Implementation of a holistic application monitoring system with QoS prediction capabilities

Modelling and Orchestrating heterogeneous Resources and Polymorphic applications for Holistic Execution and adaptation of Models In the Cloud

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 – 31 December 2022

www.morphemic.cloud

Deliverable reference

D2.2

Date
27 January 2022

Responsible partner

ICCS

Editor(s)
Yiannis Verginadis (ICCS)

Reviewers
Marta Rozanska (UiO)
Alexandros Raikos (UPRC)

Distribution
Public

Availability
morphemic.cloud

Executive summary

This deliverable provides the description of a holistic monitoring system, with self-healing capabilities, aiming to provide insights on the operation of multi-cloud applications, and allow the MORPHEMIC platform to proactively respond to their needs. The intention is to present and discuss the details of the approach for monitoring applications as well as analyse the architecture of components involved in the monitoring and prediction processes. We further provide the relevant process flow of the components which are involved in the acquisition of both real-time and predicted metrics related to the QoS specification of the application.

Author(s)

Yiannis Verginadis, Ioannis Patiniotakis, Fotis Paraskevopoulos, Andreas Tsagkaropoulos, Dimitra Tzormpaki, Vassilis Stefanidis, Jean-Didier Totow, Pawel Skrzypek, Anna Warno, Maciek Riedl, Marta Rozanska, Imen Bizid

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Table of Contents

List of Tables

List of Figures

List of Equations

List of Listings

1 Introduction

1.1 Scope

This deliverable focuses on the description of a holistic monitoring system, that aims to provide insights on the operation and health status of multi-cloud applications to allow the MORPHEMIC platform to initiate timely application reconfiguration. We discuss how the MORPHEMIC monitoring system introduces self-healing capabilities, which means it is able to cope with monitoring nodes failures or other network related issues like intermittent connectivity in order to monitor the deployed cloud application constantly and effectively. Furthermore, we describe the design and implementation of the MORPHEMIC forecasting module, which comprises several implemented time-series forecasting methods for constituting the platform able to proactively respond to the deployed cloud application needs.

The aim of this document is to report on the WP2 work with respect to the implementation and integration of a federated event processing management system with self-healing capabilities with a forecasting module that will realise the proactive adaptation capabilities of the MORPHEMIC platform. Therefore, we present and discuss the details of the approach for monitoring applications as well as analyse the architecture of components involved in the monitoring and prediction processes. We further provide the relevant process flow of the components which are involved in the acquisition of both real-time and predicted metrics related to the QoS specification of the application.

1.2 Document structure and Intended Audience

The document continues with an introductory section on the MORPHEMIC proactive adaptation approach. In section 2, we describe the overall components architecture and describe the flow for all the involved technological parts of the MORPHEMIC platform that focus on proactive adaptation support. This kind of support refers to the means for deciding on application component adaptations based on anticipated or predicted imminent Service Level Objectives (SLOs) violation. This kind of proactiveness will lead to multi-cloud application deployment optimization. The MORPHEMIC components involved undertaking: i) the aggregation, processing, and storage of monitoring information; ii) the datasets preparation for training forecasting algorithms; iii) the design of different forecasters that can be used for time-series prediction according to monitoring data characteristics; and last iv) the components involved in triggering a reconfiguration cycle proactively. Section 3 continues with detailed aspects on the design and implementation of MORPHEMIC's Event Management System (EMS), focusing on its self-healing capabilities that allow this distributed framework to recover from potential failures or network issues with the less possible disruption into the monitoring effort. In Section 4, we discuss the Persistent Storage as means for storing monitoring time-series and preparing the datasets that will be used for training the forecasters used by the platform. In Section 5, we describe the details of eight different forecasters that are used in MORPHEMIC for finding the best possible predictions on different monitoring metrics and imminent SLOs violations. Furthermore, in the same section we present an illustrative use of each of these forecasters based on the same dataset and discuss some initial comparative findings. In Section 6, we focus on the exploitation of the forecasted data for orchestrating the platform's predictions and triggering the adaptation proactively when needed. This involves implementation aspects of the Prediction Orchestrator component and the Severity-based SLO Violation Detector. Next, in Section 7, we discuss the extensions made to the Metasolver to be able to exploit the predictions provided and decide on the initiation of a reconfiguration cycle, either reactively or proactively, based on the available data. Last, Section 8 concludes this report and discusses the next steps in terms of the WP2 work.

All the sections of this deliverable should be read by all research and pilot partners to understand the current approaches and implementation work regarding the platform's monitoring system and its QoS prediction capabilities. Furthermore, this deliverable includes valuable insights and progress beyond the state-of-the-art that can be interesting to any cloud computing researcher.

2 Proactive Adaptation Approach

2.1 Overall components architecture and flow description

In terms of WP2 work, we introduce new components in the MORPHEMIC architecture to support the important proactive adaptation capability. This capability (as it was analysed in D2.3) refers to interrelated functionalities that involve the advanced exploitation of application and infrastructure monitoring data to train prominent forecasting algorithms that are able to provide time-series based predictions on critical (composite) metrics. These predictions are evaluated in real-time by our system, according to their dimensions like probability and confidence interval and analysed with respect to their potential impact on the defined Service Level Objectives (SLOs) of the cloud applications managed by our platform. These capabilities are introduced by the MORPHEMIC Forecasting module, which is analysed in the sections. A significant impact on one or more defined SLOs reveal the need to proactively start a new reconfiguration cycle that will create the necessary time space for our platform to find a new optimized configuration that will effectively cope with the predicted imminent quality of service (QoS) degradation due to expected workload fluctuation, application, network, or other infrastructure issues.

Therefore, we use as an input the monitoring data provided by the enhanced federated Event Management System (EMS) to aggregate the necessary historical records in order to calculate predictions for specific time horizons into the future. Specifically, the process flow followed along with the involved components are explained below (see also Figure 1):

i. Gather, process and propagate monitoring data (EMS);
ii. Persist monitoring data as time-series and provide the capability to automatically construct datasets appropriate for any forecasting method that should be used (Persistent Storage sub-component);
iii. Define which subset of monitoring metrics (based on the CAMEL application model) should be considered by the forecasting modules for deriving predictions (Translator);
iv. Use different prominent forecasting methods in parallel to provide predictions for the required metrics for a specific time horizon into the future, i.e., several components that implement different forecasting algorithms (TFT [4], ES-Hybrid [7], N-Beats [10], etc.);
v. Orchestrate the invocation of different forecasters in parallel per metric, aggregate predictions once ready, stop forecasters once poor results per metric are detected, filter out best predictions and/or ensemble predictions (Prediction Orchestrator);
vi. Analyse the aggregated predictions available and derive the severity of the potential SLO violation expected in case the perceived predictions will be confirmed (Severity-based SLO Violation Detector).

All the involved components are analysing in the remaining sections of this deliverable.

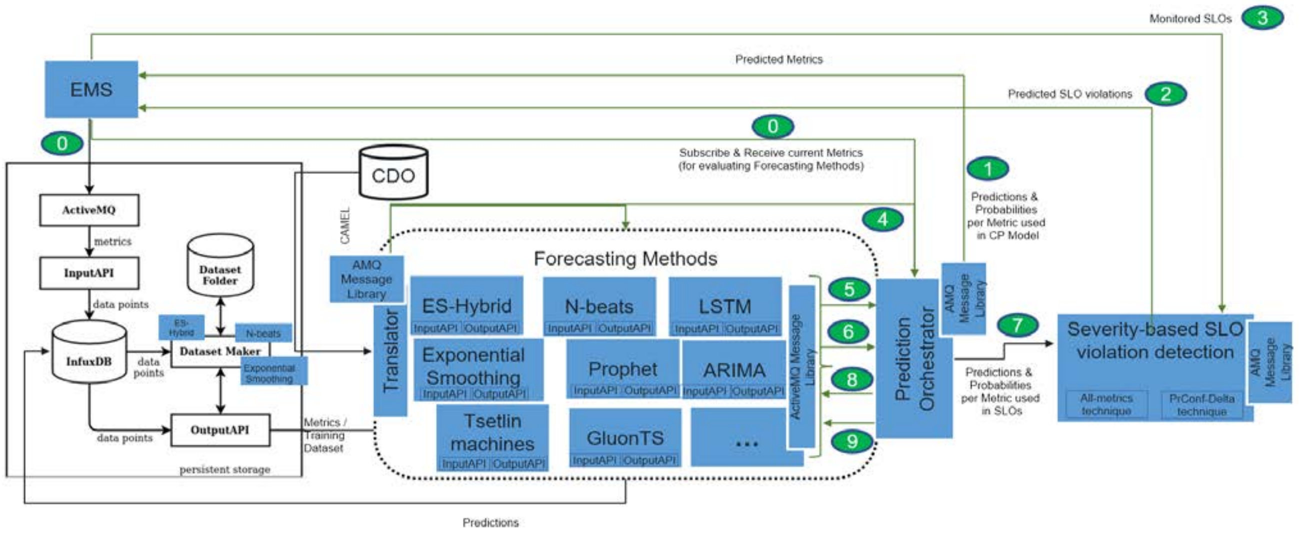


Figure 1 - Proactive adaptation architecture & flow

3 Holistic Application Monitoring System

Event Management System (EMS) is the distributed application monitoring system, used by the MELODIC and MORPHEMIC Upperware, for monitoring the operation of distributed, cross-cloud applications it deploys. EMS has the responsibility of deploying a network of agents for collecting the required monitoring information (from monitoring probes) as events, process them using complex event processing techniques, and forward the results to Upperware. The application's CAMEL model specifies the needed monitoring information and the kind of processing required.

EMS comprises of a server integrated in the Upperware, named *Event Processing Manager (EPM)*, and several clients named *Event Processing Agents (EPAs)*. EPAs are typically installed at application VMs and collocated with application components. EPM and EPAs formulate a network of nodes for distributed event processing, called Event Processing Network (EPN). This network is orchestrated and controlled by EPM. More information about EMS components and their roles can be found in deliverable D2.1 [1].

After the D2.1 submission we continued working toward improving the implementation and adding functionalities. Specifically, the following new EMS features have been implemented:

- Extension of EMS self-healing capabilities with the addition of facilities for recovering EPAs;
- Addition of EMS Administration web user interface.

3.1 EMS with Self-healing Capabilities

Self-Healing is the ability of a system to recover or repair itself without human intervention. Regarding EMS, self-healing refers to recovering or replacing monitoring system components that fail, like EPM or EPAs. To this end, the centrally-controlled EMS monitoring network of MELODIC [1] is gradually being replaced by a network of autonomously acting EMS components. Naturally, EPM will retain the responsibility of coordinating EPAs and formulating the monitoring network, but self-healing activities will be carried out by EPAs themselves or with the assistance of their nearby network nodes. For instance, an EPA might autonomously decide to restart itself in order to recover from an exceptional situation or restart an unresponsive a nearby EPA.

The steps required to complete the shift to the new paradigm of EMS monitoring network are outlined in the following Table 1.

Table 1 - EMS Self-Healing activities

Activities
Federated EMS – Local clustering of EPAs
Aggregator Selection
EPA recovery
Edge node proxy recovery
EPM recovery and/or redundancy

Federated EMS refers to the extension of EMS for partitioning EPAs into groups of nearby agents. Nearby EPAs, with the assistance of EPM, formulate clusters of peer nodes called *local clusters*. Each EPA is at the same time a cluster node and participates in a number of cluster-related operations, like checking the availability of other cluster nodes.

Aggregator builds on the notion of local cluster and introduces the role of leader node (called Aggregator), which is responsible for coordinating or carrying out self-healing actions in its local cluster. *Aggregator Selection* refers to the process of selecting one of the local cluster nodes to become leader (Aggregator). This process occurs automatically when a new local cluster is created or when current Aggregator fails or becomes unavailable.

The functionalities related to Federated EMS / Local clustering and Aggregator selection have already been implemented and are reported in detail in deliverable D2.1[1]. The EPA recovery functionalities have been added and are described in more detail in the following sections. Eventually the *EPM recovery and/or redundancy*, and *Edge node proxy recovery* will be implemented in the next period and reported in deliverable D2.4. Additionally, an extension of the Aggregator selection is also investigated in order to enable the replacement of an Aggregator based on current nodes load or other criteria beyond Aggregator failure. The latest version of EMS is available on the project's GitLab repository[1].

3.1.1 EPA recovery

EPAs are typically installed in the VMs of a deployed, multi-cloud application; thus, they are collocated with the application components running in the VMs. EPA software is a standard Java^{TM} application, which is installed when application VM is created. If for any reason EPA malfunctions or crashes it is usually enough to kill the running instance and start a new one for recovering. The new EPA instance will read the configuration and attempt to connect to EPM for receiving configuration pertaining to application monitoring as well as local clustering. In MELODIC this action was carried out manually. In the context of Morphemic, this task is part of EMS self-healing capabilities and is called *EPA recovery*.

We have designed and implemented two strategies for recovering of crashed or malfunctioning EPAs. Despite their goals are identical (i.e., the recovery of EPAs), the two strategies follow different approaches with regard to who is responsible for conducting the recovery actions. The former relies on the EPM for monitoring EPAs availability, while the latter one relies on local clusters' Aggregators for monitoring EPAs' availability. While both approaches are fast to detect an EPA loss, there are certain situations where the latter is more resilient to network disruptions and latency, since Aggregators live close to the EPAs of the same local clusters. Both recovery strategies have been implemented as EMS plugins and both take advantage of the new internal event bus framework, introduced in EPM and EPA architectures.

In order to make self-healing features more self-contained and independent from EMS core, we have introduced two new frameworks for developing EMS plugins. The former is for creating EPM plugins while the latter for EPA plugins. In fact, they can be used for developing EMS plugins for any use beyond self-healing. Such an example is the Netdata Collector plugin, which is already included in EPAs for retrieving monitoring data from the configured Netdata[2] agents.

EMS plugins must implement the *Plugin* interface, which defines the plugin start/stop methods, and also use the EMS Util library and specifically the newly added EMS internal event bus (part of Util library). Typically, EMS plugins should be activated and configured in the standard EMS configuration, although a few exceptions exist (namely the NetdataCollector plugin at EPAs). They can subscribe to certain internal event topics in order to get notified when certain events of interest occur (like connection/disconnection of an EPA to the EPM). Plugins can also use the facilities of the core EMS modules for fulfilling their goals; for instance, send monitoring events and connecting to EPA VMs.

The Plugin interface is defined as shown in the next UML class diagram:

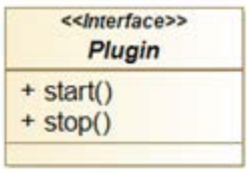


Figure 2 - UML class diagram of EMS Plugin interface

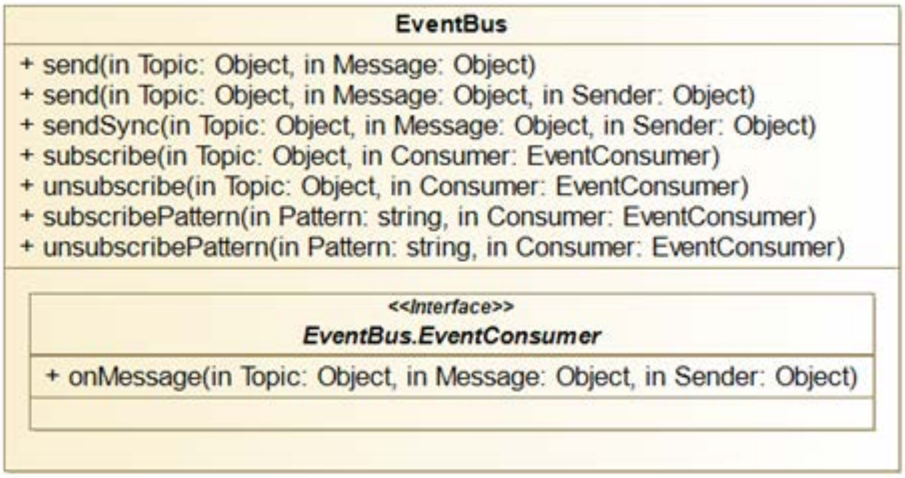
The *start()* method of all activated plugins is invoked during EPM or EPA boot time, after the core modules get initialized and before the normal execution starts. The *stop()* method is respectively invoked before EPM or EPA terminates and before the core modules are signalled to shut down.

[1] EMS in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/melodic-upperware/-/tree/morphemic-rc2.0/event-management

[2] https://www.netdata.cloud/

The internal event bus extends the EMS core with the ability to exchange internal messages, in the form of events, among EMS modules, without requiring explicit references between them. EMS modules will publish certain events at specific points in their life-cycle or when certain actions are taken. Moreover, the Baguette Server (a core module of EPM) will publish events related to EPAs, for instance when an EPA connects or disconnects. Apart from the core modules, EMS plugins can also use the internal event bus in order to publish or receive events from core modules or other plugins. The EPA recovery, self-healing plugins are particularly interested in the EPA connection/disconnection events, in order to schedule and trigger the recovery actions.

The EMS internal event bus is a lightweight bus implementation included in EMS Util package. It supports multiple topics and can convey events of any type (as long as they are Java objects). Each EPA, as well as the EPM, encompasses its own instance of internal event bus. the formal specification of internal event bus object is given in the following UML class diagram.


```
EventBus
+ send(in Topic: Object, in Message: Object)
+ send(in Topic: Object, in Message: Object, in Sender: Object)
+ sendSync(in Topic: Object, in Message: Object, in Sender: Object)
+ subscribe(in Topic: Object, in Consumer: EventConsumer)
+ unsubscribe(in Topic: Object, in Consumer: EventConsumer)
+ subscribePattern(in Pattern: string, in Consumer: EventConsumer)
+ unsubscribePattern(in Pattern: string, in Consumer: EventConsumer)

<<Interface>>
EventBus.EventConsumer
+ onMessage(in Topic: Object, in Message: Object, in Sender: Object)
```

Figure 3 - UML class diagram of Internal Event Bus

Internal event bus must not be confused with the ActiveMQ event broker of EMS Broker-CEP module, which is used for transporting the MORPHEMIC metric and prediction events from EPAs to EPM.

3.1.2 EPM-based EPA recovery

The EPM-based strategy for EPA recovery has been implemented as a new EPM plugin (ClientRecoveryPlugin class) and included in the EPM codebase. As mentioned before, EPM has been extended with an internal event bus where each core modules and plugins can publish events at certain topics. Plugins can subscribe to the necessary event topics and get notified. Based on this event bus design, the EPM-based EPA recovery plugin, receives events that signal an EPA disconnection. If this is not an expected action (i.e., EPA is scheduled for removal), the plugin will mark client as “probably lost/dead” and wait for a preconfigured period of time. If during that period EPA reconnects to EPM, a new EPA connection event is sent in the internal event bus, and the EPA recovery plugin will clear the “possibly lost/dead” mark. If the period expires then the marked EPA is considered lost/dead, and the plugin starts executing a configured EPA recovery plan. Specifically, it connects (using SSH) to the VM where the lost/dead EPA has been installed, carries out the tasks of the recovery plan, and disconnects. It subsequently marks the “recovered” EPA as “probably lost/dead” again and starts a new grace period during which the recovered EPA should boot and connect to EPM. The plugin will retry for a preconfigured number of times to recover EPA before it gives up. In this first version of ClientRecoveryPlugin self-healing plugin, we have configured a simple recovery plan that targets to Linux-based VMs and involves killing any existing instance of EPA and launching a new one. It is important to mention that the grace period must provide enough time to EPA to launch and connect to EPM.

3.1.3 Aggregator-based EPA recovery

The Aggregator-based EPA recovery strategy is similar to the EPM-based one but in this case the local cluster Aggregators are responsible for EPA recovery, instead of EPM. This strategy has been implemented as an EPA plugin

(SelfHealingPlugin class) and included in EPA codebase. Like EPM, EPAs have also been extended with an internal event bus. The plugin subscribes for events reporting removal of local cluster nodes (i.e., nearby EPAs). There are a few differences, however. First, the plugin is activated only when an EPA is the Aggregator of the local cluster, otherwise it stays idle (ignoring any incoming events). Second, the Aggregator detects and recovers only the EPAs living in the same local cluster. Eventually, the detection of disconnected EPAs is based on the facilities provided by the clustering library (currently Atomix[3]). The reason for using the clustering facilities is that there are no SSH control connections to Aggregator as it is the case with EPM. Therefore, the approach used for detecting lost/dead EPAs is based on detecting the removal of cluster nodes (which implies a VM loss or a network disruption). The clustering library uses the SWIM protocol [2] for detecting lost cluster nodes.

Aggregator-based EPA recovery is the default strategy for recovering EPAs.

3.1.4 Partitionable approach[4]

As it has been detailed in deliverable D2.1 (section 4.2.3 [1]) EPAs installed at nearby nodes are grouped together and formulate clusters (called local clusters), in order to quickly detect node failures in the cluster that can lead to monitoring capacity degradations (or interruptions), and they take certain recovery actions aiming at compensating the problem. For the creation of the local clusters, EPAs use the Atomix framework. They can be configured to use either the Raft or the Primary-Backup replication protocol, offered by Atomix. Moreover, it is possible to configure the number of partition groups used in the cluster (in EPAs' configuration files). Partition groups are sets of cluster nodes using a certain replication protocol for replicating the shared data. The default configuration of EPAs specifies one partition group using Raft for cluster management data, and one partition group using Primary-Backup for user data.

A focal concept in the EMS monitoring topology is that of Aggregators, which are EPAs with the extra responsibility of (a) collecting and processing monitoring information from their nearby EPAs (thus providing distributed processing of the monitoring data), and (b) carrying out the corrective actions when a cluster node fails (thus offering self-healing to the local cluster). The EPAs in a local cluster are configured (through a distributed selection protocol) to send their monitoring data to the cluster Aggregator.

However, when the Aggregator fails (i.e., becomes inaccessible) the EPAs of the cluster will quickly detect that fact, and select another node to become Aggregator. In the case where the nodes of a local cluster are divided into two (or more) partitions, it will happen that one partition has an Aggregator node while the other(s) do not. The nodes of the Aggregator-less partition will detect that original Aggregator has "failed" and will automatically select a new node (of the partition) to become Aggregator. Therefore, cluster will end up with two working partitions (as if we had two local clusters from the beginning).

In the case where partitions join again there will be one cluster (since all nodes will observe all other nodes), but in terms of monitoring there will be two Aggregators, and some nodes will push their monitoring data to one Aggregator, while other nodes to the other. In order to avoid this fragmentation, we plan to add a "guarding" feature in EPAs that will check for the existence of more than one Aggregator in the cluster and start Aggregator selection process in such case. Alternatively, the guarding feature can ask one of the Aggregators to retire, or one of the Aggregators can retire by itself, as soon as it finds out there is another Aggregator in the cluster with a better selection score. The remaining Aggregator will re-send configuration instructions to all cluster nodes to push their monitoring data to it.

EPAs' ability to select an Aggregator within a partition ensures full functionality in terms of monitoring. We believe the combination of this capability (i.e., having a functional partition) and the addition of the "guarding" feature that will prevent cluster fragmentation (in terms of monitoring), in the case of partition healing, will sufficiently address the consideration for a partitionable approach.

3.1.5 Dynamic aggregator election and potential performance issues[5]

In the current implementation of local clusters in EPAs, the Aggregator role is not movable, unless the node fails. In this case the Aggregator role will be assigned to another node in the cluster, picked based on a scoring function (that

[3] https://atomix.io/

[4] This section also refers to the recommendation n.8 from the MORPHEMIC Intermediate Review Report.

[5] This section also refers to recommendation n.9 from the MORPHEMIC Intermediate Review Report.

will typically consider the node resources). Still, it is possible to manually move Aggregator to another node, but we consider this as a special Developers or Administrator feature, and hence out of the scope of MORPHEMIC for autonomic management of the monitoring infrastructure.

In the future, we plan to introduce the possibility to move Aggregator role based on the load of the hosting node as well as the available resources of other nearby nodes. We will take care to prevent too frequent Aggregator moves, for instance by introducing a "cooling period" between moves, or by using strict criteria with regard to the node's current load and available resources, for moving Aggregator. Another approach would be the EMS server to assume the Aggregator role for a cluster, when there is no node capable of becoming Aggregator, or they seem to fail quite frequently. Of course, this would be a last resort since it would render local clustering meaningless for those nodes.

3.1.6 Issues by keeping monitoring data in memory[6]

It is currently possible to adjust the amount of memory allocated by EPAs in two ways; (a) setting the amount of memory the embedded CEP engine will reserve (it can be a value in bytes or a percentage of JVM's heap; current default is 20% of JVM heap), and (b) specifying the system memory the JVM will allocate, at JVM launch command (currently JVM default is used).

Apart from the previous approaches, we will extend the Broker-CEP module of EPAs to impose a limit in the number of events cached in-memory, while waiting to be sent to Aggregator or EMS server. Moreover, we can configure ActiveMQ (the embedded event broker of EPAs) to persist events in a database (KahaDB), instead of caching them in-memory. Additionally, if use case scenarios demand it, we can design an event drop-off mechanism that will eliminate events based on certain criteria (like age), when event number grows beyond a specified limit. However, such a feature must be engaged and configured by the application designer who has knowledge of which events are important for the application monitoring (and hence cannot be dropped) and which can be safely ignored.

3.2 EMS at the Edge

A notable advancement of MORPHEMIC compared to its predecessor (Melodic) is its ability to deploy multi-cloud application components to a variety of target nodes (beyond VMs) that include Edge devices. For this reason, EMS needs to be adapted accordingly (work in progress to be reported in D2.4). Edge devices pose certain constraints in installing, configuring and running EPAs, due to limited resources but due to other reasons as well (e.g., specific networking requirements, non-standard ways of collecting monitoring data, or restraint environment with regard to which tools can be deployed).

The approach chosen for including Edge nodes in EMS monitoring scheme, is to avoid installing EPAs on them. Instead, EPAs running on VMs close to the Edge will be assigned with the task of collecting, processing and propagating the monitoring data of Edge nodes. We need to distinguish between two categories of Edge devices:

- those where it is possible to deploy and run a Netdata agent, and
- those where it is not possible to deploy a Netdata agent.

Netdata is a well-known tool for real-time monitoring of various (system and application) metrics. EPAs use Netdata as a monitoring probe for collecting the required metrics and further process and propagate them.

The former category encompasses Edge devices where Netdata agent is preinstalled as well as devices where EPM can connect and install a Netdata agent. EPM requires an SSH connection to the target device and a Unix-style shell (like bash), for installing and launching Netdata agent. This category of Edge devices has already been tested with Raspberry Pi 3 units.

The latter category includes devices where Netdata cannot be used. In this case only the provided means of monitoring data collection can be used. For this reason, it is required to develop device-specific data collection EPA plugins (like the Netdata Collector plugin bundled in EPA codebase).

[6] This section also refers to recommendation n.10 from the MORPHEMIC Intermediate Review Report.

EMS self-healing also needs to be able to handle failures related to the Edge devices. This involves reassigning the collection of the monitoring data of an Edge device to another EPA when the initial one fails or gets overloaded.

3.3 Event Structure for Predictions

In order to facilitate the communication between the components participating in the forecasting mechanism of Morphemic, a number of new event types have been introduced along with the corresponding event payloads. These new event types are specific to the forecasting mechanism and are used for signalling and coordination. The event payloads are represented in JSON. More information about forecasting mechanism is provided in Section 6.

The following tables detail the payload fields for each event type. The first table details the payload of the metric event (as it has been defined in the context of MELODIC project and is used in MORPHEMIC too). The numbering used refers to the expected process flow presented in figure 1.

Table 2 - Current Metrics & SLO Event fields

[0] Current Metrics & SLO Event fields		
Topic	[metric_name]	
metricValue	Double	Actual metric value
level	Integer	The level of EPA where metric captured or computed
timestamp	Long	Event creation date/time from epoch (Coordinated Universal Time - UTC is considered in all timestamps)
refersTo	String	The id of the application or component or (VM) host for which the prediction refers to
cloud	String	Cloud provider of the VM (with location)
provider	String	Cloud provider name

Next, we provide an example event:

Listing 1 - Example Event [0]

```
{
    "metricValue": 12.34,
    "level": 1,
    "timestamp": 143532341251,
    "refersTo": "MySQL_12345",
    "cloud": "AWS-Dublin",
    "provider": "AWS"
}
```

Table 3 - Predicted Metrics Event fields

[1] Predicted Metrics Event fields	
Topic	prediction.[metric_name]

metricValue	Double	Predicted metric value
level	Integer	Level of EPA where prediction occurred or refers
timestamp	Long	Prediction creation date/time from epoch (Coordinated Universal Time - UTC is considered in all timestamps)
probability	Double	Probability of the predicted metric value (range 0..1)
confidence_interval	Double[2]	The probability-confidence interval for the prediction
predictionTime	Long	This refers to time point in the imminent future (that is relative to the time that is needed for reconfiguration) for which the predicted value is considered valid/accurate (in UNIX Epoch)
refersTo	String	The id of the application or component or (VM) host for which the prediction refers to
cloud	String	Cloud provider of the VM (with location)
provider	String	Cloud provider name

Next, we provide an example event:

Listing 2 - Example Event [1]

```
{
    "metricValue": 12.34,
    "level": 1,
    "timestamp": 143532341251,
    "probability": 0.98,
    "confidence_interval" : [8,15]
    "predictionTime": 143532342,
    "refersTo": "MySQL_12345",
    "cloud": "AWS-Dublin",
    "provider": "AWS"
}
```

Table 4 – Monitored SLOs Event fields

[2] Monitored SLOs Event		
Topic	metric.metric_list	
name	String	A string identifier of the SLO. Each SLO can contain multiple sub-SLOs, in which case the event contains the 'constraints' field and does not contain the 'metric' and 'threshold' fields
operator	String	One of the ">","<",">=" or "<=" comparison operators (when the field is present in 'simple' Event 2 JSON objects) or one of the 'AND' or 'OR' logical operators.
constraints	JSON[]	This field only exists when one or more sub-SLOs are defined for this SLO (i.e., in 'complex' Event 2 JSON objects). The value of this field is a JSON array consisting of 'simple' Event 2 JSON objects (without any sub-SLOs)
metric	String	The name of the monitoring metric for which the SLO is defined (this field is only present in 'simple' Event 2 JSON objects)
threshold	Double	The numerical value which sets the acceptable threshold for this monitoring metric (this field is only present in 'simple' Event 2 JSON objects)

Next, we provide an example event:

Listing 3 - Example Event [2]

```
{
  "name": "_",
  "operator":"OR",
  "constraints":[
    {"name":"cpu_and_memory_or_disk_too_high",
      "operator":"AND",
      "constraints": [
        {
          "name":"cpu_usage_high",
          "metric":"cpu_usage",
          "operator":">",
          "threshold":70.0
        },
        {
          "name": "memory_or_disk_usage_high",
          "operator": "OR",
          "constraints": [
            {
              "name":"memory_usage_high",
              "metric":"memory",
              "operator":">",
              "threshold":70.0
            },
            {
              "name": "disk_usage_high",
              "metric":"disk",
              "operator":">",
              "threshold":95.0
            }
          ]
        }
      ]
    }
  ]
}
```

Table 5 – Predicted SLOs Event fields

[3] Predicted SLOs Event fields		
Topic	prediction.slo_severity_value	
severity	Double	The severity value which is associated to a possible violation of an SLO
probability	Double	Probability of reconfiguration being required (range 0..1)
predictionTime	Long	Refers to time point in the imminent future (that is relative to the time needed for reconfiguration) for which the severity value is calculated (since UNIX Epoch)

Next, we provide an example event:

Listing 4 - Example Event [3]

```
{
    "severity": 0.9064,
    "probability": 0.92246521,
    "predictionTime": 1435323424
}
```

Table 6 – Translator–to–Forecasting Methods/Prediction Orchestrator Event fields

[4] Translator-to-Forecasting Methods/Prediction Orchestrator Event fields		
Topic	metrics_to_predict	
Array of:		
metric	String	Name of the metric to predict
level	Integer	Level of EPA where this metric may be produced/found
publish_rate	Long	Expected rate for datapoints regarding the specific metric (according to CAMEL)

Next, we provide an example event:

Listing 5 - Example Event [4]

```
[{
    "metric": "MaxCPULoad",
    "level": 3,
    "publish_rate": 60000,
},
{
    "metric": "MinCPULoad",
    "level": 3,
    "publish_rate": 50000,
}]
```

Table 7 – Forecasting Methods–to–Prediction Orchestrator Event fields

[5] Forecasting Methods-to-Prediction Orchestrator Event fields		
Topic	training_models	
metrics	Strings[]	Metrics for which a certain forecasting method has successfully trained or re-trained its model
forecasting_method	String	The method that is currently re-training its models
timestamp	Long	Date/time of model(s) (re-)training

Next, we provide an example event:

Listing 6 - Example Event [5]

```
{
    "metrics": ["MaxCPULoad","MinCPULoad"],
    "forecasting_method": "ESHybrid",
    "timestamp": 143532341251,
}
```

Table 8 – Forecasting Methods–to–Prediction Orchestrator Event fields

[6] Forecasting Methods-to-Prediction Orchestrator Event fields		
Topic	intermediate_prediction.[forecasting_method].[metric_name]	
metricValue	Double	Predicted metric value (more than one such events will be produced for different time points into the future)
Level	Integer	Level of EPA where prediction occurred or refers
timestamp	Long	Prediction creation date/time from epoch (Coordinated Universal Time – UTC is considered in all timestamps)
probability	Double	Probability of the predicted metric value (range 0..1)
confidence_interval	Double[2]	The probability-confidence interval for the prediction

predictionTime	Long	This refers to time point in the imminent future (that is relative to the time that is needed for reconfiguration) for which the predicted value is considered valid/accurate (in UNIX Epoch)
refersTo	String	The id of the application or component or (VM) host for which the prediction refers to
cloud	String	Cloud provider of the VM (with location)
provider	String	Cloud provider name

Next, we provide an example event:

Listing 7 - Example Event [6]

```
{
    "metricValue": 12.34,
    "level": 3,
    "timestamp": 143532341251,
    "probability": 0.98,
    "confidence_interval" : [8,15]
    "predictionTime": 143532342,
    "refersTo": "MySQL_12345",
    "cloud": "AWS-Dublin",
    "provider": "AWS"
}
```

Table 9 – Prediction Orchestrator–to–Severity-based SLO Violation Detector Event fields

[7] Prediction Orchestrator-to-Severity-based SLO Violation Detector Event fields		
Topic	prediction.[metric_name]	
metricValue	Double	Predicted metric value
level	Integer	Level of EPA where prediction occurred or refers
timestamp	Long	Prediction creation date/time from epoch (Coordinated Universal Time – UTC is considered in all timestamps)
probability	Double	Probability of the predicted metric value (range 0..1)
confidence_interval	Double[2]	The probability-confidence interval for the prediction
predictionTime	Long	This refers to time point in the imminent future (that is relative to the time that is needed for reconfiguration) for which the predicted value is considered valid/accurate (in UNIX Epoch)
refersTo	String	The id of the application or component or (VM) host for which the prediction refers to

cloud	String	Cloud provider of the VM (with location)
provider	String	Cloud provider name

Next, we provide an example event:

Listing 8 - Example Event [7]

```
{
    "metricValue": 12.34,
    "level": 1,
    "timestamp": 143532341251,
    "probability": 0.98,
    "confidence_interval" : [8,15]
    "predictionTime": 143532342,
    "refersTo": "MySQL_12345",
    "cloud": "AWS-Dublin",
    "provider": "AWS"
}
```

Table 10 – Prediction Orchestrator–to–Forecasting Methods Event fields [8]

[8] Prediction Orchestrator-to-Forecasting Methods Event fields		
Topic	stop_forecasting.[forecasting_method]	
metric	Strings[]	Metrics for which a certain method should stop producing predictions (because of poor results)
timestamp	Long	Date/time of the command of the orchestrator

Next, we provide an example event:

Listing 9 - Example Event [8]

```
{
    "metrics": ["MaxCPULoad","MinCPULoad"],
    "timestamp": 143532341251,
}
```

Table 11 – Prediction Orchestrator–to–Forecasting Methods Event fields [9]

[9] Prediction Orchestrator-to-Forecasting Methods Event fields		
Topic	start_forecasting.[forecasting_method]	
Metrics	String[]	Metrics for which a certain method should start producing predictions
Timestamp	Long	Date/time of the command of the orchestrator
epoch_start	Long	This time refers to the start time after which all predictions will be considered (i.e., t0)
number_of_forward_ predictions	Integer	This is a number that indicates how many time points into the future do we need predictions for
prediction_horizon	Long	This time equals to the time (in seconds) that is needed for the platform to implement an application reconfiguration (i.e., TR).

Next, we provide an example event:

Listing 10 - Example Event [9]

```
{
    "metrics": ["MaxCPULoad","MinCPULoad"],
    "timestamp": 143532341251,
    "epoch_start": 143532341252,
    "number_of_forward_predictions": 5,
    "prediction_horizon": 600
}
```

3.4 EMS Administration Web GUI

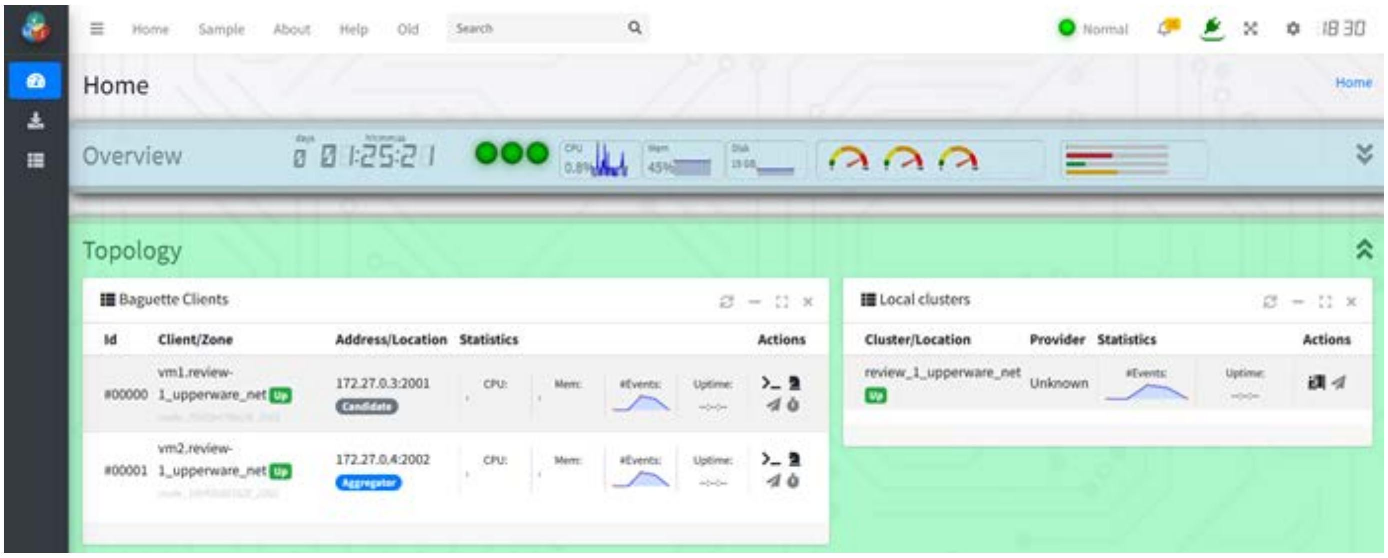
A new EMS feature, introduced in the context of Morphemic, is the addition of a web user interface for monitoring the operational status of EMS (both EPM and EPAs) as well as for carrying out certain administrative tasks. Before this feature, almost all tasks related to EMS (including monitoring, investigating problems, debugging etc.) had to be done from the command line. This required connecting to the VMs where EPM or EPAs were deployed which proved to be a quite tedious, time consuming and error prone approach. The new EMS Admin Web GUI aspires to solve several of these issues and make some of the most common EMS monitoring and administration significantly faster and easier.

EMS Admin Web GUI has been implemented as a set of Single-Page Applications (SPA), using the well-known VueJS[7] framework, as well as several Node plugins. The core technologies involved in its implementation include JavaScript, NodeJS (for server-side JavaScript), NPM and VueJS framework.

The advent of EMS Admin Web GUI required the extension of all EMS components (namely EPM and EPAs), as well as the control protocol, in order to collect and gather monitoring information. This information currently includes the system load where an EMS component runs (CPU, Memory, Disk space), the number of application metric events exchanged, the event topics, the locations of EPAs and their operational status. Moreover, the EMS REST API has been extended and is now possible to contact EPM and get EMS monitoring info. Additionally, EPM has been enhanced with Server-Side Events capability for providing a continuous stream of updates about the EMS status. EMS Admin Web GUI uses the server-side events approach for updating the displayed information.

[7] https://vuejs.org/

Below, a few screenshots of the EMS Admin Web GUI are given.



Baguette Clients

Id	Client/Zone	Address/Location	Statistics				Actions
#00000	vm1.review-1_upperware_net Up	172.27.0.3:2001 Candidate	CPU:	Mem:	#Events:	Uptime: --:--:--	
#00001	vm2.review-1_upperware_net Up	172.27.0.4:2002 Aggregator	CPU:	Mem:	#Events:	Uptime: --:--:--	

Local clusters

Cluster/Location	Provider	Statistics		Actions
review_1_upperware_net Up	Unknown	#Events:	Uptime: --:--:--	

Figure 4 - EMS Admin Web GUI screenshots of EMS network info section (1/2)

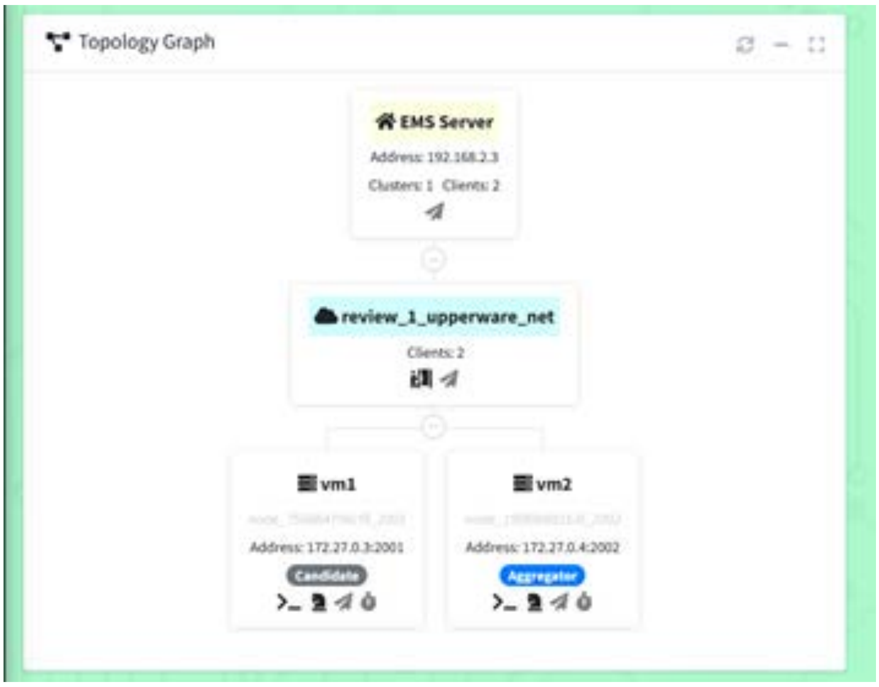


Figure 5 - EMS Admin Web GUI screenshots of EMS network info section (2/2)

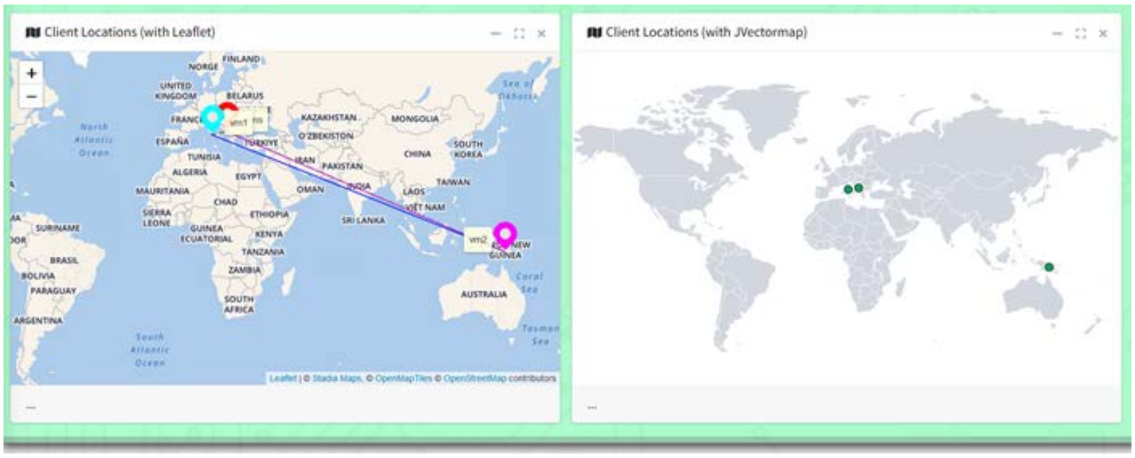
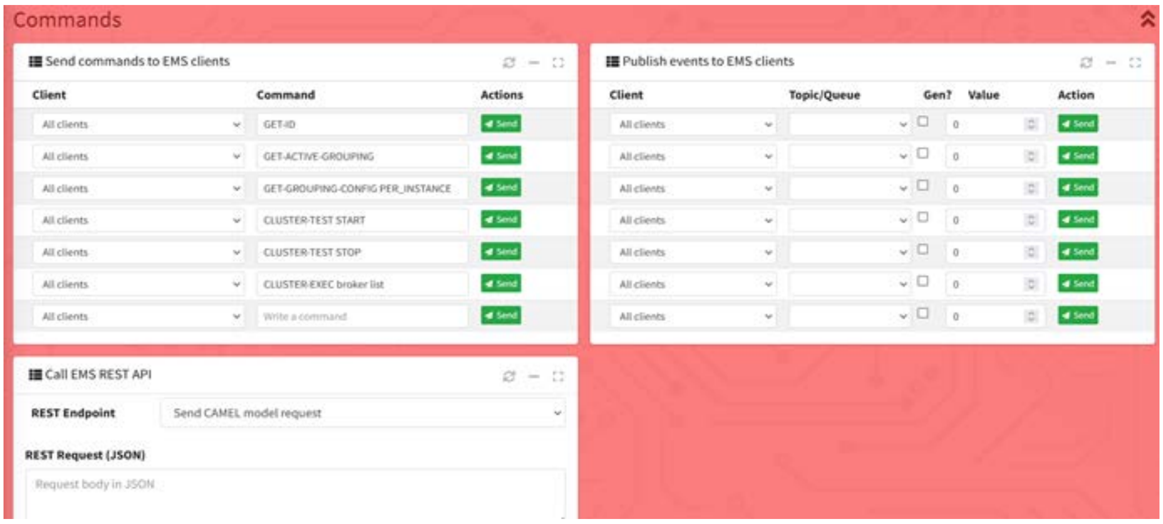

Client Locations (with Leaflet)
Client Locations (with JVectormap)
vm1
vm2

Figure 6 - EMS Admin Web GUI screenshot with EPM and EPA locations


Commands

Send commands to EMS clients

Client	Command	Actions
All clients	GET-ID	Send
All clients	GET-ACTIVE-GROUPING	Send
All clients	GET-GROUPING-CONFIG PER_INSTANCE	Send
All clients	CLUSTER-TEST START	Send
All clients	CLUSTER-TEST STOP	Send
All clients	CLUSTER-EXEC broker list	Send
All clients	Write a command	Send

Publish events to EMS clients

Client	Topic/Queue	Gen?	Value	Action
All clients		☐	0	Send
All clients		☐	0	Send
All clients		☐	0	Send
All clients		☐	0	Send
All clients		☐	0	Send
All clients		☐	0	Send
All clients		☐	0	Send

Call EMS REST API

REST Endpoint: Send CAMEL model request

REST Request (JSON): Request body in JSON

Figure 7 - EMS Admin Web GUI screenshot of Commands section

4 Persistent Storage

Monitoring a Cloud/Edge application consists of collecting Key Performance Indicators (KPI) and any other kind of metrics for analysis or reconfiguration triggering. Therefore, the implementation of a holistic monitoring system requires a pipeline composed of a sensor (device or software for capturing metrics), monitoring collector (software that aggregates metrics), a storing system (time-series or a database system) and a set of tools for analysis and evaluation such as a QoS evaluator, forecasters etc.

There is a need to establish a permanent connection with the Event Management System (EMS) for consuming metrics in real-time. The details on the type of connection and data exchanged between the persistent storage and the EMS can be found in the deliverable D2.1 [1].

The persistent storage as part of this pipeline, is the storing engine where all metrics exposed by applications running on MORPHEMIC are stored. The persistent storage has the capability of grouping metrics by application, thus easing and speeding up any retrieval operations. Metrics stored are disposed of as a dataset to different forecasters as shown in the section 2.1.

In the context of the MORPHEMIC project, a library (dataset maker) has been developed for ensuring the transformation of metrics to datasets for different analytics operations.

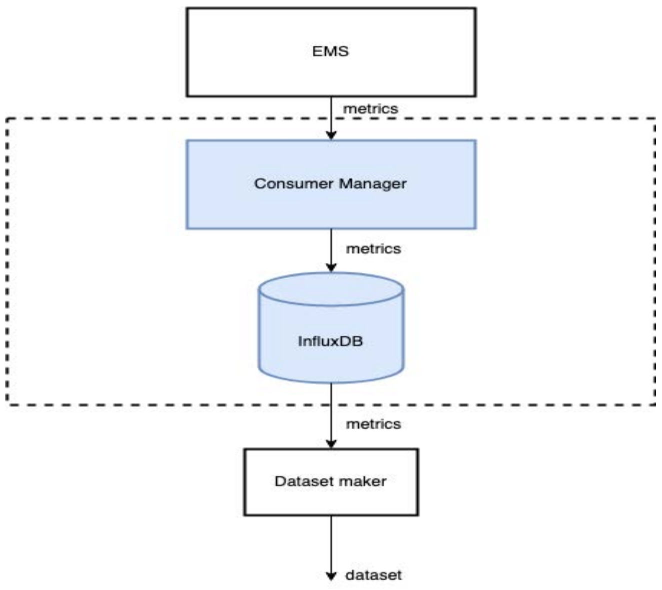


Figure 8 - Persistent storage internal architecture and flow

From the above figure, we can observe the connection between the dataset maker library invoked by the forecasters and InfluxDB. The following parameters are required for generating a dataset.

Table 12 – Dataset Generation / Configuration Details

Dataset Generation / Configuration Details	
port :	InfluxDB port 8086
username:	influxdb username
password:	influxdb password
dbname:	influxdb database name
path_dataset:	path where the dataset will be stored

5 Translator

Translator is the Proactive Adaptation’s architecture component, responsible for analysing the application CAMEL model and deducing the metrics and variables for which it is necessary to have predictions. It specifically focuses to the Service Level Objectives (SLOs) and the metric variables contained in the CAMEL model (specifically in the Utility Function), which it parses and extracts the metrics that comprise them. Subsequently, translator compiles the list of metrics that need to have predictions for them. Moreover, it generates the syntactic trees of all SLOs, necessary to the Severity-based SLO violation detector.

The extracted information (metrics list and SLO syntactic trees) is used to create special purpose events intended for use by specific Proactive Adaption architecture components; (a) Translator–to–Forecasting Methods/Prediction Orchestrator Event that contains the list of metrics for which predictions are required, and (b) Monitored SLO Event, which contains the SLO syntactic trees, and is needed in the Severity-based SLO violation detector. For more information on these events please refer to the corresponding tables in section 3.3.

6 Forecasting algorithms

6.1 Introduction

In this section, we discuss different prominent forecasting methods that have been implemented as separate components to enable MORPHEMIC's proactive adaptation support. They can be used either separately or in parallel according to the instructions of the Prediction Orchestrator to provide predictions for the required metrics for a specific time horizon into the future. The MORPHEMIC approach is open to plug in any additional implementations of forecasting algorithms that may be better with respect to datasets at hand. Currently, MORPHEMIC platform has analysed and can currently exploit the following forecasting algorithms:

- ES-Hybrid,
- N-Beats
- TFT
- Prophet
- Sarima
- GluonTS
- Exponential Smoothing

In the next sections, we provide a quick overview for each of these algorithms, their implementation details and their illustrative use and evaluation using two datasets from a Genome cloud application. This application involves a data parallel training of genome models, using Spark. The datasets which were generated contain observations for twelve monitoring metrics, collected at 30-second intervals. The datasets are available online[8]. As it is not appropriate to use only a single forecasting error metric to evaluate the quality of predictions [3], a number of well-known forecasting metrics were used: i) mean absolute error (MAE) [4] ; ii) mean squared error (MSE) - squared value of the RMSE [4]); iii) mean absolute percentage error (MAPE) [5] and iv) symmetric mean absolute percentage error (SMAPE) [6]. We note that in the results tables presented for each of the forecasters below we consider: i) the value *0* in cases where prediction for a certain metric was 100% accurate; ii) the value *inf* when the result of a KPI involves the division by zero (e.g., in SMAPE); and iii) the value - in case the dataset didn't involve adequate data for the forecaster to produce a valid prediction.

6.2 ES-Hybrid

6.2.1 Overview

The ES-Hybrid forecaster was built on top of Smyl's *hybrid* Exponential **S**moothing – Recurrent Neural Networks (ES-RNN) method[9] [7] which was the winner of the M4 competition [8]. This forecasting algorithm is still widely used and has been extended to include GPU specific implementations [9] which aim to provide a solution to the algorithms' performance, which is one of the major drawbacks of using this method.

The forecasting methods' main advantage is its effective mix of an exponential smoothing model with that of an LSTM network, and through this process the forecasts are more accurate than those of their singular counterparts, either purely statistical or ML approaches.

There are three main steps of the algorithm

(i) Deseasonalization
(ii) Generation of forecasts
(iii) Ensembling

[8] Datasets in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/time-series-data/-/tree/time-series-experiments/time-series-data/benchmark_datasets/genome/deployment-reconfiguration-range-1-to-1

[9] ES-RNN implementation: https://hub.docker.com/repository/docker/imuntua/esrun

and there are quite a few implementations in different programming languages, each with their pros and cons. Initially we compiled and re-used the original C++ implementation. Whilst this implementation worked as expected regarding the M4 data as inputs, it did not provide the necessary flexibility to be used in an adaptable software component, nor could it be used to alternative datasets.

We further examined other implementations, mainly in the python programming language in order to leverage the available ML and data handling libraries such as PyTorch[10] and Pandas[11] whilst at the same time allowing us to extend the implementation to enable us to create an adaptable component. We tested four different implementations and ended up extending and reusing the following PyTorch implementation[12].

6.2.2 Implementation Details

The forecaster is made up of four (4) main components which are used by the main *ESHybrid* controller. The architecture of the component is based on an asynchronous model. Using a call back methodology, each module defines a set of *handler* methods, which can be *plugged-in* by different implementations. The *ESHybrid* controller acts as the main *handler* implementation for each sub-module hardwiring all the logic and focusing only on the components' lifecycle.

The implementation can be found here[13]

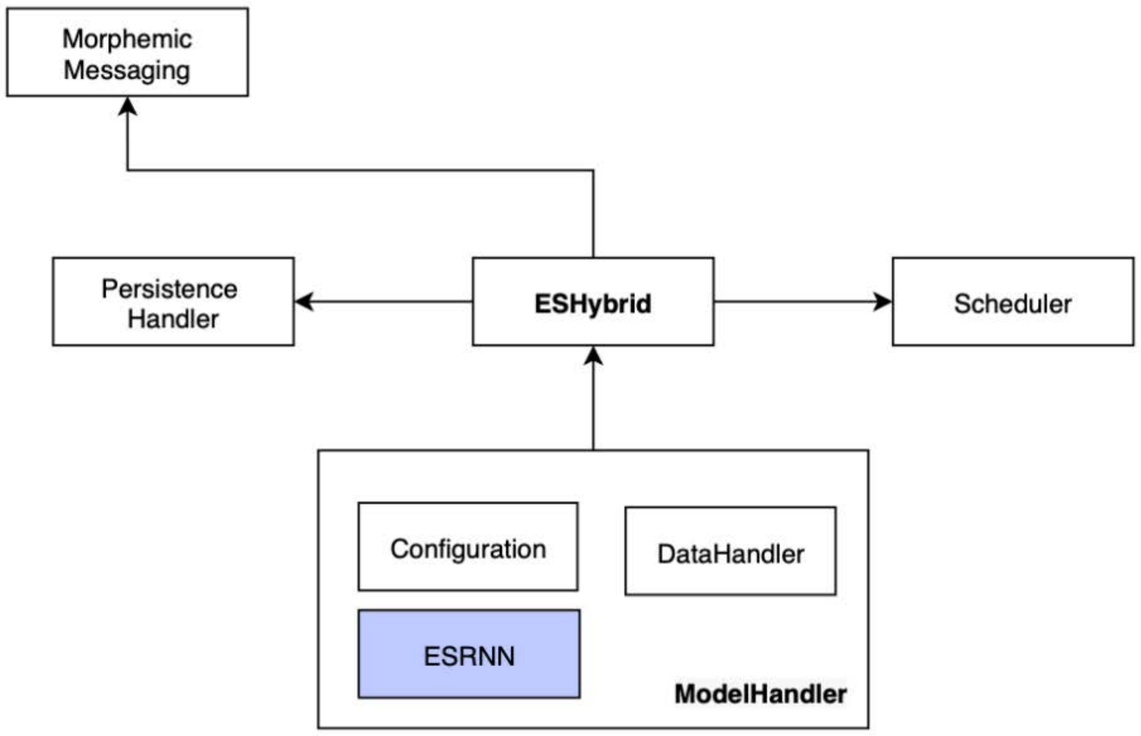


Figure 9 - ES-Hybrid Component Architecture – new sub-components denoted with white background colour

MORPHEMIC Messaging

[10] https://pytorch.org

[11] https://pandas.pydata.org

[12] https://github.com/kdgutier/esrnn_torch

[13] https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/morphemic-forecasting-eshybrid

Due to the message-based architecture of the platform, we implemented a wrapper module around stomp.py[14] to handle the necessary boilerplate required when creating a message enabled component. This provides helpers to connect to the message bus, handle named messages and provide static definition of all the topic / queue names used throughout in avoid naming issue common to this type of architecture. Furthermore, the library is self-contained and can be reused by other components using the python programming language.

Persistence Handler

This python module encapsulates the necessary logic to connect and manage data accessed through the dataset maker component (of the Persistent Storage).

Scheduler

This python module implements the necessary logic to generate the forecast timestamps based on the parameters defined in the *start_forecasting* message. Furthermore, it provides a callback handler which allows the component to determine when the forecasting should occur based on the *epoch* and *time window* defined in the *start_forecasting* message. This module is also self-contained and can be reused by other components using the python programming language.

Model Handler

One of the main requirements of the ES-Hybrid component is to be able to provide forecasting metrics as well as handle an up to date internal model based on a constantly updated stream of metric data, and as such it needs to be able to handle forecasting, and (re-)training in an asynchronous manner.

In order aid in the testing of the component we implemented a *Configuration* module as well as a *DataHandler* which allowed us to *plug-in* different implementations until reaching the final version found in the repository. Through the configuration component we are able to fine-tune the ESRNN implementation based on the requirements of the data stream.

The *DataHandler* transform the data by performing upscaling using linear interpolation up to one (1) second, such that the data used to train the model can come into the model in any variation of time. Any *n* second interval is upscaled to 1 second, providing the maximum flexibility for the forecasting timestamps.

The component is initially configured to compute 3600 future predictions, defined as the training output size. This means that it can provide results for at least one hour without receiving new data. Upon *start_forecasting* we can use the *horizon* as well as the *number_of_metrics*, in order to generate shorter output windows. For example, if we are required to provide 5 predictions for every 30 seconds, then we can set the output_size to be 10 multiples for 150seconds (since we are upscaling to 1s), this means that we will be able to maintain this model 1500seconds at which point we will need to retrain the model. The number of future predictions, which can be updated at *runtime* per new model, per retrain, determines for how long the model can be considered "active".

Once the data is properly upscaled we then initiate the model training by using the *configurable* ESRNN implementation in its own thread, by controlling its status, and thus protecting multiple training threads or obsolete trained models.

The ESHybrid component can be run as a standalone binary, however we also provided docker container. Runtime configuration can be provided by using a configuration file which can be determined through the ***–config*** parameter when running as a standalone binary.

When using the docker container, configuration can be specified using volume mounts, and mounting a local configuration file to the ***/app/sync.cfg*** mount point.

A sample config file is described below.

[14] https://pypi.org/project/stomp.py/

```
[persistence]
application=default_application
host=localhost
port=8086
username=morphemic
password=password
dbname=morphemic
path_dataset=/tmp/out/dataset-maker

[messaging]
host=localhost
port=61610
username=admin
password=admin

 [listener]
id=eshybrid
```

6.2.3 Illustrative use

To use the ESHybrid forecasting module, you can either run it as part of the MORPHEMIC platform or by implementing a handler which triggers the necessary functionality in the main controller.

We evaluated the forecaster different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 18 dataset respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 13 - ESHybrid forecaster performance on the Genome 12 dataset (Genome 12/13-2-2021)

	mae	**mse**	**mape**	**smape**
ETPercentile	3.114145566339	26.91982698733397	0.042417414324241	0.038991378668430
NotFinishedOn TimeContext	474.9234089611	403488.767200348	0.007754527087366	0.007701933421809
RemainingSimulation TimeMetric	-	-	-	-
NotFinished	474.9234089611	403488.767200348	0.007754527087366	0.007701933421809
WillFinishTooSoon Context	404.4992294520	312580.3272745923	0.008214468570769	0.008151504989457
NotFinishedOnTime	474.9234089611	403488.767200348	0.007754527087366	0.007701933421809
EstimatedRemainingv TimeContext	564.3234712783	550749.442206106	-	0.384676090693028
SimulationLeft Number	50.37160481492	4505.705204639416	-	0.361061191811129
TotalCores	0.000176144789	5.046087612295071 5e-08	0.000176144789343	0.005778379615034

MinimumCores	-	-	-	-
SimulationElapsed Time	366.7306827910	177498.9836242536	0.005800106120270	0.005778379615034
MinimumCores Context	-	-	-	-

Table 14 - ESHybrid forecaster performance on the Genome 18 dataset (Genome 20/21-2-2021)

	mae	mse	mape	smape
ETPercentile	4.677438285482	33.64481534186169	0.065609979905374	0.061811692858101
NotFinishedOn TimeContext	1101.887562814	1637167.922918425	0.019604583343204	0.019350190672584
RemainingSimulation TimeMetric	-	-	-	-
NotFinished	1101.887562814	1637167.922918425	0.019604583343204	0.019350190672584
WillFinishTooSoon Context	1049.276542870	1491155.258001562	0.023105869663074	0.022750339859257
NotFinishedOnTime	1101.887562814	1637167.922918425	0.019604583343204	0.019350190672584
EstimatedRemaining TimeContext	1298.736902284	2331138.431292067	-	0.456256100849264
SimulationLeft Number	58.63249904306	5118.422976705372	-	0.407208573625380
TotalCores	0.000173457902	4.763212257673226 e-08	0.000173457902879	0.000173436702077
MinimumCores	-	-	-	-
SimulationElapsed Time	331.1279247801	143902.9222342333	0.005753468996954	0.005732207497522
MinimumCores Context	-	-	-	-

6.3 N-beats

6.3.1 Overview

N-Beats [10] is optimized for solving the univariate times series point forecasting problem. It is a fully deep neural architecture based on backward and forward residual links and a very deep stack of fully-connected layers. The architecture has a number of desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train. N-Beats outperformed winner of M4[8] competition. It means that it achieves the best scores on 100k series from different domains.

Pros:

- Achieves good results without modification to a wide array of target domains
- Relatively fast (comparing to other deep neural networks)

Cons:

- Optimized for univariate time series (available in this form in implementations)
- Requires more data than statistical methods to function satisfactorily

6.3.2 Implementation

The implementation of the N-Beats model was taken from the pythorch-forecasting library [3] integrated with the pythorch lightning library. The training and prediction are wrapped in a docker image. It retrieves the monitoring metrics which should be predicted using the **metrics_to_predict** topic, as well as the **start_forecasting.nbeats** topic. It communicates with the Persistent Storage to retrieve monitoring data for the metrics it should predict. When predictions are made, they are published in the **intermediate_predictions.{metric_name}** topic of the broker (where {metric_name} is the name of the metric).

The rest of the N-Beats components are similar to other forecasters. Below, we provide a list of N-Beats environment variables:

- AMQ_HOSTNAME
- AMQ_USER
- AMQ_PASSWORD
- AMQ_PORT
- APP_NAME
- METHOD (nbetas)
- DATA_PATH (path were data from influx will be saved)
- INFLUXDB_HOSTNAME
- INFLUXDB_PORT
- INFLUXDB_USERNAME
- INFLUXDB_PASSWORD
- INFLUXDB_DBNAME

N-Beats config (example):

Listing 11 - N-Beats config

```
training:
      bs: 64
       max_epochs: 40
       loss: rmse (e.g. mae, mase)
       save_path:
            models
          dataloader_path:
            dataloader
      context_length_ratio:
          10 (prediction length to context history ratio)
```

The current implementation of the forecaster is available in GitLab[15].

We explain in the following the main scripts required by the forecaster.

Table 15 – N-Beats Forecaster's main script

Forecaster's main scripts	
main.py:	Is the entrypoint of the forecaster. It listens to the prediction start topic Runs independently train.py and predict.py
train.py:	Includes the source code performing the training process. Periodically performs model retraining and saves its weights. The training is performed on all available data (for the given metric) from influx. The validation set covers the last m points in the series and is not separable from the training set, which is rare in machine learning, but helps to quickly adapt to sudden changes in the series.
predict.py:	Includes the source code performing the prediction process. Periodically predicts each metric. Prediction is only possible when there are enough fresh rows in the database. Prediction module listens to stop prediction topic.
src/dataset_maker.py:	Includes the different functions required to gather the collected data from the persistent storage.
src/preprocess_dataset.py:	Preprocesses dataset from dataset_maker.py. Checks the time gaps between the consecutive timestamps. If the time gap is large with respect to metric granularity, then series is splitted, if it is short then missing values are filled with interpolation methods. Checks if there is enough of fresh data to make prediction for actual time. Applies data transformation and creates dataloaders for model.
Docker_image:	Includes the Dockerfile of the N-Beats image. It enables the installation of the different libraries required by the forecaster and the running of main file to start the forecaster. Includes the env file listing the different environment variables required by the forecaster to run and communicate with the different other services.

Like the other forecasters, nbetas forecaster depends on the following services:

- ActiveMQ is the event broker that we are using to communicate with the other subcomponents mainly the Prediction orchestrator. The library 'amq-message-python-library' is used which was developed in terms of Morphemic.
- Persistent storage is the service that manages the collection of the data related to the metrics that need to be forecasted.

[15] N-Beats forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/deployment/nbeats

6.3.3 Illustrative use

The N-Beats forecaster was evaluated using different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 18 dataset respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 16 - N-Beats forecaster performance on the Genome 12 dataset (Genome 12/13-2-2021)

	mae	**mse**	**mape**	**smape**
RemainingSimulationTimeMetric	0.7346	0.6764	0.0000	0.0000
MinimumCores	0.0000	0.0000	NaN	2.0000
NotFinishedOnTime Context	64.2549	14395.1055	0.0011	0.0011
EstimatedRemaining TimeContext	59.9467	15509.4000	NaN	0.0447
TotalCores	0.0000	0.0000	0.0000	0.0000
WillFinishTooSoon Context	65.1283	14558.8579	0.0014	0.0014
SimulationLeft Number	4.3672	71.1050	NaN	0.0384
SimulationElapsed Time	0.2734	0.1164	0.0000	0.0000
MinimumCores Context	0.0000	0.0000	NaN	2.0000
ETPercentile	0.9652	2.6898	0.0116	0.0113
NotFinishedOnTime	63.4267	13717.0021	0.0011	0.0011
NotFinished	65.4352	13661.5580	0.0011	0.0011

Table 17 - N-Beats forecaster performance on the Genome 18 dataset (Genome 20/21-2-2021)

	mae	**mse**	**mape**	**smape**
RemainingSimulationTimeMetric	1.0183	1.2597	0.0000	0.0000
MinimumCores	0.0000	0.0000	NaN	2.0000
EstimatedRemainingTimeContext	103.8079	37207.3712	NaN	0.0564
TotalCores	0.0000	0.0000	0.0000	0.0000
WillFinishTooSoonContext	93.4708	30967.0611	0.0020	0.0020
SimulationLeftNumber	4.8332	74.6092	NaN	0.0426
SimulationElapsedTime	0.4487	0.2790	0.0000	0.0000
MinimumCoresContext	0.0000	0.0000	NaN	2.0000
ETPercentile	0.9209	2.5731	0.0117	0.0114
NotFinishedOnTime	92.4963	29385.2247	0.0016	0.0016
NotFinished	94.0276	30012.8308	0.0017	0.0017
NotFinishedOnTimeContext	90.3145	29395.6334	0.0016	0.0016

6.4 Temporal Fusion Transformer (TFT)

6.4.1 Overview

TFT is an attention-based neural architecture that combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics [4]. TFT utilizes specialized components to select relevant features and a series of gating layers to suppress unnecessary components, enabling high performance in a wide range of scenarios. TFT shows significant performance improvements over existing benchmarks on a variety of real-world datasets.

Pros:

- Works with mixed type of data e.g., real unknown in feature values, categorical unknown in feature values, real known in feature values, static values etc.
- Interpretable results
- Handle well multivariate series

Cons:

- Requires more data than statistical methods to function satisfactorily

6.4.2 Implementation

The implementation of the TFT model was taken from the pythorch-forecasting library [11] integrated with the pythorch lightning library. The training and prediction are wrapped in a docker image. Docker image retrieves the monitoring metrics which should be predicted using the **metrics_to_predict** topic, as well as the **start_forecasting.tft** topic. It communicates with the Persistent Storage to retrieve monitoring data for the metrics it should predict. When predictions are made, they are published in the **intermediate_predictions.{metric_name}** topic of the broker (where {metric_name} is the name of the metric).

List of TFT enviromental variables:

- AMQ_HOSTNAME
- AMQ_USER
- AMQ_PASSWORD
- AMQ_PORT
- APP_NAME
- METHOD (nbetas)
- DATA_PATH (path were data from influx will be saved)
- INFLUXDB_HOSTNAME
- INFLUXDB_PORT
- INFLUXDB_USERNAME
- INFLUXDB_PASSWORD
- INFLUXDB_DBNAME

TFT config (example):

Listing 12 - TFT config

```
training:
        bs: 64
        max_epochs: 40
        loss: quantile (e.g. rmse, mae)
      dataset:
          tv_unknown_reals: []
          known_reals: []
          tv_unknown_cat: []
          static_reals: []
      model:
          learning_rate: 0.05
          hidden_size: 32
          attention_head_size: 1
          hidden_continuous_size: 16
          output_size: 7
      save_path:
          models
      dataloader_path:
          dataloader

      context_length_ratio:   12 (prediction
      length to context history ratio)
```

The current implementation of the forecaster is available in GitLab[16].

We explain in the following the main scripts required by the forecaster.

Table 18 – TFT Forecaster's main scripts

Forecaster's main scripts	
main.py:	Is the entrypoint of the forecaster. It listens to the prediction start topic Runs independently train.py and predict.py
train.py:	Includes the source code performing the training process. Periodically performs model retraining and saves its weights. The training is performed on all available data (for the given metric) from influx. The validation set covers the last m points in the series and is not separable from the training set, which is rare in machine learning, but helps to quickly adapt to sudden changes in the series.
predict.py:	Includes the source code performing the prediction process. Periodically predicts each metric. Prediction is only possible when there are enough fresh rows in the database. Prediction module listens to stop prediction topic.
src/dataset_maker.py:	Includes the functions required to gather the collected data from the persistent storage.
src/preprocess_dataset.py:	Preprocesses dataset from dataset_maker.py. Checks the time gaps between the consecutive timestamps. If the time gap is large with respect to metric granularity, then series is split, if it is short then missing values are filled with interpolation methods. Checks if there is enough fresh data to make prediction for actual time. Applies data transformation and creates es dataloaders for model.
Docker_image:	Includes the Dockerfile of the TFT image. It enables the installation of the different libraries required by the forecaster and the running of main file to start the forecaster. Includes the env file listing the different environment variables required by the forecaster to run and communicate with the different other services.

Like the other forecasters, TFT forecaster depends on the following services:

[16] TFT forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/deployment/tft

- ActiveMQ is the event broker that we are using to communicate with the other subcomponents mainly the Prediction orchestrator. The library ‘amq-message-python-library' which was developed for MORPHEMIC Project proposal is used.
- Persistent storage is the service that manages the collection of the data related to the metrics that need to be forecasted.

6.4.3 Illustrative use

The TFT forecaster was evaluated using different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 18 dataset respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 19 - TFT forecaster performance on the Genome 12 dataset (Genome 12/13-2-2021)

	mae	**mse**	**mape**	**smape**
MinimumCores	0.0000	0.0000	NaN	2.0000
NotFinishedOnTime	90.8372	28458.7025	0.0016	0.0016
SimulationElapsedTime	4.8891	28.1442	0.0001	0.0001
MinimumCoresContext	0.0000	0.0000	NaN	2.0000
TotalCores	0.0000	0.0000	0.0000	0.0000
ETPercentile	0.5882	1.3668	0.0073	0.0072
RemainingSimulationTimeMetric	8.1881	81.5680	0.0002	0.0002
WillFinishTooSoonContext	99.4750	34914.5868	0.0022	0.0022
SimulationLeftNumber	4.6790	86.1513	NaN	0.0427
EstimatedRemainingTimeContext	84.5214	30693.2530	NaN	0.0504
NotFinishedOnTimeContext	94.8311	26680.2110	0.0017	0.0017
NotFinished	94.2905	27321.3372	0.0017	0.0017

Table 20 - TFT forecaster performance on the Genome 18 dataset (Genome 20/21-2-2021)

	mae	mse	mape	smape
SimulationLeftNumber	5.9925	84.4109	NaN	0.0502
NotFinishedOnTime	94.1633	27117.5697	0.0017	0.0017
MinimumCoresContext	0.0000	0.0000	NaN	2.0000
TotalCores	0.0000	0.0000	0.0000	0.0000
EstimatedRemaining TimeContext	187.2061	58645.4737	NaN	0.0793
SimulationElapsedTime	3.4276	21.2428	0.0001	0.0001
WillFinishTooSoon Context	99.3047	31777.0196	0.0022	0.0022
NotFinished	87.7060	29422.6164	0.0015	0.0015
RemainingSimulation TimeMetric	8.4827	84.8350	0.0002	0.0002
MinimumCores	0.0000	0.0000	NaN	2.0000
NotFinishedOnTime Context	89.3310	30567.7021	0.0016	0.0016
ETPercentile	0.6349	1.6176	0.0080	0.0078

6.5 Prophet

6.5.1 Overview

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects [21]. It works best with time series that have strong seasonal effects and several seasons of historical data.

The Prophet uses a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

$y(t)= g(t) + s(t) + h(t) + \varepsilon t$

$g(t)$: piecewise linear or logistic growth curve for modelling non-periodic changes in time series

$s(t)$: periodic changes (e.g., weekly/yearly seasonality)

$h(t)$: effects of holidays (user provided) with irregular schedules

εt: error term accounts for any unusual changes not accommodated by the model

Using time as a regressor, Prophet is trying to fit several linear and non-linear functions of time as components. Modelling seasonality as an additive component is the same approach taken by exponential smoothing in Holt-Winters technique[17]. Prophet is framing the forecasting problem as a curve-fitting exercise rather than looking explicitly at the time-based dependence of each observation within a time series.

The main highlights of Prophet are:

- Accurate and fast, since it's built in Stan, a programming language for statistical inference written in C++.
- An additive regression model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects:
    - A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data
    - A yearly seasonal component modelled using Fourier series
    - A weekly seasonal component using dummy variables
    - A user-provided list of important holidays.
- Robust to missing data and shifts in the trend, and typically handles outliers.
- Easy procedure to tweak and adjust forecast while adding domain knowledge or business insights.

However, Prophet requires a huge amount of data (i.e. at least one year of time-series data[18]) to be able to perform good forecasting results.

6.5.2 Implementation

The implementation of Prophet was performed using the Prophet forecasting library provided by Facebook and available in Python. It provides intuitive parameters which are easy to tune.

We have applied some data transformation on the datasets before training prophet models. The main data transformers that have been used are: Log, Cube Root, Square Root, Standardization and BoxCox were applied. The prediction results were evaluated by comparing the effect of each of the transformers. MAE, MSE, MAPE, SMAPE have been used as evaluators metrics for the comparison.

Hyperparameter tuning has been also applied for choosing the best performing model. The list of hyperparameters for prophet are:

- seasonality_mode
- changepoint_prior_scale
- n_changepoints
- growth
- changepoint_range
- interval_width
- changepoints
- yearly_seasonality
- weekly_seasonality
- daily_seasonality
- seasonality_prior_scale
- mcmc_samples

The current implementation of the forecaster is available in GitLab[19]. We explain in the following the main script required by the forecaster.

[17] https://www.rdocumentation.org/packages/forecast/versions/8.15/topics/forecast.HoltWinters

[18] https://godatadriven.com/blog/facebooks-prophet-forecasting-stores-transactions/

[19] Prophet forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/forecasting_prophet

Table 21 - Prophet forecaster's main scripts

Forecaster's main scripts	
main.py:	Is the entry point of the forecaster. Runs independently the forecaster and the listener scripts that we will describe below.
Prophet_forecaster.py:	Includes the source code performing the training process and the prediction process.
Prophet_listener.py:	Includes the source code managing the subscription to the different topics to be able to receive the different messages on the one hand and the sending of messages including the prediction results over time on the other hand.
Dataset_maker.py:	Includes the different functions required to gather the collected data from the persistent storage.
Docker_image:	Includes the Dockerfile of the prophet image. It enables the installation of the different libraries required by the forecaster and the running of main file to start the forecaster. Includes the variables.env file listing the different environment variables required by the forecaster to run and communicate with the different other services.

To use Prophet[20] as forecaster, you can use directly the image published in the Gitlab registry[21] and run the following command:

```
docker run -dit --env-file variables.env --net your_network gitlab.ow2.org:4567/melodic/morphemic-preprocessor/prophet:morphemic-rc2.0
```

Variables.env refers to the file containing the following environment variables:

- ACTIVEMQ_USER: refers to the username that has to be used in order to connect to the ActiveMQ server.
- ACTIVEMQ_PASSWORD: refers to the password associated to the specified ACTIVEMQ_USER in order to connect to the ActiveMQ server.
- ACTIVEMQ_PORT: refers to the port assigned to the ActiveMQ server.
- ACTIVEMQ_HOSTNAME: refers to the hostname of the ActiveMQ server.
- APP_NAME: refers to the name of the application that we have to gather its data.
- DATA_PATH: refers to the path where we will save the collected data in the docker container.
- INFLUXDB_HOSTNAME: refers to the hostname of the InfluxDB server.
- INFLUXDB_PORT: refers to the port of the InfluxDB server.
- INFLUXDB_USERNAME: refers to the username that has to be used in order to connect to the InfluxDB server.
- INFLUXDB_PASSWORD: refers to the password associated to the specified INFLUXDB_USERNAME in order to connect to the InfluxDB server.
- INFLUXDB_DBNAME: refers to the name of the database from which, we will collect the data.

[20] https://pypi.org/project/prophet/

[21] gitlab.ow2.org:4567/melodic/morphemic-preprocessor/prophet:morphemic-rc2.0

your_network: refers to the network on which the other required services are running. Like the other forecasters, Prophet forecaster depends on the following services:

- ActiveMQ is the event broker that we are using to communicate with the other subcomponents mainly the Prediction orchestrator. The library 'amq-message-python-library' which was developed for MORPHEMIC Project proposal is used.
- Persistent storage is the service that manages the collection of the data related to the metrics that need to be forecasted.

6.5.3 illustrative use

The Prophet forecaster was evaluated using different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 18 dataset respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 22 - Prophet forecaster performance on the Genome 12 dataset

Metric	mae	mse	mape	smape
ETPercentile	1.317	3.68	0.015	0.015
NotFinishedOnTime Context	30.4	36783	0.0005	0.0005
RemainingSimulation TimeMetric	30.4	36783	0.0006	0.0006
NotFinished	162.7	85936	0.013	0.01
WillFinishTooSoon Context	157.86	69963	0.01	0.009
NotFinishedOnTime	162.79	85936	0.01	0.01
EstimatedRemaining TimeContext	165.78	76036	0.025	0.025
SimulationLeftNumber	3.78	29.5	0.007	0.007
TotalCores	0	0	0	0
MinimumCores	10.4	3539	nan	0.72
SimulationElapsedTime	30.4	36783	0.0005	0.0005
MinimumCoresContext	10.4	3539	nan	0.72

Table 23 - Prophet forecaster performance on the Genome 21 dataset

Metric	mae	mse	mape	smape
ETPercentile	1.37	4.52	0.017	0.017
NotFinishedOnTime Context	299.2	214095	0.011	0.01
RemainingSimulation TimeMetric	33.3	40284	0.0008	0.0007
NotFinished	299.2	214095	0.01	0.01
WillFinishTooSoon Context	293.3	193415	0.01	0.01
NotFinishedOnTime	299.2	214095	0.01	0.01
EstimatedRemaining TimeContext	338.25	321860	0.03	0.03
SimulationLeftNumber	3.67	27.9	0.007	0.007
TotalCores	0	0	0	0
MinimumCores	11.45	3606	nan	0.78
SimulationElapsedTime	33.3	40284	0.0006	0.0006
MinimumCoresContext	11.45	3606	nan	0.78

6.6 Sarima

6.6.1 Overview

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component [20]. It's often used as benchmark. As an input can also take array of exogenous regressors. Sarima needs to perform fit its weights before every prediction. In this work we use gridsearch for SARIMA hyperparameters search.

Pros:

- Easy to use
- Works well also on short series

Cons:

- Needs to fit before every prediction
- Requires hyperparameters finetuning to work well

6.6.2 Implementation

SARIMA implementation was taken from python statsmodels library [17]. The training and prediction are wrapped in a docker image. It retrieves the monitoring metrics which should be predicted using the **metrics_to_predict** topic, as well as the **start_forecasting.sarima** topic. It communicates with the Persistent Storage to retrieve monitoring data for the metrics it should predict. When predictions are made, they are published in the **intermediate_predictions.{metric_name}** topic of the broker (where {metric_name} is the name of the metric).

Below we provie a list of SARIMA enviroment variables:

- AMQ_HOSTNAME
- AMQ_USER
- AMQ_PASSWORD
- AMQ_PORT
- APP_NAME
- METHOD (N-Beats)
- DATA_PATH (path where data from influx will be saved)
- INFLUXDB_HOSTNAME
- INFLUXDB_PORT
- INFLUXDB_USERNAME
- INFLUXDB_PASSWORD
- INFLUXDB_DBNAME

The current implementation of the forecaster is available in GitLab[22]. We explain in the following the main scripts required by the forecaster.

Table 24 - Sarima forecaster's main scripts

Forecaster's main scripts	
main.py:	Is the entrypoint of the forecaster. It listens to the prediction start topic. Runs predict.py
predict.py:	Includes the source code performing the prediction process. Periodically predicts each metric. Prediction is only possible when there are enough fresh rows in the database. Prediction module listens to stop prediction topic. During each prediction, several sarima models are fitted on historical data and model with the best AIC is chosen for making a final prediction.
src/dataset_maker.py:	Includes the different functions required to gather the collected data from the persistent storage.
src/preprocess_dataset.py:	Preprocesses dataset from dataset_maker.py. Checks the time gaps between the consecutive timestamps. If the time gap is large with respect to metric granularity, then series are splitted, if it is short then missing values are filled with interpolation methods. Checks if there is enough of fresh data to make prediction for actual time.
Docker_image:	Includes the Dockerfile of the SARIMA image. It enables the installation of the different libraries required by the forecaster and the running of main file to start the forecaster. Includes the env file listing the different environment variables required by the forecaster to run and communicate with the different other services.

[22] Sarima forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/deployment/arima

Like the other forecasters, SARIMA forecaster depends on the following services:

- ActiveMQ is the event broker that we are using to communicate with the other subcomponents mainly the Prediction orchestrator. The library ‘amq-message-python-library' which was developed for MORPHEMIC Project proposal is used.
- Persistent storage is the service that manages the collection of the data related to the metrics that need to be forecasted.

6.6.3 Illustrative use

SARIMA is integrated with MORPHEMIC in the same way as the rest of the forecasters. To use SARIMA as forecaster, one can run morphemic. The SARIMA forecaster was evaluated using different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 18 dataset respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 25 - Sarima forecaster performance on the Genome 12 dataset (Genome 12/13-2-2021)

	mae	**mse**	**mape**	**smape**
ETPercentile	0.7636	1.6513	0.0094	0.0093
NotFinishedOnTime Context	76.6898	16498.2163	0.0013	0.0013
RemainingSimulation TimeMetric	0.2159	0.0594	0.0000	0.0000
NotFinished	76.6898	16498.2163	0.0013	0.0013
WillFinishTooSoon Context	76.8393	16573.7722	0.0016	0.0016
NotFinishedOnTime	76.6898	16498.2163	0.0013	0.0013
EstimatedRemaining TimeContext	77.6851	16911.2814	NaN	0.0782
SimulationLeftNumber	6.1957	124.5856	NaN	0.0667
TotalCores	0.0000	0.0000	0.0000	0.0000
MinimumCores	0.0000	0.0000	NaN	NaN
SimulationElapsedTime	0.1866	0.0444	0.0000	0.0000
MinimumCoresContext	0.0000	0.0000	NaN	NaN

Table 26 - SARIMA forecaster performance on the Genome 18 dataset (Genome 20/21-2-2021)

	mae	**mse**	**mape**	**smape**
MinimumCoresContext	0.0000	0.0000	NaN	NaN

RemainingSimulation TimeMetric	0.2439	0.0759	0.0000	0.0000
MinimumCores	0.0000	0.0000	NaN	NaN
NotFinishedOnTime Context	146.0565	58657.5318	0.0026	0.0026
SimulationLeftNumber	6.6988	139.5546	0.2088	0.0744
SimulationElapsedTime	0.2071	0.0547	0.0000	0.0000
WillFinishTooSoon Context	146.5062	58983.0653	0.0032	0.0032
TotalCores	0.0000	0.0000	0.0000	0.0000
EstimatedRemaining TimeContext	147.9898	59822.5965	0.2621	0.0875
ETPercentile	0.6962	1.1242	0.0095	0.0094
NotFinished	146.0565	58657.5318	0.0026	0.0026
NotFinishedOnTime	146.0565	58657.5318	0.0026	0.0026

6.7 GluonTS

6.7.1 Overview

Gluon Time Series (GluonTS) is a Python toolkit developed by Amazon scientists for building, evaluating, and comparing deep learning–based time series models [22]. GluonTS is based on the Gluon interface to Apache MXNet and provides components that make building time series models simple and efficient. GluonTS provides utilities for loading and iterating over time series datasets, state of the art models ready to be trained, and building blocks to define users' own models and quickly experiment with different solutions.

Deep learning models for time series modelling commonly include components such as recurrent neural networks based on Long Short-Term Memory (LSTM) cells, convolutions, and attention mechanisms. This makes using a modern deep-learning framework, such as Apache MXNet, a convenient basis for developing and experimenting with such models. However, time series modelling also often requires components that are specific to this application domain. GluonTS provides these time series modelling-specific components on top of the Gluon interface to MXNet.

In particular, GluonTS contains:

- Higher-level components for building new models, including generic neural network structures like sequence-to-sequence models and components for modelling and transforming probability distributions.
- Data loading and iterators for time series data, including a mechanism for transforming the data before it is supplied to the model.
- Reference implementations of several state-of-the-art neural forecasting models.
- Tooling for evaluating and comparing forecasting models.

However, GluonTS has some limitations regarding the time granularity in the time series. GluonTS requires at least 1 minute time difference between the consecutive datapoints in the time series.

6.7.2 Implementation

For the implementation of this forecaster, the Gluon Time Series Python toolkit[23] developed by Amazon is used. It includes several subpackages and submodules for different functionalities such as the gluonts.dataset package, gluonts.evaluation package, or gluonts.model package etc. The gluonts.model package includes several submodules for different types of models. In this work, we have mainly used the DeepAR for training models and forecasting.

We have applied some data transformation on the datasets before training gluonTS models. Mainly, the data transformers: Log, Cube Root, Square Root, and Standardization were applied. The prediction results were evaluated by comparing the effect of each of the transformers. MAE, MSE, MAPE, SMAPE have been used as evaluation metrics for the comparison. Knowing that GluonTS does not accept data with timestamp difference less than 1 minutes, we have applied some changes of the datapoints timestamps to adapt to this condition.

Hyperparameter tuning has been also applied for choosing the best performing model. The list of hyperparameters for GluonTS DeepAR model are:

- batch_size
- epochs
- num_batches_per_epoch
- learning_rate
- context_length

The current implementation of the forecaster is available in GitLab[24]. We explain in the following the main scripts required by the forecaster.

Table 27 - GluonTS forecaster's main scripts

Forecaster's main scripts	
main.py:	Is the entrypoint of the forecaster. Runs independently the forecaster and the listener scripts that we will describe below.
Gluonts_forecaster.py:	Includes the source code performing the training and the prediction processes using DeepAR algorithm.
Gluonts_listener.py:	Includes the source code managing the subscription to the different topics to be able to receive the different messages on the one hand and the sending of messages including the prediction results over time on the other hand.
Dataset_maker.py:	Includes the different functions required to gather the collected data from the persistent storage.
Docker_image:	Includes the Dockerfile of the gluonmachines image. It enables the installation of the different libraries required by the forecaster and the running of the main file to start the forecaster. Includes variables.env file listing the different environment variables required by the forecaster to run and to communicate with the different other services.

[23] https://ts.gluon.ai/

[24] GluonTS forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/forecasting_gluonts

To use GluonTS as forecaster, you can use directly the image published in the Gitlab registry[25] and run the following command:

```
docker run -dit --env-file variables.env --net your_network gitlab.ow2.org:4567/melodic/morphemic-preprocessor/gluonmachines:morphemic-rc2.0
```

Variables.env refers to the file containing the following environment variables:

- ACTIVEMQ_USER: refers to the username that has to be used in order to connect to the ActiveMQ server.
- ACTIVEMQ_PASSWORD: refers to the password associated to the specified ACTIVEMQ_USER in order to connect to the ActiveMQ server.
- ACTIVEMQ_PORT: refers to the port assigned to the ActiveMQ server.
- ACTIVEMQ_HOSTNAME: refers to the hostname of the ActiveMQ server.
- APP_NAME: refers to the name of the application that we have to gather its data.
- DATA_PATH: refers to the path where we will save the collected data in the docker container.
- INFLUXDB_HOSTNAME: refers to the hostname of the InfluxDB server.
- INFLUXDB_PORT: refers to the port of the InfluxDB server.
- INFLUXDB_USERNAME: refers to the username that has to be used in order to connect to the InfluxDB server.
- INFLUXDB_PASSWORD: refers to the password associated to the specified INFLUXDB_USERNAME in order to connect to the InfluxDBserver.
- INFLUXDB_DBNAME: refers to the name of the database from which, we will collect the data.

your_network refers to the network on which the other required services are running.

Like the other forecasters, GluonTS forecaster depends on the following services:

ActiveMQ is the event broker that we are using to communicate with the other subcomponents mainly the Prediction orchestrator. The library 'amq-message-python-library' which was developed for MORPHEMIC Project proposal is used.

Persistent storage is the service that manages the collection of the data related to the metrics that need to be forecasted.

6.7.3 illustrative use

The GluonTS forecaster was evaluated using different metrics provided by two datasets. In the following tables, we illustrate the results of the experiments made based on the Genome 12 and Genome 21 datasets respectively. MAE, MSE, MAPE, SMAPE were used as evaluation metrics for the following comparison.

Table 28 - GluonTS forecaster performance on the Genome 12 dataset

Metric	mae	mse	mape	smape
ETPercentile	10.3	146.7	0.13	0.11
NotFinishedOnTime Context	27796	772900714	0.46	0.59
RemainingSimulation TimeMetric	34905	1220364518	0.63	0.92

[25]GluonTs image in the MORPHEMIC GitLab repository: https://gitlab.ow2.org:4567/melodic/morphemic-preprocessor/gluonmachines:morphemic-rc2.0

NotFinished	26740	715213418	0.44	0.56
WillFinishTooSoon Context	21507	462699322	0.44	0.56
NotFinishedOnTime	23555	555541258	0.39	0.48
EstimatedRemaining TimeContext	6422	41845266	nan	1.31
SimulationLeftNumber	515	270573	nan	1.24
TotalCores	0.05	0.003	0.05	0.05
MinimumCores	1.75	3.11	nan	2
SimulationElapsedTime	29740	884800330	0.47	0.62
MinimumCoresContext	2.43	5.9	nan	2

Table 29 - GluonTS forecaster performance on the Genome 21 dataset

Metric	mae	mse	mape	smape
ETPercentile	3.53	20.2	0.047	0.047
NotFinishedOnTime Context	24575	604281846	0.438	0.562
RemainingSimulation TimeMetric	29137	849206050	0.59	0.839
NotFinished	25013	625909921	0.44	0.57
WillFinishTooSoon Context	13922	195126979	0.3	0.36
NotFinishedOnTime	22099	488746979	0.39	0.49
EstimatedRemaining TimeContext	8236	68916236	nan	1.19
SimulationLeftNumber	372	142115	nan	1.12
TotalCores	0.026	0.0008	0.026	0.027
MinimumCores	1.31	1.75	nan	2
SimulationElapsedTime	27522	757849489	0.48	0.64
MinimumCoresContext	1.39	1.97	nan	2

6.8 Exponential Smoothing

6.8.1 Overview

Algorithms based on exponential smoothing have been long recognized for their usefulness, in time series predictions. Moreover, exponential smoothing is also provided by Amazon[26], as a service which can help in workload prediction. Due to the appeal of such algorithms, we have elected to implement a forecaster which is based on the use of exponential smoothing.

The Exponential smoothing forecaster consists of a wrapper around the Holt-Winters[27] and ETS[28] R libraries. The exact library which will be used by the predictor is a configuration option. It retrieves the monitoring metrics which should be predicted using the **metrics_to_predict** topic, as well as the **start_forecasting.exponentialsmoothing** topic. It communicates with the Persistent Storage to retrieve monitoring data for the metrics it should predict. When predictions are made, they are published in the **intermediate_predictions.{metric_name}** topic of the broker (where {metric_name} is the name of the metric).

In the case of the Holt-Winters algorithm, the algorithm estimates i) a level component ii) a trend component and iii) a seasonality component. The level component tries to account for short-term workload fluctuations, the trend component tries to incorporate the influence of long-term workload trend, and the seasonality component is used to factor in workload variations due to seasonality. The addition of these three components forms the forecasted value.

In the case of the ETS algorithm, a series of models are evaluated using Akaike's Information Criterion (corrected for small sample bias) in order to determine the most appropriate forecasting model. Each model has three components - Error, Trend and Seasonal. Error can be either additive or multiplicative. Trend can be additive, additive damped or not considered. Seasonality can be either additive, multiplicative or not considered. The expressivity of ETS is greater than Holt-Winters, and in fact Holt-Winters is one of the algorithms possible to be depicted using ETS.

The detailed equations which are used in Holt-Winters as well as more details on the Holt-Winters method and possible ETS models, we refer the reader to [18].

6.8.2 Implementation

The internal architecture of the component appears in the following figure 16.

[26] https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-ets.html

[27] https://www.rdocumentation.org/packages/forecast/versions/8.15/topics/forecast.HoltWinters

[28] https://www.rdocumentation.org/packages/forecast/versions/8.15/topics/ets

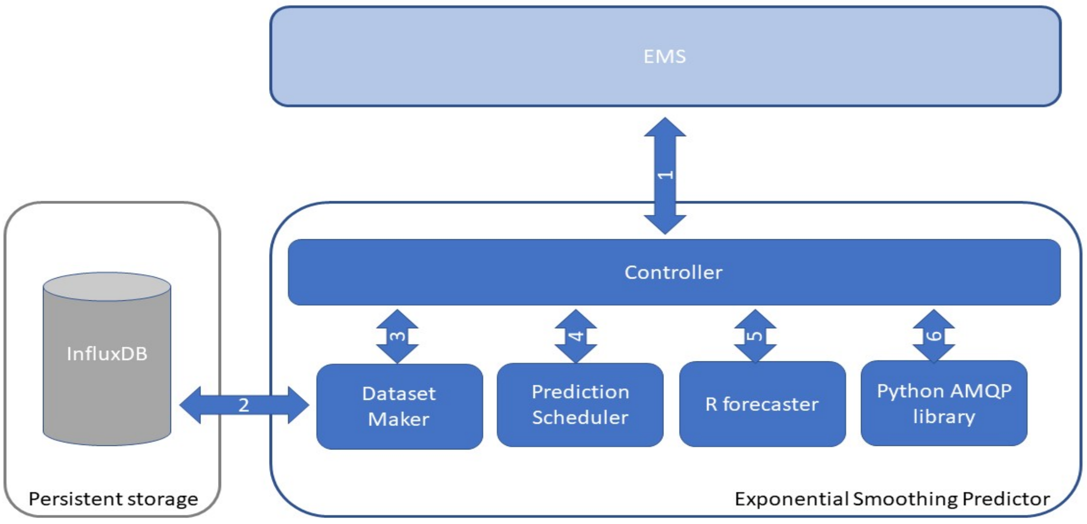


Figure 10 - Internal architecture of the Exponential Smoothing Predictor

The Exponential Smoothing Predictor is packaged as a Docker container which initializes a Python Controller responsible of coordinating the actions related to forecasting. The Predictor further includes a Dataset Maker subcomponent, a Prediction Scheduler subcomponent, an R forecaster subcomponent, and interfaces with the Python AMQP library.

The core predictive functionality of the Exponential Smoothing predictor is implemented by an R script, the R forecaster subcomponent. The predictions which are generated are propagated by the Controller to the EMS for further processing (using functionality offered by the Python AMQP library). Moreover, the Controller estimates the appropriate time point at which a new prediction process should be initiated, and then spawns one OS-level process for each attribute to be predicted, in parallel.

Regarding the communication between internal and external components, External communication 1 is performed using AMQP, and involves retrieving commands from the Prediction Orchestrator, and sending prediction events. External communication 2 is performed using REST and involves the acquisition of the dataset which will be used to generate predictions. Internal Communications 3-6 are performed using Python objects and methods.

The current implementation of the forecaster is available in the MORPHEMIC GitLab repository.[29]

When data is received from the Dataset Maker, to increase robustness, it is converted to obtain granularity at second-level. Therefore, even if a prediction is requested at a time point which is not precisely aligned with the data, the component will produce a prediction. Empty or non-existent values are estimated using the *na.approx* function which is available in R.

For the predictions created by the Exponential Smoothing forecaster to be useful, a prediction horizon greater than one second will be commonly required (to allow time for adaptation). As the Holt-Winters and ETS R predictor libraries estimate the best parameters by minimizing the MSE or maximizing the likelihood respectively (both based on the calculation of one-step ahead errors within the training dataset), it is necessary to resample the dataset in order to allow the method to estimate the best parameters assuming that predictions are made one-step ahead. We therefore include a

[29] Exponential Smoothing Forecaster in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/morphemic-forecasting-exponentialsmoothing

relevant configuration option in the properties file of the component, to set the desired resampling period. Resampled values are obtained from observations inside time intervals with an arbitrary length (in seconds).

Once all necessary data is retrieved, the data is split to training and test datasets, and a model is retrained. The amount of data which is relevant for a new prediction is specified in the configuration file. To improve efficiency, and take advantage of the presence of multiple CPUs, the implementation of the predictor, uses separate processes to calculate the prediction related to each attribute.

If the performance of the predictor is not found to be satisfactory, then an exhaustive search mechanism can be used, with a custom granularity, to allow tailoring the alpha, beta and gamma (short, long term and seasonality) coefficients of the Holt-Winters exponential smoothing method.

6.8.3 Illustrative use

To use the forecasting functionality, assuming that R is installed and the Rscript command is available, the command which should be used appears below:

```
Rscript forecasting_real_workload.R  path_to_csv_file  metric  targeted_prediction_time
```

Where ***forecasting_real_workload.R*** is the name of the R script which carries out the prediction, ***path_to_csv_file*** is the absolute filepath of the dataset, ***metric*** is the name of the metric to be forecasted and ***targeted_prediction_time*** is a Unix epoch timestamp which specifies the time point for which we wish to generate a prediction (in seconds). An example call would be the following:

```
Rscript forecasting_real_workload.R /opt/data/set1.csv cpu_usage 1613167137
```

Naturally, when predictions of multiple metrics are required, the prediction script will need to be called more than once; this process is automated by the Controller.

The predictor was evaluated using a multitude of metrics in two distinctive datasets. The predictor was configured to use the Holt-Winters library for this comparison. Its performance appears in the following tables for the Genome 12 and Genome 18 dataset respectively:

Table 30 - Exponential Smoothing Predictor performance on the Genome 12 dataset

	mae	mse	mape	smape
EstimatedRemainingTimeContext	144.0979	36605.4458	inf	0.0993
SimulationLeftNumber	8.1749	144.6138	inf	0.0746
SimulationElapsedTime	150.5000	22650.2500	0.0026	0.0026
NotFinishedOnTime	177.7620	36006.5151	0.0031	0.0031
MinimumCoresContext	1.2591	1.6078	inf	2.0000
NotFinished	177.7620	36006.5151	0.0031	0.0031
WillFinishTooSoonContext	169.5359	33449.0581	0.0036	0.0036
NotFinishedOnTimeContext	177.7620	36006.5151	0.0031	0.0031
MinimumCores	1.2591	1.6078	inf	2.0000

ETPercentile	1.6108	7.1554	0.0200	0.0191
RemainingSimulationTimeMetric	150.5000	22650.2500	0.0029	0.0029
TotalCores	0.0000	0.0000	0.0000	0.0000

Table 31 - Exponential Smoothing Predictor performance on the Genome 21 dataset

	mae	mse	mape	smape
EstimatedRemainingTimeContext	240.1213	101404.8275	inf	0.0948
SimulationLeftNumber	7.7859	132.7747	inf	0.0744
SimulationElapsedTime	150.5000	22650.2500	0.0028	0.0028
NotFinishedOnTime	265.4976	86695.0842	0.0049	0.0049
MinimumCoresContext	1.1398	1.3288	inf	2.0000
NotFinished	265.4976	86695.0842	0.0049	0.0049
WillFinishTooSoonContext	257.5010	85386.4789	0.0058	0.0058
NotFinishedOnTimeContext	265.4976	86695.0842	0.0049	0.0049
MinimumCores	1.1398	1.3288	inf	2.0000
ETPercentile	1.1722	3.4887	0.0157	0.0153
RemainingSimulationTimeMetric	150.5000	22650.2500	0.0033	0.0033
TotalCores	0.0000	0.0000	0.0000	0.0000

6.9 Discussion/Evaluation

The forecasting algorithms presented in this section cover a wide range of methods and approaches for time series forecasting. Starting from the widely used ARIMA/SARIMA models which are very popular in forecasting, through Exponential Smoothing and Prophet methods which are state-of-the-art non-deep learning time series forecasting models to advanced, deep learning algorithms (ES-Hybrid, N-Beats, TFT, GluonTS).

To compare the performance of these algorithms two important for Genom application optimization metrics have been selected. The SimulationLeftNumber metric which defines the number of the simulations remaining to finish a given set of simulations. EstimatedRemainingTimeContext metric which defines the estimated remaining time to complete a given set of simulations. The forecasting of these values will allow for better optimization of cloud computing resources currently used. To compare the accuracy of forecasting the MAE indicator has been used. As all foresting methods have been used on the same datasets, the MAE value will give the most precise comparison of the results. The MAPE and SMAPE metrics have not been used due to numerical issues with calculation of these metrics for samples where target variable is zero. The lower value of MAE means the forecasting methods are more accurate to predict future values of the abovementioned metrics.

On the genom12 dataset, the best results on individual metric level have been achieved by the Prophet method – see Table 31. As this dataset is smaller and has a lower variance for given metrics, the specialized statistical methods, which require fewer data to train, achieved better results compared to deep learning algorithms.

On the genom18 dataset, which contains bigger volume of data, a specialized deep learning method N-Beats achieved the best performance on average and on individual metric level (along with TFT method) – see Table 32. This shows that for this dataset, with the higher variance of the metrics, deep learning methods are the best choice.

The final evaluation of the forecasting methods, based on multiple examples and use case application will be presented in deliverable D2.4 "*Proactive Utility - algorithms and evaluation*" and a final conclusion about the performance of the forecasting methods will be drawn there.

Table 32 - Comparison of forecasting methods on Genom12 dataset for the SimulationLeftNumber and EstimatedRemainingTimeContext metrics

Forecasting method	**SimulationLeftNumber - MAE**	**EstimatedRemaining TimeContext - MAE**	**Average MAE value for SimulationLeftNumber and EstimatedRemainingTimeContext**
ES-Hybrid	50.3716	564.3234	307.3475
N-Beats	4.3672	59.9467	32.1570
TFT	4.6790	84.5214	44.6002
Prophet	3.78	165.7	84.7400
SARIMA	6.1957	77.6851	41.9404
GluonTS	515	6422	3468,5
Exponential Smoothing	8.1749	144.0979	76.1364

Table 33 - Comparison of forecasting methods on Genom18 dataset for the SimulationLeftNumber and EstimatedRemainingTimeContext metrics

Forecasting method	SimulationLeftNumber - MAE	EstimatedRemaining TimeContext - MAE	Average SMAPE value for SimulationLeftNumber and EstimatedRemainingTimeContext
ES-Hybrid	58.6324	1298.7369	678,6847
N-Beats	4.8332	103.8079	54,3206
TFT	5.9925	187.2061	96,5993
Prophet	27.3	304	165,6500
SARIMA	6.6988	147.9898	77,3443
GluonTS	372	8236	4304,0000

Exponential Smoothing	7.7859	240.1213	123,9536

7 Prediction Orchestration and Adaptation Triggering

7.1 Prediction Orchestrator

In section 5, different forecasting algorithms were described, each running in a different docker container. One of the goals of Proactive adaptation approach is to make use of as many forecasters as possible, meaning that the more forecasters are available, the better predictions of metrics we would get. The component that is responsible for orchestrating those forecasters and gathering produced data is called Prediction Orchestrator. Its main features are:

- Sending information to the forecasters which metrics they should predict and at what time intervals
- Gathering all of the predictions created by the forecasters
- Orchestrating received predictions: verifying if they concern valid timestamps in the future and storing them
- Based on those predictions, sophisticated ensembling methods would be launched to produced one, better prediction for each timestamp, for each metric
- Resulted predictions are further sent to the SLO Violation Detector
- If one of the forecasters would send updated predictions, Prediction Orchestrator would produce a new, updated merged prediction

7.1.1 Approach and Implementation details

Multiple, distributed forecasting containers can have different inner clocks. As consequence the event-driven approach of organizing the predictions has been chosen. This approach also allows to receive messages not in order. To communicate with the forecasters and SLO Violation Detector, AMQP messages are used. Prediction Orchestrator is started with following parameters.

Table 34 – Prediction Orchestrator's Parameters

Parameter	Description	Indicative value
forecasting_configuration.initial_prediction_horizon	This property indicates what is the time difference (seconds) between following 'predictionTime' fields, both inside the forecasters and the PO. Depending on the forecaster it may also mean how often the Forecaster would publish predictions to the PO.	30
forecasting_configuration.initial_forward_prediction_numbe	This property indicates how many forward predictions the forecaster should send at the moment of publishing predictions	8
forecasting_configuration.starting_forecasting_delay	This property indicates what is the 'epochStart' and what 'predictionTime' should have the first message send by the forecaster. It also explains the time before the forecasters start publishing. EpochStart = currentTime + starting_forecasting_delay	200
pooling.poolingStrategy	This property indicates method for checking how many forecasters' data is	STATIC_FORECASTERS_COUNT_NEEDED

	needed to be able to ensemble predictions and publish them forward	
pooling.threshold	This value indicates how many forecasters are needed to create an ensembled prediction	2
startingMethodsList	List of connected forecasters that PO expects to be working. PO would not communicate with the forecasters not on that list	es_hybrid,prophet,tft,nbeats,arima,tsetlin_machines,exponential_smoothing,lstm,gluon_machines

At the beginning of working, Prediction Orchestrator receives a 'metrics to predict' list. Then it sends a message to each of the forecasters to start working. The message is sent on topic: 'startt_forecasting.[forecaster_name]' and contains data needed to start predicting. It contains parameters that could be seen above:

`epochStart,  initial_forward_prediction_number and initial_prediction_horizon.`

All components in the system assume that every message containing prediction would have field: 'predictionTime' that needs to follow the pattern:

`predictionTime = epochStart + k * initial_prediction_horizon;` where k is integer

This field is crucial to determine which predictions concern which timestamp in the future.

From now on, Prediction Orchestrator expects to receive predictions from the forecasters on topics: `intermediate_prediction.[forecaster_name].[metricName]`

Prediction Orchestrator creates a different Listener/Thread that is responsible for handling each of those topics.

When received, they are validated and stored in the 'PredictionRegistry' which has similar work as a circular buffer. For each topic, 'initial_forward_prediction_number' of predictions are stored.

Each message is an event that can cause prediction ensembling. For the timestamp, Prediction Orchestrator verifies if there are enough predictions to ensemble (based on 'pooling' properties). If so, ensembling methods are launched to produce ensembled predictions (using data from as many forecasters as already has been received). If method succeeds, an ensembled prediction for a specific predictionTime (concerning one metric) has been created. If such ensembled values are created for each of the metric, then they are all sent to the SLO violation detector via topic: `prediction.[topic_name]`

Prediction Orchestrator is implemented as a spring boot application (v 2.4.1) with Java 11 language[30]. For the communication via ActiveMQ, the 'amq-message-java-library' (created for MORPHEMIC project) is used.

7.1.2 Model Ensembling

7.1.2.1 Introduction

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. Ensembling methods for one exampled dataset were tested.

Dataset

[30] Prediction Orchestrator in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/morphemic-rc2.0/prediction_orchestrator

Predictions on CPU usage - predictions from forecasters: N-Beats, TFT link to data[31].

Methods

Several methods are being evaluated as mentioned below:

- Classical:
  - Naive mean
  - Mean over the best n methods on k last timesteps
  - Predictions weights found with linear programming
  - Mean over the best subset (from all of possible subsets)
- Advanced:
  - FCNN (Fully Connected Neural Networks)
  - CNN (Convolutional Neural Network 1D) [16] + residual connections
  - CNN (Convolutional Neural Network 1D) + residual connections + self-attention

7.1.2.2 Experiments

Below we provide a short overview of the current experiments:

- Classical methods: The train/val/test (ratio 7:2:1) were constructed. Naive mean was directly evaluated on test split. For other classical methods (train + val) split were used for choosing the best parameters (best subset for mean over the best subset, k and n for mean over the best n methods on k last timesteps, forecasters weights for linear programming) and then evaluated on test set.
- Advanced methods: Neural networks were trained on training set, the best models were chosen based on validation test, final prediction was made on test set.

With respect to the latter methods the Neural network input involved:

- Historical real values (if available at the moment) + historical errors (if available at the moment) + historical predictions + current prediction

With respect to Neural network output:

- Weights (summing up to 1, each positive)

Below, we provide a neural network input example (which consists of 2 parts: Past and future).

[31] Dataset used for Model Ensembling in the MORPHEMIC GitLab repository: https://gitlab.ow2.org/melodic/morphemic-preprocessor/-/tree/tft_nbeats/deployment/ensembler-data

Table 35 - Neural network input part: Past

	pred_ error_1	pred_ error_2	pred_ error_3	pred_ error_4	pred_ error_5	pred_ error_5	is_past	real value
0	10.44	10.60	0	6.23	3.14	8.88	1	26.0
1	3.67	8.94	0	0.17	-3.92	7.54	1	29.0
2	1.25	12.33	0	-0.07	2.93	11.35	1	26.0
3	5.12	20.08	0	2.20	25.13	19.62	1	18.0
4	-1.50	47.69	0	-3.49	53.17	49.87	1	18.0

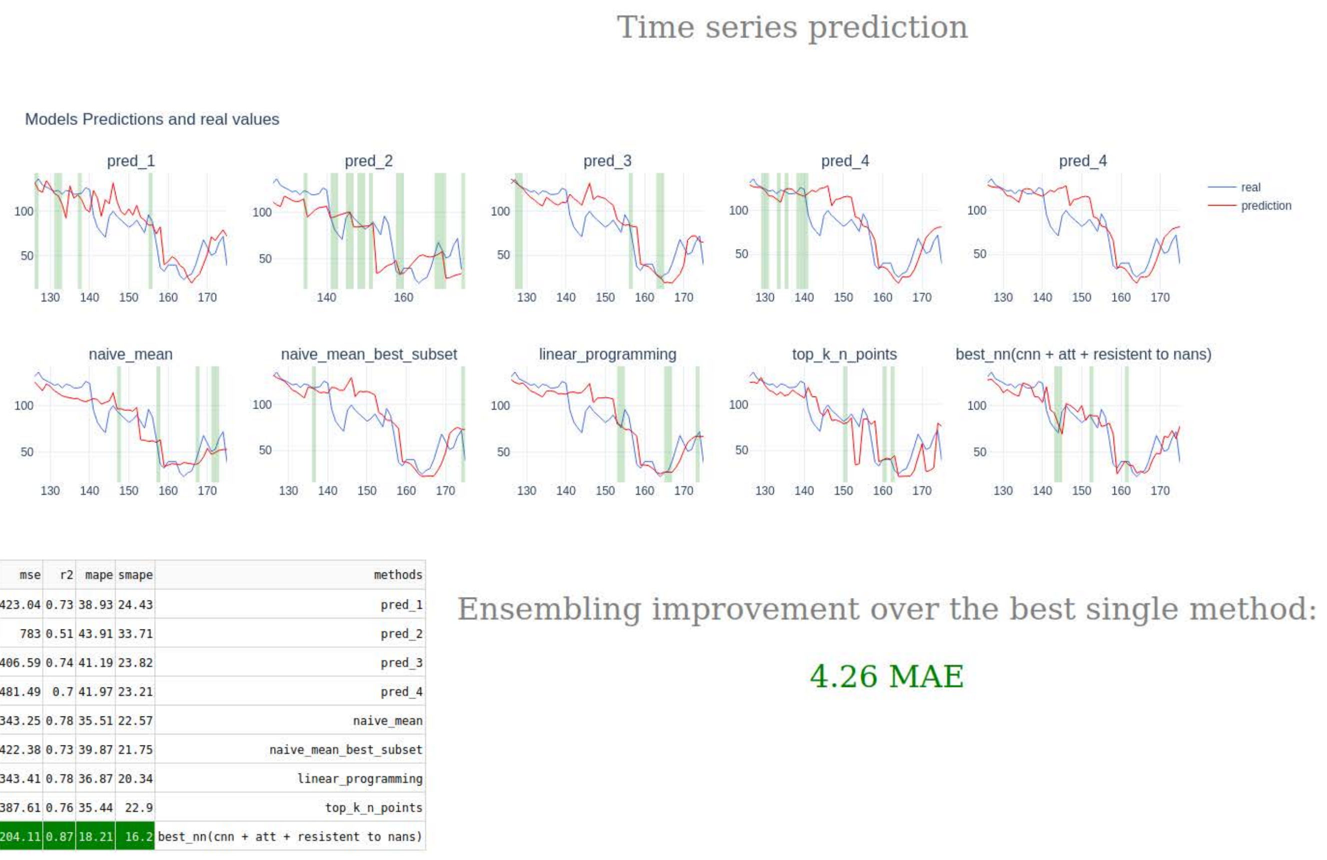
Table 36 - Neural network input part: Future

	pred_0	pred_1	pred_2	pred_3	pred_4	pred_5	is_past	real value
5	0	62.05	15.53	13.53	64.79	63.64	0	0.0
6	0	56.55	15.48	14.34	64.17	56.96	0	0.0
7	0	51.54	16.00	15.32	65.22	50.66	0	0.0

In the following table we provide the experiment results with respect to the CPU usage predicted metric.

Table 37 - CPU usage predicted metric

	mae	mse	smape	mape
Best single forecaster	16.56	631.86	0.31	0.68
Naive mean	19.94	711.26	0.36	0.83
Mean over the best subset (from all of possible subsets)	16.18	616.38	0.29	0.68
Mean over the best n methods on k last timestep	16.09	560.66	0.30	0.67
Predictions weights found with linear programming	16.07	596.51	0.29	0.69
The best neural network (CNN (Convolutional Neural Network 1D) + residual connections)	12.40	402.00	0.23	0.54



mae	mse	r2	mape	smape	methods
15.13	423.04	0.73	38.93	24.43	pred_1
21.17	783	0.51	43.91	33.71	pred_2
14.74	406.59	0.74	41.19	23.82	pred_3
15.73	481.49	0.7	41.97	23.21	pred_4
14.34	343.25	0.78	35.51	22.57	naive_mean
14.58	422.38	0.73	39.87	21.75	naive_mean_best_subset
13.2	343.41	0.78	36.87	20.34	linear_programming
14.27	387.61	0.76	35.44	22.9	top_k_n_points
10.48	204.11	0.87	18.21	16.2	best_nn(cnn + att + resistent to nans)

Figure 11 - Experiments results

Simulation of real time ensembling results, the light green highlighted rectangles represent time periods were given method gave the most accurate predictions. In the table metrics are calculated. The best method for given metric is green coloured.

As it has been shown, this approach gives promising results and therefore is being implemented as a part of Prediction Orchestrator. The final results of evaluation will be provided in D2.4"

7.2 Severity-based SLO Violation Detector

The SLO Violation Detector is the component which is responsible to trigger the proactive adaptation of the MORPHEMIC platform. It uses predicted data from the Prediction Orchestrator, and real-time data to calculate whether there is a need for an adaptation. To do so, it uses Severity, a multi-factor statistical metric. Severity is calculated whenever an adaptation opportunity is detected by the component, a situation which the platform has the time to react to. Based on the Severity value, which is determined for the current situation, the component sends a message to the Metasolver specifying the probability of a new adaptation occurring in an upcoming time period.

7.2.1 Component Architecture

Similar to other components comprising the Forecasting Module, we have adopted for the SLO Violation Detector an event-driven architecture. All input and output are received and sent using AMQP messages.

The internal architecture of the SLO Violation Detector appears below:

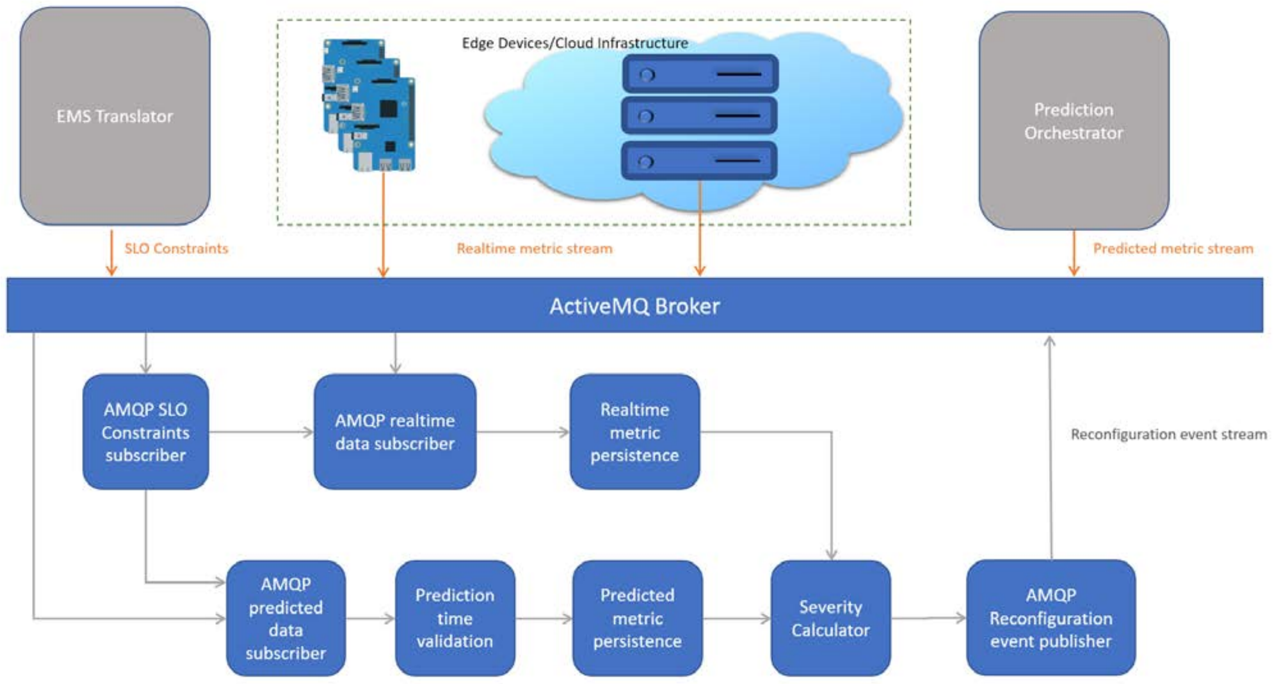

EMS Translator
Edge Devices/Cloud Infrastructure
Prediction Orchestrator
SLO Constraints
Realtime metric stream
Predicted metric stream
ActiveMQ Broker
AMQP SLO Constraints subscriber
AMQP realtime data subscriber
Realtime metric persistence
Reconfiguration event stream
AMQP predicted data subscriber
Prediction time validation
Predicted metric persistence
Severity Calculator
AMQP Reconfiguration event publisher

Figure 12 - SLO Violation Detector

The component first receives the SLO constraints which should be respected for the application which is currently deployed in the MORPHEMIC platform. Based on the monitoring attributes which are mentioned in this input, the component subscribes to the relevant AMQP topics for predicted and real-time metrics. When real-time or predicted metrics are received, they are persisted in suitable data structures. Messages containing information related to predicted metrics can also trigger a proactive adaptation, if sufficient time is available for the platform to perform an adaptation and if the data contained indicates that this is necessary. The Severity Calculator submodule undertakes the calculation of the Severity for each of the SLO constraints provided to the component. If any of these constraints is violated, a suitable message is transmitted to the ActiveMQ Broker.

The functionality of the component can be tuned using a configuration file. The parameters which can be tuned to run the component and use its stable functionality appear in Table 37.

Table 38 - Stable operation configuration parameters of the SLO Violation Detector

Parameter	Description	Indicative value
metrics_bounds	A string value which is a csv list of monitoring metrics, and the respective upper and lower bounds which are known for them beforehand. The list contains triplets which are comma separated, while elements of a triplet are separated with semicolons. Each triplet contains the name of the metric, its lowest bound and its highest bound (or the word 'unbounded' if these are not known. If a monitoring metric is not registered here, it will be assumed that it can be assigned any real value from 0 (the lowest bound) to 100 (the highest bound)	avgResponseTime;unbounded;unbounded,custom2;0;3
slo_rules_topic	A string value indicating the name of the topic which will be used to send messages containing the SLOs which should be respected by the application.	metrics.metric_list

broker_ip_url	A string value indicating the url which should be used to connect to the AMQP broker to send and receive messages.	tcp://localhost:61616?wireFormat.maxInactivityDuration=0
broker_username	A string value, which is the username to access the AMQP broker	User1
broker_password	A string value, which is the password to access the AMQP broker	userpassword
slo_violation_determination_method	A string value, indicating the method which is used to determine whether an SLO violation has occurred.	all-metrics
time_horizon_seconds	An integer value indicating the minimum time interval between two successive reconfigurations that the platform can support	900
maximum_acceptable_forward_predictions	An integer value indicating the maximum number of forward predictions for which the component will keep data	30

7.2.2 Component input

The triggering input of the SLO Violation Detector is a JSON message which informs the component about the SLOs which should be respected. The format of these SLOs is the following:

$$SLO \leftarrow \{Metric\}\{Operator\}\{Threshold\}$$

Metric: Any monitoring attribute which can be observed using the EMS can be used in the formulation of an SLO

Operator: Either greater than, greater or equal, less than or less than or equal.

Threshold: We assume that metric values used in the description of SLOs are real numbers, so any real number which can be handled by Java 9 can be used.

Multiple SLOs can be joined using an 'AND' or 'OR'-separated syntax.

Examples of AND and OR separated SLO's appear in Listing 13 and 15 below:

Listing 13 - A simple SLO rule

```
{
 "name": "_",
 "operator":"OR",
 "constraints":[
  {
   "name":"cpu_usage_too_high",
    "metric": "cpu_usage",
   "operator":">",
   "threshold": 80.0
  }
 ]
}
```

Listing 14 - An example of a complex SLO rule

```
{
  "name": "_",
 "operator":"OR",
  "constraints":[
  {

"name":"cpu_and_memory_or_swap_too_high",
     "operator":"AND",
     "constraints": [
      {
       "name":"cpu_usage_high",
       "metric":"cpu_usage",
       "operator":">",
       "threshold":80.0
      },
      {
       "name": "memory_or_swap_usage_high",
       "operator": "OR",
        "constraints": [
         {
          "name":"memory_usage_high",
          "metric":"ram_usage",
          "operator":">",
         "threshold":70.0
         },
         {
           "name": "disk_usage_high",
          "metric":"swap_usage",
          "operator":">",
          "threshold":50.0
         }
        ]
      }
     ]
    }
  ]
}
```

The simple SLO illustrated in Listing 13 states that the "cpu_usage" monitoring metric should stay ideally below 80 (percent), otherwise an SLO violation should be triggered. On the other hand, the SLO in Listing 14 is more complex and involves the use of three monitoring metrics, "cpu_usage", "free_ram" and "swap_usage", which should be below 70 and 50 (percent) respectively. The format illustrated in Listing 14 has been devised to allow nested AND-based or OR-based SLOs to be defined. The complex SLO rule in Listing 14 states that if (cpu_usage>80 AND (ram_usage>70 OR swap_usage> 50)) then an SLO violation should be triggered.

Apart from the non-periodical triggering input which configures its operation, the SLO Violation Detector also receives periodically predicted and actual metric values, from the Prediction Orchestrator and the Event Management System respectively. These values are the operands in the calculation of Severity and indicate the need to start an adaptation. Events containing the predicted values are described in Listing 2, while events containing real-time values are described in Listing 1.

When predicted metric values arrive, they are inspected to determine if the predictionTime they refer to is at least *time_horizon_seconds* away from the current time (one predction interval). Predictions should also not be too far into the future (more than *maximum_acceptable_forward_predictions* prediction intervals ahead). The values for t*ime_horizon_seconds* and *maximum_acceptable_forward_predictions* are configuration-time parameters of the component - especially t*ime_horizon_seconds* reflects the agility of the MORPHEMIC platform, its readiness to provide an updated topology.

7.2.3 Severity and Estimation of reconfiguration need

Whenever a message containing predicted monitoring attribute values over the thresholds described in an SLO is received, it is possible that a platform reconfiguration is needed. Choosing to reconfigure the platform at all times may lead to application instability, while choosing to reconfigure the application too rarely may lead to degraded application performance. To assist this decision, we assess the 'Severity' of the possible violation of the thresholds of the attributes used in the SLO, which quantifies its rough magnitude [19]. The concept of Severity is used to calculate two distinct adaptation indicators, which also correspond to relevant operational modes for the SLO Violation Detector.

In the 'all-metrics' mode, three meta-metrics are used; the confidence of the prediction estimate (PrConf), the absolute difference from the threshold (Delta) and the (sliding) rate of change of the particular attribute, ROC. To calculate Severity, we also use the sign of the Delta and ROC meta-metrics (either -1 or +1). It is reminded that higher values of Severity indicate that more pronounced changes to the application topology should be made (i.e., more VMs should be added/removed). We assume that the importance of each meta-metric is the same, therefore we assign the same weight to all meta-metrics. The calculation of normalized 'all-metrics' Severity for a PrConf, Delta and ROC triplet is illustrated in Equation 1.

$$Severity = \frac{\sqrt{PrConf^2 + DeltaSign \cdot Delta^2 + ROCSign \cdot ROC^2}}{\sqrt{3}}$$

Equation 1 - Calculation of 'all-metrics' Severity for a simple SLO

A careful observation of Equation 1 allows us to understand that depending on the DeltaSign and ROCSign values, and the meta-metric values, there may be situations in which the sum under the square root in the numerator is negative. In these cases, Severity cannot be defined.

In the 'prconf-delta' mode, only two of the three meta-metrics introduced above are used – PrConf and Delta – using the same definitions as in the all-metrics technique. This second mode is made available as an alternative allowing not to use the observed value twice as is done in the 'all-metrics' technique (the observed value there is used by the Delta and ROC metrics). While the concept of 'Severity' is used only when SLOs for multiple attributes are defined, yet we use this name to indicate the need for an adaptation in the simple case where only one SLO is involved as well. The calculation of the normalized 'prconf-delta' 'Severity' in the simplest case of one SLO appears in Equation 2

$$Severity = PrConf \cdot Delta$$

Equation 2 - Calculation of prconf-delta 'Severity' for a simple SLO

The normalized values for each meta-metric which are used in Equation 1 and Equation 2, are derived as described in Section 7.2.4.

7.2.4 Meta-metric calculation

7.2.4.1 Meta-metric value definitions

As illustrated in Equation 1, to find the Severity for a simple SLO there are three meta-metrics which first need to be calculated, PrConf, Delta and ROC. To calculate each of these metrics, we rely on the observations which have been made for a particular monitoring metric, and the predictions for its future value.

To calculate PrConf, we need to obtain the probability indicating the accuracy of a prediction, and the confidence interval width. The probability can be readily retrieved from the relevant field in prediction messages (see Listing 2) and is always positive. To obtain the non-normalized value for PrConf, we multiply this with the normalized confidence interval width as shown in Equation 3:

$$PrConf = PredictionProbability \cdot (1 - NormalizedConfidenceIntervalWidth)$$

Equation 3 - Calculation of PrConf meta-metric

where the definition of NormalizedConfidenceIntervalWidth appears in Equation 4 (assuming that the minimum and maximum Monitoring Attribute values differ):

$$NormalizedConfidenceIntervalWidth = \frac{ConfidenceIntervalWidth}{MaximumAttributeValue - MinimumAttributeValue}$$

Equation 4 - Calculation of normalized confidence interval width

To calculate the value of non-normalized ROC Equation 5 is used, with the exception of edge cases in which the real-time metric value is zero:

$$ROC = \frac{PredictedValue - RealtimeValue}{RealtimeValue}$$

Equation 5- Calculation of non-normalized ROC

To obtain the normalized percentage value for ROC, we set an upper limit of 100% to positive ROC values and lower limit of -100% for negative ROC values. Although the choice of the limit value is arbitrary, since the metric is normalized it is necessary to select appropriate limits which will allow more common, smaller ROC values to be considered, and not only the greatest ROC values which have been observed. Positive values of ROC result in ROCSign being +1, and negative values of ROC result in ROCSign being -1.

For example, if we consider that the normalized value of a metric jumps from 1 (realtime value) to 100 (predicted value), using Equation 5 the estimated non-normalized ROC value is 9900%. By contrast, a change of a metric value from 50 to 75, indicates a non-normalized ROC value of 50%. Using linear normalization, if the 9900% ROC value is assigned a normalized ROC value of 100% (for the purposes of this example), a non-normalized value of 50% would be assigned a value of ~0.51%. Having low ROC values in order to fully handle the (comparatively rare) highest ROC values significantly reduces the contribution of the ROC meta-metric in the calculation of Severity. On the other hand, using our normalization method, in the first case the non-normalized ROC value of 9900% is assigned a value of 100%, and the non-normalized ROC value of 50% is again normalized to a value of 50%.

The value of non-normalized Delta is calculated as shown in Equation 6 and Equation 7 for greater-than and less-than SLO rules respectively (with the exception of edge cases in which the maximum metric value is under the threshold value in a greater than rule, or the minimum metric value is over the threshold value in a less than rule):

$$Delta = \frac{PredictedValue - ThresholdValue}{MaximumValue - ThresholdValue}$$

Equation 6 - Calculation of non-normalized Delta meta-metric in 'greater-than' SLO rules

$$Delta = \frac{ThresholdValue - PredictedValue}{ThresholdValue - MinimumValue}$$

Equation 7 - Calculation of non-normalized Delta meta-metric in 'less-than' SLO rules

Similar to ROCSign, if the value of Delta is positive DeltaSign is +1, otherwise DeltaSign is -1.

Once the values for the PrConf, ROC and Delta meta-metrics are known, Severity can be calculated for the particular SLO.

7.2.4.2 Full Example of Severity calculation on a simple SLO

To illustrate the calculation of Severity, we will use the simple example of Listing 13 presented above. For the cpu_usage metric, the maximum metric value is 100% and the minimum metric value is 0%. Assuming that the realtime value of the metric is 50%, and the predicted value is 90%, with a prediction probability of 95% and a confidence interval of 10%, we can calculate the following meta-metrics:

$$PrConf = 0.95 \cdot (1 - \frac{0.10}{1-0}) = 0.855$$

$$ROC = \frac{0.9 - 0.5}{0.5} = 0.8$$

$$Delta = \frac{0.9 - 0.8}{1 - 0.8} = 0.5$$

Therefore, the all-metrics Severity for this simple SLO is calculated as follows:

$$Severity_{all-metrics} = \frac{\sqrt{0.855^2 + 0.8^2 + 0.5^2}}{\sqrt{3}} \cong 0.735$$

The prconf-delta Severity for this simple SLO is calculated as follows:

$$Severity_{prconf-delta} = 0.855 \cdot 0.5 \cong 0.473$$

7.2.4.3 Full Example of Severity calculation on a complex SLO

When more complex SLO expressions similar to the one expressed in Listing 14 need to be evaluated, it is necessary to aggregate the values of individual SLO expressions. The mode of the aggregation is different for each Severity calculation mode.

In the case of the all-metrics Severity calculation mode, to aggregate AND-separated SLOs we calculate the average of the Severity values of individual SLOs as shown in Equation 8.

$$OverallSeverity = \frac{slo\_1\_severity\ + slo\_2\_severity + \ldots\ + slo\_n\_severity}{n}$$

Equation 8 - Calculation of overall Severity value for a complex SLO, using the all-metrics calculation mode

To aggregate OR-separated SLOs, we calculate the maximum of the Severity values of individual SLOs.

To illustrate, consider the example complex SLO presented in

Listing 14, triggered if (cpu_usage>80 AND (ram_usage>70 OR swap_usage> 50)). Individual Severity values for the three SLOs (cpu_usage >80, ram_usage>70, swap_usage>50) are calculated as discussed in Section 7.2.3. Assuming that the Severity values for the CPU usage, RAM usage and Swap usage are found to be 0.75, 0.8 and 0.95 respectively, the overall Severity value of this complex SLO would be evaluated as shown in Equation 9:

$$OverallSeverity = average\ (0.75, \max(0.8{,}0.95)) = \frac{0.75 + 0.95}{2} = 0.85$$

Equation 9 - Calculation of overall Severity Value for an example complex SLO, using the all-metrics calculation mode

In the case of the prconf-delta Severity calculation mode, to aggregate AND-separated SLOs we use the definition of Severity, applied onto Severity values of individual SLOs using:

$$OverallSeverity = \frac{\sqrt{slo\_1\_severity^2\ + slo\_2\_severity^2 + \ldots\ + slo\_n\_severity^2}}{\sqrt{n}}$$

Equation 10 - Calculation of overall Severity value for a complex SLO, using the prconf-delta calculation mode

To aggregate OR-separated SLOs, we calculate the maximum of the Severity values of individual SLOs as in the all-metrics mode. Assuming the values used in the above example, the overall Severity value of the complex SLO would be evaluated as shown in Equation 11:

$$OverallSeverity = \frac{\sqrt{0.75^2 + 0.8^2 + 0.95^2}}{\sqrt{3}} \cong 0.80$$

Equation 11 - Calculation of overall Severity value for an example complex SLO, using the prconf-delta calculation mode

7.2.5 Component output

Having calculated the severity, the component can provide an estimate on the need for a reconfiguration of the application. In order not too trigger the reconfiguration of the platform too often, we opt to only send a reconfiguration message when the value of Severity is over the median Severity value (mSV). To calculate mSV, we calculated the Severity values which correspond to all acceptable meta-metric integer values (see set S definition in Equation 12).

$$S: (s \in Severity(roc, prconf, delta) \mid roc, prconf, delta \in (\mathbb{Z}\ \cap\ [-100{,}100] \times [0{,}100] \times [-100{,}100]\ ))$$

Equation 12 - Valid integer Severity values

The ranges for the percentage values of the roc and prconf meta-metric were chosen as shown in Equation 12 as they reflect the maximum and minimum values for the particular metric (by definition). Moreover, the range for the precentage values of the delta meta-metric was chosen to allow taking into account predictions with a high rate of change and high PrConf value while still having a value which is under an SLO threshold, or predictions with a high negative rate of change and high PrConf value, having a value which is over an SLO threshold. While we allowed negative delta values, according to Equation 6, Equation 7 and Equation 12 we do not consider predictions more than (MaxValue-ThresholdValue) or less than (ThresholdValue-MinValue) from the threshold.

Then we calculate the Severity for each element $s \in S$. The results were normalized, as they were calculated over integer percentage values – this choice was made to allow for precision while still maintaining a reasonable computing time. Based on the normalized values, mSV – equal to the normalized median value of set S – was found to be 0.4 for the all-metrics computation mode and 0.0652 for the prconf-delta computation mode. Therefore, when a Severity value is calculated to surpass mSV, we assign a reconfiguration probability as indicated in Equation 13.

$$ReconfigurationProbability = 0.5 + \frac{SeverityValue - mSV}{2 * (1 - mSV)}$$

Equation 13 - Platform reconfiguration probability

The reconfiguration probability value, which is calculated, is sent using the event specification in Listing 15.

```
{
  "severity":0.9064,
  "predictionTime":1626181860,
  "probability":0.92246521
}
```

Listing 15 - The SLO Violation Detector output event

The value of the “severity” field is equal to the value which is calculated using Equation 1,Equation 2,Equation 8 and Equation 10. The value of the “probability” field corresponds to the value which is calculated using Equation 13. Finally, the value of the “predictionTime” field corresponds to the time point for which all calculations were made by the component.

8 Metasolver

The Metasolver is the Upperware component responsible for coordinating the solving process, which generates the multi-cloud application deployments. The solving process involves several steps like, selecting the appropriate solver(s), validating the produced solution, and also triggering the application reconfiguration (i.e., re-running the solving process) when an SLO violation is detected by the monitoring system.

The MELODIC project was focused on responding to SLO violations when they actually occur, by reconfiguring a multi-cloud application, hence MELODIC implements a reactive approach. Morphemic, on the other hand, goes further by introducing the capability to proactively respond to situations that could result in SLO violations. In other words, MORPHEMIC supports both reactive and proactive handling of SLO violations.

In the context of the MORPHEMIC project, the scope of Metasolver is extended in order to assist the forecasting mechanism in generating predictions about forthcoming SLO violations. Based on them, Metasolver can also trigger application reconfiguration in order to prevent the (predicted) SLO violation from actually occurring. In this section the required updates for supporting the forecasting mechanism are reported. These updates target at two main directions:

- Update of CP model with predicted data.
- Trigger application reconfiguration on the basis of a predicted SLO violation.

8.1 CP model update with predicted data

CP model stores information that MORPHEMIC solvers use as input in order to compute a solution to the CP problem (also captured in CP model). The solution represents a new application configuration / deployment. This information can either be actual measurements and values originating from the monitoring system (EMS), or they can be predicted values the forecasting mechanism generates. This way solvers can be used to compute solutions (i.e., application deployments) that fit either to the current situation or to a predicted state of the application.

Every time an application reconfiguration is triggered, Metasolver must update CP model with the corresponding information.

- If a SLO violation actually occurred, then the new application configuration must be computed based on the (actual) information collected by monitoring system (EMS) at the time of SLO violation.
- If forecasting mechanism predicts that an SLO violation will shortly occur, then a new application configuration must be computed based on the information used by forecasting mechanism. This information includes predicted values and are also generated by forecasting mechanism.

In other words, CP model must be updated with the context (i.e., information) led/will lead to an SLO violation. For this reason, Metasolver needs to receive both the actual as well as the predicted values for each variable included in CP model. To this end Metasolver subscribes to all event topics where events pertaining to the needed information are published. By convention the names of the topics that convey predicted values are prefixed with `"prediction."` indication. Topics that convey actual values use no special indication. For instance, `AverageCpuLoad` would be a topic for actual CPU load values, whereas `prediction.AverageCpuLoad` would be used for predicted CPU load values.

The default behaviour of Metasolver can be modified by changing the following settings in configuration file.

Table 39 – Metasolver's Configuration Settings

Configuration Setting	Allowed/Default values	Description
`predictionMonitoringEnabled`	`true, false` *`Default: true`*	Flag for enabling or disabling the monitoring of predicted values
`predictionTopicFormatter`	`A valid ActiveMQ topic name pattern` *`Default: prediction.%s`*	Specifies the pattern for creating the predicted values topic names, based on their actual values counterparts. The %s is replaced with the actual values topic name
`predictionTopicPattern`	`A valid regular expression` *`Default: ^prediction\.(.+)$`*	Regular expression for extracting the actual values topic name from its predicted values counterpart. The expression part between (..) captures the actual values topic name.
`PredictionRegistryCleanupRate`	`Any long value` *`Default: -1L`*	Specifies the time-to-live of the contents of predicted values cache. After that period the predicted values cache is purged.
`PredictionRegistryCleanupAfterScaleEvent`	`true, false` *`Default: true`*	Controls whether the predicted values cache will be purged after triggering an application reconfiguration.

Metasolver is automatically configured by EMS Translator with the topics it needs to monitor. Next, Metasolver subscribes to each (configured) topic as well to its predicted values counterpart and starts caching the values received. Specifically, it caches actual and predicted values in two distinct caches. Upon application reconfiguration it will update CP model with the values of the appropriate cache. Metasolver can be configured to periodically persist the contents of the actual values cache into CP model. The default cache persistence period is 30 seconds, but it can be changed in Metasolver configuration file. The corresponding settings are presented in the table below.

Table 40 – Metasolver's Configuration Settings regarding CP Model updates

Configuration Setting	Allowed/Default values	Description
`cpModelUpdateEnabled`	`true, false` *`Default: true`*	Flag for enabling or disabling the periodic actual values cache persistence to CP model
`cpModelUpdateInterval`	`any long positive value (>0)` *`Default: 30000L`*	Specifies the actual values persistence period in milliseconds

8.2 Application reconfiguration due to predicted SLO violation

The second set of Metasolver updates concern the triggering of application reconfiguration that can be the result of either an actual SLO violation or a predicted one. The forecasting mechanism, apart from the predicted CP model variable values, it also generates predictions of anticipated SLO violation (in the near future). These predictions are published as SLO violation events in a preconfigured topic; by default, this topic is `prediction.slo_severity_value` (see also section 3.3).

Metasolver automatically subscribes to the topics where forecasting mechanism publishes predicted SLO violation events, and upon receiving one with probability higher than a preconfigured threshold (default threshold is 0.5) it starts a new application reconfiguration. As mentioned in the previous section, before triggering the reconfiguration Metasolver will first store the predicted values from the corresponding cache into CP model, and only then it will signal Upperware control plane to execute the application reconfiguration. SLO violation event with probability lower than threshold are ignored.

Table 41 – Metasolver's Configuration Settings regarding reconfiguration threshold

Configuration Setting	Allowed/Default values	Description
`pubsub.commonTopics[0].name`	`A valid ActiveMQ topic name` *`Default: prediction. slo_severity_value`*	The name of topic where predicted SLO events are published
`reconfigurationProbability Threshold`	`Double value in range 0..1` *`Default: 0.5`*	The probability threshold for a predicted SLO violation event to trigger a new application reconfiguration

In order to prevent two or more reconfiguration process instances from occurring simultaneously, Metasolver has been extended to ignore SLO violation events (either actual or predicted) when a reconfiguration process instance is still in progress. When Metasolver is notified that a new solution has been created it will re-enable reconfiguration triggering. In addition, an optional timeout period can be defined in Metasolver configuration file. This period starts when a reconfiguration is triggered and upon expiring the reconfiguration is reenabled. This is a precaution against failures in reconfiguration process that could otherwise leave Metasolver in a state where reconfiguration triggering is permanently disabled. The simultaneous reconfiguration blocking must be explicitly enabled in Metasolver configuration file.

Moreover, in order to prevent very frequent reconfigurations, which can lead to application instability, or failure, or reduced level of service, Metasolver can be configured to refrain from starting new reconfigurations for a predefined *cool down period*. The default value is 0 (i.e., no cool down) but it can be changed in Metasolver configuration file.

Table 42 – Metasolver's Reconfiguration Settings

Configuration Setting	Allowed/Default values	Description
`reconfigurationBlockingPeriod`	`Any long value` *`Default: 0L`*	Specifies the reconfiguration cool down period in milliseconds where no new reconfigurations will be started
`PreventConcurrentReconfigura tions`	`true, false` *`Default: false`*	Flag for enabling or disabling the prevention the start of new reconfigurations while a running one has not ended, or the prevention timeout period has not yet expired
`PreventConcurrentReconfigura tionsTimeout`	`Any long value` *`Default: -1L`*	Specifies the period in milliseconds where the concurrent reconfiguration prevention is active

8.3 Miscellaneous updates

A last update introduced in Metasolver, in the context of Morphemic, is the addition of the capability to publish debug events to a preconfigured topic. These events can be used to monitor and audit Metasolver activity, like CP model updates, starting of new reconfigurations, or reenabling reconfiguration after cooling down period. The debug events can be sent to an ActiveMQ broker other than that used for publishing the actual or predicted values. This feature is by default disabled, but it can be activated and configured using the following settings in Metasolver configuration file.

Table 43 – Metasolver's Debuging Configuration Settings

Configuration Setting	Allowed/Default values	Description
debugEvents.enabled	`true, false` *`Default: false`*	Flag for enabling or disabling the publish of audit events
debugEvents.topicName	`A valid ActiveMQ topic name` *`Default: Metasolver_debug`*	The topic where audit events will be published
debugEvents.url	`ActiveMQ broker URL` *`Default: n/a`*	The URL of the ActiveMQ broker where audit event will be published
debugEvents.username	`ActiveMQ broker username` *`Default: n/a`*	The username for connecting to ActiveMQ broker (if needed)
debugEvents.password	`ActiveMQ broker password` *`Default: n/a`*	The password for connecting to ActiveMQ broker (if needed)
debugEvents.certificate	`ActiveMQ broker certificate` *`Default: n/a`*	The certificate for connecting to ActiveMQ broker (if needed)
debugEvents.clientId	`ActiveMQ client Id` *`Default: n/a`*	Optional client id to be used when connecting to the ActiveMQ broker

9 Conclusions

This deliverable provided a comprehensive description of all aspects of EMS, which is a federated monitoring system with self-healing capabilities that is able to aggregate, process and propagate the current monitoring context of multi-cloud applications deployed by MORPHEMIC, to allow the platform to proactively respond to their needs (i.e., cope with SLO violations). We discussed and presented all the details of the MORPHEMIC monitoring approach, and we analysed the architecture of components involved in the proactive adaptation process. We sketched the relevant processes including the acquisition of both real-time and predicted metrics related to the QoS specification of the application and we listed all the aspects of eight forecasters that employ advanced time-series forecasting methods for proactively reconfiguring Morphemic-enabled cloud applications.

The next steps of this work involve the further improvement of the MORPHEMIC forecasting module, mainly towards the investigation of other methods for ensembling predictions from different forecasters but also examine the use and implementation of additional forecasters that enable multi-variate forecasting algorithms. Finally, the next steps involve reporting on all WP2 task implementations that also concern the description of the final algorithms used for estimating the proactive utility along with the theoretical and practical assessment of the approach.

References

[1] Verginadis Y., Patiniotakis I., Tsagkaropoulos A., Totow J.-D., Raikos A., Mentzas G., Apostolou D., Tzormpaki D., Magoutas Ch., Bothos E., Stefanidis V. (2021). Design of a self-healing federated event processing management system at the edge. MORPHEMIC Deliverable

[2] SWIM: The scalable membership protocol. Available online at https://www.brianstorti.com/swim/ Retrieved on 8/1/2021

[3] Shcherbakov, Maxim Vladimirovich, et al. "A survey of forecast error measures." World Applied Sciences Journal 24.24 (2013): 171-176.

[4] Chai, Tianfeng, and Roland R. Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature." Geoscientific model development 7.3 (2014): 1247-1250.

[5] Khair, Ummul, et al. "Forecasting error calculation with mean absolute deviation and mean absolute percentage error." Journal of Physics: Conference Series. Vol. 930. No. 1. IOP Publishing, 2017.

[6] Kreinovich, Vladik, Hung T. Nguyen, and Rujira Ouncharoen. "How to estimate forecasting quality: A system-motivated derivation of symmetric mean absolute percentage error (SMAPE) and other similar characteristics." (2014).

[7] S. Smyl, 'A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting', International Journal of Forecasting, vol. 36, no. 1, pp. 75–85, Jan. 2020, doi: 10.1016/j.ijforecast.2019.03.017.

[8] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, 'The M4 Competition: 100,000 time series and 61 forecasting methods', International Journal of Forecasting, vol. 36, no. 1, pp. 54–74, Jan. 2020, doi: 10.1016/j.ijforecast.2019.04.014.

[9] 'Papers with Code - Fast ES-RNN: A GPU Implementation of the ES-RNN Algorithm'. https://paperswithcode.com/paper/fast-es-rnn-a-gpu-implementation-of-the-es (accessed Dec. 21, 2021).

[10] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, 'N-BEATS: Neural basis expansion analysis for interpretable time series forecasting', *arXiv:1905.10437 [cs, stat]*, Feb. 2020, Accessed: Sep. 14, 2021. [Online]. Available: http://arxiv.org/abs/1905.10437

[11] Pytorch-forecasting: https://github.com/jdb78/pytorch-forecasting

[12] Lim, Bryan, et al. "Temporal fusion transformers for interpretable multi-horizon time series forecasting." International Journal of Forecasting (2021).

[13] AiY,LiZ,GanM,ZhangY,YuD,ChenW,JuY(2019)Adeep learning approach on short-term spatiotemporal distribution forecasting of dockless bike-sharing system. Neural Comput Appl 31(5):1665–1677

[14] Li J, Dai Q, Ye R (2018) A novel double incremental learning algorithm for time series prediction. Neural Comput Appl 31(10):6055–77

[15] Zheng J, Fu X, Zhang G (2019) Research on exchange rate forecasting based on deep belief network. Neural Comput Appl 31(1):573–582

[16] Zou W, Xia Y (2019) Back propagation bidirectional extreme learning machine for traffic flow time series prediction. Neural Comput Appl 31:7401–7414

[17] Python statsmodels library: https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html

[18] Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on 1/12/2021.

[19] Tsagkaropoulos, A., Verginadis, Y., Papageorgiou, N. et al. Severity: a QoS-aware approach to cloud application elasticity. J Cloud Comp10, 45 (2021). https://doi.org/10.1186/s13677-021-00255-5

[20] Durbin, James, and Siem Jan Koopman. 2012. Time Series Analysis by State Space Methods: Second Edition. Oxford University Press.

[21] Zunic, E., Korjenic, K., Hodzic, K., & Donko, D. (2020). Application of facebook's prophet algorithm for successful sales forecasting based on real-world data.

[22] Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., & Wang, Y. (2019). Gluonts: Probabilistic time series models in python.