monitored large fleet of assets. From a practical perspective, the only constraint on scalability is the computational cost of Deep Learning Algorithms. Although this is not a problem from a theoretical perspective (the system assumes a physically distributed fleet), from an experimental perspective this is found to be a limiting factor for the size of the experiments that can be tested in a single server.

4.1. Future work

Future work is derived directly from the limitations of the presented study, that is from the fact that a simulated data set is used (C-MAPPS), and that a relative low number of synthetic assets are dealt with (46). This number is low with respect to real industrial fleets but very high compared to other published research. Thus, further experimentation with larger and more varied fleets of assets is advisable. Next steps are to optimize the proposed system for a real industrial fleet with several failure types occurring at the same time, to test for the efficacy of the presented clustering algorithm in different scenarios, and to add a layer of preventive maintenance optimization in the Social Platform.

5. Acknowledgements

Sponsors: Adrià Salvador work was sponsored by a Doctoral Scholarship provided by "Fundació la Caixa", code LCF/BQ/EU17/11590049. This research was also supported by SustainOwner (Sustainable Design and Management of Industrial Assets through Total Value and Cost of Ownership), a project sponsored by the EU Framework Programme Horizon 2020, MSCA-RISE-2014: Marie Skodowska-Curie Research and Innovation Staff Exchange (Rise) (grant agreement number 645733 Sustain-owner H2020-MSCA-RISE-2014). The server used to perform the experiments in this paper 'optimusprime' was funded by the Centre for Digital Built Britain. The authors would like to thank Andrew Benton and his colleagues at Rutgers University for their help in overcoming some of the programming challenges encountered when running the experiments.

Ref.		Year	Distr.	Flex.	Adapt.	Scal.	Lean	Resil.
[51]	**Hadden G. D. et al.**	2002	Y	N	N	Y	Y	Y
[52]	**Zhou J. et al.**	2005	Y	Y	Y	Y	N	Y
[53]	**Saha S. et al.**	2008	Y	Y	N	Y	Y	Y
[54]	**Ribot P. et al.**	2008	Y	N	N	Y	Y	Y
[55]	**Chen C. et al.**	2012	N	Y	N	Y	Y	N
[56]	**Luca Fasanotti**	2014	Y	Y	Y	Y	N	Y
[57]	**Desforges X. et al.**	2014	Y	Y	N	Y	N	Y
[58]	**Wang J. et al.**	2015	Y	Y	Y	Y	N	Y
[59]	**Yu L. et al.**	2016	N	N	Y	Y	Y	N
[8]	**Canizo M. et al.**	2017	N	Y	Y	Y	Y	N
[60]	**Kiangala K. S. et al.**	2018	Y	Y	N	Y	Y	Y

Table 1: Table featuring articles that propose architectures, experiments or implementations pertinent to distributed real-time prognostics. Note how Adaptability and Leanness are the properties that appear in fewer implementations, whilst all of them are judged to fulfil scalability (although only some of them demonstrate it experimentally).

6. Appendix: other studies

Table 1 features previous research pertinent to distributed, real-time prognostics or general Multi Agent Architectures.

References

[1] T. S. Baines, State-of-the-art in product-service systems, Proc. Inst. Mech. Eng. B. J. Eng. Manuf.

[2] S. Shaheen, S. Guzman, H. Zhang, Bikesharing in Europe, the Americas, and Asia, Transport. Res. Rec.

[3] A. Neely, Exploring the financial consequences of the servitization of manufacturing, Oper. Man. Res.

[4] D. Goyal, B. S. Pabla, Condition based maintenance of machine tools-A review, CIRP J. Manuf. Sci. Technol. 10 (2015) 24–35.

[5] D. Kwon, M. R. Hodkiewicz, J. Fan, T. Shibutani, M. G. Pecht, IoT-Based Prognostics and Systems Health Management for Industrial Applications, IEEE Access 4 (2016) 3659–3670.

[6] D. Goyal, B. S. Pabla, Condition based maintenance of machine tools-A review, CIRP J. Manuf. Sci. Technol. 10 (2015) 24–35.

[7] T. Sutharssan, S. Stoyanov, C. Bailey, Y. Rosunally, Data analysis techniques for real-time prognostics and health management of semiconductor devices, 18th Europ. Mic. Pack. Conf. (2011) 1–7.

[8] M. Canizo, E. Onieva, A. Conde, S. Charramendieta, S. Trujillo, Real-time predictive maintenance for wind turbines using Big Data frameworks, in: 2017 IEEE Intern. Conf. Prog. Heal. Man., ICPHM 2017, 2017.

[9] P. Leitão, S. Karnouskos, Industrial Agents Emerging Applications of Software Agents in Industry, 2015.

[10] V. Marik, D. McFarlane, Industrial adoption of agent-based technologies (2005).

[11] K. Tuyls, G. Weiss, Multiagent learning: Basics, challenges, and prospects, AI. Mag. 33 (3) (2012) 41–52.

[12] R. Zhou, B. Fox, H. P. Lee, A. Y. Nee, Bus maintenance scheduling using multi-agent systems, Eng. Appl. Artif. Intell.

[13] Q. F. Sun, S. Li, Bo, An Intelligent Fleet Condition-Based Maintenance Decision Making Method Based on Multi-Agent, in: Int. J. Progn. Health. Manag., 2013, pp. 1–11.

[14] K. Upasani, M. Bakshi, V. Pandhare, B. K. Lad, Distributed maintenance planning in manufacturing industries, Comput. Ind. Eng.

[15] A. Brintrup, D. McFarlane, D. Ranasinghe, T. Sánchez López, K. Owens, Will intelligent assets take off? Toward self-serving aircraft, IEEE Intell. Syst. 26 (3) (2011) 66–75.

[16] H. Li, A. Parlikad, A. Salvador Palau, A Social Network of Collaborating Industrial Assets, Proc. Inst. Mech. Eng. O. J. Risk. Reliab.

[17] A. S. Palau, Z. Liang, D. Lütgehetmann, A. K. Parlikad, Collaborative prognostics in Social Asset Networks, Future. Gener. Comput. Syst.

[18] Y. Lin, S. Huang, G. E. Simon, S. Liu, Analysis of depression trajectory patterns using collaborative learning, Math. Biosci. 282 (2016) 191–203.

[19] L. Drew, Pharmacogenetics: The right drug for you, Nature.

[20] Y. Lin, K. Liu, E. Byon, X. Qian, S. Huang, S. Liu, A Collaborative Learning Framework for Estimating Many Individualized Regression Models in a Heterogeneous Population, IEEE Trans. Rel. 67 (1) (2018) 328–341.

[21] Y. Lin, S. Liu, S. Huang, Selective Sensing of A Heterogeneous Population of Units with Dynamic Health Conditions, IISE. Trans. Healthc. Syst. Eng. 5854 (May).

[22] M. Wooldridge, N. R. Jennings, Intelligent Agents: Theory and Practice (1995).

[23] P. Maes, Agents that reduce work and information overload, Commun. ACM.

[24] M. Tan, Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, in: Proceedings Tenth ICML, 1993.

[25] K. Bakliwal, M. H. Dhada, A Multi Agent System architecture to implement Collaborative Learning for social industrial assets, in: INCOM 2018, 2017.

[26] E. Martinsson, WTTE-RNN : Weibull Time To Event Recurrent Neural Network, Ph.D. thesis, Chalmers University Of Technology (2016).

[27] N. E. Zoghby, V. Loscri, E. Natalizio, N. E. Zoghby, V. Loscri, E. Natalizio, Chapter 8: Robot Cooperation and Swarm Intelligence, 2014.

[28] H. S. Nwana, Software agents: an overview, Knowl. Eng. Rev.

[29] A. Salvador Palau, K. Bakliwal, M. H. Dhada, T. Pearce, A. K. Parlikad, Recurrent Neural Networks for real-time distributed collaborative prognostics.

[30] H. T. Siegelmann, E. D. Sontag, On the computational power of neural nets, J. Comput. Syst. Sci.

[31] E. R. Lapira, Fault detection in a network of similar machines using clustering approach, J. Chem. Inf. Model. 53 (9) (2013) 1689–1699.

[32] T. F. COX, T. LEWIS, A conditioned distance ratio method for analyzing spatial patterns, Biometrika 63 (3) (1976) 483–491.

[33] B. Hopkins, A New Method for determining the Type of Distribution of Plant Individuals, Ann Bot. 18 (70) (1954) 213–227.

[34] P. J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (C) (1987) 53–65.

[35] M. Ester, H. P. Kriegel, J. Sander, X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining.

[36] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: 2008 International Conference on Prognostics and Health Management, PHM 2008, 2008.

[37] I. Fette, A. Melnikov, RFC 6455 - The WebSocket Protocol, Internet Engineering Task Force.

[38] K. Ni, M. Srivastava, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, Sensor network data fault types, ACM. Trans. Sens. Netw. 5 (3) (2009) 1–29.

[39] E. Balaban, A. Saxena, P. Bansal, K. F. Goebel, S. Curran, Modeling, detection, and disambiguation of sensor faults for aerospace applications, IEEE Sens. J. 9 (12) (2009) 1907–1917.

[40] A. B. Sharma, L. Golubchik, R. Govindan, Sensor Faults : Detection Methods and Prevalence in Real-World Datasets, ACM. Trans. Sens. Netw. 6 (3) (2010) 23.

[41] F. Chollet, Keras: Deep Learning library for Theano and TensorFlow, GitHub Repository.

[42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, G. Brain, TensorFlow: A System for Large-Scale Machine Learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16), 2016.

[43] Wtte-rnn.
URL `https://github.com/ragulpr/wtte-rnn`

[44] Y. Labrou, T. Finin, Agent communication languages: the current landscape, IEEE Intell. Syst.

[45] B. M. Gyori, PyKQML: an implementation of KQML messaging in Python. (2018).

[46] T. Finin, R. Fritzson, D. McKay, R. McEntire, KQML as an agent communication language, in: Proceedings CIKM '94, 1994.

[47] M. Steinbach, L. Ertöz, V. Kumar, The Challenges of Clustering High Dimensional Data, in: New Dir. Stat. Phys., 2004.

[48] L. McInnes, J. Healy, S. Astels, hdbscan: Hierarchical density based clustering, J. Open. Source. Softw. 2.

[49] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the Surprising Behavior of Distance Metrics in High Dimensional Space, in: J. den Bussche, V. Vianu (Eds.), Database Theory ICDT 2001, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 420–434.

[50] D. P. Kingma, J. L. Ba, Adam: a Method for Stochastic Optimization, ICLR 2015.

[51] G. D. Hadden, P. Bergstrom, B. H. Bennett, G. Vachtsevanos, J. Van Dyke, Distributed multi-algorithm diagnostics and prognostics for US Navy ships, In Proc. 2002 AAAI Spring Symposium.

[52] J. Zhou, X. Li, A. J. R. Andernroomer, H. Zeng, K. M. Goh, Y. S. Wong, G. S. Hong, Intelligent prediction monitoring system for predictive maintenance in manufacturing, IECON Proceed. (Industrial Electronics Conference) (2005) 2314–2319.

[53] S. Saha, B. Saha, K. Goebel, Distributed prognostics using wireless embedded devices, PHM 2008. International Conference on Prognostics and Health Management.

[54] P. Ribot, Y. Pencolé, M. Combacau, Prognostics for the maintenance of distributed systems, INCOM 2008.

[55] C. Chen, D. Brown, C. Sconyers, B. Zhang, G. Vachtsevanos, M. E. Orchard, An integrated architecture for fault diagnosis and failure prognosis of complex engineering systems, Expert. Syst. Appl. 39 (10) (2012) 9031–9040.

[56] L. Fasanotti, A distributed intelligent maintenance system based on artificial immune approach and multi-agent systems, Industr. Inform. (INDIN).

[57] X. Desforges, M. Diévart, P. Charbonnaud, B. Archimède, A distributed Architecture to implement a Prognostic Function for Complex Systems, Proc. Annu. Conf. Progn. Health. Manag. Soc.

[58] J. Wang, L. Zhang, L. Duan, R. X. Gao, A new paradigm of cloud-based predictive maintenance for intelligent manufacturing, J. Intell. Manuf. 25 (5) (2017) 1125–1137.

[59] L. Yu, S. Shrivastava, Distributed Real Time Compressor Blade Health Monitoring System, Proc. Annu. Conf. Progn. Health. Manag. Soc.

[60] K. S. Kiangala, Z. Wang, Initiating predictive maintenance for a conveyor motor in a bottling plant using industry 4.0 concepts, Int. J. Adv. Manuf. Technol. 0 (2018) 1–21.