www.qualiMaster.eu

QualiMaster

A configurable real-time Data Processing Infrastructure mastering autonomous Quality Adaptation

Grant Agreement No. 619525

Deliverable D4.3

Work-package	WP4: Quality-aware Configuration and Adaptation of Stream Processing Pipelines
Deliverable	D4.3: Quality-aware Processing Pipeline Adaptation V2
Deliverable Leader	SUH
Quality Assessor	Ekaterini Ioannou
Estimation of PM spent	29
Dissemination level	PU
Delivery date in Annex I	31.07.2016
Actual delivery date	01.08.2016
Revisions	18
Status	Final
Keywords:	Quality-aware configuration, quality-aware adaptation, enactment, pattern analysis, quality taxonomy, adaptation components, adaptive crawling, event detection, source volume prediction, algorithm profiles, tool support

Disclaimer

List of Authors

Partner Acronym	Authors
MAX	-
LUH	Andrea Ceroni, Christoph Hube
SUH	Holger Eichelberger, Sascha El-Sharkawy, Cui Qin, Roman Sizonenko, Christopher Voges
SPRING	Stefan Burkhard
TSI	Gregory Chrysos

Table of Contents

Executive summary

In the QualiMaster project, quality-aware configuration aims at defining the configuration options and at characterizing the adaptation space in an integrated way. Based on the quality-aware configuration, the desired pipelines can be validated and, finally, deployable adaptive pipelines can be derived in a fully automated way. At runtime, the infrastructure monitors and analyzes the execution and decides about modifications at runtime to achieve given processing performance and result quality requirements in a changing environment, e.g., due to evolving characteristics of the input data streams or due to the resource consumption of other pipelines running on the same cluster. Realizing the vision of configurable and adaptive Big Data analysis pipelines requires tailored approaches to configuration and adaptation as well as realizing components and tools.

In this deliverable, we report on both, recent improvements and extensions to the concepts underlying the configuration and the adaptation approach in QualiMaster. These include a refinement of the results discussed in previous deliverables as well as improvements of the related tool and infrastructure components realized so far. Regarding configuration, this deliverable includes a discussion of new configuration concepts for consolidating the current state and for enabling adaptation, the related derivation steps, optimization and model transformation steps as well as advances in the tool support. Regarding adaptation, we provide an overview on the state of the concepts, mechanisms and components for reactive, proactive, cross-pipeline and reflective adaptation. Finally, we discuss the validation and evaluation of the components focusing on functional and performance aspects.

1 Introduction

Quality-aware configuration and adaptation of data processing pipelines is central to the QualiMaster project. Configuration includes several activities: (1) designing the Configuration Meta Model that defines all customization options for configuration and runtime as well as their interrelationships in terms of constraints – we use here a topological model [15]; (2) defining the Configuration for a specific infrastructure installation including the resource pool, algorithms, algorithm families and pipelines; (3) validating the Configuration and determining so far undefined configuration settings by value propagation; and (4) instantiating the Configuration into deployable adaptable code. Most activities on refining the Configuration (Meta) Model[1] and the respective tooling aim at consolidating the achieved state, improving the structure of the model / the user accessible configuration options as well as integrating configuration settings needed for improved adaptations. Particular examples of modifications discussed in this deliverable are sub-pipelines for ensuring the adaptation capabilities of distributed algorithms, permissible parameters for stakeholder applications, replay sinks increasing the functionality of the stakeholder applications or further adaptation opportunities such as load shedding and re-configurable monitoring.

The quality-aware configuration model, for performance reasons also an automatically optimized version that we introduced in this deliverable, forms the foundation for the adaptation knowledge [13]. At runtime, observations made as part of continuously monitoring the infrastructure and the running pipelines are mapped into the runtime part of the configuration. In a regular fashion, the constraint-based analysis determines deviations from the expected behaviour. These deviations are signalled as adaptation triggers (also allowing triggers from internal components, external administrative triggers or even application-specific user-triggers) to the adaptation component and processed by a flexible adaptation script. The adaptation script operates on both, the configuration and the runtime information and adjusts the runtime configuration to reflect adaptation decisions. The configuration changes are finally turned into infrastructure commands changing the data processing at runtime, which may lead to complex enactments such as efficiently changing among distributed data processing algorithms [32]. In addition to reactive adaptations, this deliverable introduces predictive, cross-pipeline and reflective adaptations, the actual focus of WP4. Predictive adaptations are supported by (1) data volume prediction models for the data sources and (2) algorithm profiles capturing the quality characteristics of individual algorithms within their parameter and distribution space supporting the runtime decision. Both mechanisms are based on learning knowledge before and at runtime. Cross-pipeline adaptations particularly take resource allocations over multiple pipelines into account. The reflective adaptation operates at meta-level and aims at learning form ongoing adaptations and their effect in order to help improving configuration and adaptation.

The deliverable is structured as follows: In Section 1, we discuss improvements to the QualiMaster configuration approach including new concepts for the Configuration Meta Model, related modifications of the infrastructure instantiation process, enhancements to the reasoning support for configuration and in particular runtime, techniques for optimizing the configuration model for runtime reasoning and improvements to the tool support. In Section 3, we focus on the adaptation capabilities, including advancements for the enactment patterns as well as the state of the reflective, proactive (including source volume prediction and algorithm profiling), cross-pipeline and reflective adaptation. In Section 4, we present results from validating and evaluating the components developed in WP4 realizing the aforementioned concepts. Finally, in Section 5, we conclude this deliverable and give an outlook on future work on WP4 for the remainder of the project.

[1] As done in D4.1 / D4.2, we use the term ”Configuration Meta Model“ to refer to the definition of the possible configuration settings and their interdependencies in terms of constraints, “Configuration” to refer to one instance of the meta model and „Configuration (Meta) Model“ to refer to the combination of both, the meta-model and its instance(s).

Relation to other deliverables:

- This WP takes the adaptation-related requirements discussed in D1.1/D1.2 (refined in D4.1/D4.2) as foundation and discusses approaches and techniques for realizing the requirements as well as components realizing the approaches and techniques.
- D2.3 introduces new algorithms to be configured and, in particular, the data replay mechanism for providing stakeholder applications access to past analysis results. D4.3 discusses concepts and design options on how to configure the data replay mechanism in QualiMaster pipeline and to turn the configuration into instantiated code. Further, there is a synchronized collaboration between WP2 and WP4 regarding work on event detection and event prediction (as detailed in Section 3.3.2.2).
- Akin to D4.2 and D3.2, D4.3 and D3.3 are linked through the configuration of the hardware-based processing, the generated integration of the hardware-based processing into the pipeline execution, the protocols needed to communicate the required algorithms to hardware (required information is stated in the configuration), the hardware-related enactment patterns and the consideration of hardware in the adaptation scenarios.
- D4.3 relies on, refines, extends and implements the concepts discussed in D4.1 and D4.2. In addition, D4.3 discusses how the adaptation-specific requirements from D4.1 and D4.2 are turned into concepts, designs and implementing components.
- D4.3 has a bi-directional interdependency with D5.3 as components developed in WP4 and described in D4.3 are integrated into the QualiMaster infrastructure in WP5 and components from WP5, such as the Coordination or the Monitoring Layer are required for WP4. D4.3 will indicate links to work in WP5, such as the evolution of the QualiMaster infrastructure instantiation process, the use of components or the (early) integration of WP4 components into the QualiMaster infrastructure. The actual state of the integration will be documented in D5.3 (due in the following months).
- D4.3 is also related to D6.1 and D6.2, in particular in terms of the evaluations done there (as formal feedback to component development and the application protocol that the stakeholder applications use to subscribe to and communicate with the QualiMaster application. As part of the communication protocol, stakeholder applications can send user triggers to the QualiMaster infrastructure to indicate user requests regarding the actual processing, which can lead to adaptations of the running pipelines. For easing the creation of new stakeholder applications, WP6 and WP4 are now linked over pipeline setup files that describe the external “interface” of pipelines including the supported pipeline-specific protocol and the data delivered by the pipeline. The setup files are implicitly in the configuration (e.g., as permissible parameters), instantiated using the pipeline instantiation process from WP4 and used by the stakeholder design environment in WP6.

Responses to the reviewer comments:

- Enabling the infrastructure to “learning to switch” is an important capability and fully in line with the plans of WP4 to enable proactive and reflective adaptation. For this purpose, WP4 develops methods and components for source volume prediction (Section 3.3.2.1) and algorithm / component profiling (Section 3.3.2.3), both enabling the infrastructure to learn.
- WP2 and WP4 are working on some similar, but also different components, in particular regarding for event detection and event prediction. To improve synchronization, WP2 and WP4 share team members on these components, hoping to increase synergy. Section 3.3.2.2 details this interaction.
- Following the argumentation given in D4.2, WP4, WP5 and WP6 understand personalization as an enhancement of the stakeholder applications. Here, personalization may indicate potentially relevant adaptation triggers to the end-user. Basically, we consider quality measures on the infrastructure side as pipeline-specific rather than user-specific. If the adaptation experiments indicate that user-dependent quality measures are beneficial, we will include them. However, we also aim at avoiding an individual pipeline per user rather than balancing the individual requirements (with a priority on the internal requirements of the pipelines and the infrastructure).
- WP4 is working on methods to reduce the noise produced by the adaptive Twitter crawler and therefore to focus on extracting only the central terms for a market player. We present a new approach and further plans in Section 3.2.1.

2 Configuration

Configuring the QualiMaster infrastructure in terms of algorithms, families, and pipelines is central to the project for achieving flexibility and reuse using concepts and techniques from Software Product Line Engineering [12, 28, 31]. Due to our topological configuration approach [15], we are able to model pipelines and fully instantiate deployable code based on a type-safe composition of existing data analysis algorithms. Moreover, due to runtime variabilities and constraints, the configuration becomes a central knowledge artifact to the runtime adaptation.

In Deliverable D4.1, we explained the basics of the configuration approach and discussed the initial Configuration Meta Model. In Deliverable D4.2, we improved the Configuration Meta Model based on feedback of the QualiMaster partners. In this deliverable, we consolidate the configuration approach by refining and adding concepts that are important for the adaptation, but also aim at supporting the user, i.e., the Pipeline Designer, the Adaptation Manager and the Infrastructure Administrator. For example, we introduce the concept of sub-pipelines or a concept for detailing the external pipeline interface use by the stakeholder applications.

In Section 2.1, we discuss the extended configuration concepts. In Section 2.2, we discuss the state of validating and analyzing the validity of configurations through reasoning and an approach on how to optimize the configuration for runtime, in particular to speed up and achieve scalability of the adaptation phases monitoring, analysis and planning. In Section 2.3, we provide a brief overview on the realization of the new configuration concepts in terms of the infrastructure instantiation process. Finally, in Section 2.4, we discuss changes to the QualiMaster configuration tooling, the QualiMaster infrastructure configuration tool (QM-IConf).

2.1 Extended configuration concepts

The QualiMaster Configuration (Meta) Model plays an essential role in supporting the configuration and instantiation of the QualiMaster infrastructure and the running pipelines. In Deliverable 4.1, we introduced the basic concepts and the design of the Configuration Meta Model, which specifies configurable elements in terms of the following:

- Resource Pool supporting software- and hardware-based execution (D4.1, Section 7.2.3),
- Data Management entities including data sources, data sinks and data elements (D4.1, Section 7.2.4),
- Data Processing Algorithms performing the actual processing in terms of their input/output item types, algorithm parameters, and implementation artifact information (D4.1, Section 7.2.5),
- Algorithm Families grouping algorithms of the same functionality under the same input/output item types and algorithm parameters (D4.1, Section 7.2.6),
- Data Processing Pipelines given in terms of their topological structure linking sources, families, storage elements and sinks as a data flow graph (D4.1, Section 7.2.7) ,
- Observables for monitoring the execution at runtime (D4.1, Section 7.2.2),
- High-level Adaptation settings allowing the Adaptation Manager to influence the runtime adaptation (D4.1, Section 7.2.8),
- Global Infrastructure Settings defining the repositories to use as well as the set of active data processing pipelines (D4.1, Section 7.2.9).

In Deliverable 4.2, we extended Configuration Meta Model in order to meet new or changed requirements driven by the ongoing research work in the QualiMaster project (Section 3.2). In particular, we introduced:

- Arbitrary data item types supporting the Pipeline Designer / Algorithm Provider to define domain specific types for data items used for configuring algorithms and families (D4.2, Section 3.2.1),

- User-defined item names for a meaningful identification of input / output data items in the instantiated source code, in particular in the generated algorithm family interfaces (D4.2, Section 3.2.2),
- A textual description of algorithms supporting the user in identifying / selecting desired algorithms more easily (D4.2, Section 3.2.3),
- Extended resource information of the reconfigurable hardware for enabling adaptation of hardware-based algorithms (D4.2, Section 3.2.4),
- Artifact specifications enabling automated integration / deployment into the Processing Elements Repository via (generated) Maven build specifications (D4.2, Section 3.2.5),
- A debug mode to support development of data processing pipelines in the configuration, which can be disabled for production pipelines (D4.2, Section 3.2.6).

On the one side, the partners using the approach to configure their algorithms and pipelines gave us feedback that some concepts can be improved to simplify the development of QualiMaster pipelines, namely the need to model and instantiate distributed algorithms, but also to separate some of the concepts, such as hardware / software-based algorithms, to be able to focus better on the required configurations settings. On the other side, ongoing work on adapting the QualiMaster infrastructure requires more (runtime) configuration options in order to be able to re-configure the infrastructure and the pipelines and, finally, to enact adaptive decisions. Based on the previous version of the QualiMaster Configuration Model summarized above, we describe now the extended concepts. The extended configuration concepts are:

- **Sub-pipelines** to enable modelling, instantiation and (more detailed) adaptation for distributed algorithms. In particular, this simplifies the Algorithm Providers' work by automating tedious and error-prone implementation work for sub-topology structures, avoids the manual inclusion of monitoring probes or specific code for passing adaptation parameter signals within the distributed algorithm implementation. Moreover, generating sub-topologies enforces type-safety and algorithm interfaces also on that level, finally allowing the Algorithm Provider to focus on the algorithmic details of the data processing rather than the pipeline structure specific to the Execution System.
- **Permissible parameters** introduced in each pipeline processing node to allow the Pipeline Designer to distinguish between external algorithm parameters, for which runtime changes can be requested by the QualiMaster stakeholder applications (user-triggers) and private, internal parameters only accessible to the adaptation mechanism and the algorithms running in a pipeline (via adaptation).
- **Replay sink** supporting the configuration and instantiation of the data replay mechanism allowing QualiMaster stakeholder applications to retrospectively explore past processed results on historical data (for more details, see D2.3).
- **Algorithm concept refinement** providing a better structured design of the algorithm configuration, i.e., refining software and hardware algorithm into own types and explicitly handling specific configuration settings for the different types of algorithms. In particular, this supports the infrastructure administrator focusing on the specific (required) configuration settings for the individual algorithms.
- **Load shedding** acting as a run-time adaptation allowing processing elements to drop unprocessed data items to reduce processing load when the overall load exceeds given available resources [3, 11]. Adding load shedding to the Configuration Meta Model is required to enable this form of adaptation.
- **Adaptive monitoring** enabling the runtime re-configuration of the monitoring periods of various infrastructure parts.

Below, we will discuss the new concepts and their integration into the QualiMaster Configuration Meta Model in the sequence given above in individual subsections.

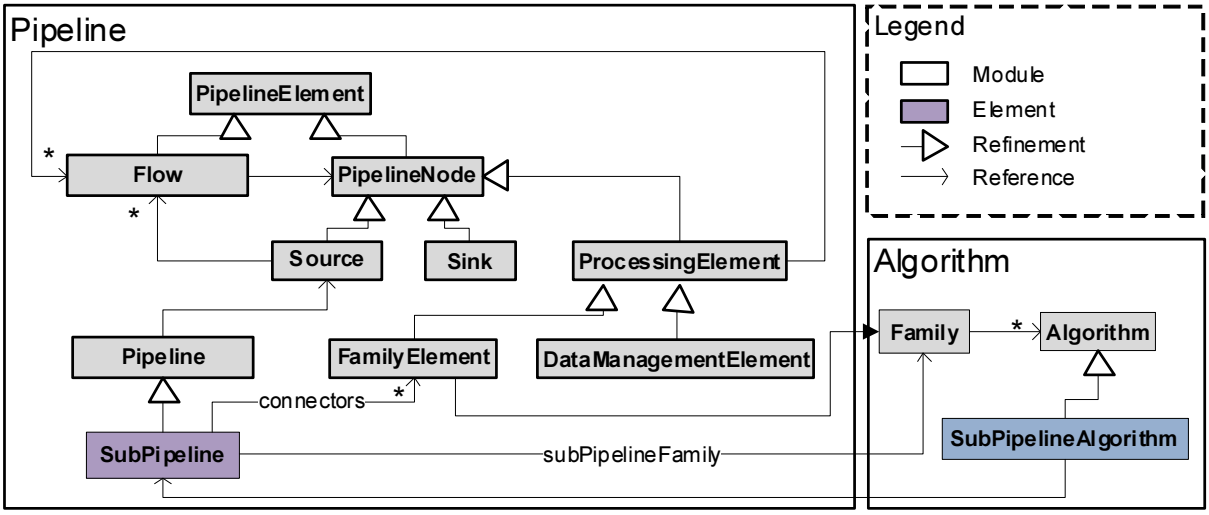

Figure 1: Fragment of the Configuration Meta Model showing concepts for sub-pipelines and their integration with to already existing configurable elements.

2.1.1 Sub-Pipelines as Algorithms

Sub-pipelines represent distributed algorithms in QualiMaster pipelines, i.e., algorithms, which consist of distributable pipeline nodes running in parallel on a cluster. Initially, sub-topologies were implemented by the Algorithm Providers using Storm concepts. As already discussed in Deliverable D4.2, such manual implementations can limit the adaptation on the processing nodes of a distributed algorithm as the structure of the algorithm is not stated in the configuration. Besides that, the consortium observed that manually implementing distributed algorithms in terms of sub-topologies is rather error-prone, i.e., minor coding errors can led to days of debugging. Moreover, the actual form of integration prevents future forms of integrating such sub-topologies into the main pipeline, e.g., to run them as separate Storm topologies to increase the control over the used resources. Such changes in the integration can easily be applied in a uniform way through the instantiation / code generation.

To overcome these obstacles, the QualiMaster consortium decided to configure distributed algorithms in terms of sub-pipelines (the configuration-level concept of sub-topologies) by reusing the existing pipeline conception and aiming at the generation of the implementing code based on the configured pipeline nodes, algorithm families and algorithms.

Figure 1 illustrates the related part of the Configuration Meta Model representing the configurable element `SubPipeline` as well as its relations to other elements. It is important to note that a sub-pipeline acts as an algorithm in the including pipeline and, thus, must become member of a respective algorithm family. Basically, a `SubPipeline` refines `Pipeline`, i.e., we consider a `SubPipeline` as a refined pipeline consisting of a data flow graph. As already mentioned above, a `SubPipeline` acts as a (distributed) algorithm (`SubPipelineAlgorithm`). In essence, such a distributed algorithm is specified through a `SubPipeline` as indicated by the reference from `SubPipelineAlgorithm` to `SubPipeline`. Following already introduced configuration concepts, the `SubPipelineAlgorithm` specifies the input/output item types as well as the algorithmic parameters, and, thus, imposes structural and type restrictions on the implementing `SubPipeline`.

Figure 2 depicts the IVML fragment of the definition of the `SubPipeline`. We equip the `SubPipeline` with a set of family elements acting as connectors. The `connectors` play an essential role in passing the input data from the including pipeline to the sub-pipeline (required by the depicted constraint). In terms of the configuration sequence, the sub-topology algorithm shall be configured before including it into families.

Enabling the user to configure sub-pipelines (this applies similarly to pipelines) also requests more detailed information about the connecting flows, in particular if the underlying execution system

shall distribute data based on information in the data items. To provide this information and to keep the model consistent, we allow linking back from a data flow to the respective output item types in the processing element where the flow originates.

It is important to note that the introduction of sub-pipelines into the Configuration (Meta) Model introduces cyclic dependencies among the configuration modules, e.g., pipelines link to algorithms via families and family nodes, while distributed algorithms link (back) to sub-pipelines. As the configuration is organized in a hierarchical fashion as explained in D4.1, the expressions assigning values to the configuration variables cause further cyclic dependencies. Respective capabilities for handling such cyclic configurations have been introduced in the Configuration Core, i.e., in EASy-Producer [14, 17], and became available to the entire QualiMaster infrastructure.

2.1.2 Permissible Parameters

As discussed in deliverables D1.2 and D4.2, we consider user-triggers as one form of adaptation in QualiMaster. In this form of adaptation, the user may request a change of the processing settings via the stakeholder application leading to a subsequent adaptation request on the related data processing pipelines. In the last version of the QualiMaster Configuration Meta Model as discussed in Deliverable D4.2, all configured algorithm parameters were considered as permissible for user-triggered adaptation. However, such a broad “interface” is typically not intended by the Pipeline Designer. As a solution, we decided to allow only selected user-triggered adaptations given in the pipeline configuration.

We introduce the concept of permissible parameters in the pipeline elements `Source`, `FamilyElement` and `Sink` as references to the parameters defined by the underlying implementing configurable element, i.e., `Data Source`, `Family` or `Data Sink`. As an alternative, we could have introduced the permissible parameters directly on the level of the implementing configurable element, but this would limit the reuse as the permissible parameters would apply to pipelines where the respective elements are used. To ensure consistency, we specify a constraint forcing that the configured permissible parameters are selected from the specific underlying element, e.g., the permissible parameters defined in the `FamilyElement` on pipeline level must be taken from the parameters of its underlying `Family`.

Permissible parameters are considered by the adaptation mechanism, which is now able to reject user-triggered events on non-permissible parameters. Moreover, WP4 and WP6 collaborate on an integration of the application design environment with the pipeline configuration in order to avoid inconsistencies and to ease the creation of stakeholder applications based on pipeline configuration information. As the infrastructure configuration happens on infrastructure / cluster side and the application design on stakeholder / end-user side, we decided to exchange an interface specification of the pipelines, which can be used to pre-configure the stakeholder applications. Two important parts of this interface specification are the output item types of the pipeline and its permissible parameters.

```
compound SubPipeline refines Pipeline {
  setOf(refTo(FamilyElement)) connectors;
  refTo(Family) subPipelineFamily;
  isDefined(connectors) and connectors.size() > 0;
}

compound SubPipelineAlgorithm refines Algorithm {
  refTo(SubPipeline) subPipeline;
}
```

Figure 2: IVML fragment defining the `SubPipeline`.

2.1.3 Replay Sink

For stakeholder applications, historical data typically represents an important insight of how stock markets changed over time. Such information is helpful for application users, in particular, to collect more intensive evidence for estimating and understanding financial risk events. For this purpose, WP2 introduced a data replay mechanism (see D2.3 for details of the underlying mechanism as well as the data aggregation) allowing to retrospectively explore processing results on historical data at different time scales and aggregation levels.

From a configuration modelling point of view, various options do exist on how to model a replay sink. We illustrate these options in Figure 3. The replay mechanism can be part of the existing sink concept enabling all sinks for replay as shown in Figure 3a). However, this can cause confusion on the Pipeline Designer side as more configurable settings are available than expected (as it was reported for algorithms and will be discussed in Section 2.1.4). Moreover, we see a potential performance issue always combining the replay mechanism with a usual data sink, in particular if the replay is active with no aggregation settings the overall output rate may drop. As an alternative, the replay mechanism could be combined with the existing data management element as illustrated in Figure 3b) and c). However, the design in Figure 3b) always requires a store node so that the replay is working, i.e., omitting the store node may lead to accidental configuration errors, in particular as sinks without store node would represent an usual sink. To avoid accidental performance pitfalls as discussed above, a separate replay sink concept in combination with the data management node would be required as depicted in Figure 3c). Paper prototyping showed us that this approach requires the pipeline designer to follow the given configuration patterns. Finally, we opted finally for creating a refined type of sink combining sink and replay functionality, which can be used optionally (in addition and parallel to usual data sinks in a pipeline) as shown in Figure 3d). So, with low effort, a Pipeline Designer can enable replay, use data sinks as usual at known performance and, if intended, can use a combined element in standalone fashion.

To support the configuration and instantiation of the replay mechanism, we include the replay sink as a special type of sink into the QualiMaster Configuration Model. The replay mechanism can be triggered in an interactive way enabling the application user to specify the parameters of the replay such as granularity and time frame, i.e., through an explicit runtime re-configuration in terms of a

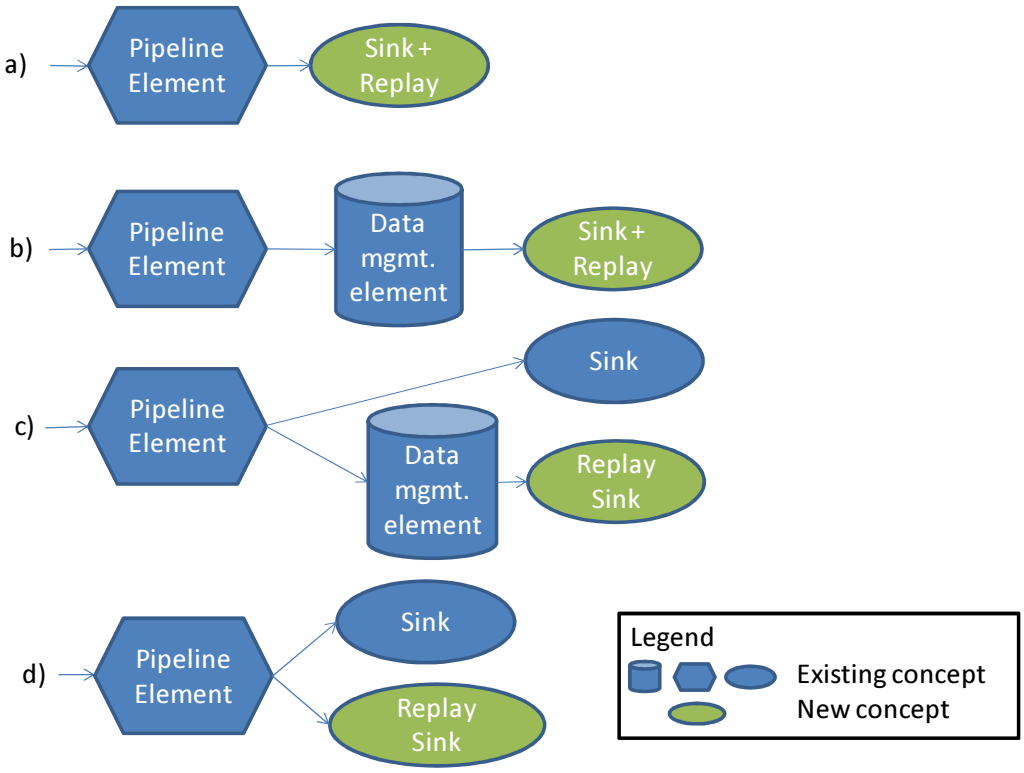

Figure 3: Configuration modelling alternatives reusing existing modelling concepts.

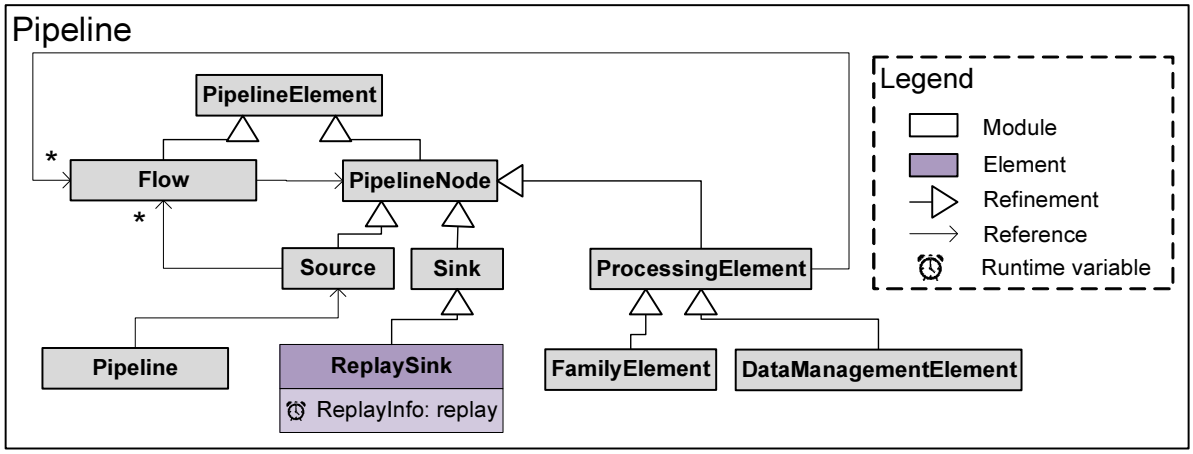

Figure 4: Fragment of the Configuration Meta Model showing the configurable `ReplaySink` and integration with already existing elements.

user-trigger. The related information must also be configurable through runtime variables.

As illustrated in Figure 4, the configurable element `ReplaySink` refines the existing `Sink` concept, i.e., a `ReplaySink` acts as a refined Sink which specifies the data sink implementation as well as the output types of data stream provided to the stakeholder application. The `Sink` is designed to emit processed results directly to the stakeholder application. In addition, the newly-introduced `ReplaySink` aims at supporting the storage of the analysis results and providing the stored historical results upon a replay request.

Figure 5 illustrates the IVML definition of both, `Sink` and `ReplaySink` including the compound `ReplayInfo`, for which instances are created/detached at runtime due to replay requests of the stakeholder applications.

```
compound ReplaySink refines Sink {
  assign (bindingTime = BindingTime.runtimeEnact) to {
    ReplayInfo replay;
  }
}

compound ReplayInfo {
  assign (bindingTime = BindingTime.runtimeEnact) to {
    Boolean active = true;
    Integer ticket;
    String start;
    String end;
    Integer speed;
    String query;
  }
}
```

Figure 5: IVML fragment defining the `ReplaySink` as well as its run-time variable `replay` allowing the applications to start/stop result replay.

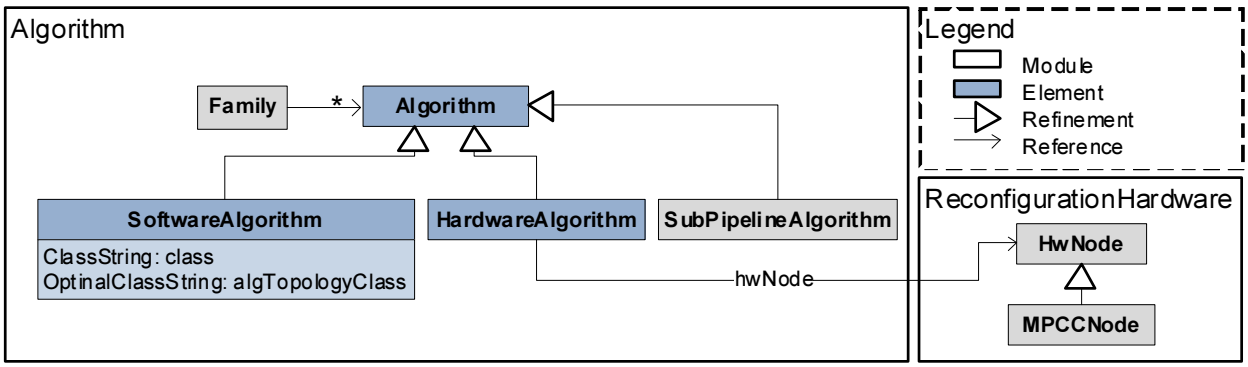

Figure 6: Fragment of the Configuration Meta model showing the updated design of the `Algorithm` configuration module.

2.1.4 Algorithm Refinement

In the previous design of the Configuration Meta Model, the configurable element `Algorithm` was responsible for representing all types of algorithms used in QualiMaster, including software- and hardware-based algorithms. As described in Deliverable D4.1, we distinguish different types of algorithms through their configuration settings, e.g., `hwNode` in `Algorithm` specifies the required reconfigurable hardware and acts as an indicator for hardware-based algorithms. However, feedback in particular from WP3 indicates that this overlap of configuration settings may confuse the Infrastructure Administrator in the configuration tool (QM-IConf). This is due to the fact that we defined all configurable settings for algorithms in one IVML compound, which are then presented in QM-IConf as one form-based editor, e.g., settings for hardware-based algorithms occur along with the settings for software-based algorithms. To support the Infrastructure Administrator, we refine the Configuration Meta Model by introducing different types of refined algorithms, so that the QM-IConf user interface, which is automatically derived from the Configuration Meta Model displays only the relevant settings for each algorithm type, in particular also for the new type of algorithms representing sub-pipelines.

We separate software and hardware specific configuration settings from the original `Algorithm` into `SoftwareAlgorithm` and `HardwareAlgorithm`, as depicted in Figure 6. Further, we change the `Algorithm` to an abstract compound, which contains all common configuration settings for the refined types, e.g., the specification of the input / output item types. It is noteworthy

```
compound SoftwareAlgorithm refines Algorithm {
  ClassString class;
  OptionalClassString algTopoClass = null;
}

compound HardwareAlgorithm refines Algorithm {
  refTo(HwNode) hwNode;
  assign (bindingTime = BindingTime.runtimeEnact) to {
    refTo(HwNode) actualHwNode;
  }
}

compound SubPipelineAlgorithm refines Algorithm {
  refTo(SubPipeline) subPipeline;
}
```

Figure 7: IVML fragment defining the `SoftwareAlgorithm`, `HardwareAlgorithm` and `SubPipelineAlgorithm`.

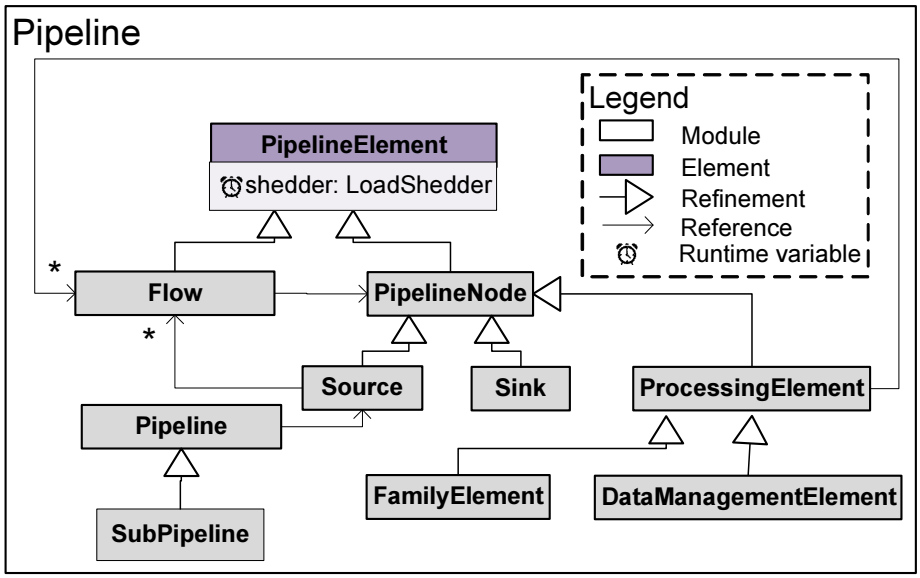

Figure 8: Fragment of the Configuration Meta Model showing the configurable `LoadShedder` as well as its relations.

that the refinement of the original `Algorithm` also releases the `SubPipelineAlgorithm` Section 2.1.1 from unneeded, optional configuration settings such as the class name required for software algorithms and the reconfigurable hardware configuration for hardware algorithms.

As illustrated in Figure 7, the `SoftwareAlgorithm` refining the `Algorithm` is equipped with specific configuration settings. For the `HardwareAlgorithm`, the `hwNode` indicates the required reconfigurable hardware server and a run-time variable `actualHwNode` specifies the actual used reconfigurable hardware allowing the QualiMaster infrastructure to re-configure the actual resource usage.

2.1.5 Load Shedding

Load shedding, i.e., throwing away data [3, 11] is the ultimate measure to protect an adaptive data streaming infrastructure. For enabling adaptive decisions about load shedding, it is important that the Configuration Meta Model allows changing the shedder settings at runtime, such as whether shedding is active or the amount of items to shed. Typically, load shedding is most effective at the sinks of a pipeline [11] as subsequent processing does not happen. However, in QualiMaster pipelines, the internal computations such as the correlation computation (D2.1 and D2.2) create massive data volumes (order of squared input), so that also overloads can occur here. Therefore, we equip each pipeline element with a potential load shedder and decide at runtime which one to activate at which impact. As depicted in Figure 8, we introduce a runtime variable `shedder` for each `PipelineElement`, which is filled with information if load shedding is needed. Figure 9 illustrates the corresponding IVML fragment. As underlying implementation, we extend the QualiMaster infrastructure with a simple, generic load shedding framework, which we will detail in the upcoming deliverable D5.3.

```
compound LoadShedder {
  assign (bindingTime = BindingTime.runtimeEnact) to {
    String name;
    Parameters parameters;
  }
}
```

Figure 9: IVML fragment defining the `LoadShedder`.

2.1.6 Adaptive Monitoring

As envisioned in D4.1, monitoring the execution of adaptive pipelines can itself be adaptive. In particular, we focus for now on the monitoring periods, i.e., the frequency monitoring happens for different parts of the QualiMaster infrastructure, such as the cluster nodes, the pipelines, and the pipeline nodes. Akin to Data Replay (Section 2.1.4) and Load Shedding (Section 2.1.5), we extended the runtime variables of the QualiMaster Configuration Meta model for the respective parts (infrastructure, pipeline, pipeline nodes) to enable the re-configuration of the monitoring periods (as indicated for pipeline nodes in Figure 8).

2.2 Optimizing the Configuration Reasoning

Reasoning over the configuration aims at validating a given configuration as well as completing a configuration through value propagation. Both operations are performed based on the constraints defined in the Configuration Meta Model. In QualiMaster, reasoning over the configuration is applied in three distinct scenarios (SC), namely:

SC1. **Configuration:** Validating the configuration before instantiation so that a structurally correct and type-safe pipeline composition can be ensured and subsequent execution errors can be prevented. In this scenario, the user executes the reasoning, i.e., reasoning must be fast enough to be considered reactive from a usability point of view (shall be completed within a few seconds). Also continuous feedback, i.e., reasoning upon each configuration change may be desirable, which imposes more demanding requirements, e.g., reasoning over a full configuration in less than 500 ms or some form of incremental reasoning.

SC2. **Adaptation Analysis:** Analyzing (violated) constraints as part of the runtime monitoring activities to trigger the runtime adaptation. In this scenario, reasoning is executed in a regular fashion by the Monitoring Layer. Currently, we run the constraint analysis once per second, i.e., reasoning over the full model shall be significantly faster than one second so that the subsequent constraint analysis can take place within the same time frame.

SC3. **Adaptation Planning:** Validating changed runtime configurations as part of the adaptation process to prevent erroneous configurations. The faster the reasoning is, the faster adaptation decisions can be made and enacted. As stated in D1.2/D4.1, the adaptation process does not necessarily require execution in (soft) real-time (in contrast to the data analysis in the pipelines), but due to the application domain, making adaptation decisions in less than one second (or faster) for the execution of the full planning is desirable.

To achieve the performance requirements for reasoning stated as part of the scenarios above, we follow two strategies (ST):

ST1. **Optimizing the reasoner**: Optimizing reasoning process while keeping or even increasing the precision of the reasoning results, e.g., by improved internal data structures or support of additional IVML concepts. In comparison to the performance results from Deliverable D4.2, we managed to increase the reasoning speed on a growing QualiMaster configuration model by 95%. We discuss this strategy in details in Section 2.2.1.

ST2. **Optimizing the model**: Optimizing the configuration model for runtime reasoning, in particular by pruning irrelevant information. This is a typical strategy for Dynamic Software Product Lines, i.e., approaches applying Software Product Line techniques at runtime [30, 35]. In QualiMaster, the optimization process enables a further speedup of around 36%. This strategy is explained in Section 2.2.2.

We discuss now the actual state of the reasoning support as well as the process for pruning configuration models. While we mention already some results in this section, we will discuss and analyze evaluation results in more detail in Section 4.1. For the results shown in this section, we conducted experiments on a Intel Core i7-3520M CPU 3.90 GHz, 6 GB RAM, Windows 7 64-bit machine. The measures were collected by executing the respective reasoner version on the Configuration Model eleven times. We omit the first result as a “warm-up” for loading classes and libraries, and report the average time calculated for the following ten iterations.

2.2.1 Optimizing the Reasoning Process

The reasoning process of the IVML reasoner was described in detail in D4.2 Section 4.1.2. In the evolved version (v2) of IVML the Reasoner, we joined pre-processing and constraint evaluation phases due to an optimized constraint dependency structure and subsequent improved constraint handling. In the most recent version of the reasoner, there are now two main phases (PH):

PH1. **Processing:** Gather information on the constraints and build the constraint base to reason on. Evaluate the specified constraints including re-evaluation of dependent constraints upon change of individual variable values until convergence.

PH2. **Post-Processing:** Create the reasoning result including detailed information about failed (sub-)expressions and involved variables. This phase is important to guide the user in identifying the root cause of configuration errors and to identify the runtime variables, which indicate adaptations.

We provide now more details on individual phases and show enhancements in the reasoning process using two optimization approaches (OA):

OA1. **Optimizing reasoning algorithm**, i.e., enhancing the way how constraints are evaluated and re-evaluated on values changed by constraint evaluation, i.e., value propagation. Details are given in Section 2.2.1.1.

OA2. **Increasing reasoning capabilities:** Taking advantage of all concepts provided by IVML, in particular concepts, which can improve the reasoning performance positively but have not been realized so far. We detail such concepts in Section 2.2.1.2.

2.2.1.1 Optimizing the Reasoning Algorithm

To optimize the reasoning algorithm we tackled two main issues:

- Improved variable-constraints dependency structure linking constraints and affected variables. The actual version of the dependency structure is inspired by the RETE-algorithm [18] and ensures constraint re-evaluation upon value changes of dependent variables. For improvement, we filter out simple assignments (assignments of constant values, container and collection initializations) and improved the variable / constraint evaluation sequence.
- Enhanced way how constraints are selected for re-evaluation if one of the variables used in a constraint is changed. By focusing on constraints that are already known to the (incremental) constraint base in the actual evaluation scope and by considering the constraints in the IVML compound refinement hierarchy, we avoid superfluous re-evaluations. Moreover, by limiting the amount of constraints that are passed into the constraint evaluation of importing scopes, we were able to minimize the (initial) constraint base.

As a result, the number of re-evaluated constraints due to (potential) value changes dropped by 82% compared with the version of the IVML reasoner discussed D4.2. Furthermore, the optimizations lead to a total decrease of reasoning time on the same model by 95% or almost 20

	IVML reasoner v1	IVML reasoner v2	drop %
Number of variables involved in constraints:	327	327	-
Number of constraints:	9050	9050	-
Number of re-evaluations:	63641	11655	82
Reasoning time [ms]:	3106	158	95

Table 1: Reasoning performance for the QualiMaster model before (v1) and after (v2) reasoning algorithm optimization.

times, dropping from 3106 ms to 158 ms. Table 1 details the comparison of performance readings by the reasoner version (v1, introduced in D4.2) and the current version (v2).

2.2.1.2 Increasing Reasoning Capabilities

In the processing phase (PH1), all constraints in the current scope are collected, rewritten for optimization (if needed) and added to the constraint base. Typically, IVML does not define the sequence of constraint evaluation, i.e., the user can state constraints in any order and the actual evaluation sequence is determined by reasoning. However, in some cases prioritizing constraints can speed up the evaluation sequence, e.g., pointing out specific initialization constraints that must be evaluated first. In IVML, this is expressed by so called eval-blocks, i.e., constraints listed in such a block must be evaluated before all other constraints in the same scope. In particular, for the validation of the topological pipeline validity constraints ([15], D4.2), such an initialization may be beneficial. So far, we emulated this behaviour by user-defined constraint functions, which lead to the same initialization effect. We introduced prioritized constraints to realize eval-blocks and to analyze the performance effects for the QualiMaster Configuration.

In more detail, we introduced the categories of *Priority project scope constraints* and *Priority compound constraints* and extended the internal evaluation sequence of a constraint base per evaluation scope as follows:

- *Default value constraints* – value assignment expressions for default values of variable declarations.
- *Type constraints* – consistency constraints defined along with type definitions.
- *Priority project scope constraints* – constraint directly defined in the project scope marked with evaluation priority.
- *Project scope constraints* – constraint directly defined in the project scope.
- *Compound constraints* – constraints defined for variables of compound type. These constraints become only effective if a variable of the respective compound type is defined. In particular, compound constraints include the assignment expressions for the default values of compound slots.
- *Priority compound constraints* – compound constraints marked with evaluation priority, with similar logic of *priority project scope constraints*.
- *Collections compound constraints* – constraints from compounds defined only in collections.
- *Constraint variables* – constraints assigned to a constraint variable.
- *Collection constraints* – constraints of constraint variable type collections.
- *Annotation constraints* – including the default values, assignments and validation of IVML annotations.

This extended categorization allows us to minimize the constraint base by shifting most of the evaluation to the specific scope. As a side effect, we do not just add constraints from imported scopes rather than adding them dynamically only if they required by dependent variables in the constraint base. This allows further reducing the size of the constraint base compared with the previous approach.

We discuss now the effects of the IVML eval-blocks. As mentioned before, we used user-defined constraint functions for emulating this behaviour. However, constraint functions have a higher call overhead, in particular if they are subject to dynamic dispatch, i.e., selection of the best alternative constraint function based on the actual parameter types. Already introducing static constraint functions to IVML, which are not called through dynamic dispatch, reduced the reasoning time. After replacing constraint functions with eval-blocks and conducting measurements we obtained the performance results summarized in Table 2.

As a result, the evaluation time decreased by further 7%, keeping coverage of constraints on the same level. In absolute numbers, the improvement by eval-blocks is “just” 14 ms, but with the

	Without eval	With eval	drop %
Number of variables involved in constraints	343	343	-
Number of constraints	10557	10557	-
Number of re-evaluations	13158	13158	-
Reasoning time [ms]	194	180	7

Table 2: Reasoning performance on the QualiMaster model before and after extension of reasoning capabilities.

overall improvements over D4.2 described in Section 2.2.1.1 we are already in a scale where improvements even of 14 ms matter.

2.2.2 Optimizing the configuration model

In order to further improve the performance of reasoning at runtime, we consider optimizing the configuration model itself. Our main idea is to minimizing the number of variables and constraints involved in the evaluation. We identified the following optimization steps (OS):

OS1. **Remove unused constraints**. For instantiating an IVML variability model, the required variables must be frozen. If a variable is frozen, it cannot be changed later on. In contrast, runtime variables for adaptation are not frozen so that monitoring and adaptation can repeatedly change them. However, constraints referring only to already frozen variables after instantiation are not needed anymore at runtime. For instantiation, these constraints are validated. In later stages, the variables used on these constraints cannot be changed, i.e., constraints referring only to already frozen variables are just re-validated at runtime without effect and can be deleted before runtime. In contrast, constraints referring to runtime variables used for monitoring or adaptation are kept.
Currently, in the QualiMaster Configuration (Meta) Model, this kind of constraints is used in two different ways, in particular for value propagation and consistency checks of algorithms, families and pipelines. After the final consistency check before instantiation, these constraints are not longer needed and can be removed from the model.

OS2. **Delete unnecessary elements**. IVML elements, which are not needed for adaptation, can be deleted to speed up the processing phase (PH1) of the reasoner, more specifically the constraints collection during this phase. For instance, this includes model comments, which are stored in the IVML object model to be able to store the complete model. It is important to mention that we do not delete configured decision variables, i.e., no configuration knowledge is lost.

OS3. **Delete inactive pipelines**. Currently, a QualiMaster configuration stores all defined pipelines, but explicitly indicates the pipelines that are active at runtime (D4.1). This is helpful in order to configure and maintain test pipelines and to turn them into active pipelines only if needed. The runtime adaptation focuses on the active pipelines only. However, also the inactive pipelines are part of runtime configuration, even if these pipelines are not deployed at all. Deleting the inactive unused pipelines before runtime shall reduce the number of (runtime) variables involved in reasoning at runtime and, thus, speed up reasoning at runtime.

We validated the expected benefit our ideas based on manually optimized models indicating a potential gain of around 30% reasoning time in comparison with the reasoner discussed in D4.2. Based on this pre-evaluation, we extended the Configuration Core EASy-Producer [14, 17] by mechanisms for automatically optimizing the Configuration in a generic manner. So far, we realized OS1 and OS2, which led to an overall decrease in reasoning time by 36% even on the most recent version of the reasoner discussed in Section 2.2.1. Based on a manually optimized

model, we evaluated that OS3 has a non-significant further impact on the reasoning time after applying OS1 and OS2, i.e., we plan to skip OS3. Detailed results of the optimization steps will be discussed in Section 4.1.2.

2.3 Extended Infrastructure Instantiation

In QualiMaster, we aim at an automated instantiation process for the QualiMaster platform based on a valid configuration. Particularly, we focus on the configuration supported by the Configuration Meta Model in terms of the basic modules of the resource pool, data management elements, data processing algorithms, algorithmic families, topological processing pipelines, the infrastructure settings (D4.1/D4.2) as well as the most recent extensions described in Section 2.1. In this section, we provide a brief overview on the changes to the instantiation. More details will be discussed in the upcoming deliverable D5.4.

Basically, the extensions on the instantiation are:

- **Generation of Sub-pipelines Implementation.** As discussed in Section 2.1.1, a recent addition is to support the configuration of sub-pipelines representing distributed algorithms. For generating sub-pipelines we reuse the existing pipeline instantiation templates in terms of a recursive traversal of the configured processing nodes through the data flow graph of the (sub-) pipeline.
- **Support for algorithm switching.** As indicated in Deliverable D4.2, switching among alternative algorithms is one of the key enactments in QualiMaster. Initially, we experimented and evaluated several switching approaches with manually-implemented code in a test pipeline. Relying on manually written code for initial experiments is reasonable, as the final code implies a rather complex pattern, which cannot be easily generated without having a working version. To enable the advanced switching approach [32] for all involved pipelines, we generate the specific switch code.
- **Support for the Data Replay.** To enable insights into historical risk events on stakeholder side, we introduced the replay sink as explained in Section 0. Based on the configuration of the replay sink, we generate specific code integrating the replay recorder for storing processed results and the replay streamer for replaying the historical results into the implementation of the replay sink. Both replay recorder and streamer are provided by the QualiMaster infrastructure (Data Management Layer).
- **Creation of setup files for stakeholder applications.** To support the design of new stakeholder applications for avoiding manually entering a series of pipeline settings (it was the case so far), we generate a setup file per pipeline containing all related configuration information. Such a setup file details the connection between pipeline and application, the schema of the result data of the pipeline sinks, supported general and pipeline-specific commands as well as the permissible parameters (Section 2.1.2) used for user-triggered adaptations as indicated in deliverable D4.2.

Besides improvements to the existing instantiation, all described changes in the QualiMaster infrastructure instantiation process are related to new configuration concepts. They are integrated into the overall instantiation process and used to derive pipelines. The instantiation process is also integrated into continuous integration to test the instantiation, the reasoning and to automatically deploy the generated pipelines to the QualiMaster processing elements / pipeline repository. We will provide a more detailed discussion of the instantiation process in Deliverable D5.4.

2.4 QualiMaster Infrastructure Configuration Tool (QM-IConf)

The QM-IConf tool addresses domain-specific configuration, deployment, and administration of QualiMaster data processing pipelines aiming at reducing the time to realize and change pipelines. We focused on the further development of QM-IConf, in particular to support the new configuration concepts introduced in Section 2.1, to simplify the usage of QM-IConf and to support Pipeline Designers, Adaptation Managers and Infrastructure Administrators in their work.

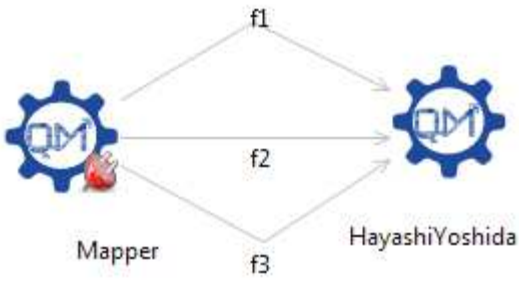

Figure 10: Sub-pipeline configuring the distributed Hayashi-Yoshida correlation algorithm.

In this section, we present the main improvements of the QM-IConf application after Deliverable D4.2. In Section 2.4.1, we discuss the realization of the new configuration concepts from Section 2.1 and how to provide (inexperienced) users assistance when configuring pipelines and the infrastructure. Section 2.4.2 details new administration and operating capabilities of QM-IConf. In Section 2.4.3, we outline our activities to make the QM-IConf application publicity available under an open-source license. Finally, in Section 2.4.4, we summarize the realization status of the requirements for the QM-IConf application as stated in deliverable D1.2.

2.4.1 Configuration Support

We continued developing QM-IConf to enable new configuration capabilities. In particular, we extended the pipeline editor to support the new configuration capabilities discussed in Section 2.1, i.e., the support for modelling *sub-pipelines* and *replay sinks*. An example for a sub-pipeline is given in Figure 10 in terms of the Hayashi-Yoshida correlation computation. In contrast to usual (top-level) QualiMaster pipelines, sources and family elements can serve as connectors, i.e., where the integrating (top-level) pipeline passes data items to the sub-pipeline (indicated by a plug in Figure 10). The pipeline editor allows pipeline developers to turn already modelled pipelines into sub-pipelines and also back into top-level pipelines if needed. The Hayashi-Yoshida sub-pipeline shown in Figure 10 is configured as a member of the correlation algorithm family.

Replay sinks collect data and replay analysis results on request (cf. Section 2.1.2). We extended the pipeline editor to enable Pipeline Designers both, the modelling of classical sinks as well as replay sinks. Figure 11 shows a fragment of the extended Priority pipeline using both kinds of sink in parallel.

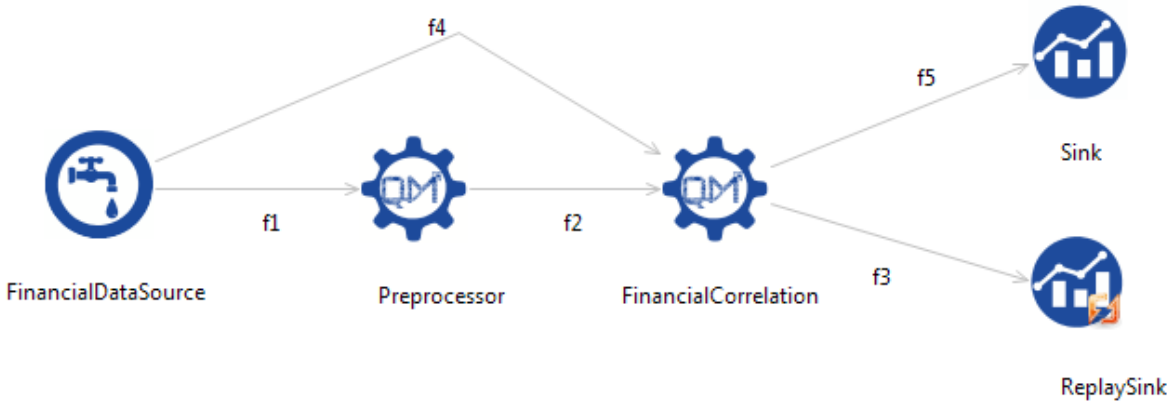

Figure 11: Modelling a pipeline with a replay sink to facilitate a later analysis of analysis results according to Figure 3d).

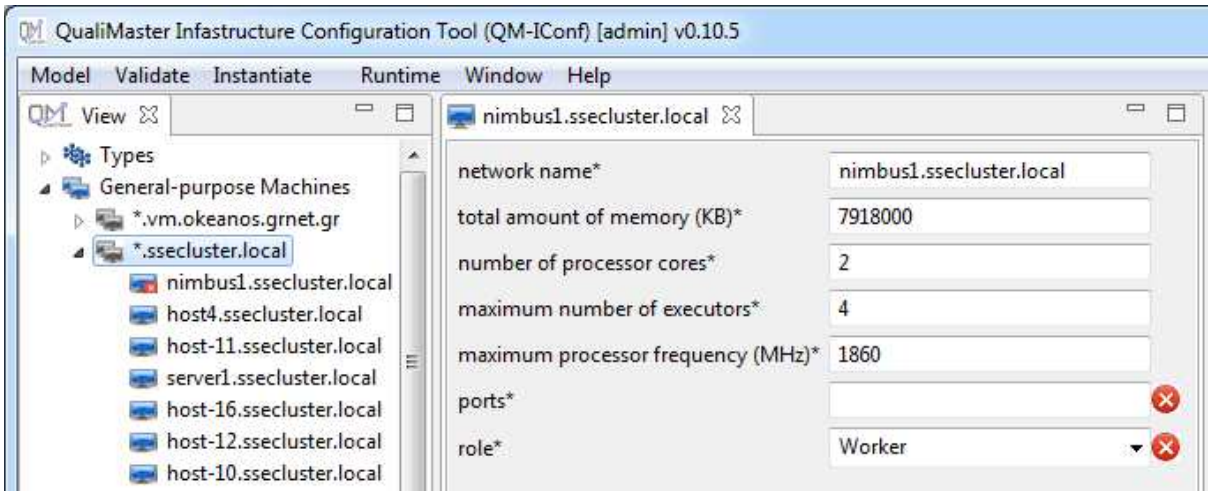

Figure 12: Visualization of a violated constraint in a textual editor (ports and role are in conflicting for the machine "nimbus1.ssecluster.local").

Further, we improved QM-IConf to further support pipeline developers during the configuration process by the following changes / additions:

- *Validation errors* are displayed textually as well as graphically. In case of the pipeline editor, the graphical display of validation errors in particular in terms of the connected elements causing the error simplifies localizing and solving the problem. This is illustrated by the right part of Figure 13 where the causing pipeline elements are marked by a red border. In addition, erroneously configured elements such as pipelines are marked with a red square in the configuration tree view (left part of Figure 13).
  Also in the form-based configuration editors, violated configurable elements are highlighted as illustrated in Figure 12. Here, ports and role are marked with a red decorator to indicate that they are configured inconsistently. The problem view as well as tool tips give an explanation for the cause of the error (not shown in Figure 12).
- Some configuration settings represent artifacts in the pipeline elements repository, in particular implemented algorithms, data sources and data sinks. Although it is possible to enter the artifact specification manually, a specific selection dialog increases the usability, avoids accidental configuration errors and saves time. QM-IConf allows the *selection of artifacts* directly from the (Maven) pipeline elements repository and *analyses the implementation* (Java class files or manifest specification for hardware-based algorithms) to simplify even subsequent configuration of details such as the implementing class (an

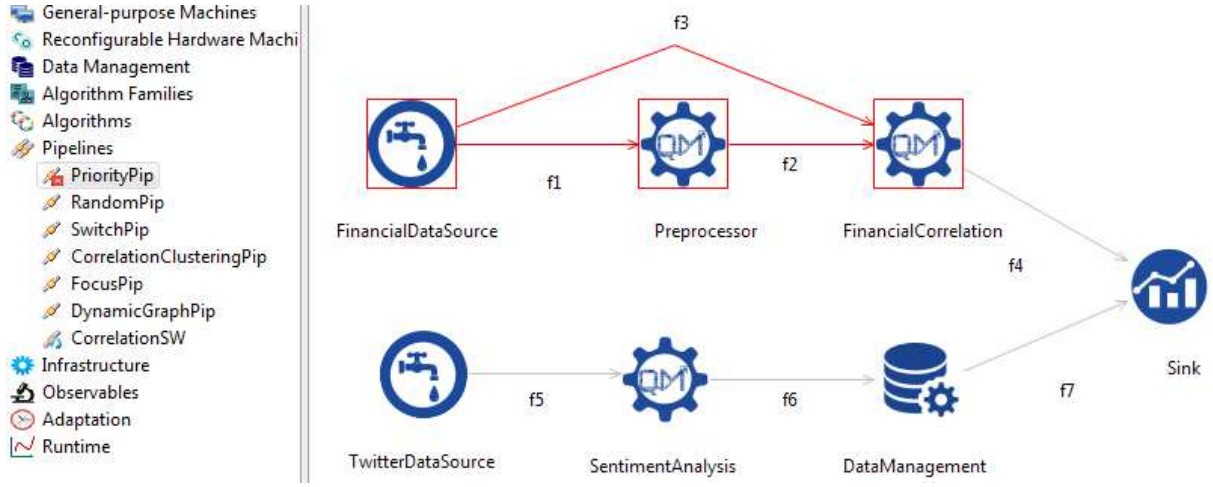

Figure 13: Visualization of an erroneously configured pipeline.

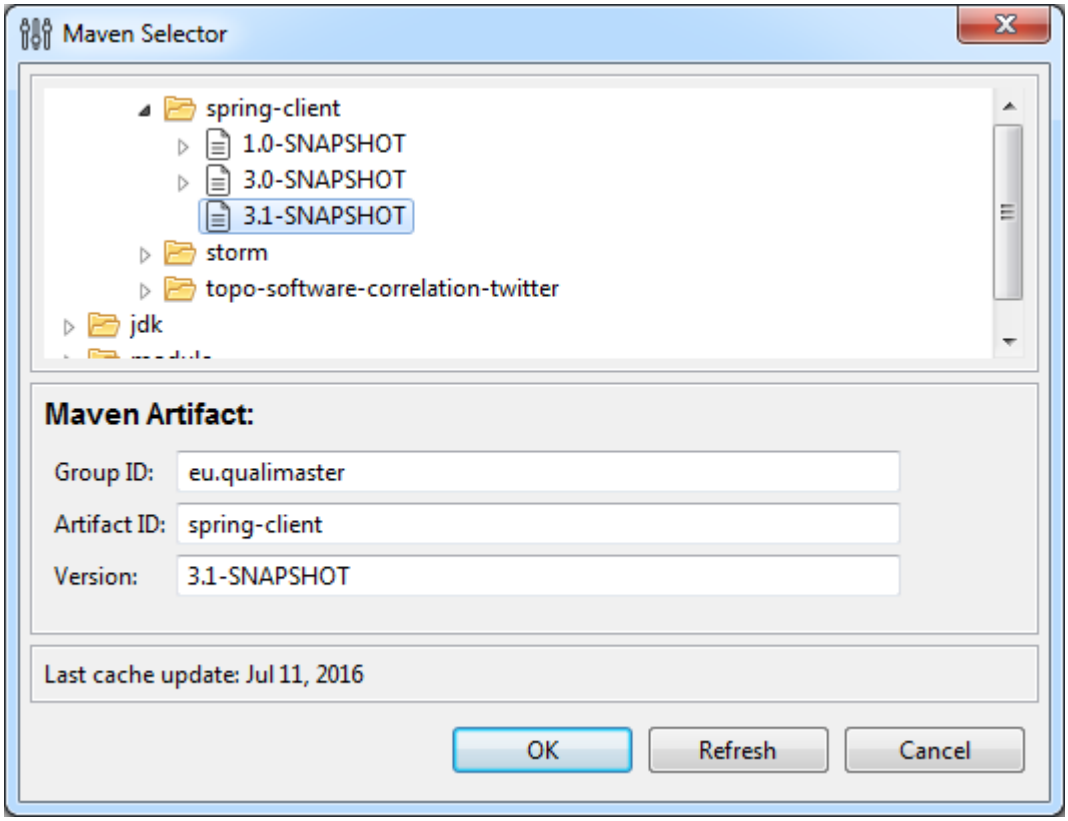

Figure 14: Selection of artifacts from the processing elements (Maven) repository to simplify its configuration.

artifact may contain multiple algorithms, sources or sinks). An example is shown in Figure 14, where the spring-client in version 3.1 is selected as a data source. A second dialog (not shown here) facilitates the selection of the provided classes, i.e., the actual implementation of the data source within the selected artifact.

- The extended Configuration (Meta) Model for sub-pipelines as explained in Section 2.1 leads to a more detailed configuration of data flows in the pipeline editor, i.e., to further references among pipeline elements / item types for specifying the data distribution mode. Basically, this could be enabled by a selection list. However, without further support the Pipeline Designer would be lost as the QualiMaster Configuration (Meta) Model offers currently the selection of more than 160 different potential item types for a flow between two elements of a pipeline. Usually, this list can be reduced to 1 or 2 entries, based on the actual context, i.e., selected sources, family elements, or sinks. For supporting the Pipeline Designer in this way, we refined the pipeline editor so that it *restricts ranges* of configurable parameters of pipeline elements to simplify the configuration process where possible.
- We realized an initial version of the *configuration support for observables* meeting requirement UC-PA1 of Deliverable D1.2. In addition to the pre-defined observables implemented by the QualiMaster infrastructure, this facilitates Infrastructure Administrators to define specific observables, related monitoring probe (based on a simple, extensible pipeline probe framework in the QualiMaster infrastructure, which will be explained in the upcoming Deliverable D5.4) and to use the new probes for monitoring and adaptation. Again, we rely on artifact specifiers and code deployed to the QualiMaster pipeline elements repository, reusing the artifact and class selector described above.
- We extended the *configuration of the adaptive behaviour* of the QualiMaster infrastructure. Deliverable D1.2 requests various access levels to the configuration of the adaptive behaviour, at least a high-level settings, e.g., in terms of weights shall be provided to adaptation managers (UC-AM6). QM-IConf now allows defining such high-level settings in terms of weights for individual observables to be considered by the adaptation. At most

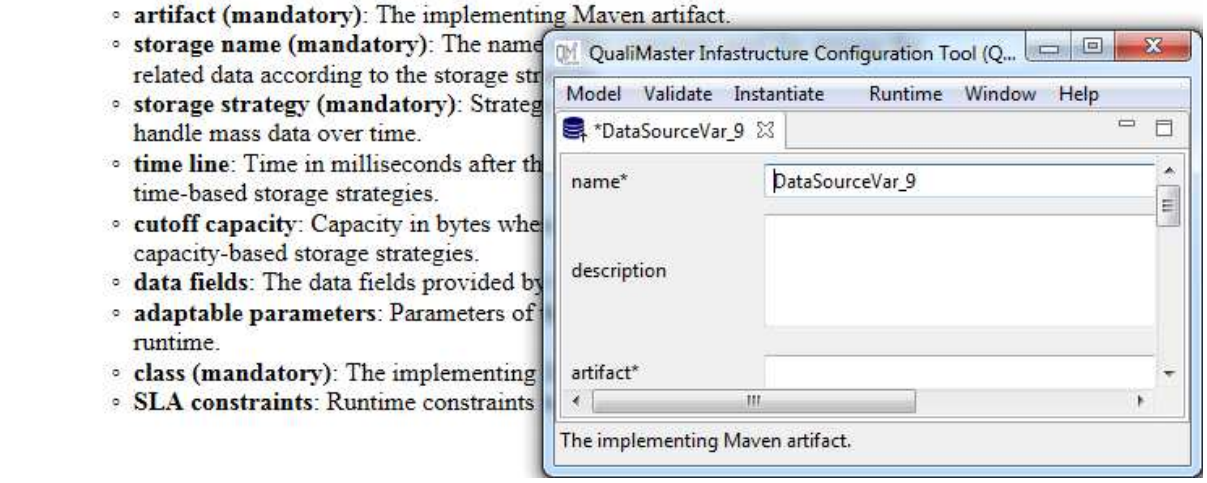

Figure 15: Help texts for configurable parameters.

detailed level, QM-IConf provides a syntax highlighting editor for adaptation specifications in rt-VIL, i.e., experienced users can even change the adaptation specification directly. For the remainder of the project, we aim at an intermediary level, e.g., disabling individual rt-VIL strategies or tactics via QM-IConf.

- We introduced *help texts* to give an explanation / rationale about the available configurable settings. The help texts are derived from the Configuration Meta Model. A complete list of all help texts can be displayed through a double click on a configuration section in the left part of the application. Further, a short description of a configurable element is presented in the status bar when hovering with the mouse on the element. It is worth mentioning that QM-IConf also supports localized help messages for different languages based on mechanisms already realized in the Configuration Core, i.e., EASy-Producer [14, 17]. An example for the help messages is depicted in Figure 15. The background shows an excerpt of a help text for a configuration section, while the screenshot of the application illustrates the help messages in the status bar, respectively.

- Some configurable elements are *mandatory* for the instantiation process, while others are optional, i.e., may not be given by the user, or depend on the configuration of other variables and can be derived during reasoning. So far, this difference was not obvious to the user (as also reported by the received feedback). The IVML reasoning (Section 2.2) is used to check that all required elements are configured. However, this can become very time consuming if mandatory /optional settings are not indicated in the user interface and reasoning must be applied by the user to determine the mandatory elements. Moreover, it can be annoying for the user to identify that a certain setting shall be made after some rounds of validation.
  A simple solution is be to mark all mandatory fields in the user interface, e.g., by an asterisk (*). However, as the QualiMaster configuration model is evolving, in particular when constraints are refined, the user interface becomes outdated. Thus, we aim at an approach which analyzes the constraints and keeps the user interface in up-to-date with the model regarding mandatory / optional settings. A configuration-level constraint analysis component (now part of the Configuration Core) analyzes the configuration and determines mandatory elements. Based on this information, QM-IConf marks mandatory elements by appending “(mandatory)” to the name in the help section and by an asterisk appended to the in UI labels (cf. Figure 15).

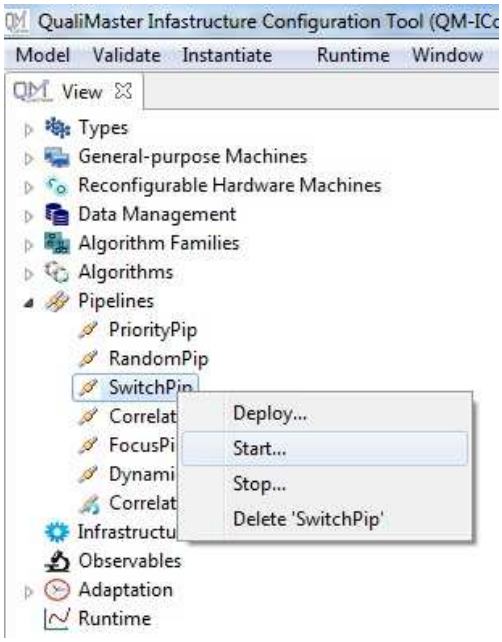

Figure 16: Integrated pipeline start and stop commands in QM-IConf.

2.4.2 Administration and DevOps

In addition to defining the configuration for a QualiMaster infrastructure before executing pipelines, QM-IConf also provides a runtime view on the running pipelines of an infrastructure installation. After connecting to a running infrastructure (`Runtime → Connect` in the menu), QM-IConf enables the user to start and stop deployed pipelines and also to monitor running pipelines. The actual state of these capabilities is described further in the next two subsections. The capabilities of these two subsections have been developed to help Infrastructure Administrators and Adaptation Managers to observe the actual execution on a QualiMaster infrastructure and, in particular, to analyze, judge and improve the adaptive behaviour. In this sense, QM-IConf provides capabilities for developing and operating pipelines, i.e., DevOps support (Development + Operations [4]).

2.4.2.1 Starting and Stopping Pipelines

In addition to infrastructure command line operations on the cluster managed by the QualiMaster layers developed in WP4 and WP5, pipelines can be started and stopped more conveniently through QM-IConf. For a pipeline in the configuration tree, the context menu offers the respective

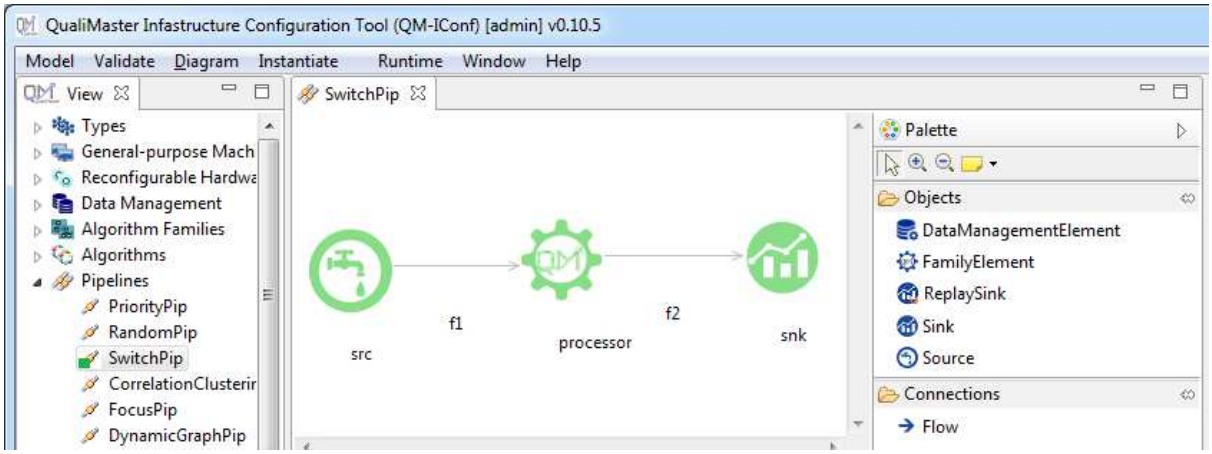

Figure 17: Pipeline workload visualization, for all pipelines left and for a specific pipeline in the pipeline editor (green = low workload, red = high workload, blue = no monitoring).

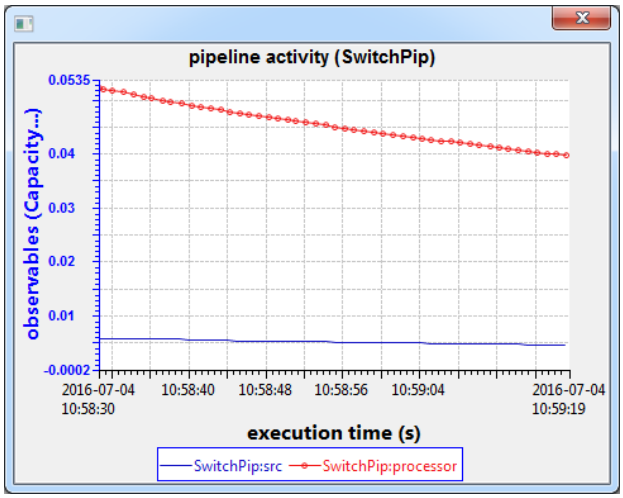

Figure 18: Comparison of different observables of a pipeline in one (dockable) window.

operations as illustrated in Figure 16. The runtime event log view (not shown in Figure 16) informs the user about the current status while starting the pipeline, in particular about changes in the pipeline lifecycle. This log also informs the user about adaptations happening during the execution of the pipeline.

2.4.2.2 Operating Pipelines

QM-IConf offers two mechanisms for displaying runtime information about currently executing pipelines in the sense of DevOps capabilities [4]. Pipeline elements of running pipelines are coloured in the configurable elements tree / pipeline editor to provide a quick overview regarding their current workload. Detailed information is presented in an extra runtime view for a more accurate overview. These two views are explained below.

For a quick overview, a summary of the average workload of all pipelines is indicated in the configurable elements tree in terms of a coloured decorator (see Figure 17 left side, the green rectangle at “SwitchPip” in the configurable elements tree). In more detail, the current load of a running pipeline is indicated in the respective pipeline editor in terms of colour coding of the individual pipeline elements. The detailed view and the colouring in the pipeline editor are illustrated in the middle part of see Figure 17. The test pipeline “SwitchPip” has a low workload. In particular the pipeline editor in Figure 17 indicates that all elements of that pipeline have a low workload.

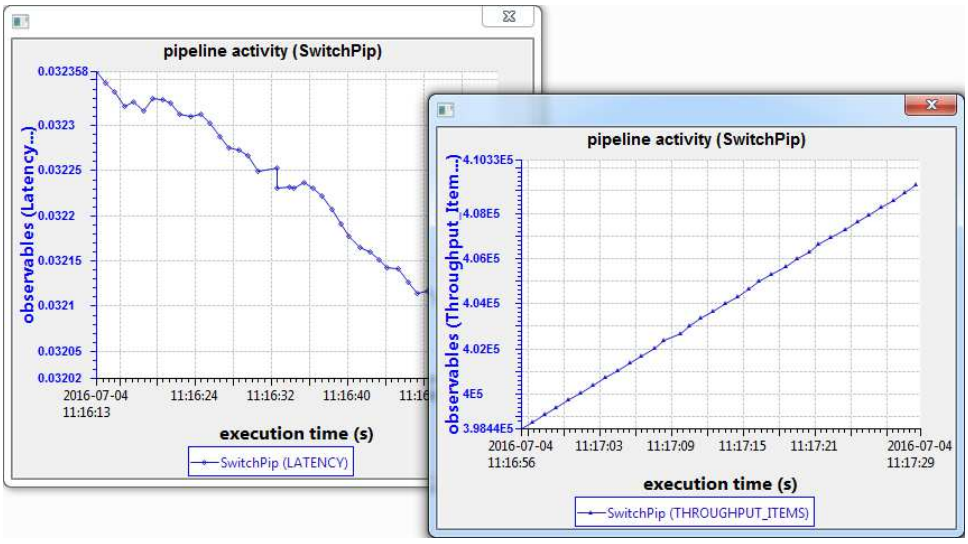

Figure 19: Comparison of different observables of a pipeline in two separate (dockable) windows.

Moreover, the runtime view of QM-IConf now provides more detailed information about the runtime state of an infrastructure. Initially, this view showed the utilization of servers and reconfigurable hardware as well as a fixed throughput graph for the running pipelines. The renovated runtime enables the user to selected combinations of monitored configurable elements (complete pipelines or their individual pipeline nodes) and the desired quality parameters (in general, observables) and to create diagrammatic views for these combinations. The view allows drawing multiple traces of monitored data into one graph (Figure 18) as well as multiple diagrams in parallel in different graphs (Figure 19). Diagrams can be docked as tabs or moved freely over the screen, the latter to create a dashboard (also storing the dashboard layout is possible). The general aim is to support the user during the inspection / analysis of potential problems and bottlenecks and to provide a high flexibility in choosing the most appropriate runtime graphs for a certain inspection task. Moreover, we developed this flexible runtime view in order to support the partners in evaluating the QualiMaster infrastructure and the adaptation in the remaining months of the project.

2.4.3 Open-Source Contribution

In the second year of the QualiMaster project, we published QM-IConf as open source under Apache 2.0 license on the GitHub space of the project[2]. Various activities have been conducted for publishing the QM-IConf application as open source:

- The *licenses of used libraries* have been *checked* regarding their compatibility with the Apache 2.0 license. If required by some licenses, further information has been added to the libraries folder in terms of additional text files.
- For the QualiMaster partners, QM-IConf version includes setup information from where to obtain the actual configuration model, currently from the project SVN. The demonstration versions are shipped with an embedded version of the configuration model. However, an external user who wants to execute the open source version can start the application, but does not have a configuration model to work on. At least an “empty” version of the QualiMaster configuration model is needed for this purpose, in particular as the current version of the model contains access tokens to external services (WP5 extended the infrastructure to internalize this information and to mitigate this issue). We performed two activities to enable an external uptake of the open source version:
    - The continuous integration process has been extended to generate and publish automatically an *empty Configuration (Meta) Model*, which does not contain project specific configuration settings of the consortium partners. Creating an empty model is required as the Configuration (Meta) Model is evolving and the empty model shall always be up-to-date with the available source code.
    - A *bootstrap* dialog helps to set up the application when running it the first time. This dialog allows to automatically download a Configuration (Meta) Model from the project repository or to import the empty Configuration (Meta) Model. Thus, consortium partners as well as external users can easily start configuring, validating and generating pipelines without need to manually change the settings of the tool.
- The *source code*, including necessary resources and libraries, of the QM-IConf application has been *uploaded to the GitHub space of the project*. More specifically, the whole development process has been changed to a public development process. This includes public issue tracking, the adjustment of the continuous integration as well as life commits.

In addition to the open source activities, we still create dedicated release versions for the partners as described in D4.2. This helps ensuring the quality of the tool, as for releasing a version of QM-IConf intensive tests are performed (see also D6.2). Moreover, the continuous integration enables us building nightly releases automatically, which can be obtained by the partners through the update functionality of QM-IConf / Eclipse RCP.

[2] https://github.com/QualiMaster/QM-IConf

2.4.4 Realization status of Requirements from D1.2

QM-IConf evolved during the QualiMaster project and realized more and more of the requirements listed in D1.2. Below, in Table 3 we indicate the actual realization state of QM-IConf with respect to the D1.2 requirements. We categorize the realization state in terms of four categories, namely:

- Full support: The realization is done and considered to be production ready. Improvements may happen due to the identification of issues or due to feedback.
- In progress: The realization is actually in progress.
- No support planned: There is a specific use case in D1.2, which indicates that no specific tool support is required or practical reasons lead to the conclusion that this use case shall not be realized for a given reason by QM-IConf.

Identifier	Name	Realization state
UC-PD1	Define new pipeline	Full support, now including sub-pipelines for distributed algorithms
UC-PD2	Modify pipeline definition	Full support, now including sub-pipelines for distributed algorithms
UC-PD3	Delete pipeline definition	Full support, now including sub-pipelines for distributed algorithms
UC-AM1	Define quality parameters of processing elements	Full support in terms of user-defined constraints
UC-AM2	Define pipeline quality characteristics	Full support in terms of user-defined constraints
UC-AM3	Define reactive adaptation rules	In progress regarding higher-level UI for enabling/disabling strategies and tactics; full support via integrated rt-VIL editor
UC-AM4	Define proactive adaptation rules	In progress regarding higher-level UI for enabling/disabling strategies and tactics; full support via integrated rt-VIL editor
UC-AM5	Monitor execution of adaptation rules	Full support, user-defined display of execution state from multiple perspectives
UC-AM6	Change adaptation settings	Full support
UC-PA1	Define platform quality parameters	Full support, in addition to pre-defined quality parameters
UC-PA2	Modify platform quality parameters	Full support, in addition to pre-defined quality parameters
UC-PA3	Add data processing algorithm	Full support, including take-over of configuration information from artifact

Identifier	Name	Realization state
UC-PA4	Modify data processing algorithm	Full support
UC-PA5	Add Hardware-based Data Processing algorithm	Full support, including take-over of configuration information from artifact
UC-PA6	Modify Hardware-based Data progressing algorithm	Full support
UC-PA7	Configure Pipeline Sources and Sinks	Full support
UC-PA8	Start Pipeline	Full support
UC-PA9	Stop Pipeline	Full support
UC-PA10	Configure QualiMaster Platform for Software-based Execution	Full support
UC-PA11	Configure QualiMaster Platform for Hardware-based Execution	Full support
UC-PA12	Start QualiMaster Platform	no UI support planned, realization via command line / operating system service
UC-PA13	Stop QualiMaster Platform	no UI support planned, realization via command line / operating system service
UC-PA14	Instantiate Platform	Full support
UC-PA15	Monitor Execution	Full support, user-defined display of execution state from multiple perspectives

Table 3: Fulfilment of D1.2 requirements in QM-IConf.

In total, the consortium collected 24 use cases for QM-IConf, out of which 20 have been fully realized, 2 are categorized as in progress of realization (supporting User interfaces for adaptation specification, in progress), and for 2 use cases no support by QM-IConf is needed (infrastructure start-up and shutdown is better managed along with the start-up of Storm by the operating system).

In terms of the specific requirements from D4.1, we conclude that most requirements out of REQ-C-1 to REQ-C-15 are realized by the Configuration (Meta) Model and the respective tooling (QM-IConf and EASy-Producer). As far as we can see by now, REQ-C-12 will be realized by fixed components in the infrastructure, which may be disabled through the Configuration.

3 Adaptation

Adaptive pipeline execution in QualiMaster aims at dynamically and autonomously mastering changes in the execution environment, in particular input stream load changes. In this section, we provide a conceptual overview on the current state, the recent work and the plans of WP4 until the end of the project. We start with an overview of the adaptation approach in Section 3.1, continue with enactment patterns and mechanisms in Section 3.2 and the adaptation control in Section 3.3, including reactive, predictive/proactive, cross-pipeline and reflective adaptation.

3.1 Overview

For defining and realizing the adaptation in QualiMaster, we follow the well-known MAPE-K cycle [9, 19, 22], i.e., Monitoring, Analysis, Planning and Execution as introduced in D4.1 and realized by the respective WP4/WP5 components. The cycle is illustrated in Figure 20. We use now the MAPE-K cycle as structure for providing an overview on the current state of the runtime adaptation in QualiMaster.

The MAPE-K cycle starts with **monitoring** the underlying system. For this purpose, we collect in the Monitoring Layer of WP5 data from various sources, including Apache Storm, generated monitoring probes, the QualiMaster infrastructure, the resource monitoring framework SPASS-meter and the re-configurable hardware. Apache Storm (via the Thrift[3] framework) provides us with information about running pipelines and the allocated cluster nodes. However, the information delivered by Thrift is pre-aggregated (by Storm) on a horizon of at least 10 minutes, which does not allow us to perform adaptation within this time frame. As mitigation, we generate monitoring probes for execution time, emitted items and volume into the generated pipelines. Moreover, these probes are extensible as we will explain in the upcoming deliverable D5.4. Furthermore, quality parameters are implicitly delivered by the QualiMaster infrastructure, e.g., the enactment time as difference between sending a coordination command and executing it by the pipeline or the infrastructure. For resource consumption of pipeline nodes and physical compute nodes in the resource pool via an additional monitoring service, we employ SPASS-meter [16], which is integrated into the QualiMaster infrastructure through monitoring events (see upcoming D5.4 for details, in particular on the optimization of SPASS-meter during the project). Finally, we poll information from the reconfigurable hardware resources via a specific protocol (see D3.2, D3.3, D5.3 and D5.4).

The information requested or received by the monitoring is raw data, which must be aggregated to

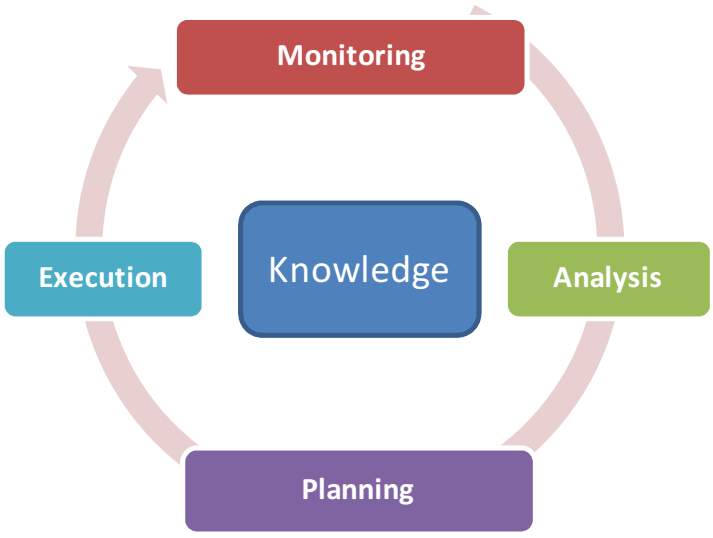

Figure 20: The MAPE-K cycle.

[3] https://thrift.apache.org/

concepts known by the Configuration, as the further steps in the adaptation cycle operate on the integrated Configuration Model [13]. To ease updating the monitored information, we maintain information delivered by the individual components separately, e.g., the execution time required by each individual Storm task (rather than just a number for all tasks per executor). However, neither the Configuration nor the information delivered by Storm is sufficient to perform all required kinds aggregation of the raw data. On the one side, the Configuration represents a rather abstract data flow graph and does not include implementation level knowledge, such as the additional nodes inserted for efficient algorithm switching (see Section 2.3). On the other side, Storm does not know about sub-pipelines or "gaps" in the pipelines in which the hardware operates (see D3.2). Therefore, we create as part of the monitoring activities a complete logical topology structure of the running pipeline, which unifies Configuration and Storm information. We use this topology to (transparently) aggregate raw values, e.g., the minimum latency over a (sub-)topology or the throughput determined by the "sinks" of the (sub-)topology. This forms a rather detailed (distributed) system state of the actual execution.

The **analysis** stage obtains in a regular fashion a snapshot of the monitored system state and runs the IVML reasoner to determine violated (runtime) constraints. As the reasoner provides detailed information about failing constraints (see Section 2.2), we can obtain the failing constraint part (typically an operator expression) and determine for that expression the deviation from the expected value (usually, one operator can be evaluated to a constant value). Failing constraints are signalled along with the causing system state to the planning stage.

During **planning**, we determine applicable adaptation strategies and tactics for the current situation and select the most appropriate strategy / tactic. As described in D4.1/D4.2, strategies and tactics are expressed in rt-VIL, a runtime re-configuration extension of the language that we use for instantiating configurations (Section 2.3). The execution of tactics / strategies may fail, e.g., implicit by violated constraints or explicitly through rt-VIL, e.g., to reject user triggers sent by the stakeholder applications. In the failing case, we use a transactional re-configuration history to restore the configuration before the failed tactic/strategy, so that in the extreme case multiple candidates can be applied in the sequence of expected success (determined by user-defined weights or constraint deviations used as prioritization). We will explain mechanisms used in the strategies and tactics to realize reflective, reactive and proactive adaptation in the following sections.

If finally the planning succeeds, the **enactment** stage starts. Here, we translate the re-configured values using rt-VIL into enactment commands of the QualiMaster infrastructure, which are turned into more detailed (coordinated) signals for the individual execution systems, such as Apache Storm or the reconfigurable hardware. The enactment can cause various changes in the data

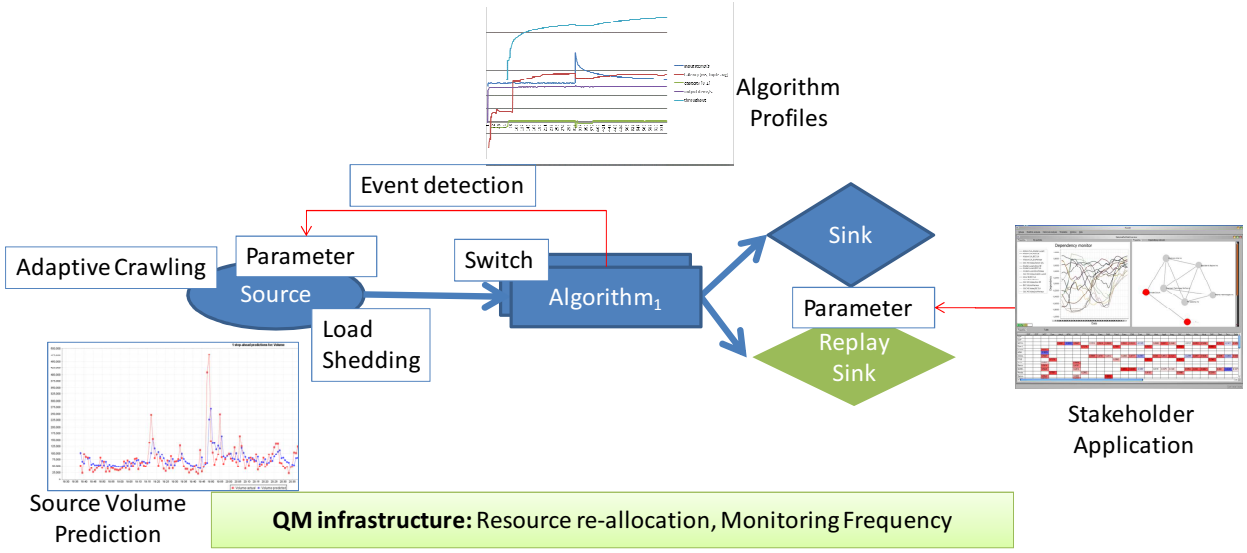

Figure 21: Overview of the adaptation mechanisms.

processing pipelines as illustrated in Figure 21. Algorithm parameters on all pipeline elements allow changing the processing in the algorithms. Parameter changes can be requested internally (as shown in Figure 21 through the Social Media event prediction) or via user triggers sent by the stakeholder application (or for maintenance by QM-IConf). Specific forms of parameters enable adaptive crawling / subscription to financial data sources, load shedding, data replay and adaptive monitoring. We will detail recent conceptual additions and improvements to the enactment mechanisms / patterns in Section 3.2. The algorithm switch mechanism [32] allows preparing alternative algorithms for execution and switches to them efficiently. On infrastructure level, resource re-allocation and changing the monitoring frequency is supported as enactment mechanisms.

In QualiMaster, adaptation is based on various forms of **knowledge**, in particular including the Configuration (with pre-runtime and runtime variables / constraints) and the runtime state collected by monitoring. Moreover, WP4 works on mechanisms that enable to learn adaptation knowledge and support predictive adaptation. For this purpose, we introduced two particular concepts (indicated in Figure 21, we will detail them in Section 3.3.2): The source volume prediction aims at estimating the expected future data volume of data sources, in particular to judge parameter changes and the need for load shedding as part of the planning stage. Moreover, algorithm profiles aim at capturing the quality properties depending on input stream characteristics, parameter and distribution settings. Algorithm profiles are intended to make dynamic trade-off decisions, to enable learning the quality properties at runtime (as we rely on an updatable statistical representation) and, moreover, are helpful during algorithm testing.

3.2 Enactment Mechanisms

In this section, we discuss the current state of the enactment mechanisms realizing the enactment patterns from D4.1 including new concepts designed and realized to fulfil the adaptation needs of the QualiMaster project. It is worth mentioning that for all mechanisms, in particular the new mechanisms, a working implementation exists or existing implementations were refined and improved. We will discuss Adaptive Crawling in Section 3.2.1, a formalization of safe/transparent algorithm switching in Section 3.2.2, enactment through data replay in Section 3.2.3, adaptive monitoring in Section 3.2.4 and load shedding in Section 3.2.5. In Section 3.2.6 we summarize the state of realizing enactment patterns.

3.2.1 Adaptive Crawling

The default approach for crawling social media, like e.g., Twitter, is to look for specific keywords and to extract only posts (tweets) containing these keywords. In most cases this strategy pays off, especially when searching for names of stocks or companies. But there are also cases where new relevant keywords emerge or where the meaning of a specific keyword changes over time. In order to approach the issue of new keywords, we introduced the concept of (self-)adaptive crawling in D2.1 and D4.2. The main idea is to maintain a dynamic list of search terms and to dynamically increase the number of relevant tweets crawled, which will then be forwarded to subsequent QualiMaster processing components. Therefore, we introduced a co-occurrence based approach in D4.2 that looks for keywords that often co-occur with one of the given seed keywords within a time window. The evaluation presented in D4.2 shows that while the approach is able to increase the number of relevant keywords, it still produces a lot of noise. The reason for this is that unrelated or ambiguous keywords can often be reached within a single step.

The problem of noise production through crawling of co-occurring keywords has been approached by [38]. The authors propose an approach that first detects candidates for adaptive crawling through co-occurrences with seed keywords and then validates these keywords using correlation of the number of tweets within a time window. The authors evaluate their algorithm by comparing it to a noisy algorithm that includes only co-occurrences. For both algorithms they manually label the tweets crawled during two known events. They show that their algorithm is able to retain 78% of the relevant tweets found by the co-occurrence based approach while removing 87% of the noise, i.e. not relevant tweets.

hashtag	financial?	Related to $AAPL?
#investing	yes	no
#stockmarket	yes	no
#options	yes	no
#applewatch	yes	yes
#finance	yes	no

Table 4: Hashtags found through adaptive crawling with seed term $AAPL.

We implemented an approach that stays close to the idea presented in [38] and tested it on our dataset of financial tweets from May 2015. We decided to only include hashtag terms. Table 4 details the first 5 hashtags found by the adaptive crawling approach when given the seed term $AAPL. While all 5 hashtags are related to the financial market only one of them is directly related to the Apple company. This illustrates the difficulty of noise reduction in adaptive Twitter crawling.
In order to further reduce the noise that is caused by the adaptive crawler we are planning to implement an approach that is able to validate candidate hashtags over time before adding them to the set of search terms. We are also planning to evaluate the different approaches on a non-topic-related dataset to avoid bias. We will use the evaluation procedure presented in [38] to validate that our algorithm is able to reduce the number of non-relevant tweets.

3.2.2 Formalizing “Safe” Algorithm Switching

Switching among alternative algorithms is one of key adaptation enactments in QualiMaster (enactment pattern EP-2 as well as EP-4 for hardware-based processing in D4.1). As analyzed in Deliverable D4.2, switching among sub-topology-based (distributed) algorithms can cause a backlog of data items in a slower alternative algorithm, which lets the slower algorithm continue processing even after the switch. This can cause duplicated data items in the output stream affecting the results of subsequent processing components. To minimize such impact, in [32] and Deliverable D5.3 presented an improved approach based on Entrance Queues. This approach synchronizes the processing with the target algorithm and transfers queued items before the actual switch. The evaluation result of this approach showed an extreme improvement on the switching time reduced from 23s in the original switch to 250 ms. However, in the Entrance Queue approach, we considered alternative algorithms as black boxes without characteristics that may influence the optimal point of time for switching, i.e., the switch is always triggered at a fixed point, e.g., immediately after warming up the target algorithm (D4.2, Section 2.5). This may still lead to certain disturbances on the output stream, e.g., switching somewhere in the analysis window of a sliding-window algorithm is processing can lead to inaccurate results.

To avoid the potential problem discussed above, we propose the concept of a “safe” (transparent, gap-free) switch, which takes the characteristics of alternative algorithms into account. The idea is that algorithm characteristics determine a feasible if not even optimal point in time for switching. For determining this “safe” point, we aim at a classification of stream processing algorithms for switching, which finally leads to the design and implementation an extensible switching framework. We believe that the envisioned framework can simplify the realization / generation of such algorithm switches as well as experimenting with alternative switching designs. Moreover, we see the switching framework as a separately exploitable asset on top of the Execution Systems, in QualiMaster on Apache Storm and Maxeler reconfigurable Data Flow Engines. In this section, we discuss the current state of our classification as well as a related ongoing systematic literature review aiming at identifying further knowledge for the classification and the framework.

In general, a stream processing algorithm can either maintain an internal state while processing or process data items independently of each other [2]. Such characteristics of processing algorithms can significantly influence in determining the safe point for switching. Basically, we assume that all

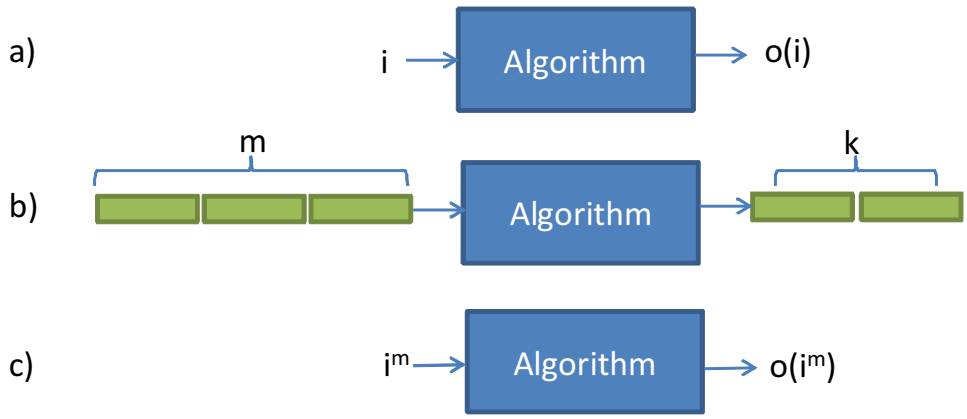

Figure 22: Three types of processing algorithms in different characteristics, a) stateless b) k-monotonic stateful and c) historic stateful.

algorithms are **terminating** for a given input item within endless time and, for now, that the underlying data processing is **free of errors**. From the viewpoint of algorithm state, we classify algorithms as follows:

Stateless algorithms. As the name indicates, algorithms of this type do not maintain an internal state for data processing. The algorithm performs calculations on an item-by-item basis without storing or accessing data structures created as a side-effect of data processing earlier. This is illustrated in Figure 22a). Stateless algorithms allow switching algorithms without need for considering the algorithm state, i.e., the algorithm switch can be triggered at any point without affecting the accuracy of the final processing result. For instance, we can perform an immediate switch on the algorithms, which only normalize, filter or project the data items.

Stateful algorithms. In contrast to stateless algorithms, a stateful algorithm creates and maintains an internal state during processing. On the one side, the current state of the algorithm typically affects the processing results the algorithm produces. On the other side, the algorithm state can be altered by each processing of individual data item. As the input stream is conceptually endless, the state maintained by an algorithm can be seen as a synopsis of the items received so far, a subset of recent items kept in a window buffer, a collection of handles to external data (from non-stream sources), or a combination of all these [2]. For stateful algorithms, state accumulation is usually used to inspect data over either a period of time or until a certain number of data items have been seen, such as an aggregation algorithm, or time series stream processing. However, the state accumulation varies among different stateful algorithms. For performing a safe switch on stateful algorithms, we need to further categorize the algorithms due to their form of state accumulations. We consider the following sub-types:

- **k-monotonic.** The state of this type of algorithms can be separated into independent partitions. As illustrated in Figure 22b), a k-monotonic stateful algorithm imposes a monotonic relation between its input data and output results. Let m and k be the number of data blocks representing individual partitions of input and output data, whereby each data block can be composed of single data items or (micro-)batches / transactions of data. The k-monotonic stateful algorithm consumes m blocks of input data and returns k blocks of output results, i.e., the relation of its input and output within an independent state is $(i_1, \ldots, i_m) \rightarrow (o_1, \ldots, o_k)$, whereby the $(o_1, \ldots, o_k)$ may be arbitrarily reordered. k-monotonic algorithms allow to perform a safe switch after each independent state. In each state we can slice the input or output streams into data partitions. The safe point for switching can then be determined either at the input side, i.e., after each input partition, or on the output side, i.e., after each output partition. Depending on whether m or k is changing during processing, there are basically two cases for determining a safe switch point: 1) if either m or k is fixed value (static), we can find the safe point by counting the block numbers. 2) if m and k are varying (dynamic), e.g., as the output only contains changes due to input data,

the monotonic relation between input and output is changing over time. In this case, we aim at relying on a safe point marker provided by algorithm as part of the output stream.

- **Historic.** A historic stateful algorithm does not have clear partition on either input or output stream as its processing relies on an aggregation of all historically maintained states. This means there is no exact safe switch point for this type of algorithms. For safe switching, we need to make an optimizing compromise, i.e., switching at a point where the disturbance caused on the output result is acceptable. One option is to rely on an algorithm-specific threshold. As illustrated in Figure 22c), let i^m be the currently processing input data, whereby the m indicates the position in the input stream. Let $o(i^m)$ be the output calculated for / until input data i^m. Also, consider ε being the threshold indicating an acceptable difference between the running and the target algorithm to switch to. In this case, switching can happen, i.e., warm-up of the target algorithm can stop if $|o(i^c) - o(i^m)| < \varepsilon$.

To obtain more insights into the adaptation of data stream processing algorithms, in particular, approaches for switching algorithms at run-time, we are conducting a systematic literature review [23, 21] on the research question of *Which runtime adaptation approaches/mechanisms exist in the execution phase of data stream processing?* Currently, the literature search includes the digital library of ACM as well as ScienceDirect. As shown in table Table 5, based on a search string derived from the survey review question, we found 98 papers in ACM and 606 papers in ScienceDirect. For the study selection process, we have completed the first stage of selection by reading the title and abstract of studies. As a result of this selection stage, we included 137 papers and 27 papers are considered as uncertain to be checked in the next stage of selection.

We summarize now the preliminary result from the included papers of the first selection stage in different categories of adaptation on data stream processing algorithms as follows:

- The typical approach in literature is the **parameter adaptation**, which aims at tuning the specific parameters or settings of a processing algorithm to achieve an optimal performance. One example is adapting the window size in a sliding-window-based Top-k query algorithm [40]. Parameter adaptation corresponds to the parameter change enactment pattern EP-1 introduced in D4.1.
- Another important adaptation we found is **changing the parallelization of operators or algorithms** to scale up/down data processing, such as [29] and [7]. In QualiMaster, we support similar adaptation through the enactment pattern for parallelizing processing elements EP-5. As the literature shows, parallelizing stateful processing is a challenging task requiring migration of the accumulated state as also discussed in D4.2.
- Moreover, **resource scheduling**, also known as resource allocation and operator placement, aims at distributing the processing tasks among the available resources to achieve the most efficient resource usage. For example, Calasanz et al. re-allocate unused resources amongst data streams to bursty streams [37]. In QualiMaster, we consider this as a specific form of EP-5.
- To cope with the situation that the overall load exceeds given available resources, **load shedding** is proposed in [10]. As discussed in Section 2.1.5, this is covered by enactment

Digital library	Number of papers (total)	First stage selection		
		Included papers	Excluded papers	Uncertain papers
ACM	98	40	52	6
ScienceDirect	606	97	488	21

Table 5: First results of Systematic Literature Review on stream processing adaptation.

pattern EP-3 and supported in the mean time by the QualiMaster infrastructure.

- Some work also changes the **processing schedule**, also known as query plan scheduling. This aims at achieving an optimal processing plan to incrementally execute streaming algorithms, such as [24]. This form of adaptation would require structurally changing the data processing pipelines / topologies (ES-9), which is more complicated, as QualiMaster pipelines are based on algorithms rather than query-operators with known semantics.

In summary, based on the adaptation categories listed above, there are interesting adaptation mechanisms, which have been already considered in QualiMaster. However, so far we do not see any adaptation approach on switching among alternative stream processing algorithms at runtime (except for our one in [32]), in particular aiming at minimizing the impact on the involved data streams.
As discussed in Section 2.3, we realize algorithm switches through generated code created by the QualiMaster platform instantiation process. Currently, the code generation produces plain code. However, to increase the flexibility also for the upcoming experiments, we are currently designing a switching toolbox framework, which eases generation and programming of algorithm switches for stream processing applications. The initial design includes in particular switching strategies, data acknowledgment mechanisms, pre-defined communication signals, and components for easing the network-based integration of algorithms into the hosting topology, e.g., for creating standalone sub-topologies.

3.2.3 Data Replay

As introduced in Section 0, the Data Replay mechanism is primarily intended to enable the stakeholder applications obtaining historical pipeline results. From the resource perspective, an active data replay consumes additional resources that can be saved in extreme situations. Thus, we consider the replay of data to stakeholder applications as an optional service, that can be denied, reduced or even be terminated as an adaptation enactment. In essence, this form of execution can be performed by re-configuring the existing Data Replay information in the configuration model at runtime and informing the Data Replay Sink about this kind of (internal) parameter change as part of the enactment. From the point of view of enactment mechanisms, adapting the data replay is then a form of changing a functional parameter (EP-1 in D4.1) on the replay sinks. As shown in D4.2, parameter changes require around 10ms enactment time depending on the underlying network topology, i.e., enacting a change of activated data replay mechanisms can be done rather quickly reusing existing mechanisms of the QualiMaster infrastructure.

3.2.4 Adaptive Monitoring

Adaptive monitoring determines the monitoring period for cluster resource monitoring, pipeline monitoring and pipeline element (resource) monitoring, if adequate for individual observables. Such observables can be those causing resource-intensive monitoring, e.g., memory allocation of algorithms or the estimation of the transferred volume based on memory consumption of the data items. This corresponds to the secondary enactment pattern ES-6 from D4.1. As mentioned in Section 2.1.6, adaptive monitoring can be enacted by re-configuring the respective monitoring frequencies, causing a change command and, if needed in case of pipeline nodes, a subsequent change signal. Technically, this can be realized as another changing an internal functional parameter (via EP-1 in D4.1), requiring around 10ms execution time as for algorithm parameters.

3.2.5 Load Shedding

Load shedding as introduced in Section 2.1.5 is a re-configuration option for extreme infrastructure overload situations, in particular if also resource re-allocation, data replay (Section 3.2.3) and adaptive monitoring (Section 3.2.5) cannot mitigate the situation. Realizing the primary enactment pattern EP-3 from D4.1, we equipped each pipeline node with a potential load shedder. Akin to data replay (Section 3.2.4) and adaptive monitoring (Section 3.2.5), enacting load shedding is performed by re-configuring the respective pipeline node in terms of an internal parameter change. Technically, this is realized by sending a load shedding command to the QualiMaster infrastructure

during enactment, which is turned into a signal and sent to the respective node. Akin to algorithm parameters (EP-1 in D4.1), enacting or disabling load shedding is similar to enactment time of a parameter change (10 ms). Depending on the respective load shedder, the impact can be configured as a shedder parameter. For example, for the default static shedder, the parameter *n* indicates the *n*-th data item to throw away. Consequently, not only the presence of a shedder but also its impact can be re-configured at runtime.

3.2.6 Summary

Realizing enactment mechanisms for the QualiMaster infrastructure is in a consolidating state, but also improving and enhancing. The relevant enactment patterns discussed in D4.1 are already realized or in refinement. In more details, all primary enactment patterns are realized and most of them are already analyzed. The secondary enactment patterns introduced in D4.1 are used to realize the primary enactment patterns, e.g., we combined data buffering (ES-1), execute tasks in parallel (ES-5), re-routing of streams (ES-8), warm-up in terms of ES-11 and synchronizing enactment and data processing (ES-12) into one efficient approach for switching among distributed data analysis algorithms [32]. In case the hardware-based algorithms are involved in an algorithm switch, also ES-2 is applied. However, it is important to recap that, as discussed in D4.1, the secondary patterns are intended to support implementing the primary enactment patters, i.e., not all patters must be implemented in the final version of the adaptation approach and the QualiMaster infrastructure.

3.3 Adaptation Control

In this section, we discuss the recent state of the adaptation control in terms of support for the four main kinds of adaptation in QualiMaster, namely reactive adaptation (Section 3.3.1), proactive adaptation (Section 3.3.2), cross-pipeline adaptation (Section 3.3.3) and reflective adaptation (Section 3.3.4). Finally, in Section 3.3.5 we discuss briefly how to achieve an overarching control over all kinds of adaptation and summarize this section.

3.3.1 Reactive Adaptation

Reactive adaptation aims at detecting a certain situation and at mitigating the situation by an immediate reaction. In QualiMaster, we rely on adaptation triggers (events) that indicate the need for an adaptation. Adaptation triggers can be caused by an analysis of configuration constraints at runtime or through programmed code. In this section, we discuss the state of making reactive adaptation decisions, a) base on constraint analysis (Section 3.3.1.1) and b) due to programmed triggers (Section 3.3.1.2). Moreover, we discuss longer lasting enactments as a consequence of realized enactment patterns and a specific adaptation prevention trigger in Section 3.3.1.3.

3.3.1.1 Configuration Constraints

Configuration constraints are a generic way of describing adaptation triggers in an integrated way with the Configuration without need for programming [13]. As discussed above and in D4.1, IVML configurations are described in terms of typed variables. Typically, variables aim at validating a configuration for instantiation, i.e., they are defined on variables which are frozen before runtime. In addition, the QualiMaster configuration model contains variables that remain undefined / unfrozen until runtime. Then, these variables are bound with values by the Monitoring Layer / the actual execution state and the related constraints can trigger adaptations. In the configuration, we currently support two types of constraints:

Infrastructure constraints define the operation boundaries for an installed QualiMaster infrastructure independent of the configured algorithms and pipelines. Examples are resource constraints, such as an upper load boundary for a pipeline (aiming at triggering load shedding) or lower and upper load boundaries for the pipeline elements (aiming at re-allocating resources). These constraints are defined as part of the Configuration (Meta) Model, i.e., they (typically) become part of every QualiMaster installation. These constraints do not relay on fixed values rather than configuration variables so that the constraints can be customized, e.g., to determine the load shedding boundary for a certain installation. Figure 23 illustrates the related configuration

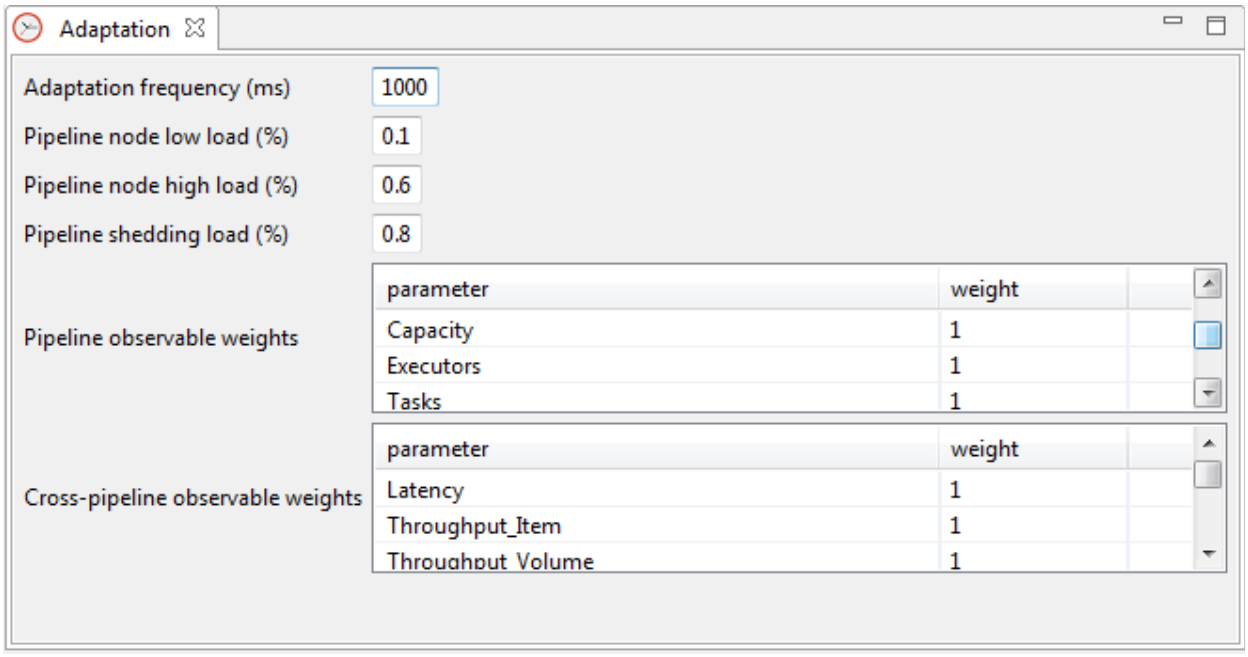

Figure 23: Configuring high-level adaptation settings in QM-IConf, in particular the load boundaries and the weights for pipeline and cross-pipeline adaptation.

options for the adaptation in QM-IConf (load settings on top of the figure) in terms of the high-level adaptation settings (already mentioned in Section 2.4.1, realizing UC-AM6 in D2.2).

Figure 24 illustrates the lower / upper bound constraint for pipeline elements. The two boundary constraints are defined on runtime variables bound by values obtained from the Monitoring Layer. The two variables `capacityHighWatermark` and `capacityLowWatermark` are global settings (defined outside Figure 24) to customize and tune an infrastructure installation (shown as pipeline node low and high load in Figure 23).

User-defined constraints capture the algorithm and pipeline specific operation conditions. In contrast to infrastructure constraints, which are defined as part of the Configuration (Meta) Model, user-defined constraints are part of the configuration. Thus, a user can define these constraints directly in the Configuration or, of course more comfortably, through QM-IConf. Figure 25 illustrates the related EASy-Producer IVML constraint editor applied to the context of a pipeline node. The main window is separated into two parts: The upper part lists the configuration variables in the selected context (here, a pipeline node) with respective descriptions. The lower part is the content-assisted constraint editor, which helps on request with the actual available variables (illustrated in Figure 25) or operations such as comparisons. The window in the background allows creating, editing, or deleting multiple constraints for a single pipeline node.

Figure 26 illustrates the configuration of the software financial correlation algorithm. In the first line,

```
abstract compound PipelineNode refines PipelineElement {
  //details omitted
  assign (bindingTime = BindingTime.runtimeMon) to {
    Capacity capacity;
    Tasks tasks;
    capacity <= capacityHighWatermark;
    executors > 1 implies capacity >= capacityLowWatermark;
  }
}
```

Figure 24: IVML fragment showing the constraints for defining the upper and lower load boundaries for pipeline elements.

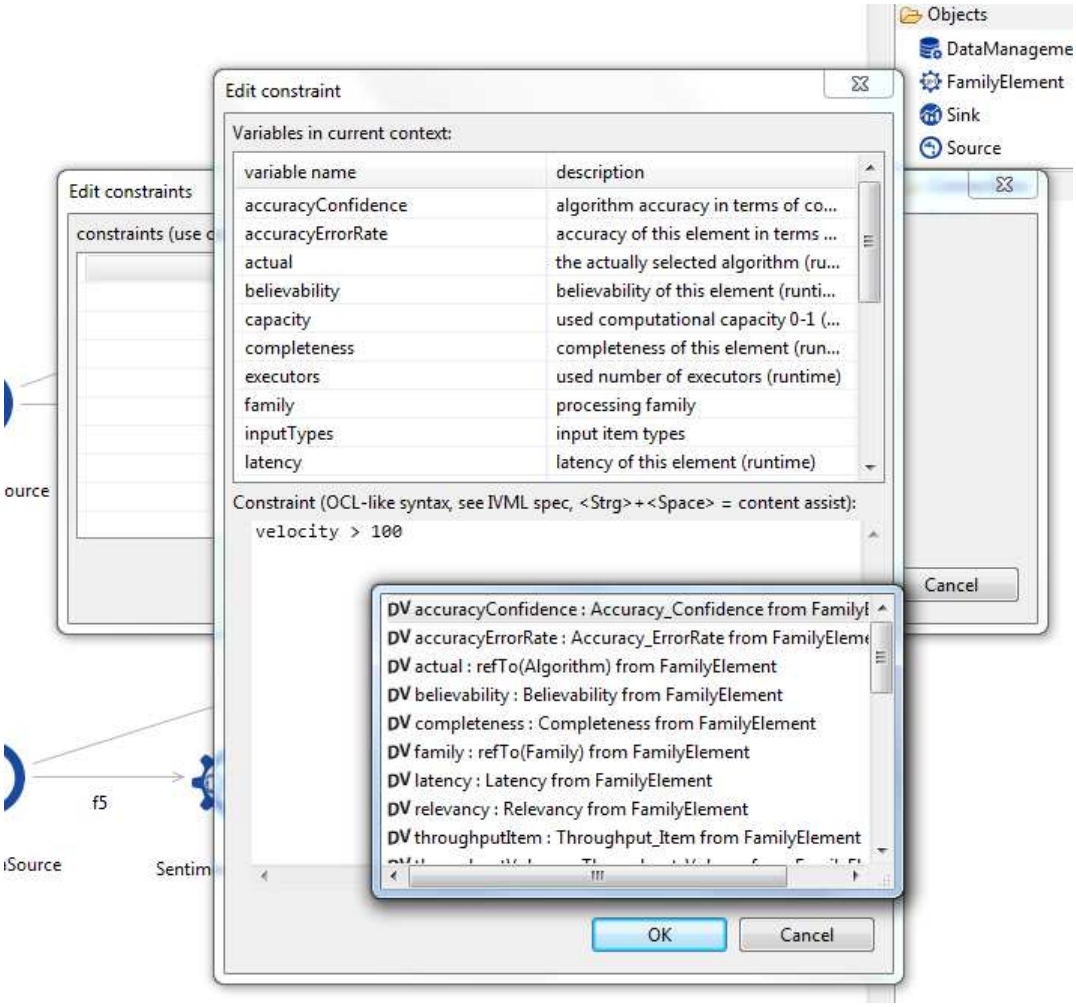

Figure 25: QM-IConf constraint editor based on capabilities of the configuration core (EASy-Producer), here applied to a pipeline node.

the configuration variable representing the algorithm is declared and basic settings such as its (logical) name and the implementing artifact are given. Technically, the constraint slot of an algorithm is declared as an IVML container variable containing constraints, here of type `setOf(Constraint)`. This enables that a user can add arbitrary (conjunctive) constraints. At the end of Figure 26, we define a simple constraint in terms of a fixed upper limit for the items per second (measured at the input side of the containing family represented by the runtime variable `family_Items`) that can be handled by the configured algorithm.

After binding actual monitoring values to the runtime variables, we execute the IVML reasoner (see

```
SoftwareAlgorithm SWTopoCorrelationFinancial = {
  name = "TopoSoftwareCorrelationFinancial",
  artifact =
    "eu.qualimaster:hy-correlation-financial-software:3.2-SNAPSHOT",
  // omitted details, input/output behavior
  constraints = {
    family_Items < 850
  }
}
```

Figure 26: IVML fragment for a user-defined constraint disabling the software correlation algorithm at a certain input rate.

Section 2.2 and D4.2 Section 4.1) to detect constraint violations. In addition to pre-runtime constraints, which must be valid for infrastructure instantiation, we must distinguish here between further points in time when constraints may become active. This corresponds to the concept of binding times in Software Product Lines, i.e., the point in time when the underlying variables must be defined. For runtime, we currently consider the following two specific binding times:

- **Runtime monitoring:** Constraints based on variables assigned to this binding time are bound by monitored values and are supposed to indicate adaptation needs (see also Figure 24). The needs are determined by the constraint analysis, which is part of the Monitoring Layer. Needs are expressed in terms of a constraint violation event indicating the violated variable, the monitoring system state, and, if possible, the degree of violation in terms of the deviation from the expected value. However, if adaptive decisions are made and about to be enacted, these constraints can be valid again, but must not be valid as the enactment did not take place so far. In this case, the underlying Execution System did not have time to execute the enactments and the Monitoring Layer was not able to stabilize a new, valid system state. As we use reasoning also for determining the validity of the re-configuration to be enacted (see D4.1 and D4.2), we must exclude constraints based on runtime monitoring variables when validating a new runtime configuration.
- **Runtime Enactment:** As opposed to runtime monitoring variables, runtime enactment variables represent the actual/new runtime configuration. Changes to these variables shall (if valid) lead to an enactment of the manifested adaptation decisions. Typically, these variables shall only be validated during adaptive decision making, i.e., the planning stage.

In addition to constraint violations, the constraint analysis can also detect that past violations disappeared. While for the constraints shown above the end of a violation is only of limited interest, in case of protecting the infrastructure against overload by load shedding (Section 3.2.1) it is essential to detect that a causing violation is decreasing in terms of its deviation or does not exist anymore. For example, for load shedding we defined a specific tactic, which gradually increases load shedding as long as the violation condition exists, decreases it if the deviations decrease and ultimately disables load shedding if the violation does not exist anymore.

Based on the binding time, the constraints defined in the Configuration (Meta) Model can cause three effects:

- **Constraint violation** of a quality property (typically an infrastructure constraint) leads to a constraint violation event and a subsequent adaptation based on the monitored system state at the time of the violation. If possible, the constraint analysis detects the deviation from the expected value (based on the first failing sub-expression reported by the reasoner) so that the rt-VIL adaptation script can prioritize multiple constraint violations. In the adaptation script, the top-level strategy receives this adaptation event and, in the QualiMaster adaptation script, handles constraint violation events through a specific tactic. This tactic distinguishes the violated constraints based on the affected observable and tries to mitigate the violation by resource re-allocation, disabling data replay or load shedding.
- **Invalid configurable element**, e.g., an algorithm is indicated by a user-constraint defined within that element as invalid. The respective indicates this state and can be filtered out of the available alternatives while processing a respective rt-VIL adaptation tactic.
- **Invalid configuration** during adaptive decision making. At the end of executing a rt-VIL strategy or tactic, the reasoner validates the changed configuration (see D4.2). To increase the performance, reasoning happens only if the actual runtime configuration was changed. This is determined through the transaction-based change log of the rt-VIL language infrastructure, which is also used to roll back the effects of a tactic or an adaptation in case of an invalid configuration.

So far, we applied constraint-based reactive adaptation to enable resource adaptation, disabling data replay, controlling load shedding as well as to realize adaptive runtime algorithm exchange, e.g., from software-based to hardware-based correlation computation. The recent state allows covering the reactive adaptation in QualiMaster based on configuration constraints. Further

triggers will be discussed in the next section (Section 3.3.1.2) and an overall evaluation of the reactive part of the adaptation will be given in Section 4.1. The reactive adaptation is complemented by the predictive / proactive adaptation (Section 3.3.2), the cross-pipeline adaptation (Section 3.3.3) and the reflective adaptation (Section 3.3.4)

3.3.1.2 Programmed Triggers

Adaptation triggers can also be caused other sources as envisioned in D4.1. Two specific triggers that are relevant to the current QualiMaster pipelines are

- **Internal (algorithm) trigger:** An algorithm requests an internal change of the data processing. One example is the event detection, which can request a change of the keywords to be used in Twitter crawling, e.g., to handle a changing analysis focus. Here, the event detection algorithm indicates the need for the adaptation and the Adaptation Layer handles this trigger by a specific (event-related) strategy. Here, a strategy can explicitly fail, e.g., in case that the requested trigger is not valid. Moreover, we aim at preventing adaptations that could overload the infrastructure, e.g., by requesting keywords that will likely lead to a massive increase of input volume. However, this already requests predictive adaptation capabilities and will be discussed in Section 3.3.2.
- **User trigger:** A stakeholder application aims at an (external) change of processing, e.g., following the suggestions of the focus recommendation. Akin to the internal trigger, the Adaptation Layer defines a specific strategy for these triggers. In contrast to the internal (algorithm) trigger, the infrastructure marks this trigger as an external request of lower priority enabling the adaptation to reject this trigger more easily (by a failing tactic).

In D4.2, we envisioned further adaptation triggers, in particular

- **Regular schedule**, e.g., to trigger an adaptation by the infrastructure based on a regular schedule to ensure that the adaptation is executed in a regular fashion. Although the concepts as well as the implementation for a regular schedule are prepared, so far no pipeline needed this form of adaptation.
- **Event-based schedule**, e.g., based on the event prediction (see Section 3.3.2.3). In this case, the adaptation request will be submitted indicating a future point in time. The Adaptation Layer schedules this request and executes it based on the indicated time and impact similar to the regular schedule. This form of adaptation will be realized as part of the integration of the final event prediction mechanism.
- **Maintenance operations**, e.g., to disable cluster machines for a certain time requiring the migration of processes. This form of trigger corresponds to the internal (algorithm) trigger with a specific adaptation strategy (or a user trigger with higher priority). We plan to validate maintenance-based adaptation as part of the evaluation of the adaptation scenarios (see Section 4.2).
- **Changes to the configuration model** correspond to re-configurations and are handled through the mechanisms explained above. In addition, we plan to equip the QualiMaster infrastructure with a mechanism to reload an evolved version of the Configuration Model to enable evolution.
- **Execution errors** may also cause adaptations. Potential strategies for handling execution errors have been discussed in D4.2 as part of the adaptation scenarios (A-4). There, we assigned a low realization priority to execution error adaptation scenario so that the related triggers and strategies will be realized if we decide to analyze scenario A-4. It is important to mention that processing elements (WP2) and the infrastructure instantiation (WP5) realized a mechanism to enable fallback of the processing to the default values in the configuration as one measure to internally manage failing executions, in particular during testing.

In summary, the proposed mechanisms for reactive adaptations have been realized, tested and are in use in the QualiMaster infrastructure.

3.3.1.3 Longer Lasting Enactments

Adaptation of stream processing systems such as the QualiMaster infrastructure imposes specific adaptation challenges, e.g., the need for transparent, nearly gap-free adaptation enactments. One particular example is the runtime exchange of stream processing algorithms, in particular distributed algorithms such as the correlation computation. As discussed in Section 3.2.2, queuing effects within the algorithms can prevent the actual enactment (exchange of algorithms) and even disturb the output streams. Therefore, we designed and evaluated a switch mechanism [32], which performs a warm-up of the target algorithm. As the warm-up time depends on the analysis window of the algorithm, enactments cannot be considered as immediate or timeless (as it is more or less the case for changing algorithm parameters as discussed in D4.2). This is characterized by the enactment time defined in the QualiMaster quality taxonomy in D4.1/D4.2. However, during such an execution time, further adaptations on the same pipeline element, e.g. caused by an increasing deviation of defined constraints, or even the same pipeline may be disturbing or even illegal. The idea here is to consider enactment responses, i.e., to extend the event paths of the QualiMaster infrastructure in a way that interested components are informed about the duration and the result of an enactment. On the Monitoring Layer, this is used to observe the enactment time. The Adaptation Layer can then effectively prevent adaptations during longer lasting enactments. Moreover, the adaptation script can determine the impact of the enactment on running pipelines and, e.g., for the time of the enactment prevent further adaptations on subsequent pipeline elements or the whole pipeline. Further, we plan to utilize the enactment time to reduce the danger of oscillating adaptations, which we plan to take further into account if they occur during the evaluation of the adaptation scenarios (Section 4.2).

3.3.2 Proactive / Predictive Adaptation

Proactive adaptation aims at a more sustainable adaptation by taking context analyses into account, e.g., using trend predictions. Predictive adaptation is in the particular focus of WP4 for the beginning of the third project period. In this section, we discuss the actual state of the predictive adaptation and follow the recommendation of the reviewers to take learning-based approaches into account, in particular for learning algorithm switching. We discuss predicting the volume of data sources (Section 3.3.2.1), the prediction of future external events (Section 3.3.2.2) as well as algorithm profiles for characterizing and learning algorithm tradeoffs (Section 3.3.2.3).

3.3.2.1 Data Source Volume Prediction

Differences in the semantic characteristics of input data can affect the functional quality of an algorithm, since different methods or parameter setups might be optimal only for particular subsets of the input domain. In streaming scenarios, as the ones considered in QualiMaster, a more basic but still crucial characteristic of input data exists: the volume. We focus on this input characteristic within this section. Since different algorithms have different temporal complexities, an estimation of the input volume in the near future allows planning adaptation strategies in advance. For instance, if the amount of input volume (and implicitly data stream volatility) is predicted to become *particularly high* in the *near future*, then a faster algorithm (perhaps less accurate) within the same family or a hardware-based algorithm can be used during. Note that, the meaning of the terms *particularly high* and *near future* depend on the domain requirements and design choices. We will experiment with different time granularities. The temporal advance given by the prediction becomes especially important in case of long enactment times (the current switching approach relies on warming up the target algorithm and requires one analysis window for that, cf. D4.2, D5.3 and [32]) and for avoiding oscillations in the adaptive decisions. We plan to integrate the prediction method into the QualiMaster infrastructure and the adaptation for the upcoming milestone and report about the integration in Deliverable D5.4.

The problem that we tackle in this section can be stated as follows: Given an input source and the series of its volumes over previous time steps, predict the volume at the next time step(s). Although external factors (e.g. events) can trigger unexpected changes, we only consider the history of volumes with the aim of keeping the complexity of the prediction low and, therefore, making it applicable at low granularities (few minutes). In the rest of this section we describe a

method that we have applied to predict input volume, while the experimental data and results are reported in Section 4.1.3. In addition, we aim at using the prediction models to estimate the impact of changes to the data sources, e.g., through subscription to more or less stocks or Twitter keywords enabling the adaptation to judge and reject related (user) triggers.

In the experiments we focused so far on predicting the volume of financial data because this domain is of primary importance in QualiMaster. However, our problem definition is generic and remains applicable to other domains, such as Social Media. The only requirement is being able to observe the Social Stream and derive a time series representing the number of tweets generated within each time period (whose duration depends on design choices) and mentioning a given stock or keyword of interest. Moreover, given the correlation between volume and volatility, this approach can be also used to implicitly predict volatility (included in the QualiMaster quality taxonomy) via the volume over time.

Differently from typical machine learning applications, where data points are assumed to be independent, data within time series has a natural temporal ordering that has to be taken into account when learning any predictive model. The prediction method that we apply consists of (i) modelling the temporal dependency among data points via additional features, and (ii) applying standard learning algorithms suitable for regression on such augmented samples. More precisely, for each point p_t at time t of a given time series, we create an augmented sample by using the values of the previous $w<t$ points $p_{t-w},p_{t-w-1}\ldots,p_{t-1}$ as feature vector $\mathbf{f_t}$ and the value of p_t as its label l_t. The idea is to use a concise but descriptive window of past points as a basis to calculate the next future value of the time series. After experimenting with different number of points in the window, we fixed w = 12 as fair balance between computational complexity and expressiveness. In our experiments we used Weka[4], which is endowed with a time series analysis environment. Doing this for each point will produce a dataset where a learning algorithm can learn how to mix feature values (i.e. recent history) to generate labels (i.e. future values) future ones. Once the prediction model M has been trained, it can be used to predict volume values $l_x = M(f_x)$ at a future time step x. Regarding the learning models, we applied and compared Linear Regression, Support Vector Machines, as well as Multi Layer Perceptrons (all implemented in Weka).

In our experiments (details will be given in Section 4.1.3) we observed that the time required to train the model is relatively short, always below 1 hour when executed on a general purpose laptop with Dual-core Intel i5. This allows retraining the model daily (every night) to always incorporate new data and possibly model new emerging trends. The time required to make predictions once the model has been trained is around few milliseconds, therefore the volume prediction can be run at high frequency (even every minute) as part of the adaptation process without being cumbersome for the whole pipeline.

3.3.2.2 Event Prediction

The occurrence of events can make a pipeline undergo high input volumes, which have to be handled properly not to lose important messages. Therefore, event prediction aims at forecasting the occurrence of events at future time points to let the pipeline adapt on time to potentially peaks of input volumes. Event prediction can be seen as complementary to input volume prediction (Section 3.3.2.1), as the former can predict events at far points in time but with coarse granularity (days at finest), while the latter looks at the immediate future with a very low granularity (up to minutes). Another difference between the two methods is that input volume prediction considers the time series of previous volumes, while event prediction analyzes the content of tweets.

An early estimation of the occurrence of events might be also helpful for adaptive crawling techniques (Section 3.2.1), which could change the set of crawled stocks/keywords in advance based on the content (e.g. participants, topics) of the predicted events.

One of the recommendations of the last review suggested a better coordination between the activities on event detection (WP2) and those on event prediction (WP4). Therefore, before delving

[4] http://www.cs.waikato.ac.nz/ml/weka/

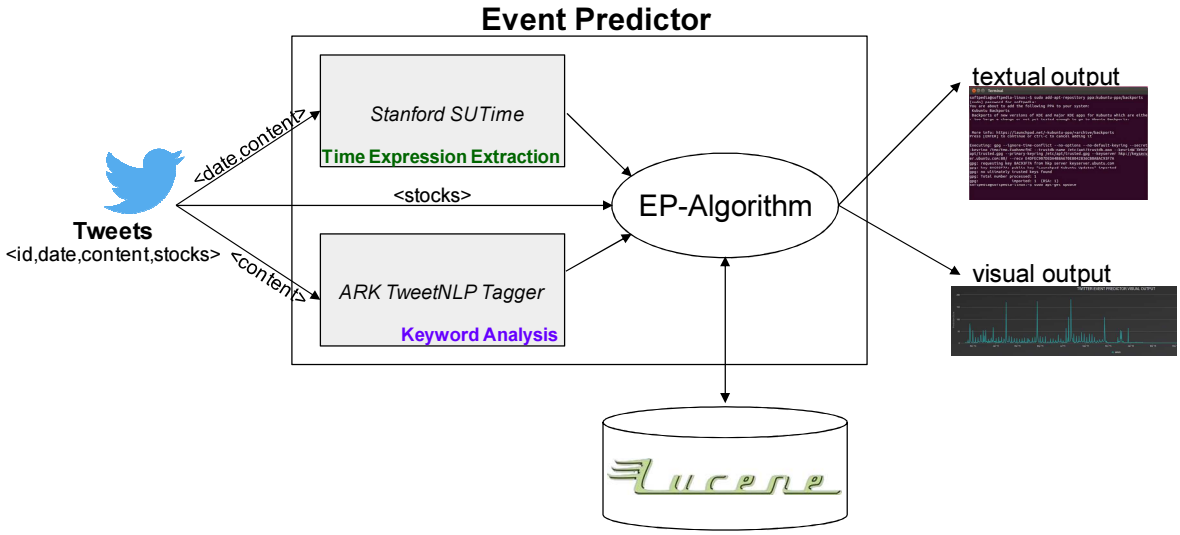

Figure 27: Overview of the event prediction.

into implementation details, we elaborate on the differences and relations between the event prediction of WP4 and the event detection methods developed in WP2. On the one hand, event detection and prediction can be different since, conceptually, the former is more retrospective while the latter is more predictive. Event detection, searching retrospectively for past events, can make use of data (e.g. tweets, news) belonging to points in time after the occurrence of events. Differently, event prediction can only exploit information up to some point in time (i.e. the present) to infer the occurrence of events in the future. On the other hand, such difference opens for collaborations between event detection and prediction and the corresponding work packages. First, thanks to the presence of SUH and LUH in both WP4 and WP2, the teams assigned to event detection and prediction constantly communicate and work in synergy, even sharing people among the teams. Second, the work on event prediction has been meant to reuse the advances achieved in event detection by regarding event prediction as a form of event detection in the future. After estimating the temporal distribution of tweets about a given stock within a future time window, based on the content of past and present tweets, it is possible to derive time series of tweet volumes and apply the event detection methods of WP2 to find events in the future. This is indeed the idea that we follow and that is described in the rest of this section.

Event prediction has been previously discussed in Deliverable D4.2, where a conceptual approach has been outlined. The main idea is to exploit thoughts, rumours, discussions about the future and evolution of market players shared on Social Media to infer the occurrence of future events. The assumption is that, when many people talk about the same stock in combination with a future date, an event might happen at that time. References to future dates in tweets are collected and stored in a calendar of future events. We have implemented that method and went beyond it by developing a bucketing and weighting method to better map the presence of future temporal expression into future events.

Our event prediction method is illustrated in Figure 27. The incoming tweets for a given stock are analyzed by extracting temporal expressions as well as keywords from the textual content. We use Stanford SUTime[5] to extract temporal expressions. It can detect both explicit (e.g., 25th November 2015) and implicit (e.g., next week) temporal expressions as well as at different granularities (e.g. “on Monday” vs “in October”). We also utilize ARK TweetNLP[6] to extract keywords from tweets and use them as summarizing information of the potential event. This extracted information is fed into the event prediction algorithm, which works as follows.

- For each market player, the future time is split in buckets with daily granularity (i.e. one bucket for each day). Each bucket is associated to a value, which represents the number of

[5] http://nlp.stanford.edu/software/sutime.html
[6] http://www.cs.cmu.edu/~ark/TweetNLP/

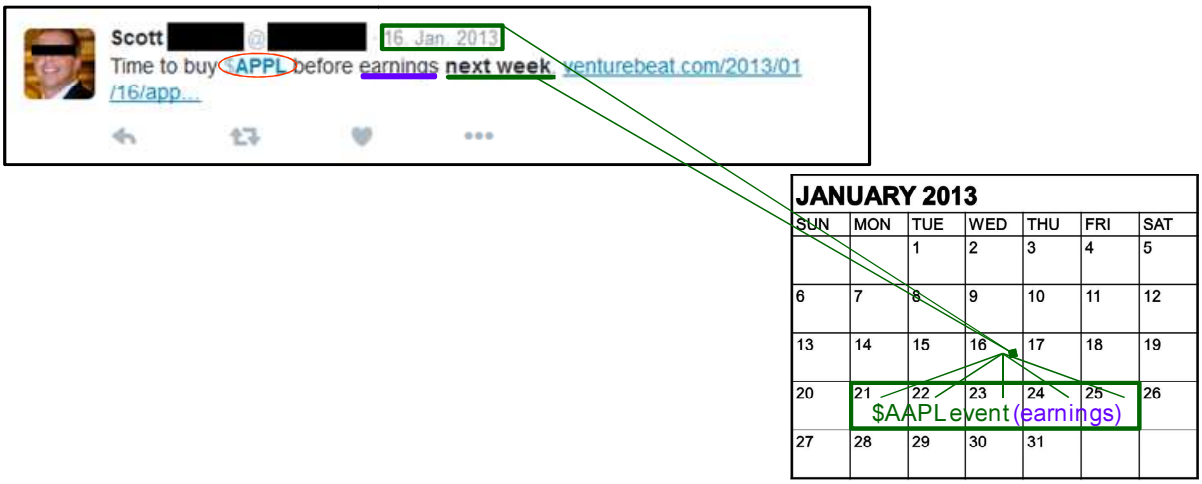

Figure 28: Example of mapping a date into buckets.

past tweets that mention the given stock at that time. The higher the value, the more likely an event will occur at that time.

- Each temporal expression is mapped to one or more buckets, depending on its granularity, and increases the associated value by an amount that depends on its granularity (see next point). We discard dates with yearly granularity (e.g., "next year" or "in 2015") as too general. Implicit dates are resolved by using the time of the tweets they belong to. An example of date-to-bucket mapping is shown in Figure 28.
- We weight the contribution of a date to a bucket based on its granularity. The underlying assumption is that a low-granularity date is more precise, involving fewer days, and should have a higher impact on the buckets it falls into. Therefore, each date is allocated the same value (equal to 1 for simplicity) and this is uniformly spread over the entire bucket it falls into. For instance, a date representing one date will increase the corresponding bucket of 1, while a date spanning one week will increase the 5 buckets (we ignore weekends) by ⅕ .
- The extracted keywords are attached to the bucket to describe what might happen for a given stock at a given time point in the future.

All the event-related information extracted from the tweets, i.e., stocks, keywords, dates, is stored in a Lucene index for later retrieval and exploration.

The output of the event prediction for a given stock is a time series of future tweets volume as the one shown in Figure 29. As mentioned previously, event detection methods, such as those implemented in WP2 can be applied to these time series to identify events as bursts of tweets volumes. We will report examples of predicted events in Section 4.1.4.

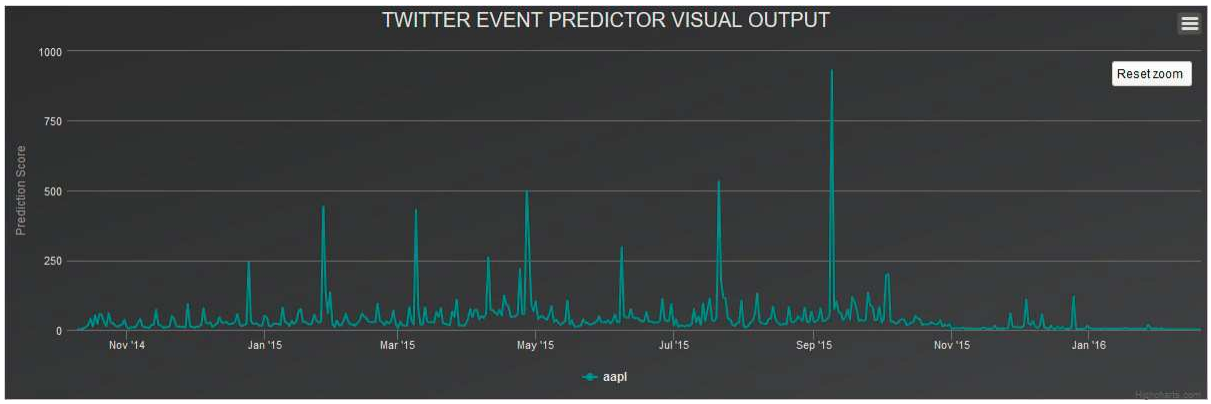

Figure 29: Example of the output of event prediction.

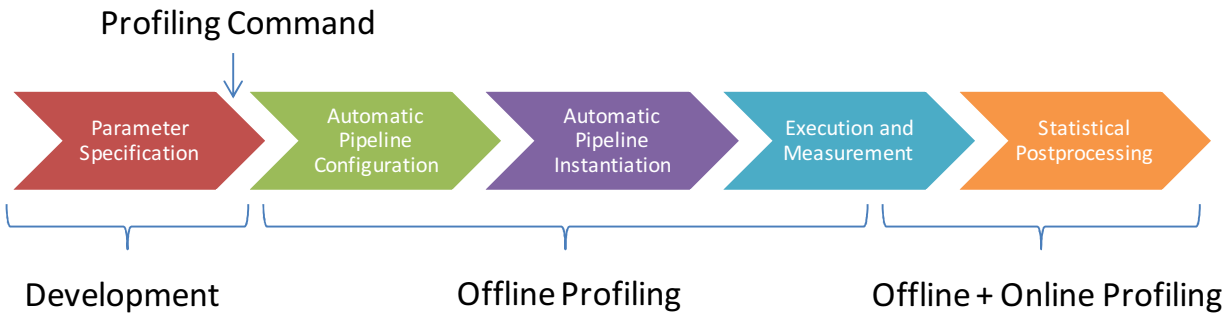

Figure 30: Profiling process

3.3.2.3 Algorithm Profiles

Determining the most appropriate algorithm at runtime, the actual algorithm parameters or the most beneficial distribution of an algorithm over a cluster are central tasks in adapting QualiMaster pipelines. These adaptation tasks can be realized using the reactive adaptation mechanisms, in particular through user-defined constraints. However, expressing the adaptation behaviour in terms of constraints requires detailed information about the individual algorithms, the expected runtime characteristics and their tradeoffs. Moreover, eliciting this information from the Algorithm Providers requires intensive testing for this purpose and is typically based on a rather optimistic mental model of the own algorithms, i.e., may include a natural bias. Such estimations can lead to erroneous runtime reconfigurations, e.g., if the behaviour of an algorithm was under- or over-estimated for certain situations. So far, we relied on elicitation / constraint-based modelling to enable adaptation for some algorithms and to realize a fallback based on reactive-adaptation according to the overall risk plan in the DoW. In this section, we discuss our approach on utilizing statistical quality profiles of the algorithms, i.e., we motivate the approach, discuss the profiling process as well as the statistical post-processing.

Our core concept for enabling more advanced, predictive algorithm decisions is the **algorithm profile**. An algorithm profile captures the quality characteristics for a given algorithm running under various parameter settings. The parameter space of an algorithm includes distribution settings as well as actual values for the functional parameters of the algorithm. The core idea of algorithm profiling in QualiMaster is to capture the quality characteristics for relevant settings and to compare the profiles to make runtime decisions for the exchange of algorithms (based on algorithm families characterizing the adaptation space), the parameters of the actual algorithm or its parallelization / distribution over the cluster. A *raw profile* consists of data measured by the QualiMaster infrastructure while running the algorithm with a profiling data set. In contrast, a *profile for making decisions* is a statistical representation of the raw profile, which is more suitable for runtime decision making.

Obtaining the profiles involves activities before and at runtime, but also requires making tradeoffs on the profiling efforts as similar to testing not the full parameter space can be profiled for a non-trivial algorithm. During development and algorithm testing, we aim at capturing profiles as basic knowledge to make initial adaptive decisions. At runtime, this knowledge may already be outdated when running a pipeline on a different cluster (as we do for evaluations in QualiMaster, see D5.3). Algorithms may also behave differently under real data and real load than expected by an Algorithm Provider so that potentially other parts of the parameter space are more relevant at runtime. As a response, we aim at updating the profiles at runtime to increase the precision of the adaptive decisions, but also at allowing decisions based on interpolations among available profiles. In more detail, we consider all configurable settings for algorithm *a* as its multi-dimensional **parameter space** including

- Distribution settings $D=\{e, t\}$ in terms of Storm executors *e* and tasks *t*,
- The adaptable parameter set $P(a)$ of algorithm *a*, $P(a)=\{p(a)_1, \ldots, p(a)_n\}$ with $p(a)_i$ being the individual parameters of algorithm *a*. Please refer to D4.1/D4.2, for more details on configuring algorithm parameters.

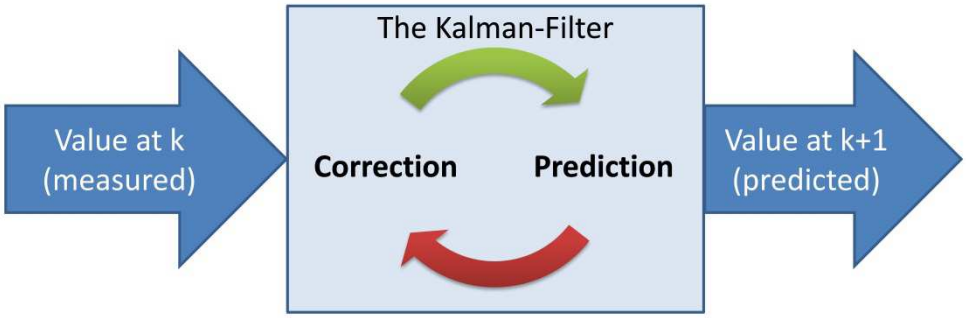

Figure 31: The general Kalman-Filter process.

A distinct point in the parameter space *D x P(a)* of algorithm *a* represents a certain (runtime) configuration of algorithm *a*, e.g., stated as a *n*-tuple of objects O^n (typically real numbers, but also Strings for some algorithm parameters) with $n = |D|+|P(a)|$. Let *Q* be the set of quality parameters defined in D4.1 and D4.2 in terms of the QualiMaster quality taxonomy. A **raw profile for a given parameter setting** is a mapping of a specific point in the parameter space to measured time series characterizing the individual quality parameters for a given profiling input stream *I*, i.e., $O^n(I) \rightarrow \{Q \rightarrow (v_0, v_1, \ldots v_n)\}$, with $(v_0, v_1, \ldots v_n)$ capturing *n* measured quality parameters. It is important to mention that also characteristics of the input data set are captured by monitoring the algorithm execution and, thus, represented as specific time series. A **raw algorithm profile** is then a set of such mappings, i.e., $p(a) = \{O^n(I) \rightarrow \{Q \rightarrow (v_0, v_1, \ldots v_n)\}\}$. Moreover, an **algorithm profile** is a statistical representation of a raw algorithm profile, e.g., represented as a matrix obtained from statistical post-processing.

The process for obtaining algorithm profiles for a given algorithm *a* is depicted in Figure 30. First, the algorithm developer equips the algorithm with a **specification of parameter settings** for one or multiple profiling runs, i.e., instances of O^n, as well as profiling data matching the input item type requirements of the algorithm. Here, the Algorithm Provider can reuse existing parameter specifications or profiling data files by referring to (compatible) existing algorithms. Both, parameter specification as well as profiling data are deployed as a profiling artifact in the processing elements repository along with the algorithm implementation.

Next, algorithm *a* must be integrated into a profiling pipeline, i.e., a pipeline consisting of a data source delivering the profiling data to the algorithm and the algorithm itself. For relieving the users of the QualiMaster infrastructure from configuring profiling pipelines for each algorithm, we automate configuring, instantiating, starting and measuring by respective infrastructure support as we will detail in the upcoming Deliverable D5.4. For profiling algorithm *a*, the infrastructure administrator issues a command stating that algorithm *a* for family *f* (one of its containing families) shall be profiled. The infrastructure now **creates a configuration** for this profiling pipeline based on the existing configuration of algorithm *a.* Then, the infrastructure **instantiates this pipeline** (a shortcut of the full QualiMaster instantiation process, see Section 2.3 as well as D5.3) to create the source code of the profiling pipeline, here in particular also the data source implementation, and to package the pipeline. For each parameter configuration defined in the profiling artifact of the algorithm, the infrastructure now **executes the profiling pipeline** until all profiling data has been processed (explicit termination through the generated data source if it runs out of data). During execution, the Monitoring Layer of the QualiMaster infrastructure **aggregates runtime information** (see D5.3) and traces the aggregated information once a second into a summary file containing the raw algorithm profile. Please note that adaptation for this pipeline is disabled during profiling to observe the plain algorithm behaviour. Moreover, the creation of the raw profile trace is enabled just for profiling pipelines and disabled during normal execution of pipelines.

The final part of the profiling process is the **statistical post-processing**. Here a statistical prediction method is used to capture the raw profile and to predict how the algorithm profile will probably evolve in the future. As the raw algorithm profile is given by a periodic series of numerical values, the prediction corresponds to the forecasting of time series. This forecasting can be done separately from the aggregation of the profiling data and was topic of a number of publications

[34]. While some work applies universal techniques such as ARIMA [8] or space-state models like the Kalman-Filter [20], there are also highly specialized approaches such as ThermoCast [25], DynaMMo [26] and PLiF [27]. ThermoCast uses autoregressive models as baseline for a combination of data mining and learning algorithms to predict the temperatures surrounding servers, based on temperature and airflow measurements. DynaMMo and PLiF focus on mining and summarizing data of multiple time series with a certain focus on effectiveness and efficiency. DynaMMo allows using algorithms, which normally cannot handle missing values inside an (input) time series through hidden variables. In this regard DynaMMo is similar to the Extended Kalman-Filter. While DynaMMo enriches non-linear sequences to be able to handle them as linear sequences, PLiF extracts essential characteristics to generate a linear meta-sequence. The later can then be further analyzed or predicted using approaches like ARIMA or the Kalman-Filter.

From the perspective of monitoring and software performance engineering, two approaches using a statistical are particularly relevant to our approach to profiling, namely RainMon [36] (for predicting monitoring time series in distributed systems) and [41], both relying on an extended version of the Kalman-Filter also aiming at learning component performance at runtime. Due to discussions of the partners in the Work Package and the availability of the implementation, we based the technical realization and the initial experiments (details will be discussed in Section 4.1.5) on RainMon.

In general, the Kalman-Filter [20] is a prediction-correction type estimator, which consists of two main steps prediction and correction. The purpose of the Kalman-Filter is to find the estimated value $\hat{x}_k$ of the (unknown) sequence x at time step k. For this a *prediction* of the next estimation $\hat{x}_k^-$ is made using the last estimation $\hat{x}_{k-1}$. Now the measurement z_k, which is the result of measuring x at timestep k is used together with the predicted estimation $\hat{x}_k^-$ to calculate the estimation for the current timestep $\hat{x}_k$. To summarize:

- $\hat{x}_k^-$ is a priori-estimation to calculate what x_k will be (prediction)
- $\hat{x}_k$ is a posteriori-estimation to calculate what x_k was (correction)

The calculation of those two estimations and their respective covariance is detailed below.

The two steps form a cycle as illustrated in Figure 31. Let A be the prediction model (matrix), $\hat{x}_{k-1}$ and P_{k-1} the last known estimated state and its covariance. During the prediction step, two values are calculated:

- The (predicted) priori estimated state of the value x for time step *k*: $\hat{x}_k^- = A\hat{x}_{k-1}$
- The covariance for the priori estimated state: $P_k^- = AP_{k-1}A^T$

  For the special case of $k = 0$ let $\hat{x}_{k-1} = P_{k-1} = 0$.

In our case, the control matrix and the control vector which are usually added to the estimated state can be omitted, because the external influence on the processed data was already handled during the pre-processing of the data in the QualiMaster Monitoring Layer. The same holds for the environmental uncertainty, which is normally added to the predicted covariance (representing the uncertainty).

The second step uses the defined models (matrices) of the Kalman-Filter, the a prioi estimated (predicted) value $\hat{x}_k^-$ and its covariance P_k^- to calculate or update the Kalman gain (K_k). Afterwards the Kalman gain and the measured new state of the value (z_k) are used to calculate the a posteriori estimation $\hat{x}_k$ for the state of the value x and its covariance P_k. These steps can be formalized as:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$
$$P_k = (I - K_k H)P_k^-$$

where H, R and I are identity matrices.

While this approach in its current form only allows calculating one time step into the future, it is possible to extend the prediction indefinitely by defining $z_k = \hat{x}_k^-$ for each correction-step. It is important to notice that the prediction quality declines with each correction-step without an actual measurement. It is also important to understand that the Kalman gain (K_k) is the only matrix that needs to be stored over multiple iterations or even externally if one wants to continue the predictions-correction circle at a later date or even with other data. Other potential interesting matrices are H and R, but in our case they are static and therefore do not need to be trained or stored. The other computations are based on the last state before the prediction, which enables a Kalman-Filter implementation to be rather memory and response time efficient. In general, complicated and/or dynamically calculated external factors may affect the performance of the Kalman approach, but those are eliminated during recording of the algorithm profile.

As stated above, we rely on the RainMon approach, which has been successfully applied to monitoring time series in distributed systems and has been intensively tested in a case study with more than 220 Gigabyte of time series data [36]. The full RainMon process consists of a *caching/crawling stage* obtaining the monitored values from the system, a combination of *trim*, *resample*, *cypress and spirit* stage to eliminate outlier peaks, to smooth the time series and to extract hidden variables, a *Kalman* stage and a *Normalize* stage.

Thanks to the data being already accumulated and pre-processed by the QualiMaster Monitoring Layer, we can directly start with the Kalman stage, in particular considering relevant spikes as they are recorded in the raw profile. Furthermore no normalization stage at the end is needed, as we aim at using the raw profile data in its original scale / domain.

For the purpose of reports, RainMon allows applying optional *draw stage* can be used to generate graphs representing the measured and predicted values. Although the draw stage is not relevant to the QualiMaster infrastructure itself, as creating and working with profiles happens without user interaction, we consider integrating views on the actual profiles into QM-IConf to enable the Adaptation Manager to get an insight into the models underlying the adaptive decisions.

Performance and prediction experiments based on raw algorithm profiles indicate that the Kalman-Filter can be applied for realizing statistical algorithm profiles. Examples and performance measurements will be discussed in Section 4.1.2. As next steps, we plan to store the Kalman-Models (i.e., the matrices used by the Kalman-Filter) along with the algorithms, to deploy them and load them at runtime to enable a smooth cold start of a given algorithm, potentially considering a kind of virtual time to cope with gaps in times when individual algorithms are not active. While running an algorithm within a QualiMaster pipeline, we aim at updating the information and storing the updated matrices in the Data Management Layer. Further, we plan to design a mechanism for automatic exploration of the parameter space in order to relieve the algorithm designer from specifying the profiling parameters and to achieve even more unbiased profiles.

Although the approach described in this section is primarily intended for algorithm profiling, it can also be used as a technique for testing or evaluating individual algorithms rather than full pipelines.

3.3.3 Cross-pipeline Adaptation

Cross-pipeline adaptation aims at observing and adapting the execution of multiple pipelines on the same infrastructure, i.e., to balance concurrent requests for the same resources. Adaptively optimizing the execution for multiple pipelines is rather similar to calculate an optimal task and resource scheduling and hard to achieve. As mitigation, we aim at pragmatic strategies that balance over- and underused resources among multiple pipelines so that multiple pipelines can be executed at reasonable performance. In particular, this includes considering the start-up phase as well as a global view on resource re-allocation.

When **starting** a pipeline, the underlying cluster resources may already be partly or completely used by other pipelines. If no resources can be freed for the new pipeline, the infrastructure must be able to reject the start-up of a new pipeline. As the decision about permitting a pipeline start-up is dynamic, relies on monitoring information and may require adaptations to running pipelines, the decision shall be made by the Adaptation Layer. This requires involving the Adaptation Layer much earlier in the pipeline lifecycle than proposed in D5.3. In other words, we suggest extending the

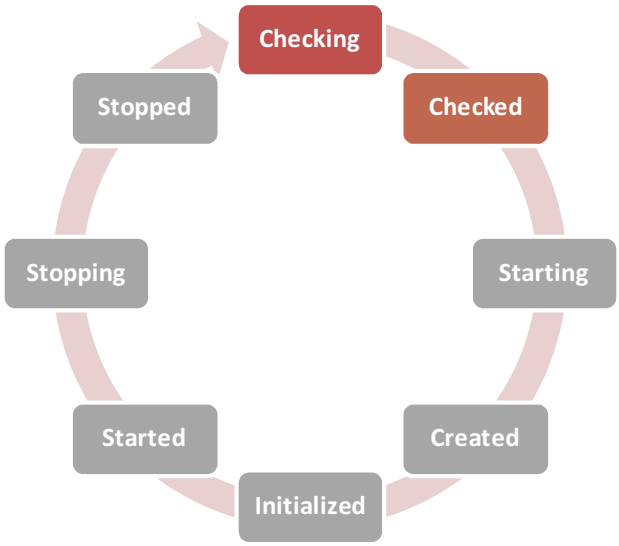

Figure 32: Extended QualiMaster Pipeline Lifecycle realizing adaptive pipeline startup through the new *checking* and the *checked* phase.

QualiMaster pipeline lifecycle by two distinct initial phases namely “checking” and “checked”, as illustrated in Figure 32.

The **checking** state requests the clearance for start-up in terms of analyzing the actual resource usage and determining the available residual resources for the new pipeline. A simple approach is to summarize the number of executors (JVMs) for all pipelines and to compare them with the configured number of maximum permissible executors for the entire cluster. A more detailed analysis can take the actual memory consumption on the machines into account. Based on the analysis, the adaptation layer can apply three tactics:

- **Reducing the resource requests** of the new pipeline by scaling down the number of requested executors and tasks. So far, we can only apply a linear scaling of the resources requested in the configuration (fixed number of executors, tasks and workers) and in source code (manually implemented sub-topologies using the same concepts). Currently, WP2 and WP4 are discussing a more advanced approach of relative resource specifications allowing for a better scalability. This would also enable running the same pipeline without manual adjustments on clusters of different size, e.g., the two clusters provided by TSI.
- **Reducing the resource usage** of the running pipelines. This can be realized through the load balancing / resource re-allocation algorithm defined as part of the rt-VIL adaptation scripts (cf. below), but running the algorithm with a reduced set of available resources (by the new pipeline) rather than all available resources. Moreover, in a second step also reducing the resource requests of the new pipeline can be taken into account.
- **Rejecting the pipeline** as the ultimate ratio if none of the previous tactics can be applied successfully. In terms of rt-VIL, this means explicitly failing the tactic, i.e., the start-up strategy and indicating the Adaptation Layer that the requested start-up cannot be performed. In this case, all infrastructure layers shall be informed about terminating the start-up of the respective pipeline.

If the new pipeline fits into the cluster or can be adjusted as discussed above, the pipeline lifecycle continues in the **checked** state. More technically, if the start-up adaptation strategy completes successfully, the Adaptation Layer indicates the lifecycle state change and informs the other infrastructure layers, which then continue with a usual start-up potentially with adjusted resources for the new pipeline. Based on the reduced resource requests, the pipeline can later be initialized with corresponding algorithms by the Adaptation Layer (initialized state).

At **runtime**, different loads can justify different resource usage as well as resource re-allocation among multiple running pipelines, including changing the allocation of algorithms to reconfigurable hardware. For the resource re-allocation, we implemented a load balancing algorithm inspired by

[1, 6, 39] in rt-VIL, i.e., based on observed resource usage, in particular load / capacity, the algorithm aims at adjusting the thread and process allocation of individual pipeline nodes. As discussed in D4.2 / D5.3, we equipped the QualiMaster infrastructure with runtime resource migration capabilities, i.e., scaling out Storm tasks into own (new) executors, scaling in tasks into other executors (terminating unused executors) or migrating tasks within the cluster. The load balancing algorithm operates on a virtual pool of executors including the allocated and free resources (as physical resources). The algorithm incrementally shifts tasks / executors from underused pipeline nodes to overused pipeline nodes along the pipeline topology. Due to design decisions in Apache Storm, the reconfiguration opportunities are bounded by the configured number of tasks per pipeline node (cf. D5.2 / D5.3). If a pipeline node is over utilized, the algorithm tries to run more tasks on physical resources than on logical resources (if not enough executors are assigned, Storm runs tasks sequentially). Executors for more tasks can be acquired from the virtual executor pool. Similarly, resources are freed in case of underutilized pipeline nodes. This algorithm is executed upon violation of resource constraints as discussed in Section 3.2. Moreover, the algorithm can be applied to multiple pipelines and that the virtual executor pool can be adjusted per execution of the algorithm, so that it can support the aforementioned resource reduction.

When **stopping a pipeline**, the opposite tactics than discussed above can be applied, i.e., after freeing the resources by a stopping pipeline, the adaptation can run the resource re-allocation algorithm to give other pipelines more resources if needed.

The extended pipeline lifecycle for cross-pipeline adaptation has been realized in all QualiMaster layers. The rt-VIL adaptation scripts have been extended to operate with the extended pipeline lifecycle and the assignment of algorithms to resources as well as the initial assignment of algorithm parameters has been implemented in rt-VIL and tested. As described above, also the resource re-allocation algorithm has been implemented in rt-VIL and tested. The cross-pipeline strategies discussed above are currently in realization. We will discuss details in Deliverable D4.4.

Moreover, besides the technical aspects of cross-pipeline adaptation, we plan to analyze how algorithm and algorithm parameter changes can affect other pipelines running on the same cluster. We also plan to analyze how to detect and control the need for sub-sequent adaptations imposed by algorithm and parameter changes as part of our work on validating the adaptation scenarios.

3.3.4 Reflective Adaptation

Reflective adaptation operates at the meta-level, aiming at improving the quality of the infrastructure configuration and the adaptation decisions took by the reactive and proactive adaptation strategies. It observes the state of the infrastructure over time, such as resource allocation and algorithm performance, to assess whether the effects of the selected reactive and proactive adaptation strategies over time are the expected ones and, actually improve the quality of the execution. In this section, we describe in a bottom-up fashion our approach for realizing reflective adaptation: The description of logged data, how it is aggregated, how to predict and measure the quality of adaptation, and how to make this knowledge reusable for other proactive adaptation strategies as well as for the user. The description is at a conceptual level, while we plan to report the implementation details and evaluation results in the next Deliverable D.4.4.

The data observed and exploited by reflective adaptation can be logically separated in the *state* of the infrastructure and the *actions* carried out by reactive and proactive adaptation strategies.

- The state s_t at a given point in time t is a vector containing all the information logged and made available in the infrastructure over time, such as the algorithm parameters (see Section 3.3.2.3), their quality parameters, the resource allocation in the pipelines, its constraints, and the input characteristics (e.g., volume, crawled stocks/hashtags).
- The action a_t is a vector containing all the adaptation decisions applied in the infrastructure at time t, as a result of reactive and proactive adaptation strategies. The action vector is in principle independent from the employed adaptation strategy, as long as their actions/decisions are logged in the same way.

The state s_t is observed and provided by the Monitoring Layer of the QualiMaster infrastructure. For the purpose of algorithm profiling, we are able to trace projections of s_t in a regular fashion. Similarly, we can trace projections of the state s_t in order to record what caused adaptations over time (e.g., critical resource allocations, algorithm quality, etc.) as a basis for assessing the actual effects of the selected adaptation strategies and tactics. Along with the realization of the profiling mechanism, WP5 already extended the Monitoring Layer with a fixed-schedule tracing mechanism, i.e., aggregated information that is relevant for this form of adaptation is persisted and made available to the reflective adaptation. Moreover, the Adaptation Layer provides a similar tracing facility for the executed strategies and tactics, i.e., the technical realization of storing the vector a_t. We further aim at extending rt-VIL with information the intent of strategies and tactics and to provide this information through the traces to the reflective adaptation.

The above mentioned amount of data (states and actions) collected over time is used by reflective adaptation to (i) have a notion of the *context* in which adaptation strategies have been enacted and (ii) evaluate their actual benefits or *quality*.

- The *context* of a given adaptation is modelled by deriving a set of aggregated characteristics from the state logs within a window of recent time steps, such as average values and trends of resource allocations, algorithm qualities, and input volume. Alternatively, in order to have a more granular description of the context, the state values are not aggregated but rather sampled in *n*-bins distributions (with *n* an open parameter). The size of the time window can be chosen according to the scenario requirements and regulates the sensitivity of the adaptation to recent or past time steps.
- Similarly to context, but looking at the future, the state variables measured within a window of time points after the adaptation decision is considered to assess the *quality* of the adaptation. Modelling the quality of adaptation can be done in different ways, with different degrees of complexity. One way is comparing the values of the variable(s) that triggered adaptation in time windows before and after the adaptation time: if the undesired trend is inverted, then the adaptation decision is judged as successful; if not, it is marked as not successful. Another way consists of taking into account more subtle aspects like the adaptation speed and magnitude (how fast and how much the value(s) under adaptation got far from the undesired value(s)). These aspects can be modelled as separate real-valued metrics or be aggregated in a single indicator representing the whole quality an adaptation decision. In addition to the enactment time, which is now recorded by the Monitoring Layer, further time measurements on the adaptation itself can be obtained at runtime as we will illustrate for validation in Section 4.1.6.

Reflective adaptation has both predictive and retroactive applications. Every time adaptation is triggered and reactive / proactive strategies have to select the adaptation decision to apply, reflective adaptation can be used to predict the quality of each candidate decision given the context. Reactive / proactive adaptation strategies can take such predictions into account as a further criterion to make decisions, for instance by using the predicted quality to weight candidate decisions. Of course, having such a predictive model requires historical training data, made of adaptation logs along with their previous context and future context (to measure adaptation quality). The machine learning problem can be formalized as either binary classification (successful vs. not successful adaptation) or regression (prediction of a real-valued quality metric), depending on the way the adaptation quality is modelled (see previous paragraph).

Besides making predictions, reflective adaptation measures the context after taking the adaptation decision (which is not available at prediction time) and assesses the actual outcome of the adaptation (i.e., the prediction goal). This knowledge is used retroactively to expand the training data with new adaptation evidence acquired online, so that the reflective adaptation model can be re-trained periodically and kept up to date. Moreover, adaptation decisions, their context and measured quality are stored as distinct tuples in a log, possibly in a more readable and user-friendly format than the rough feature vector used for learning. This log can be accessed by the Adaptation Manager, e.g., through QM-IConf, who gets aware of the actual quality of the adaptation on the long run and, possibly, can redesign the adaptation strategies according to the collected evidence.

3.3.5 Overall Adaptation Control

For specifying the adaptation strategies and tactics, we use the rt-VIL language introduced in D4.1. Over D4.2, we improved rt-VIL in various ways, in particular to

- **Enable explicitly failing strategies and tactics.** As explained in Section 3.3.1, in particular user-triggered adaptations, but also adaptations triggered by internal components such as the event detection / prediction may fail for several reasons. These reasons range from missing resources during start-up of pipelines over not permissible parameters to a predicted negative impact on the QualiMaster infrastructure. So far, strategies and tactics only failed if the reasoning indicated an inconsistent runtime configuration. In the mean time, the Adaptation Manager can explicit cause such failures and even let following alternatives fail based on the original failure (inspired by exception handling in programming languages).
- **Prevent multiple enactments on the same pipeline (node).** At each regular execution, the constraint analysis may indicate that the same constraint is still failing. Although we prevent such trigger pollution by sending triggers only on changed deviations, an accidentally re-issued trigger can cause re-adaptations on the same pipeline node. In particular, for longer lasting enactments (Section 3.3.1.3) this can lead to problems. As mitigation, we extended the type library of rt-VIL with explicit access whether there is currently an enactment going on for a specific configurable element. This information is mapped into rt-VIL by the QualiMaster infrastructure, and the adaptation scripts can consider this information to handle multiple adequately, e.g., per se it is not clear whether the re-enactment shall just be forbidden for the actual pipeline node (the current default behaviour), for the whole pipeline or for the down-stream data processors.
- **Allow for exploration of alternative tactics / strategies based on an unchanged configuration.** Initially, we considered alternative tactics and strategies only if the validation of the runtime configuration failed, i.e., the latest changes were rolled back through the transactional change history for the runtime configuration. However, alternative tactics / strategies are also relevant, if the highest ranked tactics does not find a way to change the runtime configuration. We experienced and fixed this, when integrating resource re-allocation, disabling data replay and load shedding as adaptive tactics for constraint violations.
- **Extend the rt-VIL library for QualiMaster.** As reported in D4.1, we use a specific library for binding rt-VIL against the QualiMaster infrastructure. This library allows us to express adaptations and enactments in concepts of the QualiMaster approach. Due to the evolution of the QualiMaster infrastructure, this library now includes 48 types with related operations as well as more than 20 supporting operations implemented in the library.

For combining the adaptations, in particular reactive and pro-active adaptation, we aim at applying the dynamic weighing of alternative tactics introduced in D4.1. We plan to take therefore the Configuration of weights for different observables into account (cf. Figure 23) so that the Adaptation Manager can use the high-level settings to influence the importance within the adaptive decisions. Thus, we primarily aim at weighted cost-utility-tradeoffs, which can directly be specified in rt-VIL using the dynamic weighting of tactics.

As an alternative, we plan to express the overarching adaptation control as an optimization problem and try solving it with known approaches such as genetic algorithms [33]. Initial steps towards a formalization of the problem have been taken. We plan to report about the progress and a potential comparison of the approaches in the final Deliverable D4.4.

In summary, the QualiMaster adaptation approach encompasses concepts for all tasks carried out in WP4. Most of the concepts have been implemented and initial evaluations have been carried out. Thereby, most of the requirements from D4.1 namely REQ-A-1 to REQ-A-12 are either realized or in realization. Due to the topological capabilities of the Configuration approach, several capabilities requested for the infrastructure can now even be done on pipeline level, e.g., REQ-A-6

through aggregation algorithm parameters. We will provide a more detailed discussion on the fulfilment of the requirements along with the validation in the final Deliverable D4.4.

In addition to the requirements, also the validation of the functionality and its performance are important. We will discuss recent validation results for configuration and adaptation approach in the next section.

4 Validation

In this section, we provide a validation of the components developed in WP4 and discussed in this deliverable. We start with the individual components (Section 4.1) and turn then in Section 4.2 to the adaptation scenarios introduced D4.2. Section 4.2 details the realization state as well as the validation plans until the end of the project.

4.1 Individual components

In this section, we provide a discussion of the results and, where applicable, an experimental evaluation of the components. We discuss the validation and evaluation of the following components:

- IVML configuration and runtime reasoning (Section 4.1.1)
- Optimization of the IVML model before runtime (Section 4.1.2)
- Input volume prediction (Section 4.1.3)
- Event prediction (Section 4.1.4)
- Algorithm profiling using generated sub-topologies as example (Section 4.1.5)
- Adaptation Control (Section 4.1.6).

4.1.1 Reasoning

To validate the reasoning optimizations discussed in Section 2.2 and to provide more insights on how the process changed the performance, we conducted experiments with reasoning on several artificially generated models at different variable-constraint count ratios and complexity (for the underlying evaluation approach, please refer to D4.2).
Typically, Software Product Line Models are characterized for evaluation in terms of the variable-constraint ratios. In our experiments, we use for artificially created models the following ratios:

- 10 variables and 1 constraint
- 100 variables and 10 constraints
- 1000 variables and 100 constraints

The variables in the artificial models are of type Boolean, Real and Integer. Three levels indicate the complexity of the constraints based on the number of operations performed in the constraint.

- Complexity Level 1: Simple constraints of 2 variables / constants and one Boolean operator.
  Example: `a and b` (for Booleans), `a > b` (for Integers and Reals).

Nr	Number of elements		IVML reasoner v1	IVML reasoner v2
	Variables	Constraints	Reasoning time (milliseconds)	
Complexity Level 1				
1	10	1	1	0
2	100	10	24	24
3	1000	100	305	105
Complexity Level 3				
4	10	1	1	0
5	100	10	11	24
6	1000	100	331	81

Table 6: Reasoning performance of the IVML reasoner v1 and v2.

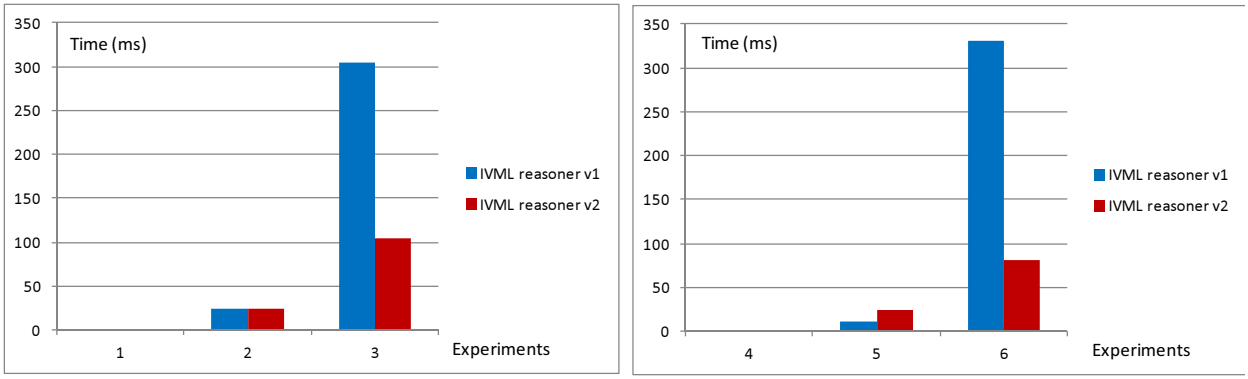

Figure 33: Performance comparison of the IVML reasoner in historical perspective (left plot for Complexity Level 1 and right plot for Complexity Level 3).

- Complexity Level 2: Concatenation of 3 variables / constants by 2 Boolean operators.
  Example: `(a = b) xor (c <= 2).`
- Complexity Level 3: Concatenation of 5 variables / constants by 4 Boolean and arithmetical operators.
  Example: `(a <= 15) xor (b - c >= d)`

In D4.2, we compared the IVML reasoner with previous implementations using frameworks such as Jess or Drools. In this deliverable, we are comparing results of IVML reasoner from D4.2 (IVML reasoner v1) with the current implementation (IVML reasoner v2) using the same artificial models. We conducted the tests with the settings described in Section 2.2.

Test results summarized in Table 6 and Figure 33 show that our optimization efforts described in Section 2.2.1 lead to an increase in reasoning performance and improve the scalability, especially

Reasoner							
Number of elements		**IVML reasoner v1**			**IVML reasoner v2**		
Variables	**Constraints**	**Total (ms)**	**Processing (ms)**	**Post-processing (ms)**	**Total (ms)**	**Processing (ms)**	**Post-processing (ms)**
variables : constraints = 1 : 3							
100	300	40	38	2	38	16	22
300	900	251	246	5	287	45	242
500	1500	614	608	6	727	189	538
1000	3000	2632	2625	7	1420	164	1256
1500	4500	5878	5868	10	2087	175	1912
variables : constraints = 1 : 1							
100	100	25	24	1	23	10	13
300	300	111	109	2	49	27	22
500	500	257	253	4	139	44	95
1000	1000	1014	1007	7	445	84	361
1500	1500	2176	2171	5	710	122	588

Table 7: Detailed performance test.

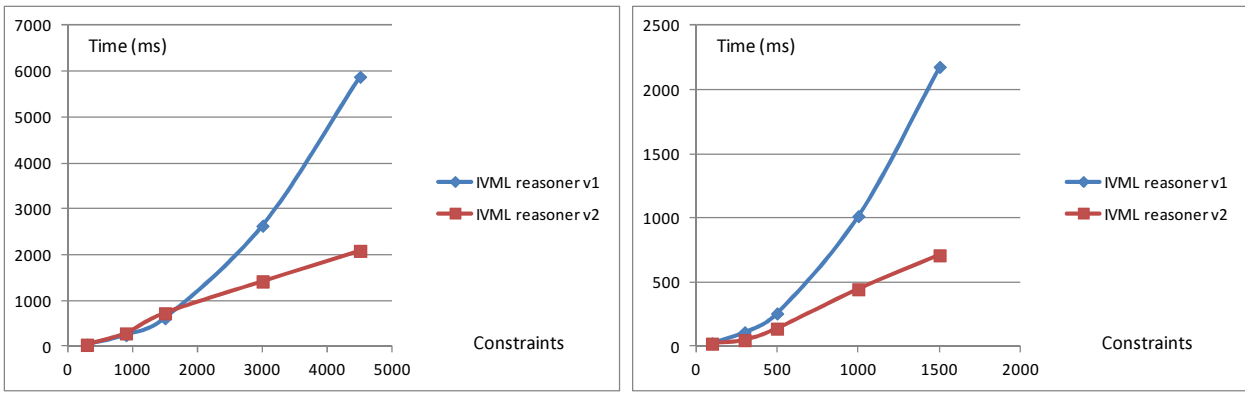

Figure 34: Total time measurement (left plot for a variables-to-constraints ratio 1:3 and right plot for ratio 1:1).

on a growing model in terms of both, variables and constraints.

For a more detailed evaluation and comparison we performed an additional series of experiments on artificially generated models with only Boolean variables, Level 3 constraints and variable-constraint ratio of 1:3 and 1:1. Again, we conducted experiments with the settings described in Section 2.2. We also measured not only the total reasoning time, but specific time for the two reasoning phases introduced in Section 2.2: Processing (PH1) and Post-processing (PH2). Table 7 summarizes the results.
As shown in Table 7 and Figure 34, the current implementation of IVML reasoner (v1) outperforms the implementation (v2) introduced in D4.2, in particular on larger scale models. This is mainly achieved by optimizing the reasoning algorithm and this optimization pays off when we are dealing with complex and large-scale models, as the one used in the QualiMaster project.

However, it is difficult to directly compare artificial models created for certain evaluations with a real-world model such as the QualiMaster Configuration (Meta) model. In the QualiMaster model, most of the variables are parts of compounds, the number of compound instances is conceptually unlimited and increases with the configured algorithms, families and pipelines and also the topological configuration imposes a different kind of complexity. However, we tried to estimate the number of variables if we would flatten it into a typical Software Product Line Model. For a version of the model with 9 pipelines and 40 algorithms, we counted around 4000 individual variables (including runtime variables), which characterizes the QualiMaster Configuration Model as a large scale model [5].

Another interesting observation is the significant decrease in processing time while keeping the reasoning capabilities stable or even increasing them in comparison to previous implementation. This can be seen in Figure 35, where the IVML reasoner v2 performance results appear to be “flat” and scalable in comparison to the IVML reasoner v1.

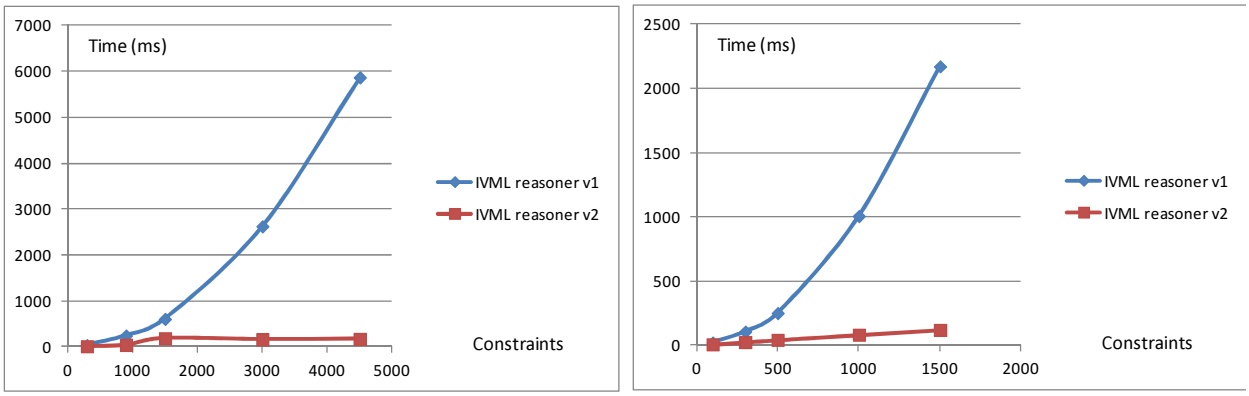

Figure 35: Processing time measurement (left plot for a variables-to-constraints ratio 1:3 and right plot for ratio 1:1).

	Normal	Pruned with inactive pipelines	% to Normal	Pruned without inactive pipelines	% to Normal
Number of variables involved in constraints:	343	190	49	185	48
Number of constraints:	10557	9371	79	8813	74
Number of re-evaluations:	13158	13066	84	11948	77
Time (ms):	149	86	64	84	63

Table 8: Comparison of reasoning performance on normal and pruned models.

However, we got also noticed dramatic increase in post-processing time with the same number of found failed constraints. This happened due to the more complex post-processing and interpretation of failed constraints in comparison to the implementation we had in D4.2. This post-processing is a pre-requisite for indicating configuration errors in detail as shown in Section 2.4.1 or for realizing the constraint analysis for adaptation (Section 3.1).

4.1.2 Optimizing the Configuration for Adaptation

To validate our effort of optimizing the configuration for runtime, we modified the testing settings described in Section 2.2 by increasing the number of evaluation runs to 51. The reason is that at runtime reasoning is executed over and over and, in particular, the first 11 performance readings can be considered as outliers due to technical “warm-up” of the code, i.e., for loading classes and libraries as well as for initializing initial instances. We determined this number by observing multiple executions of the experiment and by identifying the start of the stable phase. Based on this setup, we calculate the average reasoning time based on the following 40 iterations.

As explained in Section 2.2.2, we identified three optimization strategies, namely OS1 (removing unused constraints) and OS2 (removing unused model elements) as well as OS3 (removing unused pipelines).

We compared reasoning results on a QualiMaster model before optimization (Normal), after optimization with OS1 and OS2 (Pruned with inactive pipelines) and after optimization with OS1, OS2 and OS3 (Pruned without inactive pipelines). The results are shown in Table 8. Automated configuration optimization with OS1 and OS2 leads to a reduction of variables by 51%, constraints by 31% and re-evaluations by 16%. This is an overall improvement of the reasoning time by 36% in comparison to normal model. Based on the optimizations, we were able analyze the optimized QualiMaster Configuration in less than 100 ms of reasoning time, thus providing sufficient reasoning capabilities for adaptation in runtime. Manually applying also OS3 (to one unused pipeline in this case) leads to a decrease by only 1% (in absolute numbers by 2 ms), i.e., no significant improvement.

Based on these results, we will integrate the optimization of the Configuration using OS1 and OS2 into the next version of QM-IConf and the continuous integration so that the QualiMaster infrastructure can profit from the optimizations.

4.1.3 Input Volume Prediction

In this section, we report on the evaluation of the input volume prediction method described in Section 3.3.2.1. First, we discuss the data used in our experiments. Second, we report and compare the performances of different prediction models according to different metrics.

Method	MAE	DAC	RAE	MAPE	RMSE
granularity = 1 minute					
LR	13757.82	68.19	81.12	39.72	34977.71
MLP	10547.52	72.15	64.81	42.90	**27742.71**
SVM	**9458.66**	**73.61**	**58.98**	**34.83**	28160.95
granularity = 5 minutes					
LR	47117.97	67.59	84.76	27.34	92242.28
MLP	35338.13	72.29	66.39	24.99	**74518.80**
SVM	**30976.43**	**73.18**	**57.45**	**19.40**	77063.10
granularity = 10 minutes					
LR	88503.11	68.78	83.96	25.46	161355.48
MLP	66754.34	**72.49**	64.97	20.06	**132420.74**
SVM	**61783.55**	72.10	**59.36**	**16.89**	137568.53

Table 9: Performances of the prediction models at different granularities.

We used real-world financial data, consisting of the trade volumes of 30 stocks from the Nasdaq and NYSE stock exchanges over 1 year (May '15 - May '16). Examples of stocks in the dataset are $AAPL, $AMZN, $BAC, $GM, $GOOGL, $NGD. We consider volumes at different time granularities (1 minute, 5 minutes, 10 minutes) for sake of comparison. Working at a granularity of 1 minute, for instance, means that the trade volume of a given stock is observed, aggregated every minute, and the prediction model is used to predict the aggregated volume within the next minute. The training set is made of the first 11 months, while the last one is used for testing. Although we focus on financial time series in this evaluation, the approach can similarly be applied to Twitter data streams.

As mentioned in Section 3.3.2.1, we experimented with 3 different models (after having converted the temporal order into a set of features): Linear Regression, SVMs, and MLPs. For Linear Regression, we set the Ridge parameter to 0.1. For SVMs, we used a linear kernel and the C parameter set to 1.0. For MLPs, we used one hidden layer and squared error as loss function.

The performances of the different prediction models (Linear Regression, MLP, SVM) at different granularities (1, 5, 10 minutes) are reported in Table 9, according to the following evaluation metrics: Mean Absolute Error (MAE), Direction Accuracy (DAC), Relative Absolute Error (RAE), Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE). The absolute values of trade volumes can be order of magnitude different from stock to stock. For instance, the average volume per minute of $BAC is over 200,000, while the one of $GOOGL is 3,755. Therefore, values

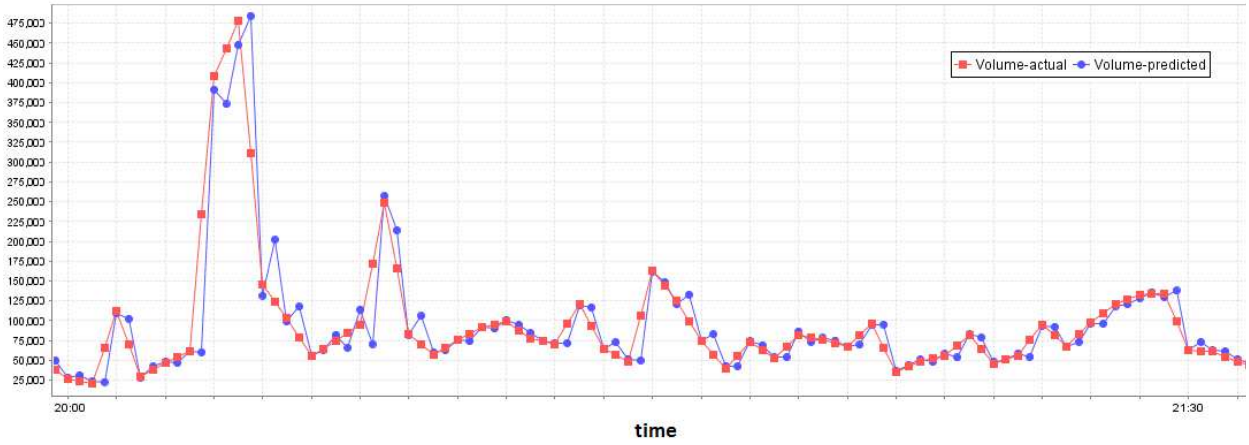

Figure 36: Prediction of $APPLE volume with SVM at 1-minute granularity.

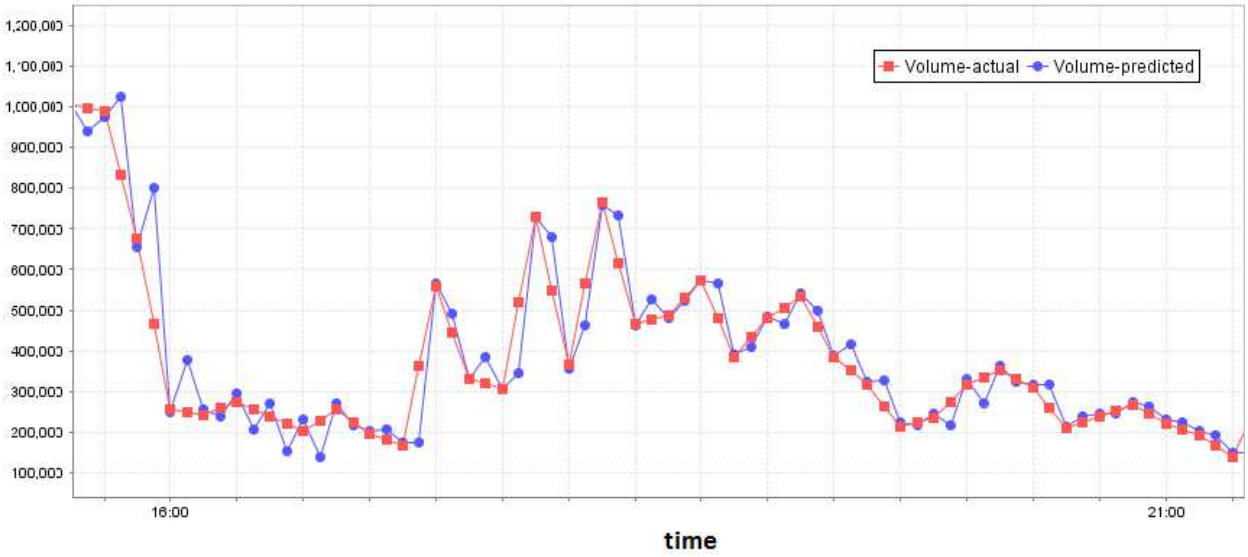

Figure 37: Prediction of $FCX volume with SVM at 5-minutes granularity.

of not normalized metrics, such as MAE and RMSE, are only reported for sake of comparison among the methods at the same granularity. Instead, the other three metrics are normalized by definition (between 0 and 100) and can tell more about the absolute quality of a method on its own.

From the results we can observe that the SVM is overall the most accurate predictor, followed by the MLP, with highest gaps for RAE and MAPE metrics. This holds for all the granularities, although the difference between SVM and MLP gets smaller at a granularity of 10 minutes. Besides the MAPE, which tends to decrease for higher granularities, the performances of the methods remain stable at the different granularities (MAE and RMSE do not count as they are not normalized). This means that the two main effects of changing granularity balance each other with almost no effect on the performance: on one side, higher granularities should make the time series smoothed and less noisy (positive); on the other side, big and sudden peaks might become more difficult to predict due to the fewer data points observable before they occur (negative).

Regarding time, we measured both the time required to train the model and the one for making one single prediction online, using a general purpose laptop with dual-core Intel i5. The former, considering 1 minute granularity (biggest dataset), is around 45 minutes for SVM and MLP, while it is around 5 minutes for Linear Regression. This difference is due to the fact that the Linear Regression is much simpler than the others. In any case, the training time is relatively short and allows the model to be easily retrained with high frequency, even every night. The latter time is, for all the methods, in the order of few milliseconds. This means that the volume prediction could be executed periodically even at low granularities (i.e., 1 minute), without a big impact on the rest of

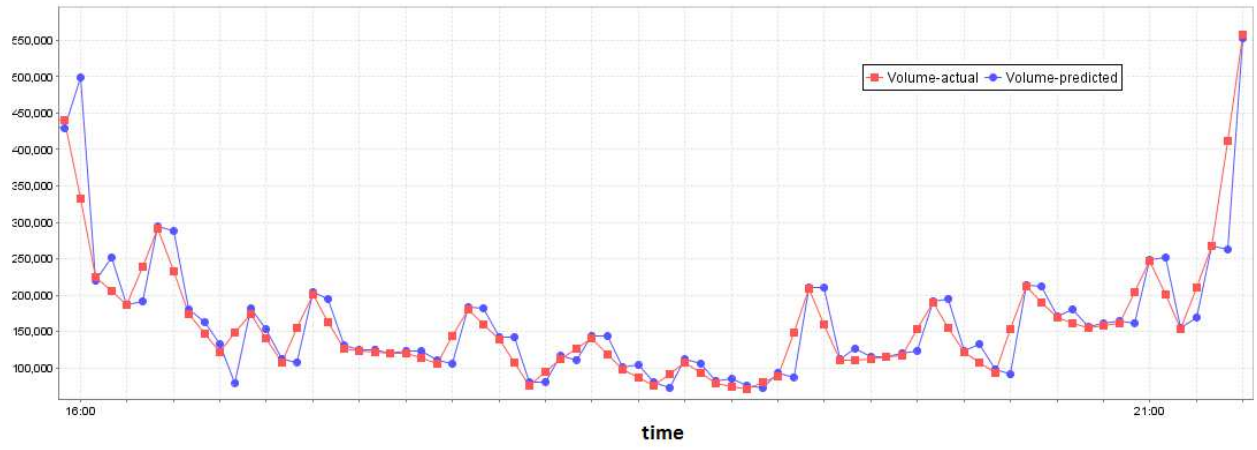

Figure 38: Prediction of $SBUX volume with SVM at 10-minutes granularity.

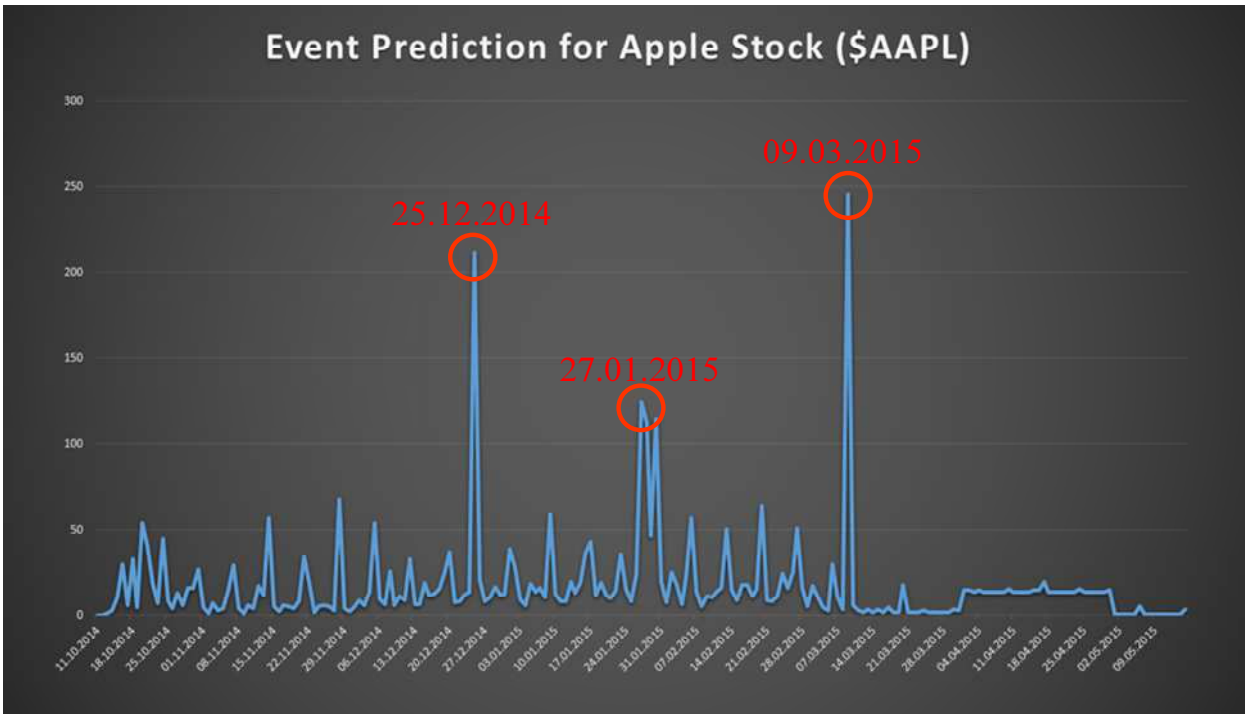

Figure 39: Predicted events for $APPL.

the pipelines or the QualiMaster infrastructure.

Finally, we show some examples to give a more tangible idea of the quality of the prediction, beyond the numerical performances. These examples are reported in Figure 36 ($APPL, 1 minute granularity), Figure 37 ($FCX, 5 minutes granularity), and Figure 38 ($SBUX, 10 minutes granularity). Red lines are the actual volumes, while the predicted ones are plot in blue. We can observe an overall good prediction, especially when the volume exhibits smooth trends. The prediction gets less precise in presence of sudden and big changes of volumes. This is an inherent limitation of the prediction methods when applied to such cases, where the peaks (either positive

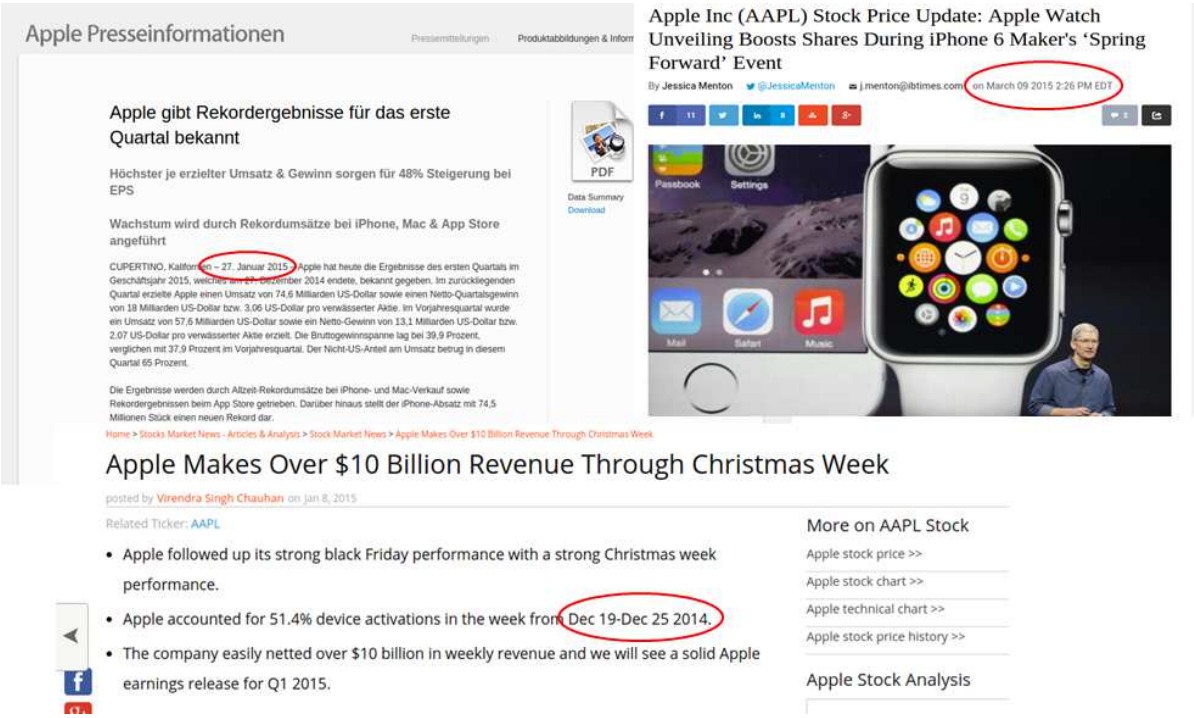

Figure 40: Evidences of the events predicted for $APPL.

or negative) occur in just one or two steps: the prediction model does not have enough intermediate data and therefore tracks the volume with some delay.

4.1.4 Event Prediction

We now discuss the results of the event prediction method described in Section 3.3.2.2, when applied to the stock $APPL between November 2014 and May 2015. The results are visualized in Figure 39: The blue line represents the distribution of future temporal expressions extracted from Tweets within a period before November 2014. The application of the event detection method (WP2) on such time series discovered 3 peaks (i.e. events), which are circled in red. From the figure, it is possible to observe that the farthest time period (starting from middle of May 2015) is characterized by fewer and less precise temporal references. This is probably because people do not talk about facts or rumours in the far future and, when they do, they use broader temporal expressions (e.g., in April 2015), resulting in more uniform values.

We manually checked the validity of the events in Figure 39 by looking for documents in the web containing their evidences. For each event we found at least one web page referring to it, they are shown together in Figure 40. The predicted events are about different domains: sales in the Christmas period, release of quarterly report, promotion and presentation of new Apple products. Two of them are periodic (Christmas, quarterly report), while the “Spring Forward” event is a special event that can change name and date over time.

4.1.5 Algorithm Profiling

In this section, we discuss preliminary results of profiling in terms of raw profiling results (see Section 3.3.2.3) and of experiments and experiences with the Kalman-filter for statistical post-processing.

The technical support for algorithm profiling, i.e., the profiling coordination command, creating a temporary pipeline configuration, instantiating the pipeline, starting it and taking control over the parameter space has been implemented and integrated into the QualiMaster infrastructure in collaboration with WP5.

We validated the functionality of the profiling with the financial pre-processor and the Hayashi-Yoshida correlation estimator developed in WP2. We selected these two algorithms, as they are fundamentally different in their design.

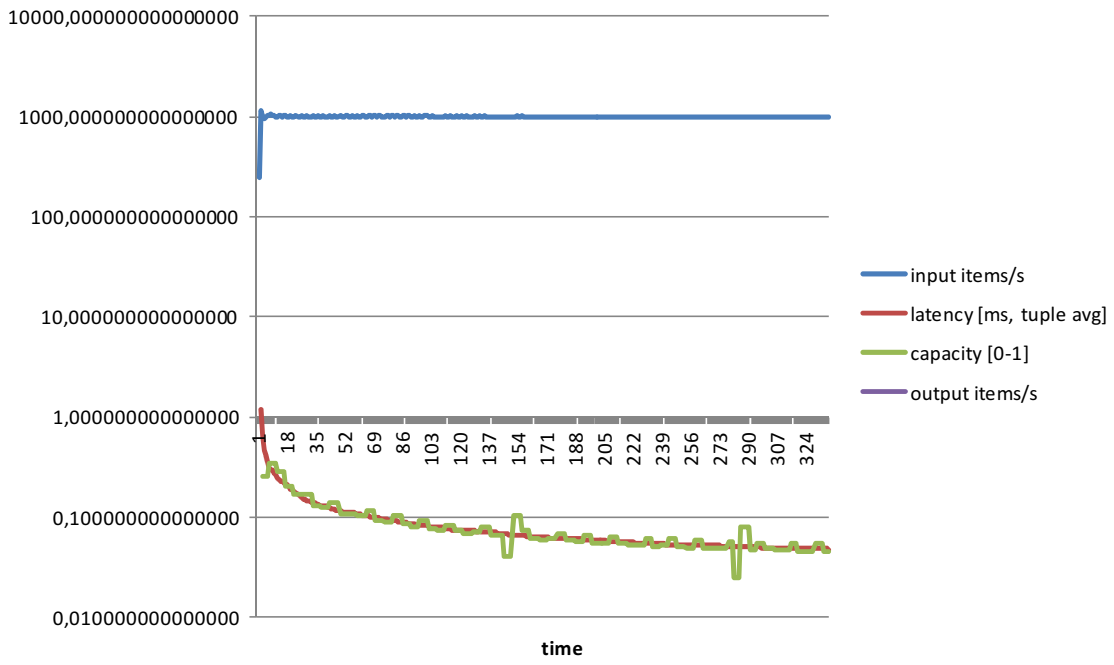

Figure 41: Raw algorithm profile for the financial pre-processor as log-linear graph.

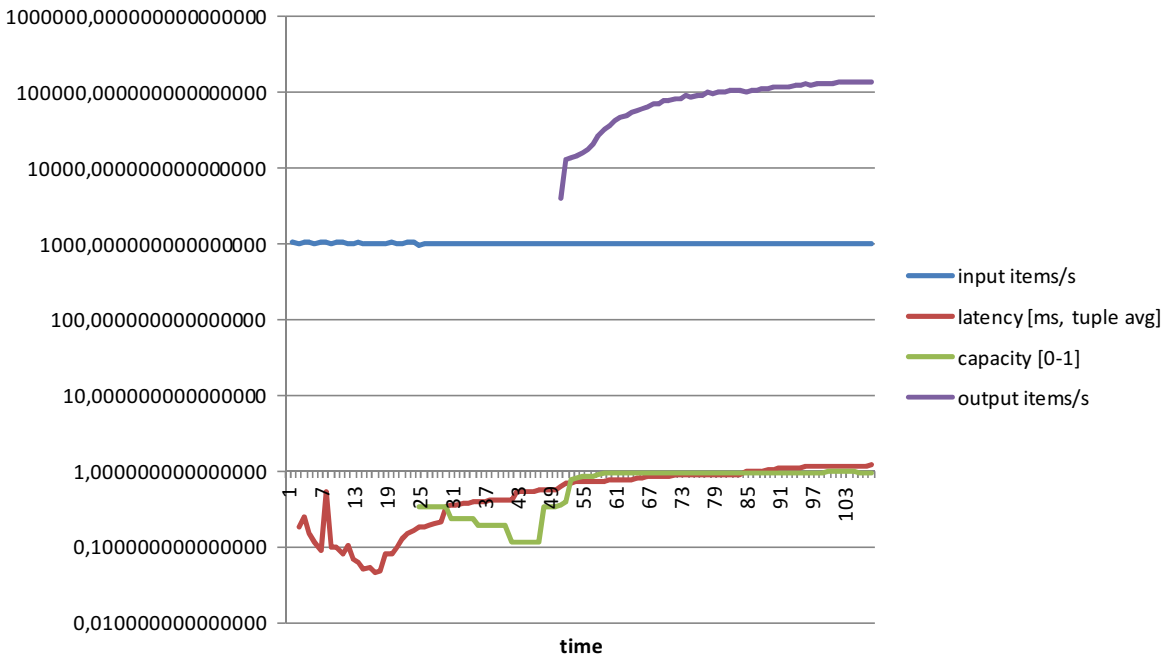

Figure 42: Raw algorithm profile for the manual implementation of the Hayashi-Yoshida correlation estimator as log-linear graph.

The **financial pre-processor** is a linear-time algorithm and typically executed as a single instance on one cluster node. Figure 41 depicts the raw profile for a data set of 1000 items / second running on a single node. As expected, the output rate is the same as the input rate (curves do overlap in VP0). Output and input rate can be considered better the higher the values are. Latency as well as capacity (both are better the lower the values are) become stable after a certain time due to the default behaviour of Storm directly connecting all the sources of a pipeline although not all nodes may be ready for processing.

The **Hayashi-Yoshida correlation estimator** is a distributed algorithm consisting of two

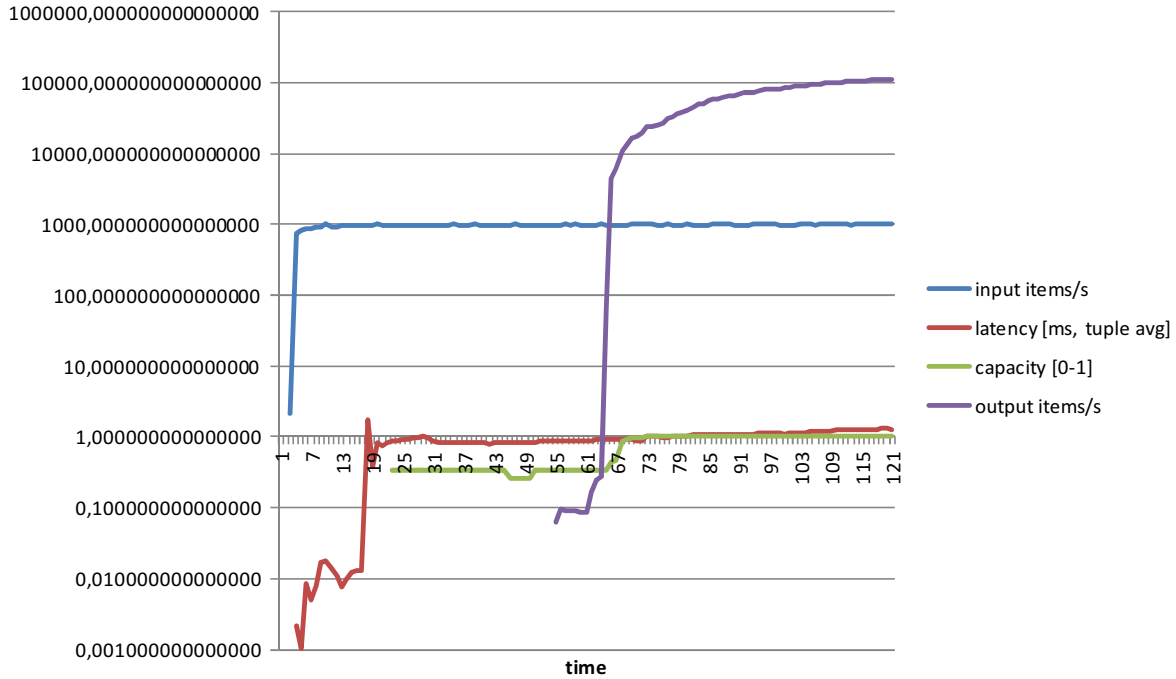

Figure 43: Raw algorithm profile for the generated implementation of the Hayashi-Yoshida correlation estimator as log-linear graph.

components connected by three different types of data streams. During a start-up period (roughly two analysis windows), the algorithm is collecting inputs and distributing the received items based on an identifier (here the identifier for market players) to its worker processing elements. After that time, the worker processing elements are emitting matrix updates at a rather high output rate.

Figure 42 and Figure 43 illustrate two raw profiles running the algorithm on 5 physical cluster nodes with 13 worker processing elements. Figure 42 shows the profile of the manual implementation and Figure 43 for the generated sub-pipeline (see Section 2.1.1 and Section 2.2). For both implementations, we performed an intensive comparison of the output produced and can confirm that they are identical. The profiles show the gap at the beginning, a stable input rate of 1000 tuples per second as well as similar throughput behaviour in terms of output items per second. However, the implementation seem to differ slightly in execution time (latency) and capacity behaviour, but this may be also be caused by differences in the processing / distribution, as also start-up artifacts in Figure 43 are recorded.

The performed experiments indicate that the (raw) profiling approach is working and allows detecting measurable differences among algorithms. However, the start-up behaviour of Storm topologies may influence at least the beginning of the profiling, which may be subject to exclusion for the statistical post-processing. We also plan to apply the explicit connection mechanism provided by the Data Management Layer, i.e., to defer connecting the physical data source until the point in time where all cluster nodes in a topology are properly initialized (as detected by the Monitoring Layer). We further plan applying the profiling to the hardware-based algorithms developed in WP3, which was not possible for this deliverable due to technical reasons.
It is also worth mentioning that the approach of automatically creating profiling pipelines seems already to pay off. Instead of configuring explicit test pipelines as we did at the beginning of the project, we are now able to derive and run a profiling pipeline based on the most recent algorithms just by a single infrastructure command. This already simplified collecting the shown profiles as well as testing the approach.

For the statistical post-processing, we apply the RainMon implementation of the Kalman-Filter, which is available as source code. In more details, we removed the phases that we identified as not relevant and execute just the relevant code in this experiment. It is important to note that RainMon is implemented in Python and that we measured the execution times from Java code calling the Python implementation, i.e., a realistic setting for the QualiMaster infrastructure.

After configuring the Kalman-Filter as described in Section 3.2.2.1, we used the raw profiles obtained from profiling the Hayashi-Yoshida correlation estimator. We aim at evaluating the performance and the predictive accuracy of the actual implementation. All tests were performed on a Notebook with Intel Core i7-4600U, 16GB DDR3-1600 MHz, Windows 7 Professional 64bit, JDK 1.7.0_70 and Python 2.7.10 (the Kalman-Filter in RainMon is implemented in Python) the following configuration and no other non-System processes running at the time of the tests:

For evaluating the performance, we evaluate two scenarios, training and prediction. The *training* scenario includes the time from the start of the execution of the Python-command by Java to the point where the Python-process was finished and its output is available to the Java. This contains generating the default matrices and loading the results of the Hayashi-Yoshida correlation from the system drive (i.e. from a Log-File). The *prediction* scenario includes the time needed for the

	Total time in ms (average time per data point)				
Number of data points	**1**	**2**	**3**	**…**	**332**
Total execution time	466	534 (267)	554 (164)	…	4126 (12,4)
Prediction-Correction	10	20 (10)	33 (11)	…	3895 (11,7)

Table 10: Execution times for the Kalman-Filter in RainMon (Python) executed from Java, average times per data point are given in parentheses.

	Total time in ms (average time per data point)				
Number of data points	**1**	**2**	**3**	**…**	**332**
Total execution Time	50	55 (22,5)	60 (30)	…	110 (0,33)
Prediction-Correction	<1	<1	<1	...	7 (0,02)

Table 11: Execution times for the re-implemented Kalman-Filter in plain Java. Average times per data point are given in parentheses.

prediction-correction cycles themselves by also calling the Python implementation from Java. Each test was done 100 times on the same input and the table shows the average result.

Table 10 summarizes our measurements for the RainMon-based Python implementation. The Total execution time (upper row in Table 10) measures a call from Java to the Python implementation for the given number of data points (including creation and destruction of the external Python environment). The prediction-correction cycle for each data point (lower row in Table 10) refers to the time for performing the Kalman calculations on an actual value and deriving the prediction for the next given number of data points, i.e., a combined training-prediction step. The results show that the maximum time for executing the full Kalman process on a single data point is 466 ms for a single data point and decreases to an average of 10-15 ms when a larger number of data points is being processed. While the processing time for one data point is fast enough for our purpose as it is less than 1 second, performance improvements are required to avoid bottlenecks in the adaptive decision making over multiple potential profiles, e.g., for different algorithms. In particular, the overhead of the overall call compared to the time for the prediction-correction cycle deserves optimization, e.g., by avoiding Python code and re-implementing the code in Java. Aside of performance reasons, this would allow for a better integration with the QualiMaster infrastructure, reduces the installation dependencies (no Python needed) and enables us to realize extensions such as gap handling if algorithms are disabled more easily.

As an experiment, we re-implemented the roughly 300 lines of Python code for the extended Kalman-Filter used in RainMon in plain Java. Relying on the Java version of the Kalman-Filter

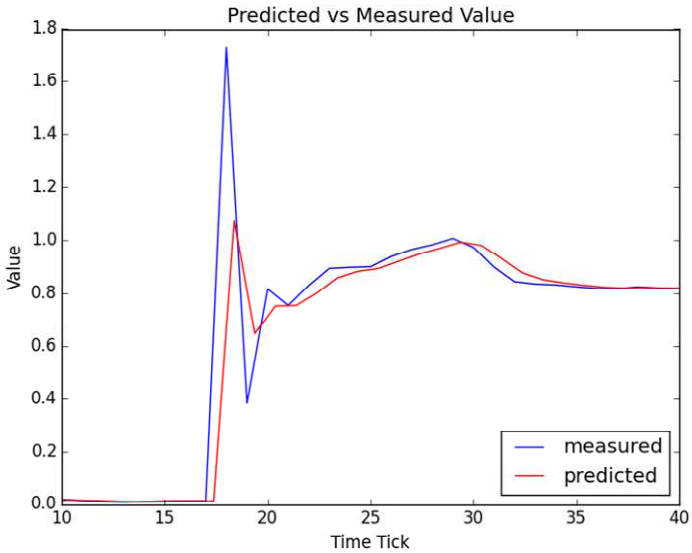

Figure 44: Comparison of measured and predicted values for execution time / latency in comparison with the raw profile values for the Hayashi-Yoshida correlation.

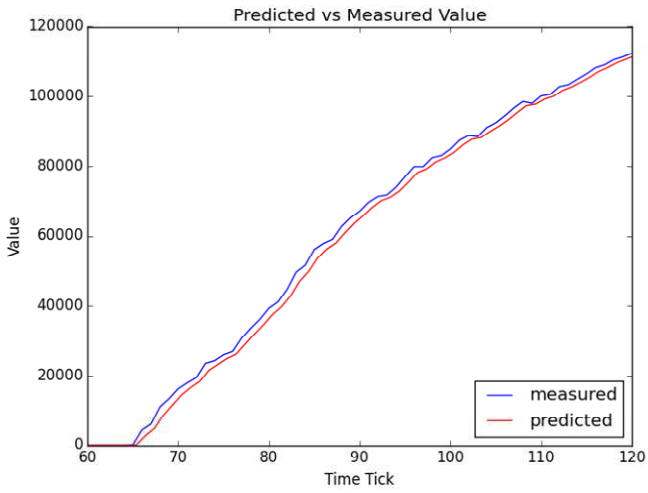

Figure 45: Comparison of measured and predicted values using the output rate in comparison to the raw profiles of the Hayashi-Yoshida correlation.

provided by the Apache Common Math library[7], we achieved this in terms of 60 lines of Java code. The performance readings in Table 11 using the same input data and compute environment described above shows the execution times for the Java version of the Extended Kalman-Filter. The execution time for a single data point is now roughly more than 9 times faster. In our experiments, the results of the Java-based implementation are quite similar to the Python-based implementation - the difference of the relative error is less than 0.8%.

Regarding prediction accuracy, we compare now the readings from the raw profile with the predictions returned by the Kalman-Filter. Figure 45 visualizes the prediction for the output rate and Figure 44 for the execution time / latency for the manual implementation of the Hayashi-Yoshida correlation indicator. Blue lines indicate the measured value, read lines the predicted values. The better the two curves overlap, the more accurate are the predicted values. It is important to note that we intentionally aim at underestimating the predictions so that an adaptation depending on the results does not react too hectically.

Both, Figure 45 and Figure 44 indicate a good prediction quality, even if we train the Kalman-Filter with just some minutes of monitoring data (2 minutes in our case). If pre-calculated, predictions can be returned with no learning phase needed at all. Even in the presence of spikes or abrupt value changes, the good prediction quality can be achieved after some iterations as shown in Figure 44. However, we aim at increasing the prediction capabilities for spikes or by considering context information when interpreting the predicted values for making adaptive decisions.

4.1.6 Adaptation Scripts

In this section, we provide an initial evaluation of the overall execution speed of the rt-VIL adaptation script for QualiMaster. We evaluate the script in terms of jUnit tests that we utilize also continuously testing against the most recent version of the QualiMaster Configuration Model. The tests cover several cases of situations where the Adaptation Layer receives a request for adaptation and decides about changes. For all test cases, this includes all phases of the adaptation cycle and can target a specific adaptation trigger or a sequence of adaptation triggers over multiple points in time. The test cases are:

- **Check before start-up:** Initial realization of the checking phase of the extended pipeline lifecycle (see Section 3.3.3). This adaptation is triggered by a lifecycle event and does not require constraint-based analysis.

[7] http://commons.apache.org/proper/commons-math/userguide/filter.html

- **Start-up:** Initialize the algorithms to use as well as the adaptable parameters. This adaptation is triggered by a lifecycle event and does not require constraint-based analysis.
- **Parameter change:** Simulates a parameter change request caused an internal component. User-triggered adaptations work rather similar. This adaptation is trigged by the causing component and does not require constraint-based analysis.
- **SW/HW algorithm switch:** The first part of the test causes a condition for switching from the software version of the Hayashi-Yoshida correlation to the hardware version based on a constraint as discussed in Section 3.3.1.1. The second part changes the environment so that a switch back from hardware to software is needed. The respective constraint violations are detected by the constraint-based analysis.
- **Resource re-allocation:** The resource constraint discussed in Section 3.3.1.1 indicates that a re-allocation is required, which is detected and signalled to the Adaptation Layer.
- **Disable data replay:** If a pipeline is under high load, disabling active Data Replay sinks is the next option, i.e., the current system state forces that resource re-allocation does not work (fall-through).
- **Enable load shedding:** If a pipeline is under high load and neither resource re-allocation nor disabling Data Replay helps, load shedding is enabled and gradually increased if the situation persists. If load decreases, also the load shedding impact is decreased and, finally, load shedding is disabled.

Due to technical reasons we cannot run the test scripts on a real cluster. Therefore, we replay the monitoring by a part of the test case enforcing a certain system state required for the respective test situation. If required by the test case at hands, the constraint-based analysis is executed determining the need for adaptation. Then, the Adaptation Layer maps the monitoring state into the runtime configuration, executes the selected strategies and tactics including reasoning and, finally, turns changed runtime variables into Coordination Layer commands as part of the enactment. As we do not run the test scripts against a real cluster, we assert the Coordination Commands without really executing them (they are tested by further tests).

For testing the adaptation scenarios defined in D4.1, we will first develop related test cases and perform more encompassing tests. Then, we will evaluate the test cases against simulated / real data on one of the clusters available to the project.

Table 12 summarizes the execution results for a Linux virtual machine with 2 CPUs at 2,2GHz and 4GB RAM for different adaptation test cases on the full configuration model, i.e., without the optimizations discussed in Section 2.2. The first column indicates the test case, the second the phase of the test case if there are multiple phases of the test as described above. The *constraint-based analysis*, which utilizes the IVML reasoner runs in the two shown cases below 600 ms, i.e., faster than the 1 second regular analysis schedule. These performance readings can be improved by optimizing the Configuration Model for runtime as discussed in Sections 2.2.2 and 4.1.2.

The typical execution time of the *full adaptation script* is below 400 ms. The execution time for the start-up test case, where the script determines the initial algorithms and parameters for a starting pipeline is higher, because more configurable elements have to be processed and more decisions have to be made / enacted. The particular execution time of more than one second is not critical, as the full start-up of a pipeline can take up to several minutes depending on the time needed to distribute the implementation of the pipeline over the cluster, to load hardware algorithms, etc. Also the full execution of the parameter change is rather high, as the adaptation script first validates the existence of the pipeline element to change. This is done by a search through the pipeline topology and can be improved.

Determining the *commands to be executed* (enactment column in Table 12) is rather fast, i.e., less than 70 ms. *Binding the monitoring state* can take in some cases nearly the same time for the execution of the strategies. As described in D4.2, we rely so far on explicitly binding all monitored values to runtime variables. This explicit mapping does not require domain-specific code, but runs over all configurable elements and pipelines. As also discussed in D4.2, we identified a realization

Test case	Phase	Analysis [ms]	Bind Monitoring State [ms]	Run Strategies [ms]	Enactment (determine commands) [ms]	Full Script [ms]
Check before Start-up	-	-	96	8	7	113
Start-up	-	-	263	869	63	1219
Parameter change	-	-	79	645	33	760
SW/HW algorithm switch	SW→HW	231	128	166	62	359
	HW→SW	202	85	113	42	242
Resource re-allocation	-	560	128	220	34	385
Disable Data Replay	-	249	73	161	20	256
Load shedding	start	239	75	122	24	224
	increase	189	70	154	13	240
	decrease	168	67	167	13	249
	stop	163	65	137	13	217

Table 12: Execution times of the rt-VIL adaptation script for different test cases.

alternative, which maps only required values into IVML runtime variables. However, this requires an extension of the IVML configuration model. We aim at implementing and validating this realization alternative until the end of the project, believing that this can reduce the binding times significantly. *Running the strategies* and tactics depends on the complexity of the situation, the decisions to be taken and the alternatives to be explored. So far, the execution time in this category fulfils our expectations, but the integration of the proactive and reflective approaches discussed in Sections 3.3.3 and 3.3.4 will probably increase the execution time.

It is worth mentioning that the individual times summarized in Table 12 are measured through a generic listener on the rt-VIL execution, which has been implemented for these tests. For reactive adaptation, we can realize a similar listener, which can provide measurements on the adaptation but also further internal insights if required.

In summary, the execution times of the adaptation and the analysis phases of the MAPE-K cycle implemented in the QualiMaster infrastructure fulfils the aims of D4.1 and, in particular, of operating within the financial time tick used in WP2/WP5 for algorithm evaluations. For the components to be integrated, i.e., the proactive / reflective adaptation components, the optimized mapping of monitored observations as well as potential runtime improvements of the existing adaptation strategies and tactics, we expect that determining adaptive decisions will operate at a similar time scale.

4.2 Adaptation Scenarios

In Deliverable D4.2, we introduced five basic adaptation scenarios, which shall be covered by the QualiMaster infrastructure and used for evaluation. In D4.2, we detailed the scenarios and

prioritized them, mentioning that we aim at some more variants of the basic scenarios. In this section, we discuss the realization state of the scenarios and the underlying mechanisms as well as the planned scenario variants to be realized and evaluated until the end of the project.

Testing and evaluating adaptation scenarios is a complex task, as for executing the scenario the related pipelines must be in a specific execution state, i.e., the input data sets must be prepared to cause a certain adaptation or a sequence of (follow-up) adaptations. Moreover, due to interrelated adaptation strategies and tactics, also effects planned for other scenarios may be caused, e.g., changing data streams (A-1) can be detected through the event detection (A-2) and end up with results planned for A-2. As far as possible, we aim at separating the scenarios and their effects during the evaluation, but take into account that cross-over effects can and shall take place.

A prerequisite for evaluating adaptation scenarios is that the underlying mechanisms are working. From an engineering point of view, the first step in validation / evaluation is to create test cases for the various scenario / strategy / enactment combinations and to test the individual parts. As already discussed in Section 4.1.6, the QualiMaster adaptation test cases execute the full adaptation cycle (due to technical reasons with simulated monitoring and without real enactment) over a single adaptation or a sequence of adaptations. Enactment tests (as discussed in D5.3) ensure that affecting the data processing and the enactment in the Execution Systems works properly. Moreover, validation of the algorithm profiling includes a validation of monitoring the QualiMaster infrastructure and the running pipelines (Section 4.1.5). Based on the individual tests, adaptation scenarios act then as a form of integration test, first as an encompassing test case akin to Section 4.1.6 and then on a real cluster with specifically prepared data sets that are replayed for the purpose of validation / evaluation. The monitoring and adaptation logs used for reflective adaptation can record and indicate the effects of the adaptation.

In this section, we list the planned scenarios in the sequence of priority from D4.2 and the respective validation measures taken so far. As detailed below, we end up with 11 adaptation scenario variants (more than the 7 variants envisioned in D4.2), including some with similarities in the underlying data or the execution mechanisms, which differ in the causes or the adaptive decision making (e.g., by considering algorithm profiles or source volume prediction). In addition to the scenario variants, we outline the planned validation for the reflective adaptation.

4.2.1 Changing Data Streams (A-1)

A typical situation in data stream processing is that the characteristics of the data streams change over time due to external reasons, e.g., the volume increases or the volatility decreases. This requires actions in particular to avoid overload conditions of the infrastructure or negative effects on other pipelines in cross-pipeline settings. As we plan to integrate the Source Volume prediction as an infrastructure component via the event path of the Social Media event detection, we plan a specific adaptation scenario variant as part of detailing scenario A-2.

For scenario A-1, We aim at two scenario variants, namely:

- **A-1.1: Monotonic input volume changes.** Starting with low-impact mechanisms such as parameter change, further adaptation opportunities shall be taken into account if the input volume change persists. Algorithm changes shall be determined by Algorithm Profiles, or, as fallback through constraints. As the stakeholder is affected, disabling data replay shall happen as one of the last alternatives. Ultimately, load shedding shall start and increase / decrease intensity over time depending on the causing situation. In contrast, changing the monitoring frequency in times of low load for a certain pipeline may happen.
- **A-1.2: Sudden change in volume / volatility in terms of peaks.** Depending on the impact of the peak and the monitored deviation, enactments of higher impact shall be chosen more quickly, e.g., directly switching to hardware rather than first changing parameters. Again, algorithm changes shall be determined by Algorithm Profiles or, as fallback through constraints. Ultimately, if not mitigated, load shedding shall protect the infrastructure.

On the one side, we aim at reactive adaptation for disabling data replay and load shedding. On the other side, we apply proactive adaptation for parameter change, algorithm change and resource re-allocation via Algorithm Profiles (or reactive adaptation as fallback).

The primary subject for the evaluations is the “Priority Pipeline” extended by the Data Replay mechanism (cf. D6.3). So far, regression tests for individual events such as parameter changes and for event sequences such as increasing / decreasing load shedding have been created and are being executed successfully as part of the continuous integration. A full regression scenario involving all related strategies aiming at validating the interaction of the mechanisms (based on simulating the monitoring) is in preparation. Further, initial evaluations on the cluster for switching algorithms and performing parameter changes have been conducted (as shown during the last review). As the algorithm profile mechanism is currently in development, the validation so far was done based on reactive adaptation. Validation for profile-based decisions using regression tests will probably need a mechanism for injecting fixed profiles the tests can rely on, i.e., learning-based effects will need evaluations on clusters.

4.2.2 Requested Resource Re-allocation (A-5)

Running multiple pipelines as well as performing administrative tasks on the QualiMaster infrastructure can require a change in the resource allocation. Actually, this may imply a (re)configuration of the resource pool. We focus here on scenarios for starting and stopping pipelines (in particular, in a cross-pipeline setting) as well as explicit administrative triggers for enabling and disabling resources. The following scenario variants are scheduled:

- **A-5.1: Pipeline start.** Starting a pipeline start in single and cross-pipeline settings applying basic algorithm and parameter assignment and, if required, the adaptations described in Section 3.3.3 for scaling down already running pipelines.
- **A-5.2: Pipeline stop.** Stopping a pipeline in a cross-pipeline setting, if required applying the adaptations described in Section 3.3.3 for scaling up other pipelines.
- **A-5.3: Administrative trigger.** Enabling and disabling compute resources on the cluster through administrative events sent by QM-IConf. This shall lead to a migration of the processing elements to the remaining resources or to new resources, respectively.

The primary subjects for the evaluations are the “Priority Pipeline” with Data Replay and the “Focus Pipeline” (cf. D6.3). The basic adaptation mechanisms for enabling and validating A-5.1 - A-5.3 are realized and tested. Sequence-based regression test cases for the scenarios are in development. Overall adaptation scenario test cases need to be created. Also the cluster-based validation needs to be conducted. For A5.1 - A5.2 no specific evaluation data sets are required.

4.2.3 Detection and Prediction of Social Web Events (A-2)

The future event detection and prediction algorithms are used to find events and related dates that may happen in the future. By analyzing the stream of messages as well as news sources that mention certain dates together with related entities and keywords, needs for (future) adaptation can be identified. The prediction is already used to inform the stakeholders as well as related pipeline processing elements, e.g., data sources to ingest more or less data. A particular task is to integrate the Source Volume prediction into these scenarios. We aim here at scenario variants for

- **A-2.1: Short term prediction.** The scenario covers a short term event prediction, which leads to a reactive adaptation. We aim at validating the adaption via the source volume prediction, i.e., change in data sources shall be checked and potentially rejected.
- **A-2.2: Long-term event prediction.** Here the long-term event prediction shall causing a scheduled adaptation, e.g., to provide more resources for a predicted duration.
- **A-2.3: Volume or volatility peaks.** Sudden change in volume/volatility in terms of peaks identified by the source volume prediction into account (as a variant of A-1.2).

As mentioned above, initial cluster-based validation for A-2.1 (as shown during the last review) has been performed. Further, regression tests for the underlying mechanisms including internal

parameter changes have been created and are successfully executed during continuous integration. Regression tests for the listed scenarios as well as detailed cluster-based validation are on the schedule for the remainder of the project, in particular, as soon as the source volume prediction is integrated. The primary subject for the evaluations of the variants of A-2 is (an extended version of) the “Focus Pipeline” (cf. D6.3).

4.2.4 User-triggered Adaptations (A-3)

In this scenario, the user explicitly requests a change in processing by modifying some settings in the stakeholder application. To some degree, it must not necessarily be obvious to the application user that such a modification may cause a change of the processing. The primary goal is to change the processing in a data source, algorithm or data sink parameter, which may lead to subsequent adaptations. We aim at the following scenario variants:

- **A-3.1: Data source parameter.** Change a data source parameter, e.g., the Twitter stream selection / filtering or the subscription to financial data sources. We aim at making adaptive decisions about user-triggers through the Source Volume Prediction and the Algorithm Profiles, e.g., leading to a rejection of the user-trigger.
- **A-3.2: Algorithm parameter.** Change an algorithm parameter, e.g., the window size of the correlation computation for the actual active algorithm. Also here, the Algorithm Profiles shall determine whether the requested variant is probably beneficial and also reject user-triggers if required.
- **A-3.3: Change the analysis focus.** Change the focus of the analysis, e.g., in terms of modified keywords. We aim at making adaptive decisions in this scenario-variant as well as rejection of user-triggers through the Source Volume Prediction.

The QualiMaster infrastructure supports receiving, validating and executing user-triggered adaptations for data sources, algorithms and data sinks. Regression tests for single events have been created and are executed as part of the continuous integration. Cluster-based scenario validations have been carried out and were successful (as shown during the last review), but detailed effects must still be evaluated. In particular, we aim at creating regression test scenarios for the mentioned scenario-variants after integrating the Source Volume Prediction and the Algorithm Profiles.

4.2.5 Processing Errors (A-4)

This scenario focuses on sustaining the processing when compute resources die unexpectedly, e.g., by switching to default values. Currently, we concentrate on working pipelines without exceptional cases due to processing errors. If all other scenarios are successfully validated, we will specify a refined scenario and validated the adaptation to handle processing errors.

4.2.6 Reflective Adaptation

We plan to execute the reflective adaptation along with each cluster-based adaptation scenario and to evaluate the functionality of the reflective adaptation, i.e., whether adaptations and their effects have been detected correctly. Based on the functional evaluation, we plan to analyze the precision and recall of the learning mechanisms used for reflective adaptation as well as the quality of the suggestions made by the implementing component.

5 Conclusions

Quality-aware configuration and adaptation are core topics of the QualiMaster project. An integrated approach to configuration and adaptation enables flexibility and consistency before runtime (configuration) and at runtime (adaptation). Moreover, instantiating a detailed configuration as done in QualiMaster even allows for deriving the implementation of deployable code for adaptive pipelines in a consistent manner. In this deliverable, we provided an update on the concepts and the realization state of the components for configuration and adaptation.

Regarding configuration, we discussed extensions of the Configuration Meta Model driven by feedback and need for further adaptation capabilities, in particular sub-topologies, permissible parameters, configuration of data replay, load shedding and re-configurable monitoring. Moreover, we provided a brief overview on the changes realized to instantiate the extended configuration, discussed the realization state of the reasoner, automated optimizations of the Configuration to increase reasoning and adaptation performance and related changes to the configuration tool support in terms of QM-IConf. In its recent version, QM-IConf realizes nearly all requirements imposed by WP1. Moreover, evaluation feedback from WP6 allowed us to increase the quality assurance for the tool, which supports the partners in designing and configuring their own pipelines

Regarding adaptation, we analyzed and realized enactment patterns for the QualiMaster infrastructure and implemented the full adaptation cycle, in particular for reactive constraint-based adaptation. Improving the adaptation capabilities is the current focus of WP4, in particular through proactive adaptation components such as event prediction and online / offline learning components such as the source volume prediction and algorithm profiling. We also introduced mechanisms for cross-pipeline adaptation and outlined the plans for the reflective adaptation.

Most of the existing components have been integrated into the QualiMaster infrastructure. All existing components have been evaluated, in particular regarding their functionality, correctness and performance. The reasoning support operating at configuration time and runtime is able to analyze an even more complex configuration model in significantly less time (in average less than 200 ms). This can be improved through automated model transformations optimizing the configuration model for runtime (speedup of more than 30%). The data source volume prediction provides good and fast estimations (<15 ms) of the upcoming data volume at low learning / training effort and the event prediction detects events that are relevant for QualiMaster pipelines. The first version of the algorithm profiling process is realized and was applied to obtain raw profiles for algorithms used in the QualiMaster pipelines, including the generated Hayashi-Yoshida algorithm. The statistical post-processing based on an extended Kalman-Filter leads to a good estimations, enables fast (online) learning as well as very quick prediction results (<15 ms) so that predictions for alternative algorithms can be obtained in reasonable time. All the components fit well into the overall decision making process, which currently operates for runtime decisions in less than 550 ms, i.e., within the stock market data tick used in WP2/WP3 for algorithm performance evaluations. Integrating the new components may increase this time, which we plan to balance by optimizing the mapping of monitoring information and by the aforementioned model transformations. For most of the adaptation scenarios, basic technical tests are available in terms of unit tests using the actual configuration model. These tests are being executed as part of the continuous integration. For some scenarios, initial evaluations on the cluster have been conducted.

Until the end of the project, WP4 sees the following final activities:

- Completing the design of safe algorithm switch / the switch toolbox and integrating it into the infrastructure instantiation process.
- Integrating, validating and improving the source volume prediction and the algorithm profiles including profiling of the relevant algorithms.
- Realizing and integrating the reflective adaptation.
- Validating of adaptation scenarios in terms of tests and cluster-based experiments.
- Summarizing the validation and evaluation results, the lessons learned as well as the documentation of WP4 components in D4.4.

6 References

[1] I. Ahmad and A. Ghafoor. Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Transactions on Software Engineering*, 17(10):987–1004, 1991.

[2] H. C. M. Andrade, B. Gedik, and D. S. Turaga. *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press, 1st edition, 2014.

[3] C. Balkesen, N. Tatbul, and T. M. Özsu. Adaptive Input Admission and Management for Parallel Stream Processing. In *Intl. Conf. on Distributed Event-based Systems (DEBS '13)*, pages 15–26, 2013.

[4] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[5] T. Berger, R. Rublack, D. Nair, J. Atlee, M. Becker, K. Czarnecki, and A. Wasowski. A Survey of Variability Modeling in Industrial Practice. In *Variability Modelling of Software-intensive Systems (VaMoS'13)*, pages 1–8, 2013.

[6] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers*, 39(10):1232–1250, Oct 1990.

[7] A. Brito. Optimistic parallelization support for event stream processing systems. In *Proceedings of the Middleware Doctoral Symposium (MDS '08)*, pages 7–12. ACM, 2008.

[8] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer Science & Business Media, 2006.

[9] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. 2009.

[10] Joong H. C. and Hye-Chung (M.) K. Frequency-based load shedding over a data stream of tuples. *Information Sciences*, 179(21):3733 – 3744, 2009.

[11] S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer, 1st edition edition, 2009.

[12] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[13] H. Eichelberger. It's about the mix: Integration of compile and runtime variability. In *FAS*W Workshop on Dynamic Software Product Lines (DSPL '16)*, 2016. (to appear, PDF available on request).

[14] H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid. EASy-Producer: Product Line Development for Variant-rich Ecosystems. In *Software Product Line Conference (SPLC '14) Vol 2*, pages 133–137, 2014.

[15] H. Eichelberger, C. Qin, R. Sizonenko, and K. Schmid. Using IVML to Model the Topology of Big Data Processing Pipelines. In *International Software Product Lines Conference (SPLC '16)*, 2016. (to appear, PDF available on request).

[16] H. Eichelberger and K. Schmid. Mapping the design-space of textual variability modeling languages: A refined analysis. *J Softw. Tools Techn. Transf.*, 17(5):1–26, 2014.

[17] H. Eichelberger and K. Schmid. IVML: A DSL for Configuration in Variability-rich Software Ecosystems. In *Software Product Lines Conference (SPLC '15)*, pages 365–369, 2015.

[18] C. L. Forgy. Expert systems. chapter RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, pages 324–341. 1990.

[19] IBM. An Architectural Blueprint for Autonomic Computing. Technical report, June 2006. http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[20] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[21] S. Keele. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. 2007.

[22] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.

[23] B. A. Kitchenham. Systematic review in software engineering: Where we are and where we should be going. In *Proceedings of the International Workshop on Evidential Assessment of Software Technologies (EAST '12)*, pages 1–2, 2012.

[24] D. Le Phuoc, M. Dao-Tran, A. Le Tuan, M. N. Duc, and M. Hauswirth. Rdf stream processing with cqels framework for real-time analysis. In *Proceedings of the ACM International Conference on Distributed Event-Based Systems (DEBS '15)*, pages 285–292, 2015.

[25] L. Li, C.-J. M. Liang, J. Liu, S. Nath, A. Terzis, and C. Faloutsos. ThermoCast: A Cyber-physical Forecasting Model for Datacenters. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1370–1378, 2011.

[26] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos. DynaMMo: Mining and Summarization of Coevolving Sequences with Missing Values. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 507–516, 2009.

[27] L. Li, B. A. Prakash, and C. Faloutsos. Parsimonious linear fingerprinting for time series. *Proc. VLDB Endow.*, 3(1-2):385–396, September 2010.

[28] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action*. Springer, 2007.

[29] K. G. S. Madsen and Y. Zhou. Dynamic resource management in a massively parallel stream processing engine. In *Proceedings of the ACM International on Information and Knowledge Management*, pages 13–22, 2015.

[30] G. G. Pascual, M. Pinto, and L. Fuentes. Self-adaptation of mobile systems driven by the common variability language. *Future Generation Computer Systems*, 47:127 – 144, 2015.

[31] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[32] C. Qin and H. Eichelberger. Impact-minimizing runtime switching of distributed stream processing algorithms. In *EDBT/ICDS Workshop on Big Data Processing - Reloaded*, 2016. http://ceur-ws.org/Vol-1558/.

[33] A. J. Ramirez, D. B. Knoester, B. H.C. Cheng, and Philip K. McKinley. Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 97–106, 2009.

[34] S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345, 1999.

[35] K. Saller, S. Oster, A. Schürr, J. Schroeter, and M. Lochau. Reducing feature models to improve runtime adaptivity on resource limited devices. In *Intl. Workshop on Services, Clouds and Alternative Design Strategies for Variant-Rich Software Systems*, pages 135–142, 2012.

[36] I. Shafer, K. Ren, V. N. Boddeti, Y. Abe, G. R. Ganger, and C. Faloutsos. Rainmon: An integrated approach to mining bursty timeseries monitoring data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1158–1166, 2012.

[37] Rafael Tolosana-Calasanz, JosÃƒÂ© Ãƒ?ngel BaÃƒÂ±ares, Congduc Pham, and Omer F. Rana. Resource management for bursty streams on multi-tenancy cloud environments. *Future Generation Computer Systems*, 55:444 – 459, 2016.

[38] X. Wang, L. Tokarchuk, F. Cuadrado, and S. Poslad. Exploiting hashtags for adaptive microblog crawling. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 311–315. ACM, 2013.

[39] Y. Xing, S. Zdonik, and J.-H. Hwang. Dynamic Load Distribution in the Borealis Stream Processor. In *Proceedings of the International Conference on Data Engineering (ICDE '05)*, pages 791–802, 2005.

[40] B. Yang, H. Huang, and Z. Wu. TOPSIS: Finding Top-K significant N-itemsets in sliding windows adaptively. *Knowledge-Based Systems*, 21(6):443 – 449, 2008.

[41] T. Zheng, M. Litoiu, and M. Woodside. Integrated estimation and tracking of performance model parameters with autoregressive trends. *SIGSOFT Softw. Eng. Notes*, 36(5):157–166, 2011.