A Transition-aware Evaluation Framework for Adaptations in CEP Mechanisms

Bachelor-Arbeit
Niels Danger
KOM-type-number

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

A Transition-aware Evaluation Framework for Adaptations in CEP Mechanisms
Bachelor-Arbeit
KOM-type-number

Eingereicht von Niels Danger
Tag der Einreichung: 18. Oktober 2017

Gutachter: Prof. Dr.-Ing. Ralf Steinmetz
Betreuer: Manisha Luthra

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation (KOM)
Prof. Dr.-Ing. Ralf Steinmetz

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor-Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die schriftliche Fassung stimmt mit der elektronischen Fassung überein.

Darmstadt, den 18. Oktober 2017 Niels Danger

Contents

1 Introduction

1.1 Introduction

With the ever increasing number of hard- and software systems that generate continuous streams of heterogeneous data it becomes more difficult to extract meaningful information from this data and act upon it in real time. To handle this problem queries on data streams can be expressed with the help of Complex Event Processing (CEP).

Data sources (producers) generate streams of data in the form of simple events (e.g. sensor readings) which can be aggregated and combined with other simple or complex events and domain knowledge to create complex events. Interested parties (consumers) can define queries on event streams as complex events that can represent situations (specific event patterns) or specify further processing (aggregation, composition) of events. If event patterns matching the query are detected in the event stream, the CEP engine will notify any interested consumers immediately. This allows for timely reaction to new information by the consumers.

If the processing of the events happens in a centralized fashion, scaling of the system (in terms of joining producers, consumers or an increase of the event rate) is obviously limited. Therefore, distributing the processing of the data over several computing nodes is beneficial, if not necessary, in many application scenarios. This becomes even more important if the producers and consumers are constrained by certain QoS demands, e.g. in a scenario with geographically spread out producers and consumers where a certain end-to-end latency must not be exceeded. In this case it is not possible to route all events to a central processing node as it will likely cause the latency limit to be exceeded. Therefore the computations involved in processing a query need to be distributed across several nodes.

To find and maintain a mapping of the query (in the form of a logical flow graph consisting of sources, operators and drains) to the network hosts that satisfies all QoS demands imposed on the query, placement strategies are utilized. These strategies primarily handle the placement of the individual operators on hosts and try to optimize one or more QoS metrics in doing so and are periodically re-evaluated to check if the placement is still optimal or needs to be adjusted. These QoS metrics can be, among others:

- latency
- bandwidth utilization
- load balance
- cost of transport and processing
- reliability
- availability
- energy consumption

Currently, most operator placement algorithms used in practice focus on optimizing latency and/or load balancing. The placement of the operators cannot happen in an arbitrary way: some operators require a minimum of processing power or specific hardware; this is further constrained by sequential dependencies between operators: if an operator combines several simple or complex events, the operators that process these events need to be placed on processing nodes located upstream (viewed from the drain) of this operator's processing node, so this can reduce the number of feasible solutions.

Further mechanisms to satisfy QoS demands involve the optimization of the operator graph as well as parallelization by replication of operators.

The choice of the appropriate strategy and its mechanisms depends on both the network conditions and the QoS demands imposed on the queries by users. As both can change over time, some form of context awareness and adaptability is necessary to guarantee a steady level of system performance.

1.2 Problem Description

While periodic re-evaluation of the placement enables the system to react to changes in the underlying network to a certain degree, there is currently no way to adapt the utilized placement algorithm (or any of the other adaptation mechanisms) to major changes of the application environment that affect the QoS demands. If any part of the application environment (e.g. network conditions, network topology, consumers' preferences of QoS metrics, ... -> application context) is changed significantly, the mechanisms (e.g. placement algorithm) that were used before may no longer be optimal as they were intended for a different context. The ability to dynamically adapt the deployed mechanisms based on the application context is necessary to assure stable and reliable system service operation in the face of changing network conditions and user demands. To achieve this, the system needs up-to-date information on the context it is operating in. Context information is any information that is relevant for determining what mechanism to use (i.e. information about network conditions, query QoS demands, current adaptation mechanisms, etc.). This information must be captured in a structured way so that it can be automatically processed. For this, context modeling techniques can be used to construct a context model representing the current system and environment state. With this it is possible to capture dependencies between network conditions, mechanisms and system performance and monitor transitions from one system state (configuration) to another. By structuring the model through constraints and dependencies the amount of configurations can be limited to valid ones. The aim is to facilitate the determination of the optimal mechanism(s) for every configuration by employing machine learning techniques on training data structured in such a way by a context model. This in turn could enable a system to adapt to known situations as well as new ones without the need for manual reconfiguration.

1.3 Goals

In this thesis the existing CEP system AdaptiveCEP will be extended. The goals are as follows:

- identification of requirements for a context model for AdaptiveCEP
- design and implementation of the context model into the existing system
- design and implementation of an interface that allows the automatic adaptation of CEP mechanisms at runtime (strategy interface proposed in AdaptiveCEP paper); choice of mechanism should be determined using suitable Machine Learning techniques

2 Paper Summary

2.1 Operator Placement

The paper Placement Strategies for Internet-Scale Data Stream Systems by Geetika T. Lakshmanan, Ying Li and Rob Strom is a survey of 8 algorithms developed to solve the problem of optimal operator placement in data stream processing systems. To compare the different approaches, the authors define a set of core components that significantly affect the performance of the system and compare the categories of these components:

- architecture (centralized, decentralized or hybrid implementation of placement logic)
- algorithm structure (centralized vs decentralized decisions)
- metric(s) to optimize (load, latency, bandwidth, machine resources, operator importance, combinations of those)
- operator-level operations (reuse or replication of operators)
- reconfiguration (triggers for operator migration: thresholds, periodic re-evaluation, constraint violation)

Based on these components, 8 placement algorithms developed respectively by Pietzuch, Balazinska, Abadi, Zhou, Kumar, Ahmad, Amini and Pandit are assigned to the discussed categories. The authors then proceed by comparing the algorithms w.r.t. their design decisions and underlying assumptions. It is concluded that decentralized approaches which are able to adapt dynamically by operator migration are prevalent, with latency and, by extension, load balancing being the preferred metrics to optimize. Based on the comparison of the assumptions a decision tree for selecting a fitting placement algorithm considering the characteristics of a given problem (node distribution, number of administrative domains, dynamics of topology and data, query diversity) is proposed.

2.2 Context Modelling

Freudenreich explicitly differentiates between situation context (information about environment) and interpretation context (information about data). Regarding the modelling of situational context of event-based systems he identifies the following required features and characteristics, based on three surveys of different context models:

- aspects:
  - absolute and relative space and time
  - application as subject
  - no context history or user profiles
- representation:
  - key-value based
  - low formality
  - fully general, variable granularity
  - no valid context constraints
- management:
  - context construction distributed and at runtime
  - basic context reasoning

- hooks for incompleteness and context information quality monitoring
- multi-context modelling

After identifying these requirements, he argues that there is a need for relationship representation and basic reasoning, but ontology-based models, which provide elaborate context reasoning, are considered to be too detrimental to EBS performance. Therefore he proposes an ontology-based metamodel that can represent domain concepts in terms of concepts, attributes and relationships which allows for context construction to happen at runtime, while the context information structure is determined at design time. The metamodel makes no assumptions about the domain model's data types or structure, making it very flexible. Situations and reactions to them can be specified declaratively as policies using the defined ontology.

E.Barrenchea et al. propose to embed context information directly into events as a first class element. Components can specify the context of their publications and/or subscriptions. Components can be grouped into contextual environments (making them "siblings") that share the same context parameters and values. Context awareness is realized by context filters in an environment middleware layer that filter according to the specified event schema and the environment they are part of, which relieves components of keeping track of context information directly. This moves context away from the component logic and towards the layer of the pub/sub scheme. An environment manager handles context updates and provides an interface to the application layer for utilizing context information.

Koripipää et al. present a software framework for context acquisition and processing in mobile scenarios. The API uses an extensible context ontology to define usable contexts. The context schema is defined in RDF syntax to ease the sharing of the ontology. Context has six possible properties: (first two are mandatory) 1. type 2. value 3. confidence 4. source 5. timestamp 6. attributes. It is possible to create context hierarchies and composite contexts. The framework consists of a context manager (blackboard, delivers context information based on subscriptions), a resource server (collects and preprocesses context data from sources) and context recognition services (create higher-level contexts from atoms, can be plugged in at runtime). The recognition services make use of naive bayes classifiers and the confidence property to create higher-level contexts.

Some authors propose using feature models, which have their origins in the domain of highly configurable software product lines (SPL), as a means to model the variability and behaviour of context-aware systems.

M.Archer et al. suggest using two separate feature models: one for the system and its variable parts, and one for the context, each of them with its own constraints (e.g. require, exclude). This allows for a homogeneous representation of system and context and the relationships between the two. The two models are then linked together by dependency constraints which represent the adaptation behaviour. The adaptation of the system happens based on event-condition-action (ECA) rules, with elements of the context feature model as condition and a re-configuration of the system feature model as the action part. The authors mention optimization of a utility function as an alternative to ECA-rules for modelling adaptive behaviour, which would require the feature model to be extended with attributes for information needed for the optimization.

Saller et al. aim to enrich feature models with context information and context constraints in order to limit the set of valid configurations further. This is done to limit the complexity of re-configuration planning and enable such calculations on resource-constrained devices. Reconfiguration is modelled and pre-planned in the form of a transition system, with transitions from one valid system configuration into the next, based on context. To further reduce the amount of possible transitions, states with different configurations that satisfy the same constraints are considered to be identical in terms of transitions (incomplete state space). This model allows for multiple contexts to be active at once.

Weckesser et al. propose an approach that extends the expressiveness of feature models as well as the possibilities to validate them. It allows for modelling of feature attribute types and constraints beyond boolean as well as UML-like multiplicities of features. Furthermore, they enable automated validation of such cardinality-based feature models through methods that check the constraints imposed on the

feature model by cardinality annotations. This is useful when dealing with systems that have multiple instances of a feature and need to take dependencies among them (and their number) and other features into account.

2.3 SPLConqueror

SPLConqueror is a collection of programs and a library for discovering the influence of configuration options of software with variable elements on non-functional properties. This is done iteratively by using linear regression learning and stepwise feature selection. It consists of four programs:

- VariabilityModelGUI: creation and editing of variability models of the configurable system; allows for exclude/require constraints between elements
- PerformancePredictionGUI: learning an influence model from a given variability model and a set of measurements (of configurations and their performance regarding one metric). Possible configuration options for sampling and linear regression algorithm are available. The result is a an influence model of terms of weighted influences on one specific performance metric in the following form: (w1*c1 + w2*c2 + ... w34*c3*c4) where w34*c3*c4 denotes the selection of both configuration options c3 and c4
- SPLConquerorGUI: GUI for visualization and analysis of the learned model with options for filtering variables and adjustment of the model. Visualization of influences and detected dependencies
- CommandLine: allows for specification and execution of experimental setups with different sampling and machine learning options

Further submodules include MachineLearning (algorithm for learning performance-influence model; interface for SAT checking of configurations w.r.t variability model and optimization of configuration for a given non-functional property and objective function) and PyML (interface to scikitlearn (a python ML framework), different regression learning techniques).

Application to AdaptiveCEP: 1. determine from measurements for every mechanism (context (network conditions) -> performance (w.r.t. mechanism's optimized QoS metric?)) the influence of every context element on the performance metric by using linear regression; -> need to restrict combinations of context features + mechanisms to valid ones to limit necessary training/measurement data 2. the resulting formula (with weights for the influence of each context feature) can predict the performance of every context feature configuration for each mechanism 3. with this, calculate the predicted performance of every mechanism for a given context configuration 4. compare the performance predictions of the different mechanisms for a given context configuration; problem: how to compare performance measured in different metrics? Can it be reasonably normalized to [0,1], if yes, what lower/upper bounds to choose for latency, bandwidth, ... to make them comparable? 5. if reasonable comparison of performance metrics can be found: choose mechanism with best (normalised) performance for the given context

3 Context Model

3.1 Motivation

If an application has to be capable of adapting to changing conditions in its execution environment, it needs information on the aspects that can change over time and are relevant for adapting its behavior. This is achieved by representing the context of an application in a context model. According to Dey, context is "*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application*". Other definitions of context exist, such as the following by Etzion: "*Context is indirectly relevant information that is useful to functioning of the service, but not provided to the service as part of its invocation*". Both definitions imply that context information has to be collected and stored in a way that enables the application to ensure it behaves according to expectations. As every application has different notions of what relevant context information is, the required features of a context model need to be identified based on the desired behavior of the application.

3.2 Context Model Design

3.2.1 Application Requirements

In the case of AdaptiveCEP, the system should be able to detect the conditions of the network that hosts the operators of its queries. In order to ensure the compliance with any QoS demands placed on the queries, it should then be able to dynamically choose appropriate mechanisms (operator placement strategy, ...) that fulfill the demands and optimize their relevant QoS metrics even if the network conditions change. To do this, the application needs knowledge about the status of any nodes currently hosting operators (to detect performance deterioration) as well as alternative nodes currently unoccupied (for possible reconfiguration). Some of this information could be abstracted into coarser categories to facilitate decision making. Furthermore, it should be able to cope with joining and leaving hosts. The adaptability of the application is given by the different strategies therefore it should be possible to include new strategies, possibly with new types of QoS metrics. Useful dimensions of network status information are, among others: latency, bandwidth, utilization, throughput, location/velocity (in case of mobile network entities), energy consumption.

3.2.2 Context Model Requirements

To capture the requirements of the context model in a structured manner, the analysis framework proposed by Bolchini et al. is used. This categorizes the requirements as follows:
1. Modeled Aspects

Feature	Analysis Result
space	yes, necessary for proximity demands
time	maybe, for "freshness" of information
absolute/relative space and time (coordinates/timestamp vs. near/after something)	abs.+relative space, absolute time
context history (does the current context state depend on previous ones)	unclear, maybe for stability metrics of links; needed for machine learning purposes?
subject (what is the point of view -> user or app perspective)	app centric, system needs context information to optimize strategy usage
user profiles (are user preferences relevant for the context, how are they represented)	Qos demands can be viewed as user preferences

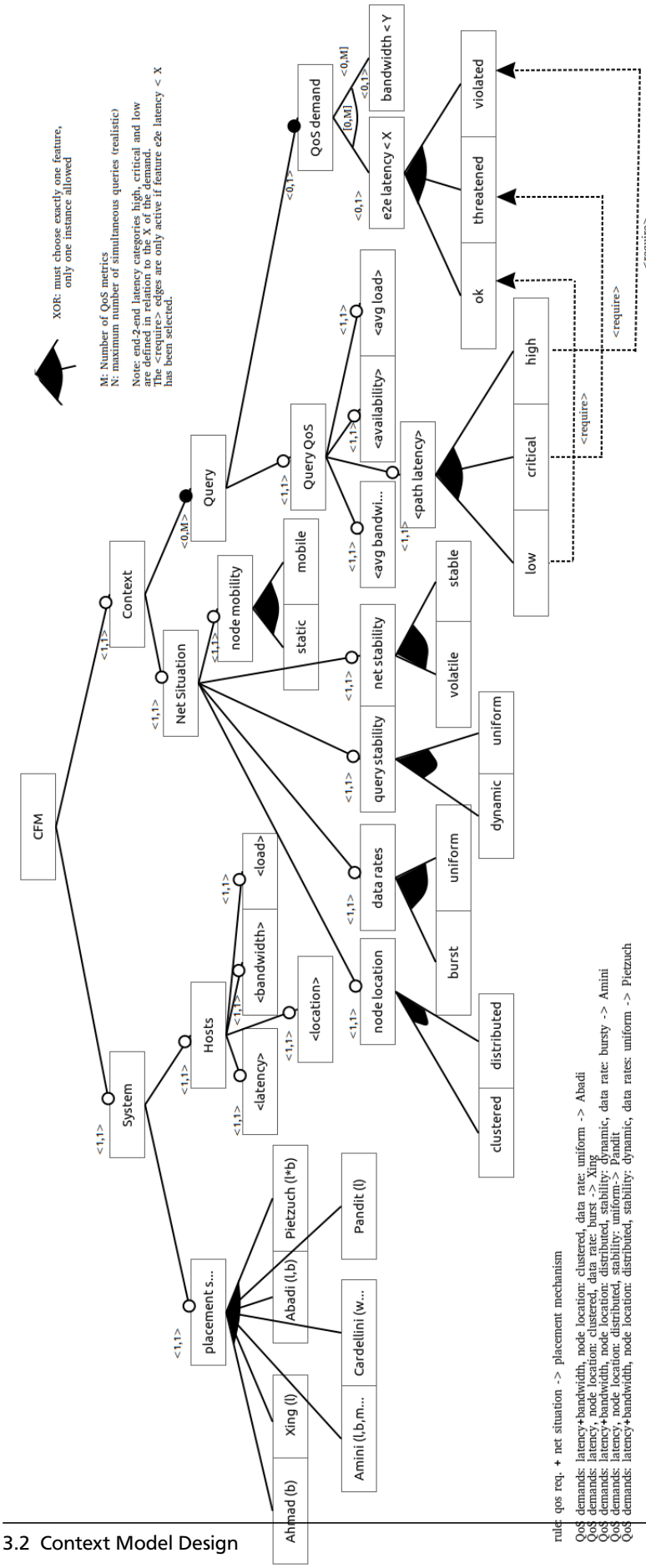
2. Representation Features

Feature	Analysis Result
type of formalism (key-value, markup, logic, graph, ontology -> different intuitiveness, reasoning possibilities)	Context Feature Model == graph-based, logic for constraints
flexibility (domain specific vs general)	specific to application, with enough flexibility for extensions
variable context granularity (ability to represent context at different levels of detail)	-
valid context constraints (can the number of possible contexts be restricted by semantic constraints)	necessary to restrict space of system configurations; inherent to CFM

3. Context Management and Usage

Feature	Analysis Result
context construction (central description of possible contexts at design time vs distributed construction of the current context at runtime)	central description of context dimensions, distributed construction by collection of current values
context reasoning (ability to infer properties or construct higher-order context information from sensor readings)	maybe, abstract from measurements to situations
multi-context modeling (one instance of the model can represent all possible context, not one instance for each context)	yes, need to model changes between contexts

3.2.3 Context Model Proposal



The model limits itself to those elements of the system that are relevant for the detection of or reaction to network environment changes. The graph modeling technique used to represent the context model as a feature model was first used in the domain of Software Product Lines (SPL) and allows definition of variable parts (features) of both system and context. At runtime, the active features constitute the configuration of the system/context; the space of possible configurations is restricted by constraints between features (xor-, or groups, require- and exclude relations). This allows the restriction of the model to valid configurations, which is helpful when gathering measurement data for different possible configurations. The main elements of the context are the queries executed by the system, their QoS metrics, the QoS demands placed upon them, and a set of situations that characterize the network conditions of the nodes in the system. These situations abstract from measurements on single hosts to the overall state of the queries' environment, and can be used in the definition of constraints and adaptation rules. Their mutually exclusive subcategories (like dynamic, uniform) can be defined by threshold values (in some relation to QoS demands on operators) and can be defined more finely granular if necessary. Note that multiple Situations can be active at the same time. The variable parts of the System are the ones that can be directly reconfigured: the placement mechanisms, each optimizing a set of QoS metrics and suited to a certain situation. Although they can not be directly reconfigured, the Hosts/Nodes are an integral part of the system, so they are included on this side of the model. Every Host in the System has a set of associated QoS measurements. For the sake of readability, constraints among the features of the model can as well be noted separately as follows:

(C1) Situation:data rate:burst **and** Situation:node location:clustered **and** Query:QoS demand: e2e latency < X **requires** placement:Xing

with Xing being a placement strategy that further increases the latency. Adaptations from one Strategy to another could be triggered in a similar manner:

(A1) Query:QoS demand:latency **and** Query QoS:path latency:high **implies** other latency optimizing strategy

Implementation Considerations:

- context information could be distributed via context events; a context manager subscribes to context sources and the application (or adaptation strategy) subscribes to the aggregated events from the context manager
- context source monitoring could happen similar to the existing MonitorFactories
- create a new interface for the context components (metrics, attributes, constraints) and its management
- implement the strategy interface described in the AdaptiveCEP paper to allow for specification of mechanism behavior