Masaryk University
Faculty of Informatics

Semantically Partitioned Complex Event Processing

PhD Thesis Proposal

Filip Nguyen
Supervisor: doc. RNDr. Tomáš Pitner, Ph.D.

Brno, January 2014

Statement

I declare that this thesis proposal is my original copyrighted work, which I developed alone. All resources, sources, and literature, which I used in preparing or I drew on them, I quote in the thesis properly with stating the full reference to the source.

Contents

1 Introduction **1**

1.1 Complex Event Processing 2

1.2 Aims of Thesis 3

2 State of the Art **5**

2.1 Complex Event Processing 5

2.1.1 Basics of CEP 6

2.1.2 Notable Extensions 8

2.2 Performance and Distributed Complex Event Processing 10

2.3 DEBS Grand Challenge 14

2.4 Middleware Support 15

2.5 Applications 16

3 Proposed Research **18**

3.1 Experiments 21

3.1.1 DEBS Grand Challenge 2014 Dataset 22

3.1.2 Intelligent Buildings and Smart Meter Data 25

3.2 Time Plan 25

4 Achieved Results **27**

4.1 Complex Event Processing 27

4.2 Information Systems and Middleware 27

Bibliography **28**

A List of Results **39**

A.1 List of Papers 39

A.2 Presentations 40

A.3 Teaching 40

A.4 Thesis Supervision 40

A.4.1 Bachelor Thesis Supervision 40

A.4.2 Diploma Thesis Consultant 41

A.5 Full Papers 41

A.5.1 IDC 2013 [13] 41

A.5.2 Scaling CEP to Infinity 52

A.5.3 BCI 2012 65

A.5.4 Control and Cybernetics - ADBIS extended paper 2012 . . 72

A.5.5 NotX service oriented multi-platform notification system . 88

A.5.6 Co-authored IDC 2013 97

1. Introduction

My thesis proposal targets Complex Event Processing (CEP), an area that shares many aspects with stream processing. CEP is mainly application oriented and dedicated to enabling the processing of vast amounts of data online while the data are being generated.

The goals of my thesis, which are outlined in this proposal, are aimed towards both theory and application. In theory, I would like to introduce and validate a new model for distributed CEP called *Semantically Partitioned Peer to Peer Complex Event Processing* (PCEP). The application oriented goal is to develop a processing engine that uses pillar ideas from PCEP. Such an application could then be used to solve problems in the real world, within the energy distribution industry and the monitoring of smart buildings.

The structure of this proposal is as follows. The rest of this introductory chapter summarizes the proposal on a high level and lists the aims of the thesis.

The chapter 2 State of the Art investigates current approaches to CEP, distributed CEP and their usage in the field of monitoring. It also investigates some middleware tools, that are used to support event processing.

The chapter 3 Proposed Research introduces PCEP and defines goals of the thesis in more detail. It also proposes experiments and metrics used to evaluate them. The chapter concludes with a time plan for the rest of my doctoral study.

The chapter 4 Achieved Results highlights my accomplishments during the doctoral study. The chapter is designated as an input for Advanced (in Czech: Rigorózní) State Examination. It is divided into two sections, the first of which describes contributions in CEP. The second one proceeds to describe contributions in the area of Information Systems and middleware. The reason for this

split is that the CEP contributions are more directly related to this proposal and thesis, but the latter area is also relevant for Advanced State Examination.

1.1 Complex Event Processing

CEP was introduced in [1] in a very comprehensive fashion, it focuses mainly on terminology and basic architecture of how to use temporal operators to reason about streams of events (selecting events that are related according to the time of occurrence, in other words, searching for a pattern in events)

Even in this publication about CEP, the author acknowledged the importance of distributed aspects of event processing. There are immense differences in definitions of distributed CEP from various authors ([1, 2, 3, 4, 5]). The meaning of the word *distributed* ranges from a distributed collection of events to a distributed processing of events on independent CEP processing nodes that communicate with each other.

The applications of CEP are mainly from the field of monitoring. This is because monitoring is implicitly a field that generates a high volume of data which is hard to process and it is difficult to extract meaningful information in reasonable time. One of the sources, that produces large volumes of data is Cloud Computing, where multiple clients (tenants) use the same hardware on a computing node. Another prominent field of CEP application areas is the monitoring of sensor data. Hardware sensors produce a high volume of data with a high enough resolution to overwhelm current processing techniques. Examples of such sensors are RFID sensors, smart meters, NetFlow network units. The nature of the data collected from sensors introduces additional problems to event processing. Sensors may introduce duplicate or in other way corrupted data, they may be used as a tool to inject fraud data into the overall system, and the sensors may become unavailable (stop producing the data). These challenges must be faced in a consistent fashion with CEP limitations and philosophy - online and fast.

When the CEP solution is being deployed, the first step is to decide what kind of user queries/questions should be answered by the solution. This may include detecting a peak values of a metric (e.g. power consumption measured on a smart

meter) or complex time correlations between multiple events (e.g. fraud detection by correlating several events, separated by larger time intervals). Another example might be computing the total power consumption of a device for billing purposes.

1.2 Aims of Thesis

Within my research, I specifically target the non-deterministic queries (queries that if not answered are not critical for the client - these queries will be later defined in this proposal as *result scalable*) and architecture/solutions that should answer this type of queries. The reasons for this, along with some examples, will be described in chapter 3 Proposed Research. I believe that such queries are best suited for event processing. On the other hand, I cannot leave this decision unexplained, since deterministic queries are so heavily studied - in research [6, 7, 8, 9, 10, 11], also in the monographs [12, 1].

The main aim of the thesis is to focus on PCEP, which is also covered in detail in my recent publication [13]. It is an architecture of event processing that views an event producer as the most important element of the whole event processing network. In the thesis, I aim to provide arguments in support of the feasibility of PCEP. A prototype implementation of PCEP will also be validated against real world data and a standard data set of Lasaris laboratory.

Even though the validation may be done largely theoretically, it is important to conduct experiments with real implementation of the theoretical model in order to validate more complicated properties, e.g. PCEP performance on real data. This is even more emphasized by the fact that important problems in event processing come from data pollution and other specifics that are hard to model theoretically.

In order to evaluate the prototype implementation, there are three data sets that will be used. Two of them are under direct control of laboratory Lasaris. These are smart meter data (about 100GB) and data from intelligent buildings (40GB). The third data set comes from the event processing community - data set from smart plugs. The third data set is a collection of real world data, that were collected across 40 houses in Germany last year. The data are collected from

hardware sensors called smart plugs. Smart plug is a device between electric power outlet in a wall and an electronic device. The smart plug publishes data each second. During my evaluation I will use this data to show prediction capabilities of PCEP and also search for outlier smart plugs (those with high electric consumption). The data sets and proposed experiments will be described in more detail in chapter 3 Proposed Research.

2. State of the Art

The research aspect of the distributed CEP may be approached from several points of view. To most properly align the state of the art with my research goals, the following areas had to be explored: Complex Event Processing, Distributed CEP, Middleware Support for CEP and Monitoring.

2.1 Complex Event Processing

The advent of Big Data, Cloud Computing, and Internet of Things indicates a need for the processing of large amounts of data. Since more and more storage is available, the organizations tend to store all data. Many fields of research generate vast amounts of data, e.g. oceanography, astronomy (Large Synoptic Survey Telescope is producing 15TB of raw data per night, resulting in 150PB of data over ten years[14]). Another example, as noted in [3], that one popular web site that recommends famous Japanese restaurants supports 70,000 e-shops, if each one would correspond to a CEP rule, the service would have to accommodate 70,000 CEP rules which would exceed capabilities of today's CEP engines. Furthermore, extracting information in a meaningful time frame might be of a high interest. CEP is an area, that emphasizes the importance of near real-time processing of data and extracting meaningful information from them. This highly contrasts with other efforts in data analysis that aim to process stored data by distributing a computation (e.g. Map Reduce). Note that Map Reduce has other advantages over CEP. Also, there are even research efforts to combine the CEP and Map Reduce [7].

CEP researchers also mention the importance of a data producer. This is largely neglected in other data processing approaches.

There are two comprehensive monographs in the field. The first view on CEP

was introduced in David Luckham's book, *The Power of Events* [1], in 2002. This publication still provides common terminology, design patterns, and techniques for researchers that focus on CEP. The second comprehensive view is in book Event Processing in Action [12].

To conclude the history of CEP, at the same time as the book Power of Events from 2002 was published, there was a first workshop called Distributed Event-Based Systems (DEBS) 2002, it took place in Austria, and was co-located with ICDCS. After several years, the first specialized conference in the field, ACM Inaugural DEBS 2007, took place in Toronto, Canada. Presently (2013), the conference is organized annually, and includes a forum dedicated to the dissemination of research related to event-based computing.

2.1.1 Basics of CEP

In this subsection I will introduce the most important concepts of CEP, together with definitions that suite the purposes of my research the most. The definitions in this section are my own contributions.

Event processing centralizes around the notion of an Event.

Definition 1. ***Event*** *is a record of an activity in a system. The event has two aspects: the content (carries static data) and the time stamp. Formally, an event* E *is a tuple* $E(t, p)$ *where* t *is the time stamp and* p *is a the set of key-value properties* (k, v) *of an arbitrary type.*

The above definition is a modified version of a more elaborate and less formal version in [1].

The events are usually mapped to understandable, real world events, such as measurements of a value on a sensor or the sale of a stock in an automatized stock market. An important concept that allows the extraction of information from a flow of events is called a pattern ([1]). Here I present a very broad definition of a *CEP query* that contains a pattern.

Definition 2. ***CEP Query*** *is a template that matches a set of events. Formally, a CEP Query* $Q(P, W)$ *is a tuple, where* P *is a pattern and* W *is a temporal*

length (time window, e.g. 10 min). The pattern P is a logical formula in the propositional calculus. Let $E_0(a, b)$ and $E_1(c, d)$ be events. A variable in the formula takes the form of comparison of two elements from a set: $\{a, c\} \cup b \cup d \cup CONSTANTS$, where $CONSTANTS$ is a predefined set of constants.

An example of a pattern P over two events

$$E_1(e1.timeStamp, props1), E_2(e2.timestamp, props2)$$

might be a pattern:

$$e1.timestamp > e2.timestamp \wedge props1[producer] =' SensorX'$$

The CEP query is detecting a pattern in the flow of events and is usually associated with a trigger action. The action will happen after the pattern is found in an event stream. The resulting action of a query may be a creation of a new event.

For purposes of this thesis, the CEP applications may be divided with regards to the nature of the queries to two disjoint sets: result binary, result scalable.

Definition 3. ***Result Binary*** *A CEP query is result binary if and only if the receiver of the query results, requires the results to be deterministic with regards to the input events.*

The complementary categorization of a query is *result scalable* query. Every query is either *result binary* or *result scalable*. It can be observed, that many queries, mainly from monitoring, are result scalable, because the receiver of the results regards the results as additional information (e.g. uncovering credit card fraud).

There are many queries that are result binary. A large amount of research and software solutions today focus on such queries. An example of a result binary query is a computation of the total power consumption of a device for billing purposes. The receiver of these results (possibly a distributor of the electric power) expects this information to be precise with only a few marginal errors in order to successfully bill its customers.

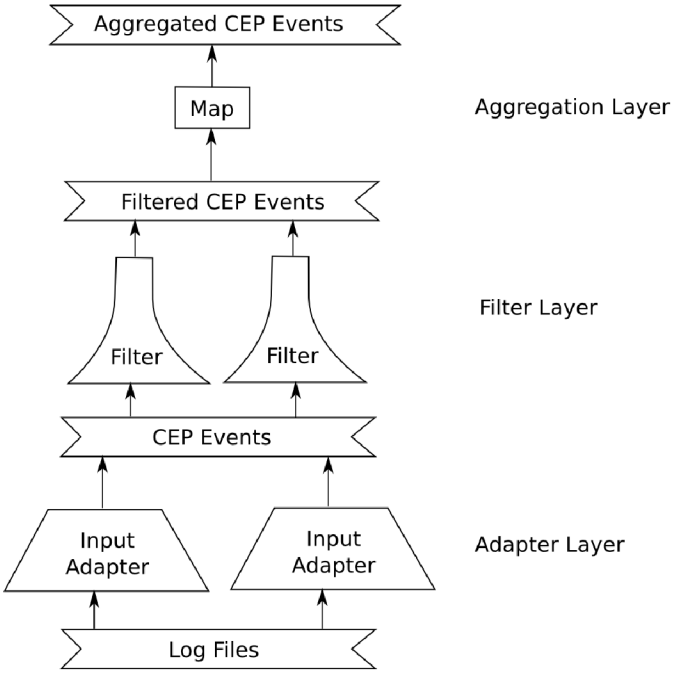


Figure 2.1: Example Event Processing Network for Log Files Processing

Very often, an abstraction of an Event Processing Network (EPN) is used to explain the architecture of an event processing solution. A network is using layers and is very similar to a graph, where the nodes are called Event Processing Agents (EPA) [1]. An example of such an EPN might be the processing of network event logs on Figure 2.1. This example is based on a notation from [1]. The first layer of this EPN converts the log records (lines of log files) into the common format that can be understood by the CEP engine. The EPAs in the second layer (Filter Layer) can be absolutely independent from each other and can function as filters that discards corrupted log records. The third layer of EPAs (Aggregation Layer) take this filtered input and may discover more sophisticated patterns between the events. Advantages of using EPNs are re-usability of EPAs and the possibility to distribute the EPAs across a network.

2.1.2 Notable Extensions

The following two examples demonstrate recent (2008, 2013) development of the CEP theory.

One direction of research in CEP is adding new extensions to the CEP language. These extensions are accompanied by new semantics that add capabilities to CEP or improve its performance. An example of such efforts is portrayed in the publication [15], where authors create a language extension for RFID (Radio Frequency Identification) events. A contribution of high importance is the possibility to address the non-occurrence of an event in a query.

Another interesting, and intuitive research direction in language extensions is the enrichment of events [16]. This enrichment is automatic and uses semantic web. The motivation for this is that events are usually very encapsulated and don't contain much context. This research direction uses standard knowledge technologies to automatically enrich an event with necessary information. This unified approach to enrichment is very interesting in regards to my research because provides the necessary tools to extend algorithms that discover similarities between events of producers - which in turn discovers similarities between the producers.

Some theoretical advances in the field deal with the uncertainty of events, which relates strongly to my work. In [17] the authors use *intelligence-based event processing* that supports probabilistic reasoning. The need for such techniques is driven by the fact that many uncertainties typically occur within the area of event processing (e.g. latency between producers, peaks in measured sensor data, even a direct possibility of some event occurring in the future - based on historical data). With the possession of more probabilistic tools, it may be possible to answer queries that take into account the possibility of random event - not just a binary occurrence of it. Event processing systems have to deal with various types of uncertainty: incomplete event streams, imprecise time stamps associated with recognized events, inconsistent event annotation, etc. Having such inconsistent and sometimes unpredictable data indicates that CEP result receivers should incline towards asking only result scalable queries.

Similarity functions may be used for correspondence between items over data sequences. This has recently been studied in [18], and I may use some of the verified similarity functions (from this publication) in my research.

The idea of PCEP is to uncover similar producers and connect them. Similar

ideas can be found in today's research within relation to *proactive-event driven computing* and also in event processing for mobile applications.

Proactive event-driven computing [19] plugs in prediction models into the the event processing. This approach is not focused on extending CEP language, but on improving the event system on a more conceptual level. It introduces a proactive approach that is trying to mitigate or eliminate undesired future events. Because elimination is based on uncertain prediction, the results tend to be less reliable - similarly to PCEP. It can be observed that many current CEP solutions are solely reactive. Even from the definition of CEP query, it is clear that the intent is firstly to detect a composite event and then react to it. A further improvement to the overall concept is to introduce event sources that may be predicted (e.g. weather conditions can be predicted) which will trigger preventive changes in system policies. This proactive event-driven computing also supports my view that *result scalable* queries are important, because I think the predictions are already perceived by the clients as uncertain.

Recent mobile application event processing research is focused on the clustering of device space ([20, 21]. Mobile devices work as sensors and the tendencies of groups are exploited in order to begin analysis within smaller groups.

The connection between CEP and middleware solutions for enterprise applications is evident. Recent publications [22] show how integration solutions leverage ideas from event processing. I believe this is mainly because the majority of research in CEP applies directly to real problems and is usually validated by extending current systems. This argument is also supported by [23] in an advanced tutorial on design patterns in event processing. The tutorial identifies routing and compositions as one of the advanced CEP patterns. These are key aspects of the enterprise integration [24].

2.2 Performance and Distributed Complex Event Processing

In this section I will review the current approaches to CEP scaling as well as introduce distributed CEP. I will use the definition of distributed CEP that is most suitable for comprehension in the context of this proposal.

Definition 4. *Distributed Complex Event Processing is the collection and processing of events on several computing nodes, divided by a computer network. The addition of a new processing nodes is carried out with the goal to increase the performance of the whole system.*

There are many approaches to scaling CEP that don't exploit distributed computation. For example, research is currently taking place by use of using parallel pattern matching with multiple cores [25]. The aim of this research is to create fine grained parallelism while preserving matching capabilities. Similarly, [4] presents results on the topic of vertical scaling possibilities by analyzing typical client code in event processing systems. This contribution focuses on specific cases but generalizes the observations.

The tier approach to achieve high throughout (e.g. in [10]) is another way how to increase performance. This approach is similar to Map Reduce implementations, where each layer of computation may be done massively in parallel.

[3] delivers a SCTXPF platform that handles a large number of events from different event sources. It allocates CEP rules only to some of the computing nodes and achieves a throughput of 2.7 million events per second. The idea is to enhance typical load balancing so that it is aware of the internal intricacies of CEP rules.

The *publish subscribe* paradigm may be viewed as a CEP specialization. This can be observed even in early contributions to the publish subscribe field [26]. The results that improve publish subscribe performance may be used for distributed CEP. Many problems and research in publish subscribe systems also answer result scalable queries.

The system outlined in [27] is used for distributing events among subscribers of a notification framework. This work ensures QoS by using user-defined cost functions, so this service - a query for notifications - may be viewed as *result scalable*.

To see more distributed features that are similar to distributed CEP, it is possible to draw references to event routing in publish subscribe. A recent study

[28] concluded that content based routing shows many benefits to the publish subscribe system. Another argument for *result scalable* queries may be found in [29]. The authors developed an Event Subscription Recommender (ESR) in order to dynamically produce new event subscriptions.

There are recent publications (2012) that touch upon an idea similar to mine (the core of PCEP - the idea of semantically partitioning event space). The distribution/expressiveness trade-off that is part of my research is being studied by Tariq in [30, 31, 32]. Authors use spectral clustering for publish subscribe scenario and reduce the cost of event dissemination by grouping subscribers in order to exploit the similarity of events. Theirs method also uses P2P networking. Notable characteristics of their solution are:

- The usage of the spectral graph theory in distributed settings
- The evaluation of the proposed distributed spectral mechanism
- The effectiveness evaluation of event dissemination

In the described method, similarity is identified between different subscribers of events. These subscribers are then grouped into clusters, and event dissemination within those clusters can be very efficient with regards to false positives. The clustering is done based on the Jaccard similarity function with regards to the similarity of subscriptions. I suspect that a similar approach may be tested with my method by clustering producers based on generated events. After the clusters are created, the method outlined in [30] periodically re-establishes clusters of peers.

Other authors have also touched upon the idea of clustering event processors, but in a more engineering manner. In [33], the authors use CEP queries as an input for their algorithm that divides EPN into so called strata. In each strata, processing agents are completely independent, and can therefore run in parallel. This stratification has its own limits in scaling. Because it is possible that in a specific scenario no event processing agents are independent. In this worst case scenario, the stratification fails to bring any benefits.

Balis et al. [34] apply distributed CEP that answers result binary queries. The queries find answers to monitoring questions about a computational grid. This work features the true distributed nature of CEP, distributing queries to nodes that are divided by a computer network. The researchers took the approach where they distribute a possibly complicated query such as the calculation of a max aggregation function to several nodes. Each node deploys a sub-query that computes the maximal value for a limited number of producers. From the resulting stream, a higher level CEP engine computes the maximum among those.

The ideas of PCEP can also be related to scalable management and self-organization capabilities that are central requirements for large-scale, highly dynamic distributed applications studied by authors in 2003 [11]. This research resulted in the creation of the Astrolabe CEP engine. This system has information propagation delays of tens of seconds on a very large scale.

Several research groups are concerned with studying publish subscribe and CEP with geospatial extensions. This area is promising for the application of PCEP, because it provides simple partitioning rules (based on the location of the producer).

The geospatial extensions for CEP were studied recently in [35]. This extension adds another dimension besides time, and that is space.

Another possible geospatial clustering is described in [21]. Crowds of people usually gather at important events like concerts or conferences, or in a traffic congestion. When such events occur, it is interesting to gather events from these groups of people for analysis.

Among other distributed CEP systems that improve performance, it is possible to look at the work by Randika et al. [2]. This research group has created an engine called epZilla. This system is interesting in regards to peer to peer properties. It uses a *leader election algorithm* to establish a coordinator of CEP engine clusters. The engines in this system are grouped into clusters with leaders which are being assigned CEP rules for processing.

In [5], authors use query rewriting techniques to enable queries to be distributed more efficiently. The idea gives good insight into the limits of distributing general

CEP queries.

Last important note is that my technique is not to be confused with key-based partitioning recently studied in [36].

To summarize, the distributed CEP and monitoring is a promising area. Publish subscribe systems are heavily investigating these ideas [37, 38, 39]. Clustering itself is already being done in the CEP community, through the use of geospatial techniques - and I believe it would be beneficial to have more automatized techniques to provide more general support for the partitioning. I would like to neglect the area of vertical scaling (which was touched upon in this section) and focus solely on horizontal scaling in the same fashion as [30], but with much simpler query dissemination than is presented in other recent work (e.g. [34]).

2.3 DEBS Grand Challenge

As of 2011, there exists a series of challenges driven by the research community, known as the DEBS Grand Challenge. This challenge enables a research group to demonstrate the strength of event processing systems.

Each year, the assignment changes significantly. Thusly, merely by observing the Grand Challenge assignments and their solutions, it can be observed in which direction event processing is continuing. In DEBS Grand Challenge 2011 [40], the DEBS community identified broad goals for the series of Grand Challenges. The first assignment for the year 2011 was published [41] and made public on a website [42]. This first assignment was rather artificial, assigning a social game with 3 event producers: a question generator, player, and a system which generated *control* events. Since 2012, the challenges are being regularly published as peer reviewed papers with detailed descriptions of the solutions.

The challenge for year 2012 [43] featured a high tech manufacturing domain with real world data from the plant. The 2013 challenge was concerned with tracking and analyzing of football match statistics. The raw data that served as an input for the statistics was tracking data from the RedFIR tracking system [6]. The data were collected at Nuremberg Stadium in Germany. The results of this Grand Challenge are already published as regular papers. The solutions are very inter-

esting, because they use novel approaches. The map reduce paradigm was used in one of the solutions [8]. Another approach was the development of a custom processing engine [7], and its comparison to the state of the art Esper engine.

The most recent Grand Challenge 2014 [44], that has not been solved yet, is the most interesting in regards to my research. This Grand Challenge features a data set from smart plugs. A smart plug is a device which is attached between a power outlet and an electric device. It measures the consumption of energy and publishes this information every second. The goal of this Grand Challenge is to answer two result scalable queries. I plan to address this data set as a baseline during my research. Because the solutions to this Grand Challenge will be published, it will be possible to make a scientifically sound comparison of my solution to the others. A more detailed description of this data set is given in chapter 3 Proposed Research.

2.4 Middleware Support

Both commercial implementations and the research community tend towards an SQL-like pattern matching syntax [15]. Many different processing engines exist. I suspect this is due to the fact that the field is still unsettled and researchers don't find all of the necessary features in the existent engines. Also, instead of building on top of existing engines, research usually revolves around creating new features into theirs own implementation.

The history of event processing engines that truly represent Complex Event Processing, dates back to the year 2002 and the RAPIDE project [45] that was created by a group led by David Luckham. This software package was used as a referential implementation for the book Power of Events[1].

Currently there are several well known implementations of CEP technology. The most notable ones are Esper, Storm and Drools Fusion [46, 47, 48, 49]. All of these systems function in a centralized fashion. They allow clustering to some degree, but they still need centralized coordination.

It is interesting to note that the research community has identified some event processing capabilities of other open source projects such as Apache Camel [22].

2.5 Applications

Most CEP applications are from the area of monitoring ([50]).

A great amount of research revolves around analyzing RFID data. Notably, the authors of [51] created their own engine for this task. This engine featured typical query operations (large sliding windows and intermediate result sizes), and everything was meant to be applied to supply chain management, surveillance, and facility management, health care, etc.

Some of the applications use existing CEP platforms. The authors of [52] used the IBM InfoSphere platform for analyzing CDR records. This analysis is particularly hard due to business and operational constraints of the legacy system used in the deployment environment.

Another proprietary high tech manufacturing application [53] features the automated and timely detection of anomalies which can lead to failures of the manufacturing equipment.

More high level studies are also available. The authors of [54] created a system that aimed to increase the comfort of passengers in public transportation. In the city of Helsinki, Finland, public transportation vehicles were monitored using real-time sensors, and the system analyzed individual driving styles.

Sensors in the form of set-top box event processing [55] were used to analyze the real-time tendencies of the television viewers.

Another big area of application for CEP is network monitoring. Network monitoring tools usually generate a lot of information of which only some is relevant in a very short time frame (e.g. during a network attack). Recent work in this area portrays benefits of event processing technologies ([56, 57, 9]).

Cloud monitoring using CEP is also a promising area of application ([34, 58]).

A model based validation of streaming data is used for comparing a model of event streams to an actual event stream [59]

The advent of mobile devices and software that accompanies them also spurred a rise in CEP applications to monitor and ensure QoS for them. In [60], the latencies combined with data location and bandwidth capacity usage are exploited

to ensure better QoS.

3. Proposed Research

My initial publications on the subject [13, 61] introduce the idea of PCEP and explain the motivations behind it. As seen from the state of the art discussed in this proposal there is a consensus in the event processing community about the methodology to advance event processing. Researchers usually develop a prototypical event processing engine and validate its usability on fabricated or real world data. The comparison is then done against existing, general purpose, CEP engines.

In this section, I will briefly introduce the the PCEP and outline extensions to be made to PCEP. Next, I will discuss planned data sets and the experiments to be conducted.

An exact definition of PCEP is out of the scope of this proposal. However, I will list several typical characteristics of a PCEP solution. PCEP solution exhibits:

- Producer-centric architecture, where producer also acts as a CEP engine.
- Each *peer* (producer/CEP engine) uses the same algorithms as others and all the peers communicate in peer to peer fashion
- Event dissemination may not be complete, some aggregate events may not be detected in favor of performance

Basic building blocks of PCEP are shown in Figure 3.1. The notation allows to model event processing systems that feature event producers and possibly centralized engines. The Figure 3.2 shows a simple centralized architecture. This architecture is typical for existing CEP engines. The notation allows to show P_4 as a centralized engine that has an active CEP query. The producers are connected by edges along which events, numbered by discrete time values, flow

Figure 3.1: Basic PCEP building blocks

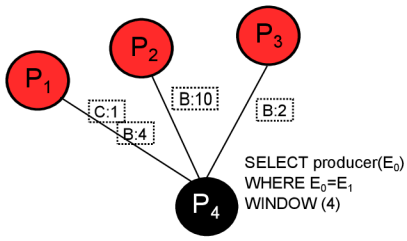


Figure 3.2: Centralized CEP

from producers to the engine.

There are queries that in an ideal case, need a centralized engine. For example, the query *SELECT producer(E0) WHERE E0=E1 WINDOW (4)* needs all events in the time window of size 4 to decide whether a new event should be matched against the pattern. The core of my method is to find only those producers that will most likely produce such same events and run event processing among them. To see such dynamic reconfiguration, please refer to Figure 3.4. On the diagram, P_6 is the ideal centralized engine. By uncovering that P_1, P_3, P_4 are related, I am able to place a new engine P_7, to conduct event processing on a smaller scale - thus faster. This idea is exploited in the form of so called *partitioning algorithms* (my original work) that divide the producers into clusters in which they relate to each other. Still, the layout of the event processing network on 3.4 is useful only for explaining the problem and the core idea. The real PCEP has to fully utilize peer to peer architecture without fixed centralized coordination. Such a situation may be depicted as in Figure 3.3.

The extensions to be made to the current model are:

1. The development of standalone Java daemon (called peer) that implements the PCEP protocol
2. The usage of spectral clustering methods for partitioning

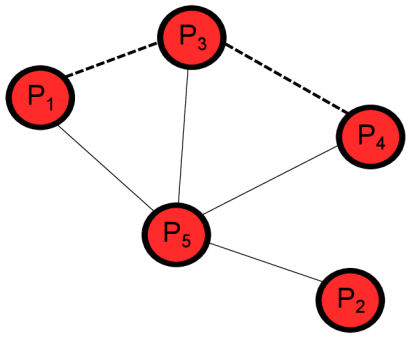


Figure 3.3: Peer to Peer Architecture

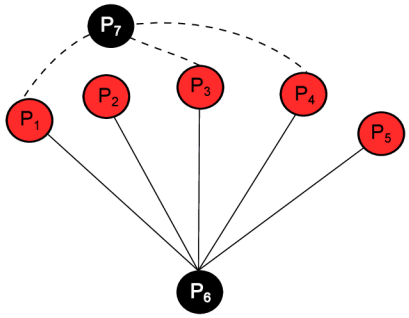


Figure 3.4: Event Space Correlation Problem

3. Simplification of the Monte Carlo partitioning method

The need for arose as a consequence of the evaluation in my publication [13]. The current implementation of the partitioning algorithm is too complex in regards to memory consumption.

Using the spectral clustering methods mentioned in item 2 for partitioning is a very promising automatized way how to conduct the clustering. It may uncover unforeseen relations between producers. This idea arose from my discussions with peers during the international conference IDC 2013.

Lastly, the item 1 is a logical consequence of my efforts. It is beneficial to validate the resulting architecture in a real distributed environment. This approach of developing a proprietary implementation of a CEP engine is a standard approach in the field of event processing.

As for concrete application purposes, I plan to apply PCEP to smart grid infrastructures, smart buildings, and network intrusion systems. The effort and feasibility of this idea can be observed in recent publications of event processing. As described in [62], smart grid infrastructures are being transformed into a system with advanced sensors featuring two-way communication.

3.1 Experiments

I plan to continue with the experiments that began in [13]. The experiments were conducted in a simulated distributed environment. The gist of the evaluation was the comparison of two partitioning methods (Monte Carlo and CEP based Partitioning Algorithm) against an ideal, not partitioned, CEP engine. The results 3.6 show that Monte Carlo partitioning performs better in regards to semantic power but suffers from memory consumption. CEP Based is also, subjectively, more intuitive to use for an inexperienced user.

The simulation environment that I will use for evaluation is depicted in Figure 3.5. Part of the simulation environment was developed in the bachelor thesis of Štefan Repček, which was under my supervision. This simulation environment will be loaded with 2 sets of data. One data set is available at the Lasaris laboratory, at the Faculty of Informatics, Masaryk University. The second data set will be the

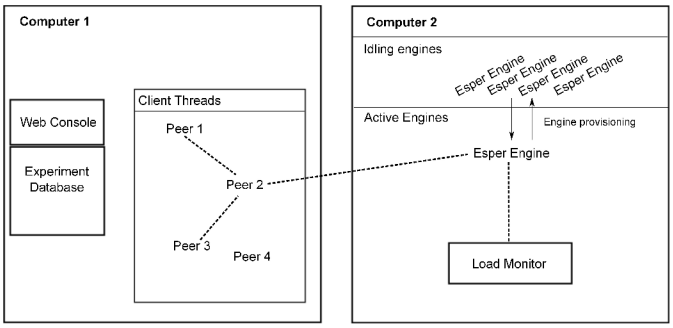
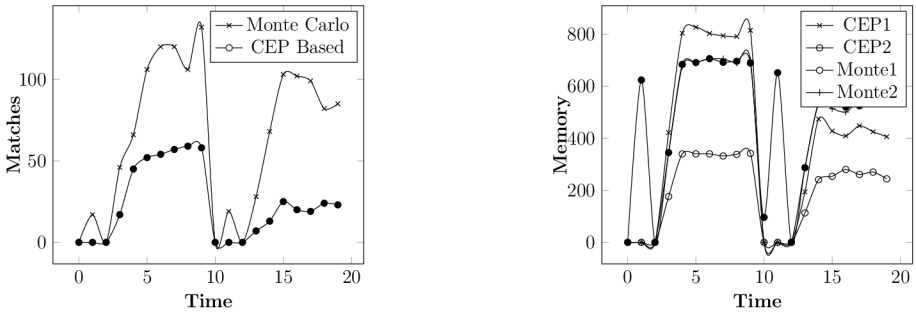


Figure 3.5: Test Bed

publicly available smart plug data from DEBS Grand Challenge 2014. The data sets are described later in this section.

Metrics used to evaluate the solution will be: performance and semantic power against current event processing platforms. The performance is more than just a non-functional property of CEP. Performance for CEP is more of a functional property, because it uncovers important events faster and at the same time when it is possible to react to them. Consequently, the reaction generates new events that would otherwise be unknown.



(a) Pattern Matches

(b) Memory Consumption

Figure 3.6: Monte Carlo vs CEP based Partitioning Algorithm

3.1.1 DEBS Grand Challenge 2014 Dataset

The assignment for this year's Grand Challange is currently available online [44]. The assignment and solutions to the challenge will be available after the this year's conference and will allow for scientific comparison to my method. A small part

of the data set (approx. 5GB, collected during the first 16 hours) is also available online [63]. The data are collected from real environment, from 40 houses in Germany. The data set is a text file where each line represents a measurement record. The structure of the data set is as follows:

- id
- timestamp
- value
- property
- plug_id
- household_id
- house_id

Each such record corresponds to low level sensor data from a smart plug. The *property* component of the record differentiates between two types of records: load, work. The *load* is current consumption measured by the smart plug, and the *work* is cumulative consumption. In reality the data may be corrupted (missing, peaking) because this data set is collected in a real environment. The final solution should deal with these problems.

The final data set was collected during the span of one month in 40 houses.

I plan not only to use the data set, but I will also implement the two queries required by the DEBS Grand Challange, so that my solution is comparable to the community solutions. There are two queries required by this Grand Challenge [44].

The query 1 is concerned with a load prediction. The task is to predict an average load in a time slice. The size of the time slice varies and the solver of the problem should provide predictions for each time resolution (a size of the time slice). The time resolutions are 1min, 15min, 60min, 120min. The solution should generate an output stream for each time resolution. The organizers are giving a baseline prediction model for predicting the average slot for the upcoming time slice. The

prediction formula is:

$$load_{t+1}(t) = (load_{t-1} + median(\{load_i | i \, mod \, daylength = (t+1) \, mod \, daylength\}))/2 \tag{3.1}$$

The result should generate two types of output streams. One for a household and one for individual devices. There will be 4 streams for each type (one for each time resolution). The format for events in the first output stream type is as follows:

- ts - The time stamp of the starting time of the slice for this prediction
- house_id - The id of the house
- predicted_load - The predicted average load in the time slice

The output stream for individual devices will generate:

- ts
- plug_id
- household_id
- house_id
- predicted_load

The query 2 is concerned with answering the question "How many outlier smart plugs are in any given household?". An outlier plug is a plug that has its median load in a time window greater than the global median load in the time window. Global median load is a median load among all the existing devices in the system. The solvers should answer this question by generating event streams (one for 1 hour window and one for 24 hour window), where each event is in the following format:

- ts_start - The time stamp of the start of the window
- ts_stop - The end of the window
- house_id

- percentage - The percentage of plugs in the house that are outliers

An event should be generated to an output stream when a percentage for the given house changes.

In my opinion, this creates an ideal opportunity to employ a PCEP solution to solve this problem by clustering the producers (smart plugs) into smaller clusters and measuring the median and outliers only in these small clusters.

3.1.2 Intelligent Buildings and Smart Meter Data

The Lasaris laboratory has several available data sets. Because the CEP research group shares similar goals, each individual project can reuse and compare solutions with each other.

The volume of the smart meter data available at the laboratory is 100GB. They are heavily masked, but it will still be possible to draw some conclusions after using this data set.

The data from intelligent buildings are currently stored in a relational database, and the volume is 40GB. The data contain various information from many sensors deployed throughout Masaryk University: security, temperature, and other measurements. These data contain events and measurements very similar to those found in the DEBS Grand Challange 2014. However, these data are still very raw and unconnected and therefore not very usable in initial research stages.

3.2 Time Plan

I am currently in the fourth semester of my doctoral studies. And I have approximately two years remaining.

I plan to dedicate the upcoming two semesters to finishing the prototype of the outlined extensions. Given that my theoretical basis for PCEP is largely done and published, I envision that at the end of my 3rd year, I should be able to focus solely on the comparison of my implementation with solutions from the DEBS Grand Challenge 2014. Simultaneously, I will have time to compare implementations that use Intelligent Building and smart meter data with my own implementation. This would leave me with one semester to finish writing my PhD thesis and,

possibly, some time would be left for deploying my implementation into an existent project as a monitoring solution - either to analyze computer network traffic or an environment that provides data from smart building sensors at Masaryk University.

4. Achieved Results

4.1 Complex Event Processing

I have published 3 papers in the area of Complex Event Processing. Two papers, [13] and [61], are concerned with the ideas outlined in this thesis proposal. I have also given a presentation for these papers at respective conferences and workshops. The third publication [64] discusses a possible approach to drive research in CEP today.

My contribution in all of these publications is ninety percent.

I am also a co-author of [58] with a contribution of ten percent.

4.2 Information Systems and Middleware

In the area of information systems and middleware, which is closely related to CEP, I have published three publications. In the current research [65, 66], I have been following state of the art notification systems.

My oldest publication [67] (seventy percent contribution) is a report on the implementation of a novel scalable notification system which was generating events. Software that accompanied the publication is currently in the production environment of the Takeplace [68] Information System.

The second publication, which I co-authored [69] (thirty percent contribution) for an international conference, evaluated state of the art methods for implementing a Social Network System. I also attended the conference and presented the paper.

The last publication [70] (forty percent contribution) was a journal contribution - an extension of the second publication. In this extended version I covered possible uses of CEP to increase the flexibility of a Social Network.

Bibliography

[1] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, New York, 2002.

[2] H. C. Randika, H. E. Martin, D. M R R Sampath, D. S. Metihakwala, K. Sarveswaren, and M. Wijekoon. Scalable fault tolerant architecture for complex event processing systems. In *2010 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 86–96, 2010.

[3] Kazuhiko Isoyama, Yuji Kobayashi, Tadashi Sato, Koji Kida, Makiko Yoshida, and Hiroki Tagato. A scalable complex event processing system and evaluations of its performance. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 123–126, New York, NY, USA, 2012. ACM.

[4] Shoaib Akram, Manolis Marazakis, and Angelos Bilas. Understanding and improving the cost of scaling distributed event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 290–301, New York, NY, USA, 2012. ACM.

[5] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 4:1–4:12, New York, NY, USA, 2009. ACM.

[6] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. The debs 2013 grand challenge. In *Proceedings of the 7th ACM International Conference*

on Distributed Event-based Systems, DEBS '13, pages 289–294, New York, NY, USA, 2013. ACM.

[7] Hans-Arno Jacobsen, Kianoosh Mokhtarian, Tilmann Rabl, Mohammad Sadoghi, Reza Sherafat Kazemzadeh, Young Yoon, and Kaiwen Zhang. Grand challenge: The bluebay soccer monitoring engine. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 295–300, New York, NY, USA, 2013. ACM.

[8] Kasper Grud Skat Madsen, Li Su, and Yongluan Zhou. Grand challenge: Mapreduce-style processing of fast sensor data. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 313–318, New York, NY, USA, 2013. ACM.

[9] Vikram Kumaran. Event stream database based architecture to detect network intrusion: (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 241–248, New York, NY, USA, 2013. ACM.

[10] Raphaël Barazzutti, Pascal Felber, Christof Fetzer, Emanuel Onica, Jean-François Pineau, Marcelo Pasin, Etienne Rivière, and Stefan Weigert. Streamhub: A massively parallel architecture for high-performance content-based publish/subscribe. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 63–74, New York, NY, USA, 2013. ACM.

[11] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, May 2003.

[12] Opher Etzion and Peter Niblett. *Event Processing in Action.* Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.

[13] Filip Nguyen, Daniel Tovarňák, and Tomáš Pitner. Semantically partitioned peer to peer complex event processing. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of *Studies in Computational Intelligence*, pages 55–65. Springer International Publishing, 2014.

[14] Z. Ivezic, J. A. Tyson, R. Allsman, J. Andrew, R. Angel, T. Axelrod, J. D. Barr, A. C. Becker, J. Becla, C. Beldica, R. D. Blandford, W. N. Brandt, J. S. Bullock, D. L. Burke, S. Chandrasekharan, S. Chesley, C. F. Claver, A. Connolly, K. H. Cook, A. Cooray, C. Cribbs, R. Cutri, G. Daues, F. Delgado, H. Ferguson, J. C. Geary, P. Gee, D. K. Gilmore, W. J. Gressler, C. Hogan, M. E. Huffer, S. H. Jacoby, B. Jain, J. G. Jernigan, R. L. Jones, M. Juric, S. M. Kahn, J. S. Kalirai, J. P. Kantor, D. Kirkby, L. Knox, V. L. Krabbendam, S. Krughoff, S. Kulkarni, R. Lambert, D. Levine, M. Liang, K. T. Lim, R. H. Lupton, P. Marshall, S. Marshall, M. May, M. Miller, D. J. Mills, D. G. Monet, D. R. Neill, M. Nordby, P. O'Connor, J. Oliver, S. S. Olivier, R. E. Owen, J. R. Peterson, C. E. Petry, F. Pierfederici, S. Pietrowicz, R. Pike, P. A. Pinto, R. Plante, V. Radeka, A. Rasmussen, W. Rosing, A. Saha, T. L. Schalk, R. H. Schindler, D. P. Schneider, G. Schumacher, J. Sebag, L. G. Seppala, I. Shipsey, N. Silvestri, J. A. Smith, R. C. Smith, M. A. Strauss, C. W. Stubbs, D. Sweeney, A. Szalay, J. J. Thaler, D. VandenBerk, M. Warner, B. Willman, D. Wittman, S. C. Wolff, W. M. Wood-Vasey, and H. Zhan. LSST: from Science Drivers to Reference Design and Anticipated Data Products. *ArXiv e-prints*, May 2008.

[15] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 147–160, New York, NY, USA, 2008. ACM.

[16] Souleiman Hasan, Sean O'Riain, and Edward Curry. Towards unified and native enrichment in event processing systems. In *Proceedings of the 7th*

ACM International Conference on Distributed Event-based Systems, DEBS '13, pages 171–182, New York, NY, USA, 2013. ACM.

[17] Alexander Artikis, Opher Etzion, Zohar Feldman, and Fabiana Fournier. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 32–43, New York, NY, USA, 2012. ACM.

[18] Mert Akdere, Jeong-Hyon Hwang, and Uğur Cetintemel. Real-time probabilistic data association over streams. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 219–230, New York, NY, USA, 2013. ACM.

[19] Yagil Engel and Opher Etzion. Towards proactive event-driven computing. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 125–136, New York, NY, USA, 2011. ACM.

[20] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Opportunistic spatio-temporal event processing for mobile situation awareness. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 195–206, New York, NY, USA, 2013. ACM.

[21] Ioannis Boutsis, Vana Kalogeraki, and Dimitrios Gunopulos. Efficient event detection by exploiting crowds. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 123–134, New York, NY, USA, 2013. ACM.

[22] Christoph Emmersberger and Florian Springer. Tutorial: Open source enterprise application integration - introducing the event processing capabilities of Apache Camel. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 259–268, New York, NY, USA, 2013. ACM.

[23] Adrian Paschke, Paul Vincent, Alex Alves, and Catherine Moxey. Tutorial on advanced design patterns in event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 324–334, New York, NY, USA, 2012. ACM.

[24] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, New York, 2002.

[25] Cagri Balkesen, Nihal Dindar, Matthias Wetter, and Nesime Tatbul. Rip: Run-based intra-query parallelism for scalable complex event processing. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 3–14, New York, NY, USA, 2013. ACM.

[26] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *SIGMOD Rec.*, 30(2):115–126, May 2001.

[27] Anthony Okorodudu, Leonidas Fegaras, and David Levine. A scalable and self-adapting notification framework. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part II*, DEXA'10, pages 452–461, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. Efficient content-based routing with network topology inference. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 51–62, New York, NY, USA, 2013. ACM.

[29] Yiannis Verginadis, Nikos Papageorgiou, Ioannis Patiniotakis, Dimitris Apostolou, and Gregoris Mentzas. A goal driven dynamic event subscription approach. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 81–84, New York, NY, USA, 2012. ACM.

[30] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, and Kurt Rothermel. Distributed spectral cluster management: A method for building dynamic publish/subscribe systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 213–224, New York, NY, USA, 2012. ACM.

[31] Muhammad Adnan Tariq, Gerald G. Koch, Boris Koldehofe, Imran Khan, and Kurt Rothermel. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part I*, EuroPar'10, pages 458–470, Berlin, Heidelberg, 2010. Springer-Verlag.

[32] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, Imran Khan, and Kurt Rothermel. Meeting subscriber-defined qos constraints in publish/-subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(17):2140–2153, 2011.

[33] Geetika T. Lakshmanan, Yuri G. Rabinovich, and Opher Etzion. A stratified approach for supporting high throughput event processing applications. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 5:1–5:12, New York, NY, USA, 2009. ACM.

[34] On-line grid monitoring based on distributed query processing. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics - Volume Part II*, PPAM'11, pages 131–140, Berlin, Heidelberg, 2012. Springer-Verlag.

[35] Michael Olson, Annie Liu, Matthew Faulkner, and K. Mani Chandy. Rapid detection of rare geospatial events: Earthquake warning applications. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 89–100, New York, NY, USA, 2011. ACM.

[36] Martin Hirzel. Partition and compose: Parallel complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 191–200, New York, NY, USA, 2012. ACM.

[37] S. Bianchi, P. Felber, and M.G. Potop-Butucaru. Stabilizing distributed r-trees for peer-to-peer content routing. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1175–1187, 2010.

[38] E. Casalicchio and F. Morabito. Distributed subscriptions clustering with limited knowledge sharing for content-based publish/subscribe systems. In *Sixth IEEE International Symposium on Network Computing and Applications, 2007. NCA 2007.*, pages 105–112, 2007.

[39] M. Guimaraes and L. Rodrigues. A genetic algorithm for multicast mapping in publish-subscribe systems. In *NCA 2003. Second IEEE International Symposium on Network Computing and Applications, 2003.*, pages 67–74, 2003.

[40] Pedro Bizarro, K. Mani Chandy, and Nenad Stojanovic. Event processing grand challenges. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 361–362, New York, NY, USA, 2011. ACM.

[41] Nenad Stojanovic. Debs challenge. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 369–370, New York, NY, USA, 2011. ACM.

[42] DEBS Grand Challenge 2011
3.1.2014 <`http://debs2011.fzi.de/index.php/challenge`>.

[43] Zbigniew Jerzak, Thomas Heinze, Matthias Fehr, Daniel Gröber, Raik Hartung, and Nenad Stojanovic. The debs 2012 grand challenge. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 393–398, New York, NY, USA, 2012. ACM.

[44] DEBS Grand Challenge 2014 Assignment
3.1.2014 <http://www.cse.iitb.ac.in/debs2014/?page_id=42>.

[45] Rapide event processing engine
3.1.2014 <http://complexevents.com/stanford/rapide/>.

[46] Drools Fusion Software
3.1.2014 <http://www.jboss.org/drools/drools-fusion.html>.

[47] Esper Engine
3.1.2014 <http://esper.codehaus.org/>.

[48] Storm
3.1.2014 <http://storm-project.net/>.

[49] IBM Infosphere Stream
3.1.2014
<http://www-03.ibm.com/software/products/en/
infosphere-streams/>.

[50] SangJeong Lee, Youngki Lee, Byoungjip Kim, Kasim Selçuk Candan, Yunseok Rhee, and Junehwa Song. High-performance composite event monitoring system supporting large numbers of queries and sources. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 137–148, New York, NY, USA, 2011. ACM.

[51] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 407–418, New York, NY, USA, 2006. ACM.

[52] Eric Bouillet, Ravi Kothari, Vibhore Kumar, Laurent Mignet, Senthil Nathan, Anand Ranganathan, Deepak S. Turaga, Octavian Udrea, and Olivier Verscheure. Processing 6 billion cdrs/day: From research to production (experience report). In *Proceedings of the 6th ACM International*

Conference on Distributed Event-Based Systems, DEBS '12, pages 264–267, New York, NY, USA, 2012. ACM.

[53] Yuanzhen Ji, Thomas Heinze, and Zbigniew Jerzak. Hugo: Real-time analysis of component interactions in high-tech manufacturing equipment (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 87–96, New York, NY, USA, 2013. ACM.

[54] Pekka Kaarela, Mika Varjola, Lucas P.J.J. Noldus, and Alexander Artikis. Pronto: Support for real-time decision making. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 11–14, New York, NY, USA, 2011. ACM.

[55] Bin Cao, Jianwei Yin, Shuiguang Deng, Yueshen Xu, Youneng Xiao, and Zhaohui Wu. A highly efficient cloud-based architecture for large-scale stb event processing: Industry article. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 314–323, New York, NY, USA, 2012. ACM.

[56] Vojtech Krmicek and Jan Vykopal. Netflow based network protection. In Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *Security and Privacy in Communication Networks*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 543–546. Springer Berlin Heidelberg, 2012.

[57] Pavel Minarik, Jan Vykopal, and Vojtech Krmicek. Improving host profiling with bidirectional flows. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 03*, CSE '09, pages 231–237, Washington, DC, USA, 2009. IEEE Computer Society.

[58] Daniel Tovarňák, Filip Nguyen, and Tomáš Pitner. Distributed event-driven model for intelligent monitoring of cloud datacenters. In Filip Zavoral, Ja-

son J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of *Studies in Computational Intelligence*, pages 87–92. Springer International Publishing, 2014.

[59] Cheng Xu, Daniel Wedlund, Martin Helgoson, and Tore Risch. Model-based validation of streaming data: (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 107–114, New York, NY, USA, 2013. ACM.

[60] Mauricio Arango. Mobile qos management using complex event processing: (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 115–122, New York, NY, USA, 2013. ACM.

[61] Filip Nguyen and Tomáš Pitner. Scaling CEP to infinity. In *9th Summer School of Applied Informatics*, Brno, Czech Republic, 2012. Masaryk University.

[62] David A. Wollman. Frameworks and data initiatives for smart grid and other cyber-physical systems (invited keynote). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 1–2, New York, NY, USA, 2013. ACM.

[63] DEBS Grand Challenge 2014, sample dataset
3.1.2014
<`https://docs.google.com/file/d/0B0TBL8JNn3JgV29HZWhSSVREQ0E/edit`>.

[64] Filip Nguyen and Tomáš Pitner. Information system monitoring and notifications using complex event processing. In *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, pages 211–216, New York, NY, USA, 2012. ACM.

[65] Kyuchang Kang, Jeunwoo Lee, and Hoon Choi. Instant notification service for ubiquitous personal care in healthcare application. In *International Con-*

ference on Convergence Information Technology, 2007., pages 1500–1503, 2007.

[66] C. Schmandt, N. Marmasse, S. Marti, N. Sawhney, and S. Wheeler. Everywhere messaging. *IBM Systems Journal*, 39(3.4):660–677, 2000.

[67] Filip Nguyen and Jaroslav Škrabálek. Notx service oriented multi-platform notification system. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 313–316, 2011.

[68] Takeplace
3.1.2014 <`http://www.take-place.com/en`>.

[69] Jaroslav Škrabálek, Petr Kunc, Filip Nguyen, and Tomáš Pitner. Towards effective social network system implementation. In Mykola Pechenizkiy and Marek Wojciechowski, editors, *New Trends in Databases and Information Systems*, volume 185 of *Advances in Intelligent Systems and Computing*, pages 327–336. Springer Berlin Heidelberg, 2013.

[70] Jaroslav Škrabálek, Petr Kunc, Filip Nguyen, and Tomáš Pitner. Towards effective social network system implementation. *Control and Cybernetics*, 41/2012, 2013.

[71] Daniel Tovarnak and Tomas Pitner. Towards multi-tenant and interoperable monitoring of virtual machines in cloud. In *Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '12, pages 436–442, Washington, DC, USA, 2012. IEEE Computer Society.

[72] NotX Notification System
3.1.2014 <`http://sourceforge.net/projects/notx/`>.

A. List of Results

A.1 List of Papers

- Filip Nguyen and Tomáš Pitner. Information system monitoring and notifications using complex event processing. In *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, pages 211–216, New York, NY, USA, 2012. ACM.
- Filip Nguyen and Tomáš Pitner. Scaling CEP to infinity. In *9th Summer School of Applied Informatics*, Brno, Czech Republic, 2012. Masaryk University.
- Filip Nguyen, Daniel Tovarňák, and Tomáš Pitner. Semantically partitioned peer to peer complex event processing. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of Studies in Computational Intelligence, pages 55–65. Springer International Publishing, 2014.
- Filip Nguyen and Jaroslav Škrabálek. Notx service oriented multi-platform notification system. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 313–316, 2011.
- Jaroslav Škrabálek, Petr Kunc, Filip Nguyen, and Tomáš Pitner. Towards effective social network system implementation. In Mykola Pechenizkiy and Marek Wojciechowski, editors, *New Trends in Databases and Information Systems, volume 185 of Advances in Intelligent Systems and Computing*, pages 327–336. Springer Berlin Heidelberg, 2013.
- Jaroslav Škrabálek, Petr Kunc, Filip Nguyen, and Tomáš Pitner. Towards effective social network system implementation. *Control and Cybernetics*,

41/2012, 2013.

- Daniel Tovarňák, Filip Nguyen, and Tomáš Pitner. Distributed event-driven model for intelligent monitoring of cloud datacenters. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII, volume 511 of Studies in Computational Intelligence*, pages 87–92. Springer International Publishing, 2014.

A.2 Presentations

- 2012 Advances in Databases and Information Systems September 17-20, 2012: Poznan, Poland
- 2012 Bedrichov, 9th Summer School of Informatics
- 2013 7th International Symposium on Intelligent Distributed Computing - IDC'2013 IDC Prague

A.3 Teaching

- PA165 Enterprise Applications in Java - being a tutor for 2nd consecutive year
- PB138 Modern Markup Languages and Their Applications - being a tutor for 2nd consecutive year
- PV226 Seminar LaSArIS - I did two presentations in two different semesters

A.4 Thesis Supervision

A.4.1 Bachelor Thesis Supervision

Defended thesis:

- Bc. Štefan Repček *CEP Portal for Simulation*
- Bc. Jan Brázdil *Automatic Pull Request Integration*
- Bc. Pavel Dedík *Graphical User Interface for Thesis Management System*
- Bc. Roman Jakubčo *Configuration Conversion of JBoss AS5 to JBoss AS7*

Already submitted thesis, awaiting defense:

- Radek Koubský *JBoss Teiid konektor pro NoSQL databázi Apache Cassandra*
- Jakub Senko *Effective Testing of Local Git Branches Using Remote Execution*
- Tomáš Skopal *Usage of Analytical Patterns for Development of Small Business Application*

A.4.2 Diploma Thesis Consultant

These theses were defended, with me as a consultant:

- Mgr. Andrej Vaňo *Reactive Resource Management in Red Hat lab*

I have been in a position of a consultant on, still undefended, thesis:

- Bc. Radim Hopp *Relational Access to Amazon SimpleDB*
- Bc. Monika Gottvaldová *Modern open source Java EE-based process and issue tracker*

A.5 Full Papers

A.5.1 IDC 2013 [13]

Semantically Partitioned Peer to Peer Complex Event Processing

Filip Nguyen, Daniel Tovarňák and Tomáš Pitner

Abstract Scaling Complex Event Processing applications is inherently problematic. Many state of the art techniques for scaling use filtering on producers, vertical scaling, or stratification of an Event Processing Network. The solutions usually aren't distributed and require centralized coordination. In this paper, we are introducing a technique for scaling Complex Event Processing in a distributed fashion and by taking semantic information of events into account. We are introducing two CEP models for scaling CEP architectures, providing core algorithms, and evaluating their performance.

1 Introduction

In this paper, we are concerned with a new way of scaling Complex Event Processing (CEP) applications. This section introduces CEP, presents the motivation for our work, and introduces our contribution.

1.1 Complex Event Processing

CEP is both a theoretical and a practical research area that studies events and event processing in current computing systems and businesses. Examples of such events may be:

- A payment using a credit card. This may be regarded as relatively infrequent event.
- A barcode reading of a product code. This may be regarded as a frequent event.
- A motion sensor on electronic doors to a supermarket.

All of these events may be correlated together and may thusly cross both technological and domain boundaries. The process of correlating is referred to as *pattern matching*. Such a pattern might be: two payments made by the same credit card in different supermarkets within a time frame of 1 hour.

Another possible view of Complex Event Processing is an upside down version of standard data processing. Instead of data sitting in a database, waiting for queries to be submitted, CEP may be viewed as queries sitting and waiting for data to be

Filip Nguyen e-mail: `xnguyen@fi.muni.cz` · Daniel Tovarňák e-mail: `xtovarn@fi.muni.cz` · Tomáš Pitner e-mail: `tomp@fi.muni.cz`
Masaryk University Faculty of Informatics, Botanická 68a, Brno, Czech Republic

submitted. That is why CEP is used as a monitoring tool ([5], [6]). An interesting method of using CEP as a monitoring was introduced in [8], where it is usable even for data mining.

Motivation for CEP nicely correlates with the current advent of Big Data, which is a data centric approach to extracting meaningful data. The problem here is not to store such data, but to retrieve it and to extract meaningful information [2]. CEP is able to carry out extractions of data regarding time, a frequent component in data. In CEP, data that is not processed is simply discarded.

CEP was motivated and introduced in a very comprehensive publication [4]. The book provides a basic framework for many CEP usages, including dynamic complex event processing. The book introduces the so called *Event Processing Agent* (EPA). EPA is defined as an object that monitors events to detect certain patterns in them. Another abstraction is then introduced in the form of an *Event Processing Network* (EPN) which connects many EPAs into a network where each EPA sends events to another EPA. EPNs are added to the model for flexibility and allow for the easy reuse of EPAs and for the building of dynamic EPNs. In these dynamic EPNs, processing agents may enter or leave the network at the time of processing.

1.2 Peer Complex Event Processing

We exploit the fact that events from certain producers are related to each other in some way. We do this by breaking up the event space by identifying related producers using CEP itself. We then exploit this information and put processing only where it is most needed. For example, knowing that two retail stores were visited by the same customer in the last two days signifies some connection between the two retail stores. With information that the producers are related, we are able to add event processing between these two retail stores. This event processing between these two stores may consider much more fine grained events. By using this approach we may lose some information because fine grained analyses are not done everywhere. A similar probabilistic view of CEP was also discussed in [7] using an example of a hypothetical surveillance system that is monitored using CEP.

To show how we help scale CEP, we are introducing two graph oriented event processing abstractions: *peer model* and *centralized model*. In both models, we view a graph of CEP engines and producers as a basic building block for creating scalable event processing architecture.

To uncover possible correlations between producers, we use *partitioning algorithms*. These two algorithms help to identify which producers should have the engine among them (using the centralized model). We later also map this centralized approach to the peer model

We have developed two partitioning algorithms. Their properties are discussed and they are studied on an experimental basis in a simulated CEP environment.

2 State of the Art

In this chapter we are introducing current models of Complex Event Processing that relate to our research. The word ”distributed” is being attached to CEP in many publications, but its meaning varies. In [4] the interpretation is: the processing of heterogeneous events, generated from various sources in an enterprise environment. In this environment, events are generated very often and they travel via various channels (a private computer network, the Internet) to be processed.

There are several approaches to scaling CEP. A straightforward way is by analyzing bottlenecks in current CEP systems. The authors of [10] did extensive profiling of the Borealis CEP system and identified that communication is one of the biggest bottlenecks. Another approach was introduced in [11] where traditional query planning was applied to CEP and optimizations. Sliding window push downs were successfully applied. Scalability via hierarchy was introduced in [8].

Stratification is a particularly interesting way of scaling CEP. It was introduced in [13]. Stratification significantly improves event processing capabilities by parallelizing certain matching operations. The input for a stratification algorithm is EPN with dependency information. Dependency information states which EPA depends on the input of another EPA. Using a simple algorithm, the authors were able to cleanly separate EPAs into sets called strata, which contain agents that can run in parallel.

In this paper, we understand distributed CEP as distributed collection, distributed processing and most importantly, distributed matching of events.

3 Peer Complex Event Processing

In this section we will introduce our CEP models. We will also discuss the *event space correlation problem* as well as our approach to solve it. Our notation has two parts: graphical representation and query language.

First we will look at graphical representation. Figure 1 shows the basic building blocks of our model. The black node represents the centralized engine that accepts events. The producers of the events are represented by red nodes with light rings. The producers only produce events and do nothing else. An example of a producer is a server computer in a supermarket that produces barcode reading events and credit card payment events. Red nodes with bold rings represent peers. A peer is a CEP engine that is placed directly on a producer and replaces the centralized engine in our model. Our notation further includes an event which may be augmented with time information pertaining to its creation as well as with the name of the producer.

Fig. 1: CEP Graph Model

In our model, we denote the set of producers, event engines, and peers using capital P, e.g. $P_1, P_2, ..., P_N$. Set of events is denoted E and discrete events may be denoted by syntax $name : time : producer$. Events are described using two functions: $time : E \rightarrow N$ for the time when the E originated and $producer : E \rightarrow P$ to get the producer of the event. The producers may be connected in an undirected graph and a producer may send some of his events to his neighbor. The function $events : P \times P \rightarrow 2^E$ is used to retrieve a set of events that have traveled between the two producers. The event engine is a node in our graph that, upon reception of an event (E_0 signifies incoming event), tries to use this event to match a pattern. Patterns MP are represented by a simple SQL like syntax, e.g. WHERE E_0=E_1 WINDOW(10). The WINDOW part of SQL syntax denotes the time frame in which the matching takes place.

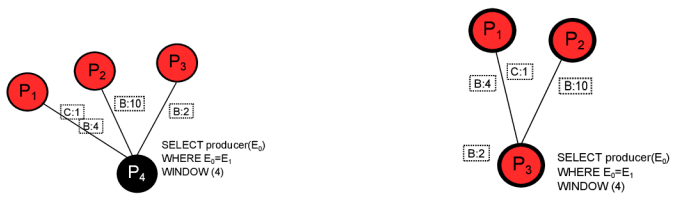
Function $patterns : P \rightarrow MP$ returns subset of patterns running on the peer.

Function $matches : MP \times N \rightarrow 2^E$ returns matched set of events matched by a pattern since specified time.

There are two possible ways to model an event processing situation using these building blocks: peer model and centralized model. Centralized model uses centralized engines and producers. In this case, events flow from the producers to the engines. Figure 2a shows a simple CEP network where producers P_1, P_2, P_3 send their events to P_4. The P_4 in this example contains only one matching rule. This rule fires only once when event $B : 4$ arrives to P_4. The $B : 10$ doesn't match the rule because the sliding WINDOW is only 4 time units in this case.

The second possible way to model an event processing situation is *peer model*. The basic building block is peer: both a producer and an event processing engine. In this peer model, a graph is formed only by peers and it is possible to achieve the same matching capabilities as with the centralized model. In Figure 2b, you can see an example of peer model notation. P_3 works both as the producer and the engine that works instead of P_4. Note that P_3 also produces events back to itself.

It is apparent that these two models are interchangeable to a certain degree. Using the centralized model is better when explaining the concepts. On the other hand, the peer model resembles our targeted implementation more as well as introduces additional advantages.

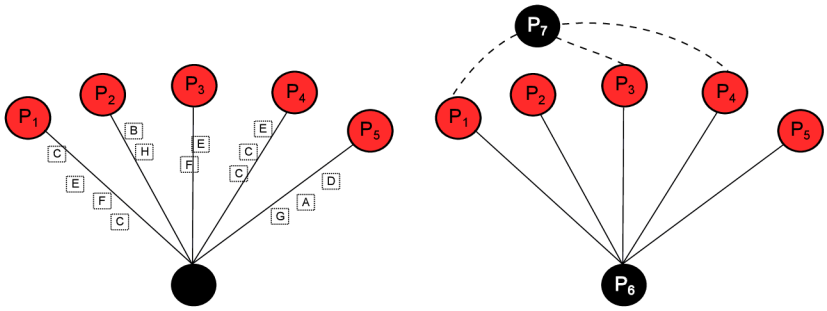


(a) Simple centralized CEP (b) Simple Peer CEP

Fig. 2: CEP Models Comparison

3.1 Event Space Correlation Problem

Suppose we are building a CEP solution for 5 producers which are producing events. Figure 3a shows such a situation. Suppose we want the engine to answer to the following query: SELECT $producer(E_0)$, $producer(E_1)$ WHERE $E_0 = E_1$ WINDOW(10).



(a) Centralized CEP (b) Dynamic CEP deployment

Fig. 3: CEP Models Comparison

Consider the technological implementation of P_6. To be able to match the equality, the CEP engine needs to accumulate all of the events in the time frame and continually analyze whether a new event is correlating with any of the other events thus far accumulated. Because all of the events are related to each other, no stratification is possible here. This problem is common among CEP engines and was also discussed in [8], where the authors used rule allocation algorithms in their distributed CEP model. They considered a structure of rules to circumvent the problem. A different approach to solving a special case of this problem is to design a special data structure to hold shared events [9].

We have exploited the observation that producers are related. In the given example on Figure 3a, we can see that producers P_1, P_2, P_4 are correlating together more than the other producers. The fact that these 3 are related in some way can be exploited by placing a new P_7 engine between them. This engine will have to consider a lower number of events and thus its performance will also be better.

Methods for detecting some related producers are discussed further in this paper. We call these methods *partitioning algorithms*.

In this section we discussed the dynamic placement of engines between producers and we made use of a classic centralized CEP model. This facilitates easy understanding, but it is important to note how this can be reinterpreted into our peer model. In the peer model, we do not deploy additional engines, but we only connect the appropriate engines and assign roles. Figure 4 shows an example of peer model. The peer P_5 in this situation assumes the role of P_6 from the previous example. When our algorithms decide that P_1, P_2, P_3 are related, the three peers connect (dashed line) and P_3 takes responsibility of correlating events between them.

Let us portray the example in real world scenarios.

One example might be the detection of network attacks [14]. Computers and network elements are the producers of network logging information. It is possible, by monitoring coarse grained events in a network, to find out that some limited set of computers is vulnerable to attack. Using this knowledge, we can deploy an additional engine among the limited number of producers and process more fine grained events.

Another example would be CEP engine used for monotoring public transportation [15]. Sensors in vehicles are producers. By detecting the coarse grained event of "user comfort by temperature", we can detect that some vehicles are good candidates to be closely analyzed and we can deploy an additional engine in the vehicle computer itself and analyze more fine grained events (speed changes, vehicle technical data, time table precision of the driver).

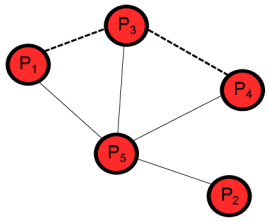


Fig. 4: Peer Model

The peer architecture of Complex Event Processing has some obvious downsides:

- In the real world, it may be challenging for a producer to be able to see every other producer, e.g. every supermarket server may not have access to every other supermarket server.
- Some event patterns might not be detected when the partitioning algorithm doesn't connect the correct producers.

The advantages of this architecture are mainly:

- The previously mentioned scaling capabilities. It is easy to dynamically connect several nodes and thusly begin processing among them.
- Because producers feature an event processing engine, it is possible to easily filter events.
- Each added producer will improve the power of the whole system because of the addition of another processing engine.

3.2 Partitioning Algorithms

We are introducing and comparing two partitioning algorithms: the CEP based algorithm and the Monte Carlo Partitioning Algorithm. The algorithms dynamically add/remove CEP engines among peers to conduct pattern matching.

We have used **wait** in the algorithms, which signifies waiting either for a specific guard or to wait for a specified time. While this wait is in progress, matching continues in the CEP environment. This waiting happens only to the partitioning algorithm. Note that the algorithms are shown in a pseudo code style for clarity.

The CEP based algorithm uses complex event processing of coarse grained events (less frequent, not consuming too many resources) to later deploy more fine grained events. The present matching patterns are configurable by the user and should be prepared for a specific purpose. The algorithm uses *centralized model*.

INPUT: COARSE GRAINED MP_1. FINE GRAINED MP_2, MP_3. CEP GRAPH $G(P, channels)$

2. $channels := P_c \times P$; $patterns(P_c) = \{MP_1\}$; T:=0;
3. **WAIT** UNTIL $|\{p|\exists e \in matches(MP_1, T).p \in producer(e)\}| >= |P|/2$
   $X := \{p|\exists e \in matches(MP_1, T).p \in producer(e)\}$
   T:= CURRENT TIME
   $channels := channels \cup \{(x, P_{left})|x \in P\} \cup \{(x, P_{right})|x \in P/X\}$
   $patterns(P_{left}) := \{MP_2\}$
   $patterns(P_{right}) := \{MP_3\}$
4. SAME WAIT AS IN (3)
   IF $\{p|\exists e \in matches(MP_1, T).p \in producer(e)\}$ DIFFER SIGNIFICANTLY FROM X
   GOTO (2)
   ELSE **GOTO** (4)

The Monte Carlo algorithm uses a centralized engine at the beginning. It derives the interesting producers and correlates with the current set of matching rules. It then divides the event space. A big advantage of this algorithm is that it doesn't need any input from the the CEP user. It works with matching patterns which have already been provided for the centralized version of CEP. The algorithm uses *centralized model*.

INPUT: EXISTING MATCHING PATTERNS MP. T:=0. CEP GRAPH $G(P, channels)$

1. RANDOMLY SELECT $MP_{small} \subset MP$;
2. $channels := P_c \times P$; $patterns(P_c) = MP_{small}$; T:=0
3. **WAIT** DEFINED TIME. THEN $\forall x \in matches(MP_{small}, T)$
   $STAT[producer(x)]$**+=**1
   LET SH, SL BE SET OF PRODUCERS. $|SH| - |SL| < 1 \wedge \forall x \in SH, \forall y \in SL.x >= y$
   $channels := channels \cup \{(x, P_{left})|x \in SH\} \cup \{(x, P_{right})|x \in SL\}$
   $patterns(P_{left}) := MP$; $patterns(P_{right}) := MP$
4. T:= CURRENT TIME; CONSTRUCT STAT2 SAME WAY AS IN (2)
   IF STAT2 FROM STAT DIFFERS SIGNIFICANTLY
   GOTO (2)
   ELSE **GOTO** (4)

3.3 Evaluation

We have implemented and measured these two algorithms in a simulated event processing environment. This environment consisted of 100 event producers, each pro-

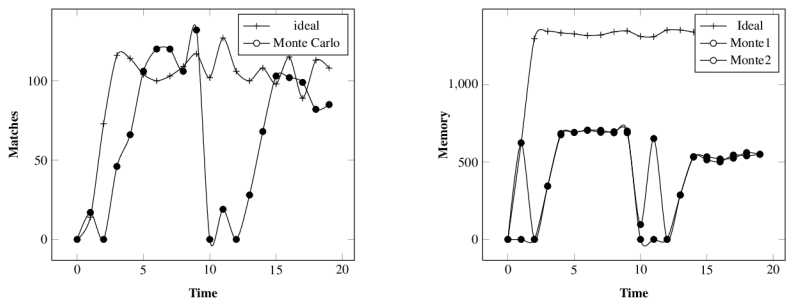
ducing up to 10 000 event types at random times from a non-uniform distribution. The experiment was divided into two time periods.

In the first period, a random 20% of the producers were related. They produced only 1 000 event types.

In the second time period, a different 20% of producers were chosen to be related.

As we can see from Figure 5a, the Monte Carlo is even better than ideal pattern matching at a point. This is due to the fact that the two engines operating in the parallel display had an overall better performance than the one overloaded ideal engine - from time to time.

From Figure 7, we can see that both algorithms drop in matching performance when different events become related. However, the Monte Carlo is better in overall matching capabilities. On the other hand, the CEP based algorithm provides better overall memory characteristics because it selects smaller event spaces to consider, and it also uses coarse grained events to decide where to put fine grained engines. This operation doesn't consume much memory. The Monte Carlo, on the other hand, has to work as a centralized and memory overloaded engine.

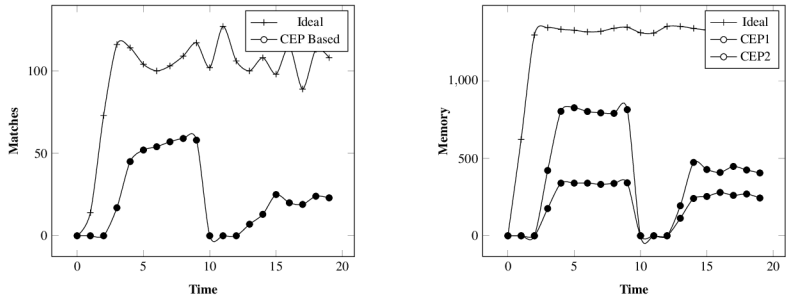


(a) Pattern Matches (b) Memory Consumption

Fig. 5: Ideal Engine vs Monte Carlo

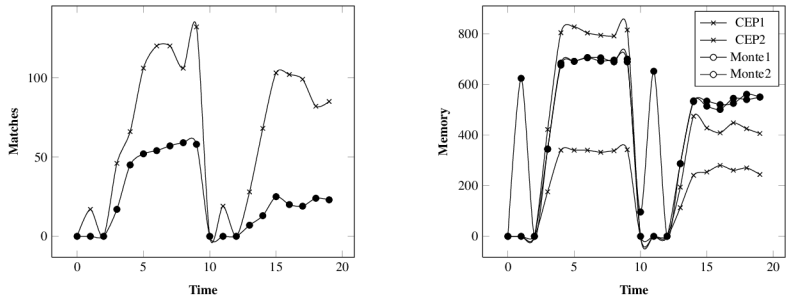
4 Conclusion and Future Work

In this paper, we have introduced a simplified view of Complex Event Processing called *peer model*. Peer model aims to simplify the way of looking at CEP to easily introduce conceptual scaling optimizations. With this model, it is easier to think about distributed event collection and processing. We have also described practical mappings of this model in real world situations. Further, we have identified *the event space correlation problem* that stops current CEP models from scaling conceptually. We have introduced two algorithms and evaluated their performances in a simulated environment. Experimental results show that the Monte Carlo partitioning algorithm performs better with regards to matching capabilities, but suffers from memory consumption. The CEP based algorithm provides an overall better memory performance



(a) Pattern Matches (b) Memory Consumption

Fig. 6: Ideal Engine vs CEP based Partitioning Algorithm



(a) Pattern Matches (b) Memory Consumption

Fig. 7: Monte Carlo vs CEP based Partitioning Algorithm

and is easier to deploy into high volume environments, but has significantly lower matching capabilities.

In the future, we plan to extend the peer model from a query semantics perspective. We will introduce distributed algorithms that will allow the deployment of CEP rules into the peer model and also allow for the collection of matching results from it. Additional studies of partitioning algorithms are also needed. We hope that the CEP based algorithm may be improved to give better matching performance. All of these efforts are aimed to create an Open Source platform Peer CEP that is written in the Java Programming language. The simulator which was used to conduct experiments for this paper will also be part of the suite - for testing and research needs.

References

1. Bin Cao, Jianwei Yin, Shuiguang Deng, Yueshen Xu, Youneng Xiao, and Zhaohui Wu. 2012. A highly efficient cloud-based architecture for large-scale STB event processing: industry article. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12). ACM, New York, NY, USA, 314-323.
2. Adam Jacobs. 2009. The pathologies of big data. In Communications of the ACM - A Blind Person's Interaction with Technology CACM Homepage archive. Volume 52 Issue 8, Pages 36-44
3. Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. 21, 2
4. David Luckham. 2002. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. 376 pages. Addison-Wesley Professional, New York.
5. Daniel Tovarňák. 2012. Towards Multi-Tenant and Interoperable Monitoring of Virtual Machines in Cloud. In 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Timisoara, Romania
6. Filip Nguyen, Tomáš Pitner. 2012. Information System Monitoring and Notifications Using Complex Event Processing. In Proceedings of the Fifth Balkan Conference in Informatics. Serbia : ACM. ISBN 978-1-4503-1240-0, pp. 211-216. Novi Sad, Serbia.
7. Alexander Artikis, Opher Etzion, Zohar Feldman, and Fabiana Fournier. 2012. Event processing under uncertainty. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12). ACM, New York, NY, USA, 32-43.
8. Kazuhiko Isoyama, Yuji Kobayashi, Tadashi Sato, Koji Kida, Makiko Yoshida, and Hiroki Tagato. 2012. A scalable complex event processing system and evaluations of its performance. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12). ACM, New York, NY, USA
9. SangJeong Lee, Youngki Lee, Byoungjip Kim, Kasim Seluk Candan, Yunseok Rhee, and Junehwa Song. 2011. High-performance composite event monitoring system supporting large numbers of queries and sources. In Proceedings of the 5th ACM international conference on Distributed event-based system (DEBS '11). ACM, New York, NY, USA, 137-148.
10. Shoaib Akram, Manolis Marazakis, and Angelos Bilas. 2012. Understanding and improving the cost of scaling distributed event processing. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12). ACM, New York, NY, USA, 290-301.
11. Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-performance complex event processing over streams. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data (SIGMOD '06). ACM, New York, NY, USA, 407-418.
12. Randika, H. C., Martin, H. E., Sampath, D. M. R. R., Metihakwala, D. S., Sarveswaren, K., Wijekoon, M. 2010. Scalable fault tolerant architecture for complex event processing systems. In International Conference on Advances in ICT for Emerging Regions (ICTer). Colombo, Sri Lanka.
13. Ayelet Biger, Opher Etzion, Yuri Rabinovich. 2008. Stratified implementation of event processing network. In 2nd International Conference on Distributed Event-Based Systems. Fast Abstract. Rome, Italy.
14. Pavel Minařík, Jan Vykopal, Vojtěch, Krmíček. 2009. Improving Host Profiling with Bidirectional Flows. In Computational Science and Engineering. CSE '09. International Conference on, On page(s): 231 - 237 Volume: 3, 29-31 Aug.
15. Pekka Kaarela, Mika Varjola, Lucas P.J.J. Noldus, Alexander Artikis. 2011. PRONTO: support for real-time decision making. In Proceedings of the 5th ACM international conference on Distributed event-based system (DEBS '11). ACM, New York, NY, USA, 11-14.

A.5.2 Scaling CEP to Infinity

Paper [61]

Scaling CEP to Infinity

Filip Nguyen, Tomáš Pitner

Masaryk University, Faculty of Informatics, Lab Software Architectures and IS
Botanická 68a,
602 00 Brno, Czech Republic
{fnguyen,tomp}@fi.muni.cz
http://lasaris.fi.muni.cz

Abstract. Scaling CEP applications is inherently problematic. In this paper we are introduce solution for scaling CEP applications that is fully distributed and aspires to scale CEP to the limits of current hardware. Our solution simplifies existent Event Processing Network abstraction and adds features on the level of CEP that change direction of its usage.

Abstrakt. Škálování CEP aplikací je ze své podstaty problematické. Tento článek předvádí řešení pro škálování CEP aplikací, které je plně distribuované a aspiruje na škálování CEPu až k limitům současného hardwarového vybavení. Naše řešení zjednodušuje existující abstrakci událostních sítí a přidává nové vlastnosti na úrovni CEPu, které mění směr jeho použití.

Keywords: CEP, scalability

Klíčová slova: CEP, škálovatelnost

1 Introduction

Complex Event Processing (CEP) brings real-time processing of massive amounts of events with aggregation capabilities. This aggregation enables to correlate between events in real-time over large sliding windows. This is crucial capability of today high-end CEP technology and theory. Idea of correlating large amount of events in single window raises interesting questions about scalability of any implementation of such idea. Is it possible to scale such processing? Do it in distributed fashion? These questions are hard to answer. Our results show that move from centralized CEP to distributed model requires changes not only in technology. The move would require to evaluate basic CEP assumptions and workflow of the developer who leverages CEP.

Fig 1 illustrates simple CEP application. One might argue that CEP application uses so called Event Processing Network (EPN) thus is distributed in some sense. This is not true, because EPN behaves towards producers as monolithic engine and its behavior is static.

adfa, p. 1, 2011.
© Springer-Verlag Berlin Heidelberg 2011

In this paper we revisit the problem of scaling context aware CEP. Our approach is based on creating peer-to-peer model of processing agents without centralized coordination.

2 Complex Event Processing

Complex Event Processing (CEP) is computing abstraction for working with events and streams of events. There is a big body of literature about the topic [13][7]. The research areas in field of CEP is very wide and encompass theoretical, business oriented, technology oriented topics. In our research we are concerned with scalable pattern matching over event streams, which is a specific area of CEP. We believe this area is also the most important in whole CEP. Simpler scenarios that are concerned with filtering of stateless channels with single events can be relatively easy handled in proprietary fashion. In these simple cases CEP helps mostly by giving common terminology.

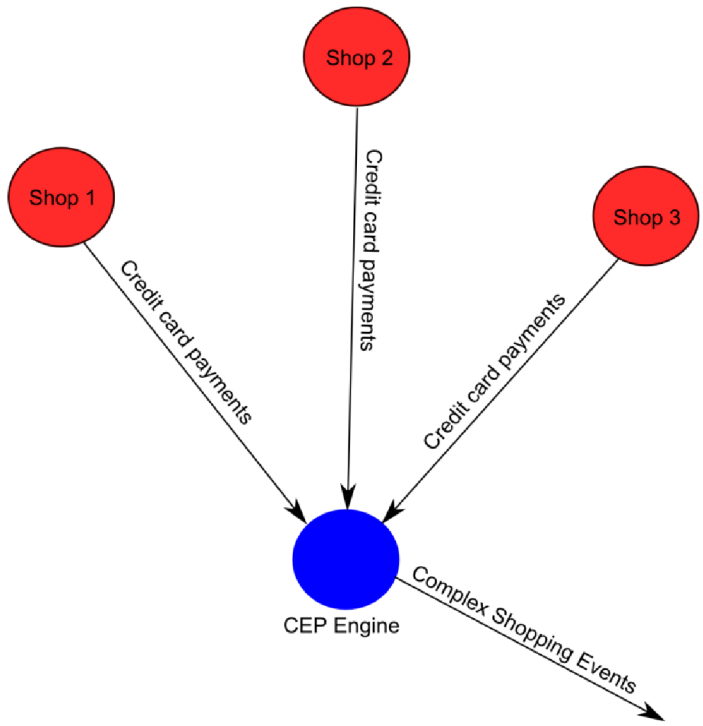


Fig. 1.

It is appropriate to begin with simple example, using terminology that is mostly guided by intuition. We want to give basic understanding of CEP concepts. Simple example of pattern matching over event streams is depicted in Fig 1. Red circles represent the *event producers* and blue circle represents *CEP engine.* The event producers in our example are retail business shops that produce events representing *payment by*

credit card. We view any such possible event e as variable of the first order logic. Accompanying predicate symbol $twindow(x, y)$ is designating two events *x,y* that happen in same time window (time window size etc. is part of the query). Function $cardnum(e)$ is returning card number and $shop(e)$ has information about the shop in which the purchase was done.

The events are entering CEP engine. The engine is able to detect specific relation between events. Intuitively but more precisely: it can select subset of events such that for the set, following logic sentence is true $\forall x, y(twindow(x, y) \wedge cardnum(x) = cardnum(y) \wedge !(shop(x) = shop(y)))$. In example on Fig. 1 such pattern will be selecting payments *x,y* which are done by 1 credit card with card number *cardnum(x)* in different stores. Whenever such subset is identified on the stream of data, new events are generated (*Complex Shopping Events*). This concrete pattern matching have clear business oriented motivations:

- stores for A and B might be in geographical proximity
- stores for A and B might have common products for specific type of customer

To extract this kind of information is relatively easy even with existing tools (relational databases, data mining methods). So why CEP? What CEP brings is stream processing. The data are never stored, they are processed "on the go". IN terms of first-order logic pattern matching model we say that the latest subsets matched by pattern is considered in current moment. Lateness of the set is defined by the timestamp of the latest event in that set. Expressing patterns in first order logic formula frees ourselves in expression power, but for CEP there are (similarly to SQL) vast amount of language dialects that declaratively define the event patterns. All CEP dialects have one thing in common. They incorporate time of the event occurrence as first class citizen, together timewindow abstraction and with many operators that help understanding the time correlation between events, e.g.:

- event A happened before event B
- event A happened while B

Some CEP dialects even include spatial pattern recognition.

Real time access to processed data is one of the main advantages for Complex Event Processing [10]. This advantage is even strengthen if the data are mostly relevant in current moment.

2.1 Technological Tools

CEP is enabled by many software tools today. The main Open Source tool is Esper for Java and .NET. Esper is centralized CEP engine. Main advantage of it is object oriented nature and embedability into Java process. The patterns in Esper takes form of SQL-like declarative rules that are given to the engine in the form of uncompiled String, e.g.:

```
String epl = "select avg(price) from
OrderEvent.win:time(30 sec)";
EPStatement statement =
epService.getEPAdministrator().createEPL(epl);
```

IBM Active Middleware (AMiT) is another successful software suite used for CEP [21]. AMiT architecture is closer to theoretical model in current CEP literature. It eclipse based IDE with possibility to construct Event Processing Network. Event patterns in AMiT takes form of logical formulae similar to this:

```
If transaction.type="cash_check" and
transaction.amount>=transaction.parameter_check_threshold
```

2.2 Scaling Complex Event Processing

Scaling of CEP is regarded as nonfunctional property of CEP. We are concerned with horizontal scalability in the following categories:

- Volume of processed events
- Quantity of agents
- Quantity of producers
- Quantity of context partitions
- Availability

The volume of processed events is most important criterion for sliding window correlation. With any complex pattern, it is immediately clear that CEP engine has to correlate between all the events in the time window thus have them in the memory.

Scaling in quantity of agents is naturally given by our method because our assumption is that event producers ultimately become CEP engines.

Quantity of producers is also scaled for free by our method, because we leverage peer-to-peer model where each new producer is regarded as yet another building block of CEP application.

Quantity of context partitions is tricky scaling criterion. However, after careful look, it lies in the heart of our method. Partitioning contexts of events is the key to distribute load to infinity.

Current approaches to scalability can be categorized as follows:

1. Optimizations related to EPA assignment
2. Optimizations related to coding of specific EPA
3. Optimizations related to execution process

We believe the (1) is most important and promising area of getting true horizontal scalability for CEP applications. In this area several approaches already took place:

- Stratification
- Peer-to-peer scaling

- Vertical scaling ([2])

Current approaches to scale Complex Event Processing using stratification were studied in [6]. By stratifying the CEP architecture, it is possible to distribute load of events among more than one engine. This stratification is static way of scaling CEP. The input Event Processing Network (EPN) is stratified by algorithm into so called strata. This method mainly benefits from the fact that some event processing agents work as filters thus are independent of the context and other agents (which are put into different strata). High throughoutput is achieved using this method, but it is limited by static nature.

Another way to distribute CEP is to push event processing to producers which was already studied in [9][16]. We believe this to be very promising direction. In [4] distributed query engine is motivated by peer-to-peer file sharing. Authors identify peer-to-peer approach to be very well suited for monitoring scenarios and intrusion detection. Relaxation of design principles is identified as main way to bring good distribution of query engine.

In [5] authors identify self-organization capabilities as central requirement for large-scale distributed applications. However, CEP theory and even the implementation doesn't encompass this ideas.

3 Research Problem

CEP engines help to handle massive amounts of events. However, the inherent limitation of the idea behind pattern matching are that correlation between events must happen in centralized fashion. This paradox leads to discussion about how to scale CEP. In our example, as number of stores connected to our engine grows, one might deploy more intelligent approaches to handle number of events. In this paper we are concerned how to scale CEP to the limit of existing hardware.

In previous section we have identified current approaches to scaling. The problem with current approaches is that they still understand centralization of CEP processing and use static approaches to grow query engines. Some work has been done in peer-to-peer distribution and theories resemble existing NoSQL databases with limited context pattern matching and stream processing capabilities.

We focus on creating system that scales dynamically with the quantity of producers, context partitions, while giving guarantee of availability. Our research neglects any optimizations in vertical fashion (stratification, anything related to current communication stack or process execution optimizations).

In next section we introduce Collaborative Fully Distributed Complex Event Processing (CFD-CEP) which is our solution to this problem

4 Collaborative Fully Distributed Complex Event Processing

In this section we introduce our model for CEP. Our method is fully distributed. That means each node has exactly same semantics and is no different from any other node. The Fig 2. shows difference between typical CEP engine and our method. On the left, typical engine correlates between events from all the producers. The engine can be part of EPN but that partitions contexts (we revisit this limitation in next section). Our solution on the right is free of any EPN abstraction. Each producer is simultaneously a CEP engine.

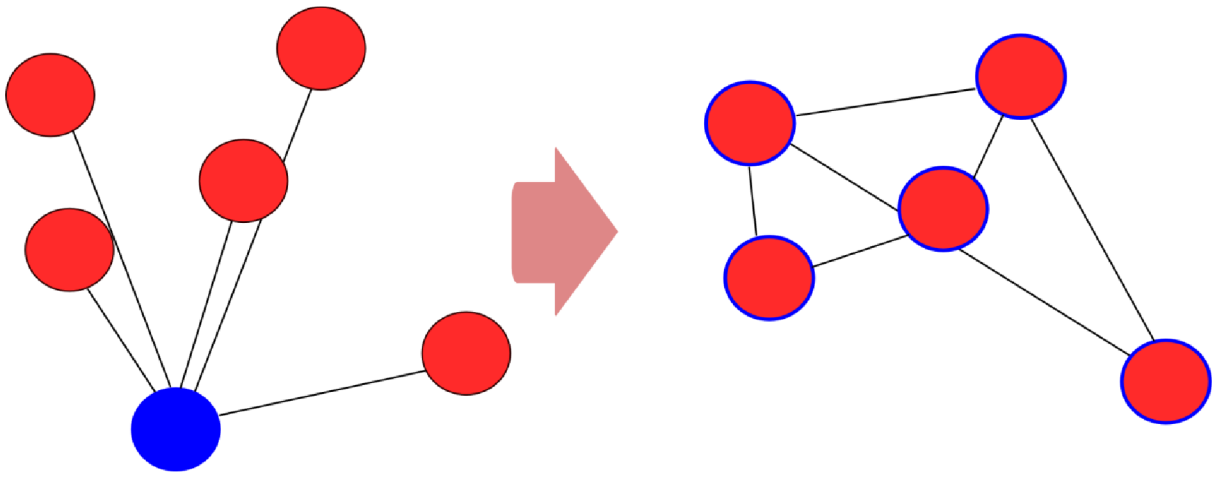


Fig. 2.

Our CFD-CEP model becomes distributed application in classical sense with dynamic properties and asynchronous communication channels. To achieve centralized cooperation for specific event processing tasks we use *leader election* distributed algorithms that dynamically select coordinator of specific task. Each instance of CEP engine has two main differences to typical CEP application.

1. Implementation
2. Pattern reaction syntax

Implementation has to support creation of new CEP nodes directly on producers. This alone is non-trivial requirement and by far most important problem to tackle.
Another problem of the implementation, application of distributed algorithms, is not that complex in the light of the fact that communication links are established between the nodes.
Pattern reaction syntax in current systems can generate higher level event. We introduce new events that will be used to alter structure of the graph that makes up the distributed CEP engine:

- *Node event* creates/deletes processing node (engine) or adds new event pattern to specific engine
- *Link event* adds/deletes link between two nodes

These two events grant possibility to manipulate the distributed CEP graph. Beside these, we allow to insert a new producer/engine into a graph manually.

4.1 Collaborative Subset Identification

This section describes very important problem that arises with sliding windows. Suppose we have to match following, very simple, pattern $\forall x, y\ (x = y)$ in environment that is depicted on Fig 3. Producers 1-5 are generating letters from alphabet. To do such pattern matching, it is necessary that CEP engine holds all the events that are in specific time window in memory and does matching among them. As number of producers grow, more computation power is needed to enable the matching.

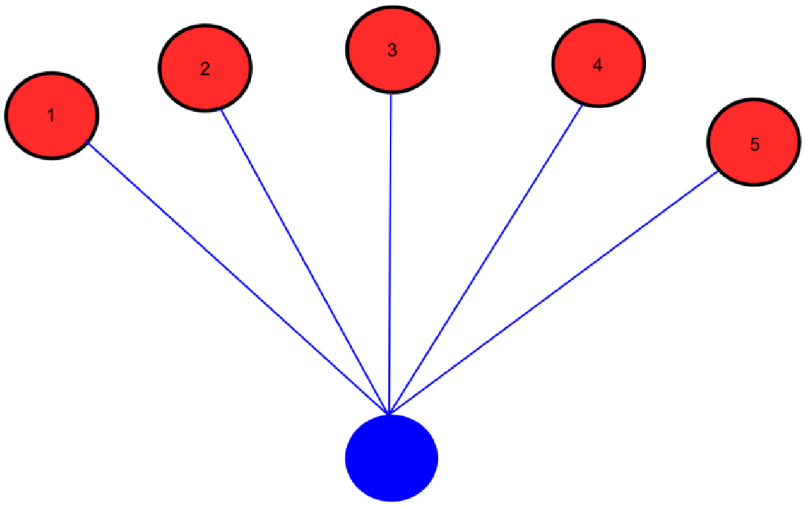


Fig. 3.

We argue, that distribution of letters from different sources is different. Suppose that producers 1 and 3 have higher probability of generating letter *A* than any other producers. Knowing that statically, one might add yet another engine to the application just for this simple case as depicted on Fig 4. New engine *AE* has very simple semantics. It only detects A events over producers *1, 3* and correlates between them. Because we knew that *1,3* have higher probability of generating such events it is also more probable that the *AE* is useful.

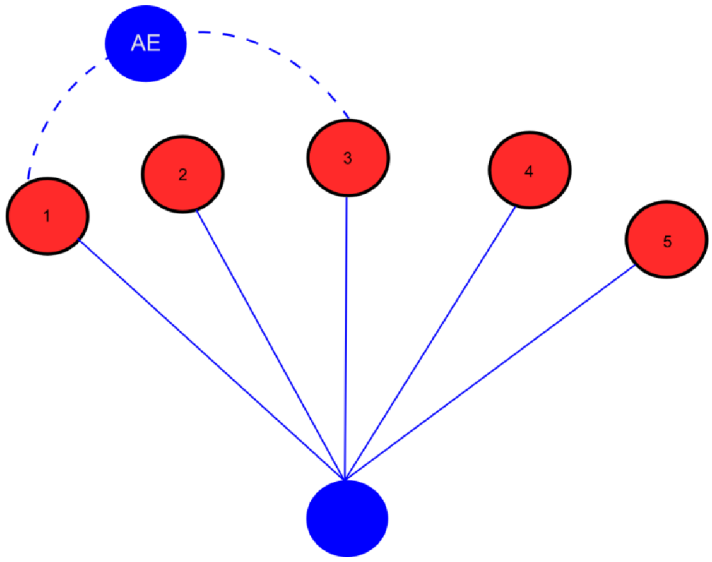


Fig. 4.

By adding *AE* we have created another context for pattern matching. It is significantly smaller than the context of the main CEP engine in the example. We could now delete links between 1, 3 and main CEP engine. That would unload event burden on main engine, but simultaneously we would decrease probability that some events that were sent by 1 and 3 would be correlated with other engines. Nevertheless, we think that this tradeoff is necessary to scale such applications.

Interesting fact to note about matching $\forall x, y\ (x = y)$ is, that theoretically it would be possible to match such rule, without loss of any information by putting engines between every binary combination of producers as depicted on Fig 5. This is obviously another extreme as opposed to previous solution. Number of such *AE* engines grow exponentially - $\binom{n}{2}$, where *n* is a number of producers. Clearly, the tradeoff between number of *AE* engines and information extraction capabilities of whole CFD-CEP engine has to be make. To make this tradeoff it is necessary to identify subsets of producers that are somehow related and create context above each of such subset for pattern matching. We call this problem *Collaborative Subset Identification Problem* and our solution is CFD-CEP. Identification of specific subsets is made by CEP patterns and context creation is done by generating *Node event* or *Link event*.

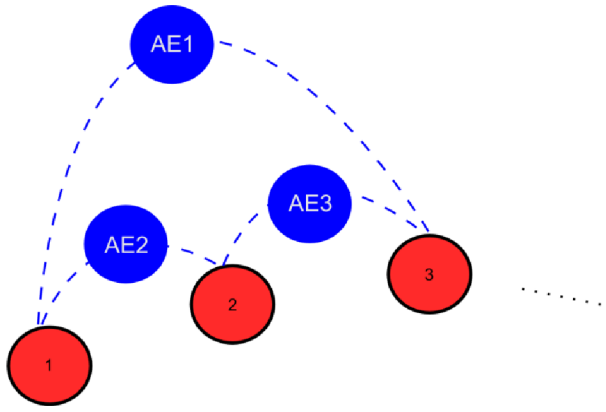


Fig. 5.

4.2 Evaluation Metrics

This subsection discusses possible metrics that will be part of experiment to measure performance of CFD-CEP. In distributed computing *message complexity* is known metric. This will, for sure, be used to complexity of context creation over identified subsets.

Most important measurement will be concerned with information loss relatively to performance gain of the whole system. This metric will therefore measure tradeoff situation and will prove that tradeoff has been made on reasonable basis.

4.3 Challenges

There are few, hard to solve, challenges that we face with CFD-CEP:

1. How to measure uncertainty in CFD-CEP
2. How to map statistical methods to CEP patterns to solve *Collaborative Subset Identification*

Uncertainty in CFD-CEP is given by the fact that information is lost. Measuring such loss in controlled experiment will be very demanding on high quality data set.
To solve *Collaborative Subset Identification* it is necessary to create clear guidelines how to find out that some producers are related to each other. Initial idea is to correlated coarse grained events on central CEP engine (selected by *leader election* distributed algorithm) and afterwards deploy new contexts between identified engines.

4.4 CFD-CEP Implementation

For experimental evaluation it is necessary to create prototype implementation. Implementation consists of daemon written in Java programming language. The dae-

mon is runable on any device with Java Virtual Machine. It is a node in the distributed application graph with knowledge only about its neighbors.

To monitor and run experiments we develop web portal for running historical data that simulate event streams. We send them into the distributed network from the web application. This web application is absolutely independent of the idea itself and acts as monitoring tool for the experiments.

We do not plan to write the daemon from scratch. After evaluation, we have selected two existing software packages/projects to base our daemon upon:

- Smartfrog
- Esper

Smartfrog is Java project developed at HP labs. It is devoted to deploy distributed applications. We leverage some of its abilities to deploy distributed system and ensure communication links between nodes. Smartfrog was intended to mostly static deployments. Our use case is highly dynamic. That might be possible obstacle, but benefits from object oriented nature of internal Smartfrog API is convincing fact to use it. Esper is Java based CEP engine that features SQL like syntax for patterns. Together with Smartfrog we will integrate it into the daemon and distribute such package to every node in the distributed system. Thanks to lightweight implementation of Esper and possibility to embed Esper into JVM it is perfect candidate for such implementation.

We believe that our approach can be further extended to fully functional software suite, because foundations of our solution will be based on industry-proven software packages.

5 Conclusion

In this paper we have presented novel approach to CEP scaling. Collaborative Fully Distributed Complex Event Processing (CFD-CEP) allows to scale CEP applications on peer-to-peer basis. This way, CFD-CEP solution is seen as a network of identical EPAs. We further simplify notation to describe EP networks known from the literature, because we believe pattern matching over sliding windows to be the most important area of CEP research. On the other hand we add more focus on dynamic distributed aspects of EP networks and we enrich the CEP description with rigorous descriptions of CEP pattern matching rules. Rules are given in formal way, but not only that, they also feature new construct to enable distributed processing to be captured.

Because EPA is close to the producer we also achieve effective vertical scaling regardless of computing architecture being used. While experimental evaluation is still pending, we have presented design of target system and identified possible implementation paths.

From performance point of view our distributed solution may be seen as a tradeoff between aggregation capabilities with sliding windows and scalability. On the other hand one might argue that we lose some knowledge about specific correlations be-

tween events. In the end we agree with this argument. We do not see this property of our system to be problematic. Actually, we believe that this is property of the massive event processing itself. It is not possible to aggregate infinite amount of events with finite resources. So it seems necessary to sacrifice some aggregation capabilities to gain processing capabilities. Allowing such tradeoff on the level of theoretical CEP model gives great flexibility in the design of such applications.

6 References

1. Artikis, A., Etzion, O., Feldman, Z., Fournier, F. (2012). *Event processing under uncertainty.* Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems - DEBS '12
2. Akram, S., Marazakis, M., Bilas, A. *Understanding and Improving the Cost of Scaling Distributed Event Processing Categories and Subject Descriptors*
3. Balis, B., Dyk, G., Bubak, M. *On-Line Grid Monitoring Based on Distributed Query Processing*
4. Balis, B., Slota, R., Kitowski, J., Bubak, M. *On-Line Monitoring of Service-Level Agreements in the Grid*
5. Barbosa, V. *An Introduction to Distributed Algorithms* The MIT Press, 1996
6. Biger, A., Rabinovich, Y. *Stratified implementation of event processing network*
7. Etzion, O., Niblett P. *Event Processing in Action* Manning Publications, 2011
8. Hirzel, M. *Partition and Compose: Parallel Complex Event Processing*
9. Huebsch, R., Hellerstein, J. M., Shenker, S. *Querying the Internet with PIER.*
10. Isoyama, K. *A Scalable Complex Event Processing System and Evaluations of its Performance*
11. Kowalewski, B., Bubak, M., Bali, B. *An Event-Based Approach to Reducing Coupling in Large-Scale Applications.*
12. Lee, S., Lee, Y., Kim, B., Candan, K. S., Rhee, Y., Song, J. *High-performance composite event monitoring system supporting large numbers of queries and sources.* Proceedings of the 5th ACM international conference on Distributed event-based system - DEBS '11
13. Luckham, D. *The Power of Events* Addison-Wesley Professional, 2002
14. Luckham, D. C., Frasca, B. *Complex Event Processing in Distributed Systems*
15. Randika, H. C., Martin, H. E., Sampath, D. M. R. R., Metihakwala, D. S., Sarveswaren, K., Wijekoon, M. *Scalable fault tolerant architecture for complex event processing systems.* 2010 International Conference on Advances in ICT for Emerging Regions (ICTer)
16. Renesse, R. V. A. N., Birman, K. P., Vogels, W. Astrolabe : *A Robust and Scalable Technology for Distributed System Monitoring , Management , and Data Mining*
17. Schilling, B., Koldehofe, B., Rothermel, K. *Distributed Heterogeneous Event Processing Enhancing Scalability and Interoperability of CEP in an Industrial Context Categories and Subject Descriptors*
18. Tel, G. *Introduction to Distributed Algorithms* Cambridge University Press, 2000
19. Vera, J., Perrochon, L., Luckham, D. C. *Chapter Event-Based Execution Architectures for Dynamic Software Systems*
20. Wu, E., Diao, Y., Rizvi, S. *High-performance complex event processing over streams.* Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06
21. Magid, Y., Sharon, G., Arcushin, S., Ben-Harrush, I., Rabinovich, E. *Industry experience with the IBM Active Middleware Technology (AMiT) Complex Event Processing engine* In

Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems - DEBS '10

A.5.3 BCI 2012

Paper [64]

Information System Monitoring and Notifications using Complex Event Processing

Filip Nguyen
Masaryk University
Faculty of Informatics, Lab Software Architectures and IS
Botanická 68a
602 00 Brno, Czech Republic
xnguyen@fi.muni.cz

Tomáš Pitner
Masaryk University
Faculty of Informatics, Lab Software Architectures and IS
Botanická 68a
602 00 Brno, Czech Republic
tomp@fi.muni.cz

ABSTRACT
Complex Event Processing (CEP) is a novel approach how to process streaming events and extract information that would otherwise be lost. While tools for CEP are available right now, they are usually used only for a limited number of projects. That is disappointing, because every Enterprise Information System (EIS) is producing a high number of events, e.g. by logging debug information, and industry is not taking an advantage of CEP to make these information useful. We pick two concepts that seems to be from a different category - notifications - a ubiquitous way how to notify user of an EIS and EIS monitoring. With notifications we define a new abstraction upon notifications with respect to a separation of concerns to create a more maintainable implementation. In our research we show that this is a typical example of a possible future application of CEP and that the industry requires specific service oriented tools that can be used for both, notifications and monitoring. When these service oriented tools would be introduced into the industry it would promote EIS maintainability and extensibility.

Keywords
CEP, EIS, monitoring

1. INTRODUCTION
This paper is an outline of a research intent and aims to address the Complex Event Processing (CEP) in the area of Enterprise Information Systems (EIS).

The CEP is a relatively new data processing approach (introduced in [2]) to extract interesting events out of possibly large amount of meaningless events. To do so, CEP uses temporal operators. There are many solutions on the market that support CEP scenarios [3] [8] [14] [9] [15]. These solutions allow to run CEP rules on streams of events and reveal interesting patterns. The CEP abstraction is very low level and to gain any business value it is necessary to add fairly complex integration logic and non-trivial amount of work.

There are several areas in which CEP research is currently taking place. One is Business Activity Monitoring (BAM), failure detection, risk detection, capital market monitoring. All these areas are very specialized and are not embodied in every organization or every EIS. It seems that CEP solutions are only interesting for these specialized applications and require high technical expertise to deploy them.

Notifications are next area we are interested in and research is also taking place. This area seems to be distant but it is easy to see that any notification is just a reaction to events inside an EIS. Be it IPDC in DVB-H [20] or even notifications using other notifications [10]. Notifications are studied in many contexts, mainly in connection to web services [13][19][18]. Notifications are so important in EIS development because today, user is connected to many different systems and from practical reasons he is not log in to every one of them. Instead it is common practice that system notifies user of interesting events and he then may choose to log in to control the situation. Place where these notifications meet (common place is an e-mail) can be thought of as a one meeting place for all interesting events. This underlying concept of notifications brings question, why the user should be limited to have just e-mail as a common engine to receive messages. Because of this and other topics, that are out of the scope of this paper, it is beneficial to utilize so called Notification System (NS). In our research we want to combine CEP solution with NS and EIS monitoring in such a way that it is usable in almost every EIS. NS is able to inform an end user of an EIS about complex events or can store the complex event for review by an administrator of the system (monitoring). Further, we want to analyze existing systems and show, that our approach will create cleaner separation of concerns for almost every existing EIS.

Application monitoring is also gaining more importance. Reason may be that lot of bleeding edge systems today are still very complex and without user interface. Cloud computing, heavy lifting application servers, complex business process orchestrations including human tasks or monitoring via sensor networks. All these systems and concepts needs common way how to monitor them to detect failures, notify

users and administrators of important events. Research in this area focuses on performance, e.g. [4], or pro-active detection of undesired results: detection of aggressive drivers in cities or more theoretical research in this area, e.g. [6]. But no research is related to giving a platform for developers to truly show developers how to use CEP for the monitoring and consolidate knowledge.

The basic idea with regards to notifications is to detect a *notification lead* from business events using CEP and generate *notification*. By creating loosely coupled CEP Engine for Notification (CEPEN) that will receive events within an organization it is possible to encapsulate complicated notification triggering rule, in such a way that it is better maintainable and reusable. Creating such an abstraction is similar to an underlaying idea of BPM - abstraction from underlying system. In BPM implementations today, it is common to coordinate multiple resources and systems, even human tasks, to carry out such a process. However, in the area of NS these implementations are usually hardcoded in EIS itself. CEPEN easily consolidates events from multiple systems and gives a developer tools to maintain event rules, share them with community and provides developer with body of knowledge necessary to use CEP in his application effectively.

Benefits of CEPEN are thus twofold. First usage is on business level, where developer generates business events from various systems in organization and CEPEN will be meeting place of such events to generate notifications. The second usage is EIS monitoring. In this case CEPEN will be notifying administrator of the EIS about important event that happen inside of the EIS. Both these usages leverage common architecture of CEPEN and thus prove the architecture to be general. Also, in both cases, developer is looking at one stream of events, consisting of all events in the organization.

This paper is organized as follows. In the second chapter we specify our goals in each area of the interest. The third chapter describes deliverables of our research and describes them in detail. In later chapters we lay out high level research plan and shows how deliverables should be created with discussion of methods to achieve them. The last chapter, Conclusion discusses possible research directions and summarizes benefits of EIS monitoring and notifications using CEP.

2. GOALS OF RESEARCH

We aim to review state of the art EIS and examine how CEP can bring value added. Two areas of the interest that we see particularly interesting are *monitoring* and *notifications*. These two abstractions are important for almost every EIS.

First goal of the research is to gather input from existing EIS developers and passively gather EIS notifications to find existing usages of CEP in a real world development. This can be done in number of ways but the goal is to differentiate between theoretical possibilities and real motivations of the development industry. An active approach can be taken when interviewing developers and stakeholders of small scale systems. By analyzing large systems such as world wide social networks we aim to analyze possible CEP opportunities by studying notifications coming out of these systems.

Identification of existing events and their types in organizations is next necessary goal to analyze CEP usage in any EIS. Catalog of types will be created and most interesting events identified. Such a list can be also beneficial as a final product for any manager or developer in EIS development.

One of the most important goals of the research is to show that it is possible to monitor activity in EIS as one stream of heterogeneous events consisting of business events, low level logging, hardware activity events and third party systems' events. The more flows of events developer had, the more complex patterns he can detect. To motivate him in looking for such pattern we will develop CEP engine platform and collaboration tools for sharing CEP rules.

Already mentioned goal is to create a prototype implementation of a platform for CEP application for Java EE stack, CEPEN, to easily analyze events from existing EIS. There are many CEP engines and while we want to make use of them we also want to avoid vendor-lockin and base the architecture of such CEP engine on open and easy to understand standards. Imperative is easy to use interface. The service will have a web interface that will allow a user or a researcher to enter CEP rules and based on this rules it will allow to send notifications or log important events. The prototype will support processing events from multiple systems by design. This is probably the most important difference between monitoring via CEP and using CEP in the system itself. By focusing on monitoring one can consolidate many events from different sources without worrying about EIS boundaries.

To send notifications, CEPEN will use our, already developed, notification system NotX [1]. This system is briefly introduced in next chapters.

By meeting goals above, it is possible to predict future challenges, possible outcomes and points of interest of CEP in EIS applications. All the information from this research will be published in publicly available Wiki system. Also the prototype will be available for professionals to share event processing patterns and to try out new patterns that will emerge in their EIS.

3. RESEARCH DELIVERABLES

Research outputs can be depicted in standard WBS (Work Breakdown Structure) diagram (Figure 1).

First level contains our goal - bring value to EIS development with regards to NS and monitoring.

3.1 Knowledge Base

Knowledge base will serve two purposes. It will be input for a development of CEPEN and it will serve to EIS development community to integrate CEP ideas into their solutions.

Because patterns are common practice how to capture repetitive solutions in software industry, we believe that *CEP patterns* should be also captured for typical EIS.

EIS List will contain case studies that are either proprietary for given organization or well known social networks. While for proprietary EIS it is possible to contact small number

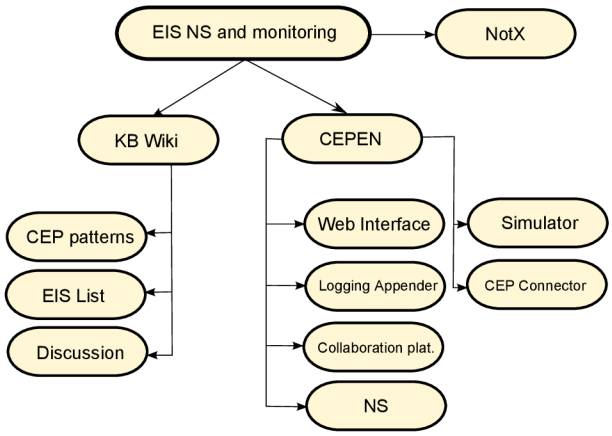


Figure 1: Work Breakdown Structure

of core developers and discuss CEP usages in the area this may not be the case for services such as Facebook or Twitter. For such services, we choose to study notifications that are being generated by such systems and show how they can be easily developed, maintained, revised, reused using CEPEN or standard CEP solution.

It can be argued that knowledge base is only building of content. We believe that important part of any knowledge base is tight integration of technology with actual knowledge. Allowing users to start up CEP rules presented in the knowledge base, navigate to actual collaborators that created the rules. Also motivation features such as promotion to moderators and domain experts known from services such as Stack Overflow are beneficial for the community. Also Knowledge Base should encourage direct use of patterns. CEP patterns that are usually hard to replay and experiment with. Web interface with pre-recorded CEP data should ease up this task.

We wish to establish a *Community* for CEP development in the area of NS, by inviting EIS to publish their events or specify them at least.

3.2 NotX

There is no open notification system in the community right now. Research has been done in specific sub-areas of notifications as *web services* to *instant messengers* integration [11] and more general concepts as WS-Notification were considered. In our research we want to leverage more low level service, because of that, as part of our research we developed advanced notification system called NotX that will be used as a service for CEPEN. Overall architecture of NotX can be seen on Figure 2 [1]. Due to fact that NotX is queue centric and all its components are horizontally scalable, it is ready for high load and prepared to provide all features that CEPEN requires including storing of monitoring information.

From business point of view the NS have great deal of benefits for user himself and for a developer. User can use more complex ways how to route notifications to different engines. He can send all important messages to his cell phone via text message. Some of them can be routed directly to social

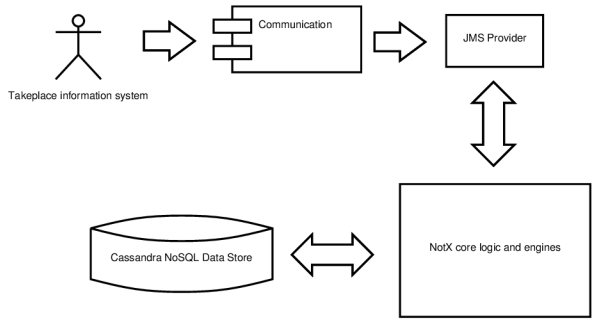


Figure 2: NotX architecture

network if there is a notification engine available for that specific social network. Developer can leverage

- Internationalization
- Out of the box prepared web interface for administrator and user
- Prepared engines (Voice, SMS, e-mail) and he can develop his own
- Scalability and performance (because of low coupling)

3.3 CEPEN

Events will be consumed by CEPEN in text representation, which is easy to generate. There has been research about extensions to CEP ([16]) that should mitigate technological hurdles of such a processing. It can be argued that XML or more structured key/value pairs should be used. However, it is necessary to find middle ground between various sources as system log files and human input.

EIS implementations that are based on Java platform use a logging of development messages. That will enable us to directly consume lot of events.

Basic architecture is depicted on Figure 3. The arrows visualize event streams. The main flow comes via logging appender, to promote easy to use interface. Developer of the EIS in the organization can generate more complex notifications and detect more complicated event patterns when multiple event streams in an organization are consolidated. To achieve this we allow to supply events into CEPEN via ESB which enables easy way to consolidate third party EIS through any protocol supported by ESB, log files from operating system or submitted by human (Human task). The online architecture on the Figure 3 shows the CEPEN in the context of other services and environment. To understand concrete capabilities of CEPEN it is necessary to consider inner structure and services provided.

Main components are depicted on Figure 4. The Figure shows 3 areas of business interest:

1. **Base CEP** Area responsible for rules management and execution.

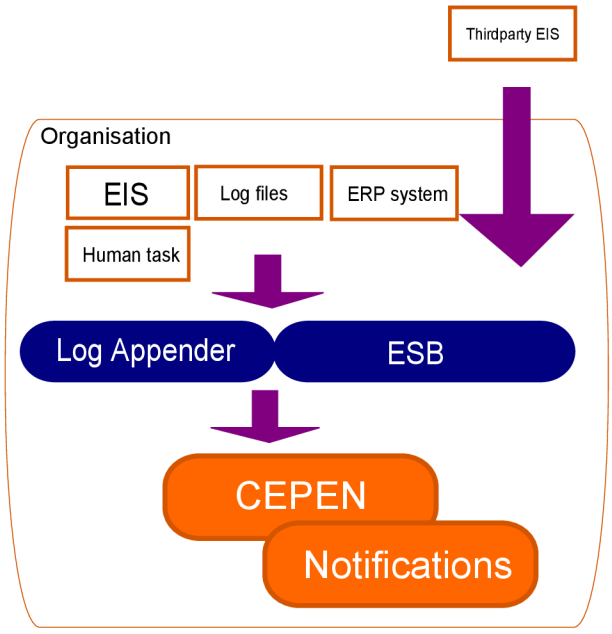


Figure 3: CEPEN online architecture

2. **Data Streams** Area responsible for creating data streams from typical EIS and promote debugging with pre-recorded data.

3. **Collaboration** Area responsible for rules sharing and collaboration with creating CEP analysis over EIS.

Inside these areas there are specific use cases that will be implemented to support CEP in EIS development. Web interface will be, as usual, place where these three areas meet. Rules sharing will enable reuse of general CEP patterns. Use case *Static datasets* represents possibility to use historical data as a data stream. *Common CEP* works for

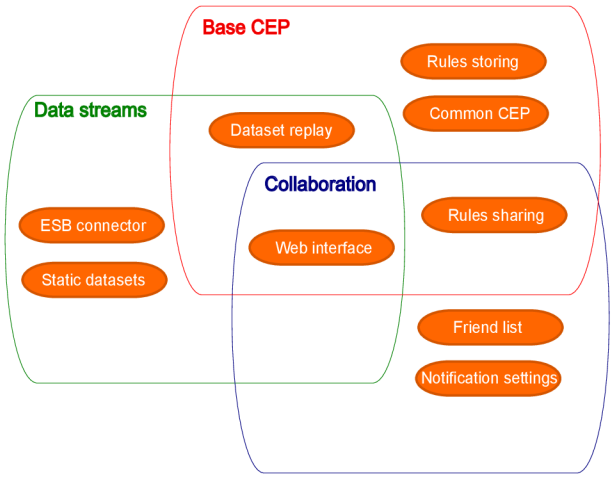


Figure 4: CEPEN internal design

3.4 Event streams

Event streams will be obtained from existing systems by implementing standard Java technology by implementing standard logging appenders. This way we will gain lot of events for free without need of modification of EIS code base. It is very interesting way of utilizing large amount of events, that are currently available in every EIS. Moreover, because logging in EIS is usually very verbose on *debug* level, this level is not used in production environment and it's potential is wasted. Also, because application logging is natural to every Java developer we argue that developer should also use same interface (Java logging) to publish business events.

The main reason debug level is disabled in large systems is because logs would consume too much of a disk space. This risk will be mitigated by processing the logs right away. Also, to give a developer more options to reason about problems in the system we will use ESB centric architecture. The events will be picked up with ESB and routed to service that will represent the CEPEN connector.

3.5 Monitoring

Monitoring is interesting mostly because of the power of reuse. Java EE systems can turn on debug level notifications. These are useless when read manually because of verbosity. We examined several logs of existing information systems and outputs of JBoss Application Server version 6. Most important things that can be monitored on running servers we identified so far:

1. **Hardware problems** In a given time window, client machine, or one of client servers is getting multiple warnings or errors

2. **Performance problems causes** Correlation of performance usage with certain type of SQL queries

3. **Malicious behavior** DoS attacks prevention (trivial use of *time window* concept in CEP). Suspicious processes running with thick client while performing highly restricted queries with the EIS.

4. **Performance problems** Running of complex SQL queries in short time

5. **Clustering problems** One node is getting too many requests while system's performance log shows full CPU usage.

6. **Failover problems** Messages is received using *modjk* apache module and delivered into a cluster but suddenly is missing when node goes down

Another benefit of using monitoring is it's distributed architecture that stems from CEP *stream* abstraction. Application servers are usually deployed in clusters consisting of tens nodes. Each node contains own log file and runs on different machine that has it's own performance/system logs. All these can be easily combined into one stream with CEPEN and complex processing logic run on them.

With regards to information systems, next advantage we identified is possibility to gather logs from clients easily. We didn't experimented with this option yet but we see that this can be very promising area of research by itself.

4. RESEARCH PLAN

To address chronological organization of the research it will be conducted in three distinct phases as depicted on Figure 5. *Phase 1* is mainly resource oriented work, that will require to enlist systems, contact developers and conduct interviews. This phase should be as long as possible to gather most input. Outputs will be document/publications oriented. We will publish requirements from the industry and reason about their motivations.

The second phase revolves around brainstorming conducted by research team and selected participants from EIS developers. This brainstorming should have a very strong output that will define future research directions and, most importantly, define the CEPEN. Development of CEPEN (CEP Engine for Notifications) will be guided by the Scrum methodology [17]. Product owner is not yet known but he will be selected from interested parties.

The third phase is all about revising existing system, concluding whether benefits and goals were met and putting all the information into public knowledge base. Plan is that involved parties will by this time be using the CEPEN routinely and thus certain parts of knowledge base will have it's owners.

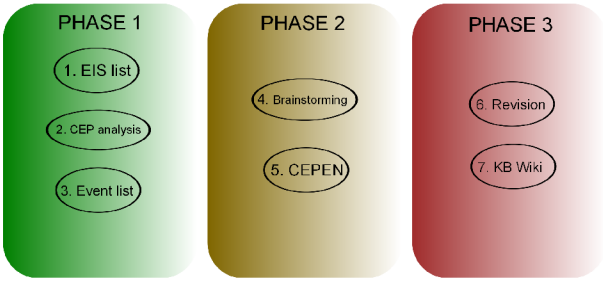


Figure 5: Phases of research

1. Create initial list of EIS implementations
2. Create a list of CEP solutions in those EIS
3. Identify interesting events in various systems
4. Interview EIS developers and brainstorm events and CEP ideas. This stage should be done after initial list of events to be able to give the examples to the developers during the interview
5. Create a CEPEN prototype with a web interface
6. Connect EIS implementations or EIS simulators (described in next chapters) to examine CEP applications with regards to logging and notifications. Finish the list of EIS implementations and list of events
7. Create initial knowledge base on public Wiki system with description of CEP patterns that are particularly useful for logging and notifications from any EIS. This knowledge base will contain concrete examples and it will be possible to test those on the simulator

5. CONCLUSION

In this paper, we have outlined the intended research in very promising area of EIS. Our approach relies on the tangible connection with real world EIS and aims to identify and redefine the way CEP is currently perceived. CEP offers strong basis for gathering patterns that would be hard to gather offline or would lose their importance when gathered lately.

We identified use cases for notifications and monitoring of EIS. Further, we have elaborated use cases of monitoring and we identified key feature of this monitoring to be looking at the whole system as vast amount of heterogeneous events that need to be searched in real time using and using CEP is the best approach to do so.

We explain that notifications, that are usually direct consequences of business events, shouldn't be hardcoded and can be reused, which is novel approach, not yet implemented in EIS. Notifications could also be used to report important events detected by monitoring.

We introduced a new way of looking at *separation of concerns* in yet another ubiquitous area of EIS development. Both monitoring and detecting of notification leads is technically possible to solve by using CEP. However, to step beyond academical usage it is necessary to incorporate complex approach and build much more than just a proof of concept. Deliverables to achieve such a goal were identified and described.

Our research already begun, bringing enterprise level notification system NotX that will be used as a service for CEPEN.

Nevertheless, the future research (after developing CEPEN) cannot be predicted right now. Only after the research finishes, based on results of we may:

1. Continue with new enhancements to the prototype
2. Extend CEP theory to better support applications for information systems
3. Create more CEP patterns for EIS

6. REFERENCES

[1] Nguyen F., Škrabálek; J. *NotX Service Oriented Multi-platform Notification System* In M. Ganzha, L. Maciaszek, M. Paprzycki. FedCSIS 2011 proceedings. 10662 Los Vaqueros Circle: IEEE Computer Society Press, 2011. od s. 313-317, 1011 s. ISBN 978-83-60810-39-2.

[2] David C. L. 2001. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] Esper - Event Stream Intelligence, `http://esper.codehaus.org`

[4] Lee, S., Lee, Y., Kim, B., Candan, K. Selçuk, Rhee, Y. and Song, J. *High-performance composite event monitoring system supporting large numbers of queries and sources* In Proceedings of the 5th ACM

international conference on Distributed event-based system, 2011

[5] M. Olson, A. Liu, M., Faulkner, K. M. *Rapid Detection of Rare Geospatial Events: Earthquake Warning Applications* In International Conference on Distributed Event-Based Systems, 2011

[6] Engel, Y., Etzion, O. *Towards proactive event-driven computing* In Proceedings of the 5th ACM international conference on Distributed event-based system, 2011

[7] Okorodudu, A., Fegaras, L., Levine, D. *A Scalable and Self-adapting Notification Framework* In Database and Expert Systems Applications, 2010

[8] Drools Fusion `http://www.jboss.org/drools/drools-fusion.html`

[9] HStreaming `http://www.hstreaming.com`

[10] Minakuchi, M., Miyamori, H. *Richbiff: E-Mail Message Notification with Richer Clues* In Human Interface and the Management of Information. Designing Information Environments, 2009

[11] Chi-Huang Chiu; Ruey-Shyang Wu; Chi-Io Tut; Hsien-Tang Lin; Shyan-Ming Yuan; Nat. Chiao Tung Univ., Hsinchu *Next Generation Notification System Integrating Instant Messengers and Web Service* In International Conference on Convergence Information Technology, 2007

[12] Pradeep G.; Pyarali, I.; Gill, C.D.; Schmidt, D.C. *The design and performance of a real-time notification service* In Real-Time and Embedded Technology and Applications Symposium, 2004

[13] De Labey, S.; Steegmans, E.; Dept. of Comput. Sci., K.U. Leuven, Leuven *Extending WS-Notification with an Expressive Event Notification Broker* In IEEE International Conference on Web Services, 2008.

[14] Twitter Strom, `http://engineering.twitter.com/2011/08/storm-is-coming-more-details-and-plans.html`

[15] Oracle CEP, `http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html`

[16] Alves, A. *A General Extension System for Event Processing Languages* In DEBS 2011

[17] Schwaber, K., Beedle, M. *Agile Software Development with Scrum*

[18] Yi H.; Gannon, D. *A comparative study of Web services-based event notification specifications* In International Conference on Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006

[19] Vinoski, S. *More Web services notifications* In Internet Computing, IEEE. 2004

[20] Hornsby, A.; Bouazizi, I.; Defee, I. *Notifications: Simple and powerful enhancement for services over DVB-H*, 2008

A.5.4 Control and Cybernetics - ADBIS extended paper 2012

Paper [70]

Control and Cybernetics

vol. **41** (2012) No. 4

Towards effective social network system implementation*

by

Jaroslav Škrabálek, Petr Kunc, Filip Nguyen and Tomáš Pitner

Masaryk University, Faculty of Informatics,
Lab Software Architectures and IS
Botanická 68a,
602 00 Brno, Czech Republic
{skrabalek, xkunc7, xnguyen, tomp} @fi.muni.cz
http://lasaris.fi.muni.cz

Abstract: In this paper we present our latest research in the area of social network system implementation. Both business and technological aspects of social network system development are considered. There are many tools, languages and methods for developing large-size software systems and architectures represented by social network systems. However, no research has been done yet to uncover the reasons behind the selection and usage of such systems in terms of choosing the right architecture and data storage. We describe effective approach to developing specific parts of social network systems with special attention to data layer (using Hadoop, HBase and Apache Cassandra), which forms the foundation of any social network system and is highly demanding for performance and scalability.

Keywords: NoSQL, architecture, social networks, complex event processing

1. Introduction

Social networks – a millennium's first decade phenomenon – have enabled users to connect with people they usually have never seen in person and to live virtual lives, promoted progressive networking, helped people to find a job or just supported gamification (defined as the infusion of game design techniques, game mechanics, and/or game style into anything to solve problems and engage audiences; see Zichermann & Cunningham, 2011) of regular products and services as a very engaging marketing channel.

*Submitted: October 2012; Accepted: November 2012

Size matters. The social networks, unlike common web-based information systems, represent a supreme discipline of software development. No other information system, application or web service could attract millions of users with immensely progressive potential. This unique opportunity cannot be built without a deep requirement analysis including users as main decision makers from the very early phase and, of course, without a careful selection of functionality. For designing a social network system, user-centered approach is particularly crucial. A *platform*, which social network actually is, takes human perspective into account. If customers are satisfied, they are more likely to use new services and recommend the platform to other potential users, which enables the growth of the user basis and provide the foundation for future network. The high number of users indicates high popularity of the platform, which brings more people in. Furthermore, if people use the product, they provide feedback, in particular, they report errors and require new features. Their feedback can lead to significant improvements of the social network and as a result, it can enhance platform as a whole (see Škrabálek et al., 2011).

Architecture is the key. Architecture of the social network system is like a backbone. If it is crooked, the growing potential will never be reached and only a small number of pioneers will use the platform for a limited time and then they will leave it. Scalability and robustness as well as universal analysis with advanced level of flexibility will ensure future enhancement, and eventually will help to completely change the initial intention if users require quite a different functionality. This is, for example, the case of *Takeplace* (see *Takeplace*, 2012) – a digital and mobile event management platform helping event organizers in all event management processes. Thanks to precise definition of core functions followed by an open-minded and foresighted analysis, Takeplace becomes a platform supporting both organizers and community of event attendees to manage any kind of events of any size from small seminars, consultancy meetings with 20—40 participants up to the conferences with hundreds of attendees, or even trade shows, fairs and festivals inviting thousands of people. Soft part of the development such a social network functional requirements, analysis and design is just the first part within the process of social network system development. We need to know *What* to develop but the question *How* follows immediately.

Persistence. Selecting a proper back-end technology is crucial. In the starting phases of the platform adoption by users, it is very easy to handle the demand by standard tools and software approaches one was used to employ in numerous previous projects. The turning point is different in every project but such a time will certainly come and everybody will learn how restrictive our past decisions may be in the case of a social network system. It may even cause the end of the previously well-evolving platform. Therefore, it is indispensable to consider modern persistence tools and frameworks like *Hadoop* from the very beginning since they support handling big data volumes. The non-relational, distributed DBMS HBase and NoSQL database solution Cassandra (described later in the

paper) help developers to keep up with the technological development in time and preserve the direction with respect to contingencies.

Mobile platforms. Such needs are also accentuated by tremendous advent of modern mobile platforms. While the front-end development is simplified thanks to strict usability approaches required by the companies standing behind iOS, Android or Windows Phone 7, the requirements for cloud-friendly back-ends increase continuously. Regarding the existing social networks *Facebook*, *Twitter*, *Instagram* or *Pinterest* the platforms the most people speak about catch around 50 % or more traffic from mobile smartphones and tablets*. Tablets are increasingly appearing and starting to be used as the main working tool. This change of ICT utilization paradigm (see Gartner, 2012) will cause enormous demand for well-developed, not only social network, services and platform solutions in next five years, capable to handle millions of inquiries as well as huge data storages in the backend.

This paper is organized as follows. In this section we have discussed business aspects of social networks. In the second section *Technological demands of social networks and case study* we will introduce one important case study together with its implementation details. The third section is focused on maintainable implementation of social networks using Cassandra NoSQL database. In the fourth section we fill in the gaps in social network implementations by describing monitoring of social networks.

2. Technological demands of social networks and case study

In the domain of social network services, data-oriented architectures and technologies are widely used as those services demand high data throughput. The architectures are designed for heavy loads, concurrent requests, and database can store billions of rows. This section introduces the key features of *Hadoop* and *HBase* (see White, 2009) and describes them in a real application written in Java using HBase as a persistent storage.

2.1. Hadoop and HBase

The use of NoSQL databases means that the data loses relations, developers cannot use the Structured Query Language with joins, triggers, or procedures. Here comes the question why the system architect would want to choose any of NoSQL databases. The main reason is *scalability.* The following story describes how a growing service based on RDBMS usually evolves. Initially, the developers move the system from local environment to the production one with predefined schema, triggers, indexes and in normalized form (3NF or 4NF). As the popularity grows, the number of reads and writes increases. Some caching service is used to improve the read time and the database loses ACID. To improve write

*While Facebook and Twitter register around 40—60 % of mobile accesses, Instagram is a purely mobile social network with 90+ % mobile traffic.

time the components of the database server must be enhanced. New features are added and database schema must be changed – either de-normalized or the query complexity increases. If the popularity grows further, the server has to be more powerful (thus expensive) or some functionality must be omitted (triggers, joins, indexes) (see White, 2009). This is where software framework Hadoop, developed by Apache, is clearly a better solution as it offers automated and linear scaling, automatic partitioning and parallel computing. Hadoop consists of two basic parts:

- MapReduce
- HDFS (Hadoop Distributed File System).

The *MapReduce* model has been introduced by Google. It consists of two phases that both read and write data in a key-value format. The map phase divides the problem into smaller pieces that are then sent by the master node to other distributed nodes in order to be processed. After solving the current problem at distributed nodes, data is sent back to the master node which processes the responses and assembles the solution of the original problem. This model is suitable in situations when the application needs to "write once" and "read many" while traditional RDBMS are designed for frequent data writes or updates. The MapReduce model is also designed to run on commodity hardware so it deals with node dropouts: whenever (in the map phase) the node does not answer in time, the master node just reschedules the problem to other instance. Thus, the master node is the only bottleneck but there can exist more master nodes in the MapReduce model. *HDFS* was designed to store huge files across the network. The default block size is 64 MiB which improves the seek time and increases transfer rates for vast data. Finally *HBase* is a non-relational distributed database based on the Hadoop framework. Tables are automatically distributed in the cluster in the form of *regions*. Each region is a data subset defined by the first row (included), the last row (excluded) and the region identifier. When the size of data grows so much that it cannot be stored on one machine, they are automatically split and distributed (usually using HDFS) across the nodes.

HBase data model The basic HBase data model is inspired by Google's *BigTable*. Tables are in fact four-dimensional persistent sorted maps. The first dimension is a row - any row has a predefined (on the table level) second dimension – column families. Column families can have variable number of columns (third dimension) containing the data. The fourth dimension can also be used: the version – each column can remember x older versions of data stored in the column. The HBase data model is designed to store billions of rows and millions of columns. Data have no relations so joining the tables is impossible. Keys and values are arrays of bytes, so any data can be maintained. A very simple way to obtain certain data is to access the value of the map by specifying table row, column family and column. The only possibility to obtain more rows is to use sequence scanner fetching rows from some interval as the row keys are stored lexically.

2.2. Takeplace: case study of the architecture

The architecture will be demonstrated on a working example of a social network service designed and implemented for the event management platform *Takeplace* (see *Takeplace*, 2012). However, this subsystem can be used in any service. This goal is achieved by using simple interfaces defining the services. Developer should implement a communication layer dealing with remote calls (for example, REST, JSON-RPC or SOAP) or can even call the services directly when using Java. The communication layer must provide a secure user session identifying the current user.

2.2.1. Inner architecture

The system alone consists of three-layer structure (see Fig.1) connected by interfaces. The service layer provides services for external calls and also takes care of basic authorization of operations to be executed. Data access layer retrieves or stores data from/into the database and its main goal is to transform business objects into data structures and vice versa. The last layer provides basic CRUD methods and creates a simple framework for any non-relational databases or cache. *HBase* and *Memcached* are used in this project.

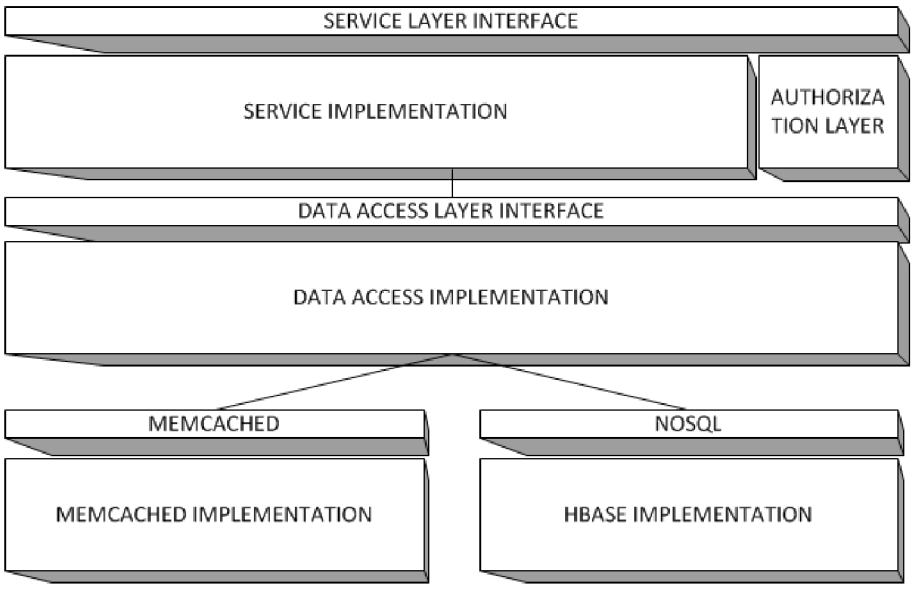


Figure 1. Inner architecture

There are four services available for external calls: *Follow* (managing interactions among users and providing information about the relations), *Wall* (providing interface related to the users' posts, their own walls and news feeds),

Discussion (comments connected to certain post) and *Like* (managing users' favorite posts).

2.2.2. Data model

The data model of the application comprises three tables: *walls*, *entities* and *discussions*. The table *entities* (modeling any user) contains five column families: *followers*, *following* and *blocked* contains user ids in columns. The *news* column family contains ids of posts on news feed (page which shows posts of people the user is following). The last column family *info* contains redundant data about the numbers of followers, following etc. The table *walls* stores posts created by users in the system. The column family *info* stores basic information about the post. In the *text* column family this is only one column containing the text of the post and the *likes* column family contains user ids of people who like the post. The table *discussions* is similar to the table *walls*.

2.2.3. Data storage

While working with non-relational databases the key aspect of design is to choose row identifications as their choice heavily affects performance. The identifications can describe a relation among data and lexical sorting also defines region in which this information is stored. Furthermore, the only way to obtain more rows from the database is the sequence scanner when the rows are sorted lexically. This is why the row identification has to be chosen wisely. For example, table *walls* uses concatenation of the user id and the time of the post, so the posts are grouped by users and then sorted by time from newest to oldest post. Lexically it would be the oldest-to-newest posts so we have to invert bytes in the date format to enable the sequence scanning from the newest posts. Fetching the wall is fast for each user and there is higher probability that it is stored on the same region server. Also each entity can view history of its own posts. There are only weak relations among data. The users (entities) have their posts and they have their comments. These relations are displayed in names of keys and developer is responsible for fetching the correct data.

2.2.4. News feed

The only problem with performance occurs when loading the news feed. These rows of data will be stored across region servers as the row identifier can vary a lot – as the id of any post is assembled from user identification and time – so there would be the need to load a post by post to be displayed from each followed user's wall – or the system could store the posts in each profile creating great redundancy. In this case it is suitable to use memory-caching tool. While sending a new post to the server it is inserted in the cache in a minimized form (also time to live is set) and the link to it is put into the cached news feed to every interested (following) user. Thus, we can obtain a news feed in two cache queries. The first one fetches the list of posts and the second one (batch

query) returns the posts. Memcached is a hash table in random access memory providing fast read/write operations. Once the data gets old or Memcached is full, the expired posts are deleted first and subsequently the least recently used. The backup of the news feed is stored in HBase as the permanent storage.

2.3. Testing the application

The application was tested using *Jakarta JMeter*, *Netbeans Profiler* and private pilot run. The application was tested on three virtual computers, each simulating very old commodity hardware (2 GHz and 1 GB RAM) connected with a 100 Mb/s LAN network. On average, one simple follow invocation took 0.5 ms, fetching the list of one hundred followers and thirty random followers took 2.2 ms (data access time was less than 1 ms). Sending one post to the server consumed 2.58 ms on average and loading wall of 35 posts for a single entity took about 4 ms. Loading time of an entity's news feed was 7 ms. A view of profiler is shown in Fig. 2. On the servers, we got throughput of almost 50 requests per

Call Tree - Method	Time [%]	Time	Invocations
http-thread-pool-8080- (3)		270 ms (100%)	1
consys.social.core.Walle.doGet (javax.servlet.http.HttpServletRequest, ja		270 ms (100%)	1
consys.social.core.service.WallServiceImpl.postPost (consys.social.cor		129 ms (48%)	50
consys.social.core.dao.WallDaoImpl.savePost (java.util.List, consys.s		104 ms (38.7%)	50
consys.social.core.service.FollowServiceImpl.getAllFollowersUid (c		23.0 ms (8.5%)	50
Self time		2.1 ms (0.7%)	50
consys.social.core.bo.Entity.equals (Object)		0.128 ms (0%)	50
consys.social.core.service.AuthenticatedUser.getUser ()		0.017 ms (0%)	50
consys.social.nosql.dao.hbase.HBaseTable.<init> (String)		128 ms (47.8%)	3
consys.social.core.service.FollowServiceImpl.follow (consys.social.core		3.94 ms (1.5%)	5
consys.social.core.service.WallServiceImpl.loadEntityWallNewest (cc		3.85 ms (1.4%)	1
consys.social.core.service.WallServiceImpl.loadEntityWall (consys.s		3.78 ms (1.4%)	1
Self time		0.061 ms (0%)	1
consys.social.memcached.dao.spymemcached.SpyMemcachedImpl.<i		2.18 ms (0.8%)	1
Self time		1.22 ms (0.5%)	1
consys.social.core.dao.WallDaoImpl.<init> ()		0.064 ms (0%)	1
consys.social.core.service.AuthenticatedUser.setUser (consys.social.c		0.053 ms (0%)	52
consys.social.core.bo.WallPostType.<init> (int)		0.018 ms (0%)	50
consys.social.core.dao.FollowDaoImpl.<init> ()		0.009 ms (0%)	1
consys.social.core.bo.Entity.<init> (String)		0.005 ms (0%)	6
consys.social.core.service.FollowServiceImpl.<init> ()		0.004 ms (0%)	1

Figure 2. Profiling

second and the median for loading a simple page which performed one follow operation was 290 ms. The most important page's (News Feed) throughput is 44 requests per second and median was 354 ms. Memcached heavily improved the loading time as data load operation of getting all followers improved almost 1000 times (considering only data read time) as the operation needs a single request to RAM. The testing data from the pilot run and load tests look really promising, as the load times are short for common hardware. Next tests we would like to perform will show the results of this system on cloud services, robustness of horizontal scaling under heavy load and we also want to perform tests to compare HBase with MySQL or any other relational database. This software architecture and data model show how the social subsystem can be

implemented using non-relational databases to allow simple horizontal scaling as the data amount grows, high throughput and handling of heavy data loads.

3. Maintainable NoSQL data model using Apache Cassandra

This section shows a simple method for developing Data Layer and Data Access Layer for Social Network written in Java using Cassandra NoSQL database (see Lakshman & Malik, 2010). To get more information about NoSQL data stores see Stonebraker (2010, 2011). To access the Cassandra we use Hector API (Echague et al., 2012) – the most mature API for accessing the Cassandra today. The data layer is a crucial part of any implementation of social network. Such implementation must meet the criteria of

1. maintainability
2. usability
3. transparency
4. compile time checking
5. verification.

The (3) above means that the layer is understandable and comprehensible for new developers, similarly (1) and (2). We are inspired mainly by Fowler (2002) and Larman (1997).

Lastly, the (4) and (5) are something more unique. Java programming language is a language for which an extensive unit testing is very typical. It is mainly due to the fact that pioneers of the Test Driven Development (TDD) come from the Java background (Koskela, 2007). To achieve (4), the developers must use well structured, object oriented, Java API to access their data model. The definition of "well structured" may be disputed, but a programmer with average experience can relatively easily - after a short period of experience with the API - say whether compile time checking may reveal possible problems in this API (see Martin, 2008). If the API does not meet this well structured criterion, the developer has to develop a general implementation. We have come to the conclusion what it means to be the general wrapper. This can be summarized by *Definition of DAL API Generality*: "The Data Access Layer API implementation is said to be general when the implementation need not be changed due to the business requirements".

The Generality is a necessary condition that has to be met for a maintainable NoSQL Data Model.

The (5) is a well known requirement for any piece of software written today. To achieve it, we suggest TDD. In fact, from empirical experience, the TDD works very well with DAL implementations. There are many reasons for that:

- DAL has well defined inputs and outputs.
- Implementation (querying, creating, deleting) is more complicated than testing.
- It is time consuming to test DAL manually, mainly because of the setup/teardown of test methods. Also note that manual DAL testing is error

prone because the tester will give a lot of false positives.

Justification of reasons why TDD works well with DAL is unfortunately out of the scope of this paper.

Simply by using Java programming language and Cassandra NoSQL database, the solution gets several extra benefits for free:

- openness
- multi-platformity
- easily test driven
- robustness.

Both Java and Cassandra NoSQL database are free of charge and multi-platform – running under JVM. The robustness of Cassandra may be claimed because it was used as Facebook's backing storage for inbox search. Finally, it is possible to embed Cassandra into automated tests.

The goal of this section is to explain how to achieve data model for a social network. A small part of data model for the social network is given. A test driven approach is applied for this model and data access layer (DAO) classes are defined. The DAO layer is a way for a programmer to access the actual data. The rest of the section is devoted to explaining aspects of the DAO layer and TDD to give a detailed insight into the techniques.

3.1. Example

In this section we show how data can be queried. We also present basic objects to access the DAL layer of the social network. Almost every social network should contain the entity *Person*. Assume that Person has two attributes: *name* and *email*. Such a data model is implemented in Java by creating POJO (plain old java object) – object that has no dependencies but Java SDK (standard development kit). Listings for *Person.java* shows a possible person implementation.

Person.java

```
public class Person  implements Serializable {
  private String id;
  private String name;
  private String email;
  //Getters and setters
  ...
}
```

Another part of the data model is the Cassandra Layer. To create an entity in the NoSQL database, Column Families abstraction is used. It is out of the scope of this paper to introduce this concept. The basic idea is that column families are created from source code. This allows good automation and maintainability. Following listing shows such a usage for the Person class:

CassandraBootstraper.java

```
public class CassandraBootstraper {
    public void recreateKeyspace(){
    ...
```

```
      ColumnFamilyDefinition cfd =
        HFactory.createColumnFamilyDefinition(
            keyspace,
            DBConstants.CF_PEOPLE,ComparatorType.UTF8TYPE);
      c.addColumnFamily(cfd);
    }
}
```

Note that when working with the NoSQL database, we are not creating any column definitions. We just define the column family for holding collection of Person data objects. We do not define the schema for attributes (name or email) in any way. Those are added at runtime of the application. The last important piece of code is DAO – Data Access Object. This object is responsible for CRUD operations (Create Read Update Delete) on the entities. There is one DAO for each Entity, hence PersonDAO. Lets take a look at an example of PersonDAO.

PersonDAO.java

```
@Repository
public class PersonDAO extends DAO {
   ...
   public List<Person> findUsersByName(Set<String> names) {
     Rows<String, String, String> result =
          findRows(DBConstants.CF_USERS,
          names, new StringSerializer());
     List<Person> users = parseUsersFromResult(result);
     return users;
  }
}
```

The example shows the read method for getting Person data objects from NoSQL database by their names. The PersonDAO extends the base *DAO* class that contains utility methods like *findRows*. The method *parseUsersFromResult* is a private method of PersonDAO for parsing the *name* and *email* from the database.

3.2. Data layer

While basic Entity-Relationship modeling techniques are well known and studied for years, the NoSQL databases require a different approach. Data in NoSQL database is highly denormalized to gain performance. The data also allow great flexibility in adding new attributes to entities already in the database. When building social network, it is advisable that no proprietary scripts are introduced.

By the Generality Definition we can create a general implementation of *bootstrapping mechanism*. The mechanism is based on the idea that all the test data + schema of the database will be created using the same programming techniques (Java) as is used at runtime. The programmer should use a dedicated class. We name this class *Bootstrapper* in our case. This class has the following responsibilities:

- connect the Cassandra instance

- create schema in empty Cassandra database
- insert test data.

Bootstrapper helps better maintainability because information about schema are versioned in this Bootstrapper class (using Subversion). Bootstrapper also helps testability of code, because unit tests can directly invoke Bootstrappers methods.

To implement a DAO Layer to access Cassandra Database it is advisable to use Hector API because it gives a lot of enterprise level features out of the box. The API introduces a lot of clutter. That is why the best approach to implement DAO is to create base class with general implementation for following actions:

- findRows (columnFamily, keys, resultSerializer) - finds rows with given keys
- findAllRows (columnFamily) - all rows in given family
- findAllObjectRows (columnFamily, objectColumnName) - deserializes the object from given column
- deleteColumn, addColumn, findObjectColumn.

By creating this abstraction the developer can tinker it for a specific social network.

Another important technique to be used for Java+Cassandra DAO layer is *Aspect Oriented Programming* (AOP). We introduced *ErrorHandlingAspect* that is responsible for improving error handling on DAL. The AOP is invaluable in these situations. Reasons stem from the nature of DAL. DAO objects have methods specific for the given entity (Person may have *attachMessage*, which is unique for this entity). It is important that the exception be caught inside of these methods to have bigger picture of the error (parameters, name of the method) and not having to inspect the stack trace. AOP gives us this flexibility. The only thing needed is to declare *AfterThrowing* aspect on all DAO methods. Apart from the error handling, by using AOP we can easily log parameters entering each DAO method.

TDD is a very advisable technique for DAL implementation for applications using NoSQL Database. As the data is highly denormalized and unstructured, it is easy to introduce regression bugs into the code. In-memory Cassandra instance can help to create test suites.

4. Reactive social network monitoring

When studying implementations we came to the conclusion that high volume traffic applications like social networks need very specific approach to their monitoring and also special reactive rules for certain actions happening in the system. This section lays out our findings and latest results from this area. Our results contain both the general framework for deploying such monitoring/reactive infrastructure and best practices to use them.

4.1. Monitoring

The monitoring is a basic activity done in any system being in the production or testing phase. We have found out that in many cases, current monitoring options for data model state changes are not sufficient for such large scale systems because of the following reasons:

- The amount of data generated by social networks is enormous.
- Data changes in social networks may not be deterministic. Furthermore they can be without strict time successions due to a distributed nature of the data model. This is acceptable from the point of view of the user because social networks do not usually contain any time critical information.

Having the data on a *logical node* is very convenient (logical node is, for example, one pluggable component in our layered architecture). The convenience materializes itself in terms of having the right monitoring information in one place for processing. Several problems arise with this approach. A badly written component (*logical node*) may create a bottleneck itself for monitoring information processing. However, the biggest problem is that logical nodes do not respect availability of data over the cluster of data nodes. Data must be moved back and forth between nodes causing serious performance issues.

Data node is, for example, Apache Cassandra cluster node instance. This is the best option from the performance point of view. Having the monitoring information on this node data is not transferred via network nor moved across different file systems for processing. There are, however, certain subtle problems with this approach. No correlation can be done for disparate monitoring information. However, these disparate monitoring information pieces are often related.

4.2. Reactive rules

During the implementation of our case studies – social networks – we were often faced with the necessity to react to specific extreme cases in a dynamic fashion. For example, when a user sends too many notifications per minute, his/her throughput of messages should be limited until his/her payment credibility is checked. Another example might be a situation where continually more respected user has his/her limits relaxed (usual functionality of cloud services from Amazon).

4.3. Methods

At first, we thought that our problems with monitoring and reactive rules are not related. We believed that a combination of *business process execution language* and modification of Apache Cassandra logging system will solve the issues. However, we have found a common solution that solves both problems. We introduced *Complex Event Processing* into the monitoring processing and reactive rules processing. The CEP concept is introduced in Etzion (2011) and Luckham

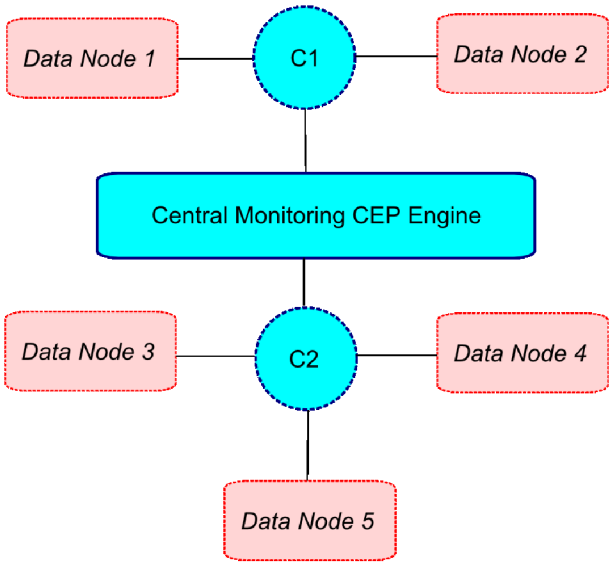


Figure 3. Data monitoring

(2002). The viability of using CEP in distributed environment is supported by several studies in this area, e.g. Lakshmanan, Rabinovitch & Etzioni (2009) and Luckham & Frasca (1998).

CEP solves the dilemma of placing monitoring onto the *logical node* and a *data node*. We choose to put the CEP engines on data nodes and connect them based on logical partitioning done by Cassandra. In this way, different Cassandra nodes are monitored together. The situation is depicted in Fig. 3. In this figure we see five data nodes. A central monitoring CEP engine is used to correlate all monitoring information on high level of abstraction. It does not get all the fine levels of logging messages (e.g. save of particular discussion post in social network) but receives only high-level events that will indicate that some data nodes need a finer analysis. In Fig. 4 we see that central monitoring engine decided that *Data Node 1* and *Data Node 2* should be analyzed together. Analogically, the rest of the data nodes were assigned with *C2* monitoring engine. This architecture allows us to scale monitoring infrastructure very flexibly and automatically.

The issue with reactive rules is solved by publishing events to the service tier of the application from the data tier. Fig. 4 depicts this connection. The connection is in fact the same approach as the solution to the monitoring. We use the monitoring information to gather "hard to get" information without need to go deep into the application logic. In this way we can achieve both scenarios mentioned in the previous section. We can easily monitor how many notifications user sent in a given time window, and through connection to the service layer we can publish this monitoring information. The service layer picks

up this event and acts accordingly by limiting user quota.

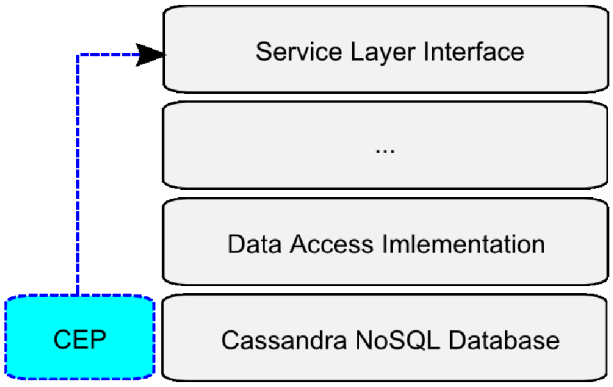


Figure 4. Data layer to Service layer connection

5. Conclusion

In this paper we have presented our current research and development results in the area of social network system development. We showed proper usage of existing frameworks, languages and rationale behind their usage. For a developer, project manager or businessman, being aware of existing tools, their proper implementation and foresight of the future development with close collaboration with users will help to achieve success of the platform and its establishment on the market.

References

Echague, P., McCall, N. et al. (2012) *Hector – A high level Java client for Apache Cassandra.* http://hector–client.github.com/ hector/ build/ html/ index. html. Visited at 11/1/2012.

Etzion, O., Niblett, P. (2011) *Event Processing in Action.* Manning Publications, Shelter Island, USA.

Fowler, M. (2002) *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, USA.

Gartner (2012) Gartner Identifies the Top 10 Strategic Technologies for 2012 http://www.gartner.com/it/page.jsp?id=1826214. Visited at 11/1/2012.

Koskela, L. (2007) *Test Driven: TDD and Acceptance TDD for Java Developers.* Manning Publications, Shelter Island, USA.

Lakshmanan, G. T., Rabinovich, Y. G., Etzion, O. (2009) A stratified approach for supporting high throughput event processing applications. *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, Article 5. ACM, New York, USA.

Lakshman, A., Malik, P. (2010) Cassandra - A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review archive*, **44**, 2, 35-40.

Larman, C. (1997) *Applying UML and Patterns.* 1st edition. Prentice Hall, Boston, USA.

Lin, J., Dyer, C. (2010) Data-intensive text processing with Mapreduce. *Synthesis Lectures on Human Language Technologies*, Morgan and Claypool Publishers, USA.

Luckham, D. (2002) *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Professional, USA.

Luckham, D., Frasca, B. (1998) *Complex Event Processing in Distributed Systems.* Standford University, **28.**

Martin, R. (2008) *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall, Boston, USA.

Stonebraker, M. (2011) Stonebraker on NoSQL and enterprises. *Communications of the ACM*, **54**, 10-11.

Stonebraker, M. (2010) SQL databases vs. NoSQL databases. *Communications of the ACM*, **53**, 10-11.

Škrabálek, J., Tokárová, L., Slabý, J. and Pitner, T. (2011) *Integrated Approach in Management and Design of Modern Web-Based Services.* Springer, New York, USA.

Takeplace (2012) An Event Management System http://take-place.com. Visited at 11/1/2012.

White, T. (2009) *Hadoop: The Definitive Guide.* O'Reilly Media, California, USA.

Zichermann, G. and Cunningham, C. (2011) *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps.* O'Reilly Media, Canada.

A.5.5 NotX service oriented multi-platform notification system

Paper [67]

NotX
Service Oriented Multi-platform Notification System

Filip Nguyen[1] and Jaroslav Škrabálek[1]

Faculty of Informatics Masaryk University, Brno, Czech Republic
nguyen.filip@mail.muni.cz,skrabalek@fi.muni.cz

Abstract. This report describes NotX - service oriented system, that applies ideas of CEP and SOA to build highly reusable, flexible, both platform and protocol independent solution. Service oriented system NotX is capable of notifying users of superior information system via various engines; currently: SMS engine, voice synthesizer (call engine) and mail engine. Adaptable design decision makes it possible to easily extend NotX with interesting capabilities. The engines are added as plug-ins written in Java. There are plans to further extend NotX with following engines: Facebook engine, Twitter engine, content management system engine. Also the design of NotX allows to notify users in theirs' own language with full localization support which is necessary to bring value in today's market. Most importantly, the core design of NotX allows to run under heavy load compromising thousands of requests for notification per second via various protocols (currently Thrift, Web Services, Java Client). Thus NotX is designed to be used by state of the art Enterprise Applications that require by default certain properties of theirs' external systems as scalability, reliability and fail-over.

Key words: information system, soa, cep, notifcation system

1 Introduction

Notifications have been studied as valuable tool in context of *ubiquitous computing* [4] and little more simplistic version of them (email notifications) are present in almost every *information system* as a standard approach to notify (and prompt) user in the case of password change, registration approval or account state change. But the real power of notifications come when there is more sophisticated business logic associated with generation of these events such as in [5]. Other useful applications of such notification service are areas where traditional paper based communication/notification means are used [6]. Consider simple example - in information system dedicated to organize academic conferences user would expect to receive notifications about paper submission and paper approval or rejection. They would also expect to be notified about other more real-time events like reschedule of certain presentation. This kind of business logic is usually system-specific but means of delivering these notifications

are usually the same: email or SMS. There is one additional channel that we find very useful (also indicated in [1]) and that is voice channel, namely text to speech synthesis delivered into cellular network. Because of repetitive use of this notification infrastructure (e.g. [7]) it would be beneficial to create service that would provide all these notification means.

In this report we describe service that complies to above criteria - NotX. In first part of paper we describe business requirements that are relevant for such service. Then the actual architecture and technological details of NotX are presented. Last part of paper is dedicated to discuss development process used to drive NotX development and possible directions of further work on NotX.

2 Business requirements

NotX's first deployment hence first real use case is to serve as a notification service to Takeplace information system to send various notifications including:

- emails with password change/registration
- rescheduling of presentation (this is typically delivered by SMS or voice)

Notification is sent dynamically via appropriate engine according to user setting and global NotX setting. Voice and SMS engines are very fast way to notify user but use of these engines are charged so theirs use is not unlimited and must be controlled.

Voice notifications can be delivered into cellular network or to SIP. Motivation for SIP can be found in [8].

Because it is anticipated that use of notifications will be massive and certain groups of users will be repetitively notified (for example attendees of certain conference) we demand *tagging* of users. Information system developer (IS developer) should be able to send notifications to either specific user or to specific tag.

Required operations to be performed via NotX are:

- tag (userid, tag)
- unTag (userid, tag)
- sendNotification (dest, msgType, templateName, placeholderVals)

The *tag* parameter in *tag* and *unTag* is text with '.' characters permitted, e.g. *ConfernceA.attendee* or *ConferenceA.speaker*. The first part of *tag* parameter up to '.' is called *domain*. Tag doesn't have to include the domain, the domain is used only for billing and statistical purposes. The *userid* is unique identifier of user to be tagged. The *sendNotification* operation is used to send notification itself. The parameter *dest* is used to specify to which entity the message should be sent. It can be either userid prefixed with ':' or it can be tag. When tag is used in this parameter the message will be sent to all tagged users. Parameter *msgType* is used to add more semantic to message, e.g.: *important*. Administrator of NotX can use this semantic parameter to configure NotX to send all *important* messages via predefined engine (for example TTS). Next parameter *templateName*

is used to specify which message should be sent, for example template for registration approval *registration_approval* is template of message that is sent when registration process is successful. Lastly *placeholderVals* is associative array that is used to inject values into template.

Takeplace itself is distributed web application and has many different developers that are experienced in various technologies (ranging from PHP to Servlets) hence every one of these developers is used to different way of accessing services. NotX should take this into account and make it as easy as possible to access NotX service.

Next important requirement is concerned with internationalization. Because academical conferences are usually attended by participants from various countries it's convenient for them to receive notification's in their own language. This is important equally for voice, sms and email notifications.

Because NotX uses charged services like cellular network the NotX has to keep track of sent notifications with information about domain to which they were sent.

Regarding nonfunctional requirements, the most important is to handle peaks of notifications with persistent fail-over. Usually if there is one big notification for all participants of major conference there can be thousands of various messages sent via email, SMS or voice synthesizer (TTS). It's not necessary to deliver all notifications at once but system should not render unresponsive or shouldn't crash and all messages should be delivered.

3 Architecture

NotX is developed to be scalable platform and protocol independent Service. Currently the NotX is deployed to serve as a service for Takeplace so thousands of messages can arrive per second at peak hours.

Necessary attribute of reusable software service is it's platform independence. That's why NotX and it's components are built using Java programming language. NotX itself is web application that is built using build tool Maven [2].

Main output of build process are 2 war archives one is Notx.war and Communication.war which is web application that exposes protocols used as a interfaces into NotX. These main components are depicted on figure 1. These wars correspond to main components of NotX architecture - the core logic itself (NotX) and communication module (the Communication.war).

JMS (JSR 914) is specification for messaging API between loosely coupled components of information system. We are using Apache Active MQ implementation which supports persistent fail-over. Considering fail-over there are several fails that can happen during NotX's lifecycle:

1. Problem with external engine provider (SMS or voice)
2. Problem with connectivity to communication module with NotX
3. Bug in NotX logic
4. Any fail of hardware while processing notification

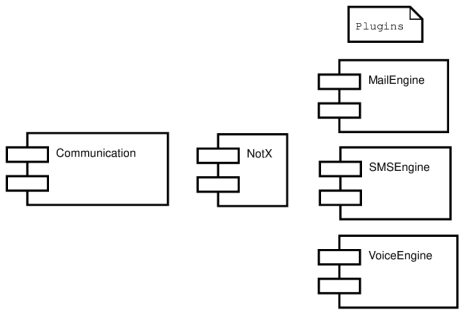


Fig. 1. Components of NotX

To address all of these problems our architecture is queue centric as seen in overall design in figure 2. After receiving request for notification the communication module immediately sends the request into persistent queue.

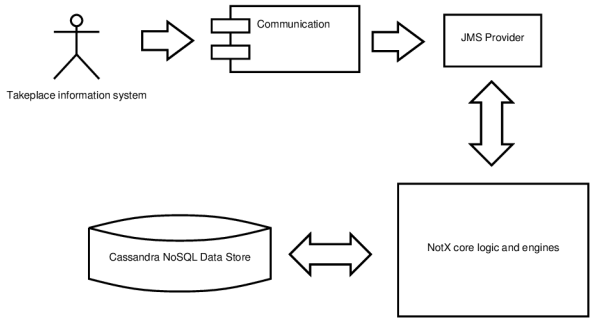


Fig. 2. Overall design

The most important operation of NotX is *sendNotification*. We will describe core logic behind this operation in more detail. As noted in business requirements this operation takes 4 parameters: *dest*, *msgType*, *templateName*, *placeholderVals*. Important logic takes place when *sendNotification* is called and destination is set to some specific tag, for example *ConferenceA.attendees*. Following steps take place after request has arrived to communication module:

1. Communication module recognizes the request as notification request and puts new notification request message A into message queue
2. NotX logic starts processing A by looking up N users which are to be notified by notification in A. Then NotX generates N messages $\{A_1, ..., A_N\}$ and puts them back into message queue.
3. Note that up to this point there was no interaction with any engine. Now NotX will be continuously receiving messages $A_x \in \{A_1, ..., A_N\}$ from message queue and each such message is processed as follows:
   a) Finds language L of user for whom A_x is dedicated.

b) Looks up template for A_x according to L
c) Now NotX injects *placeholderVals* into template and uses selected engine to notify the user. If this whole process is successful then notification statistic is saved into data store.

If there would be any kind of problem with data store or connectivity the messages are kept in message queue for administrator to manually decide how to deal with them.

Very important is 2. The generation of $A_1, .., A_N$ messages helps to more evenly distribute load on the system and also helps traceability of the system. For example when notification request is to be processed for *ConferenceA.attendees* that can mean notification of 1000 users. When even 1 notification fails it is beneficial to know which one failed and why. After bug fixing it's important to be able to swiftly retry sending exactly the same notification as failed previously.

3.1 Protocol access

Requirement for protocol access may occur in many contexts. NotX provides following interfaces (as depicted in figure 3):

- JSON interface via HTTP POST for simple notification sending
- Thrift interface for higher level languages (framework for cross-language service development)
- native Java libraries
- web services

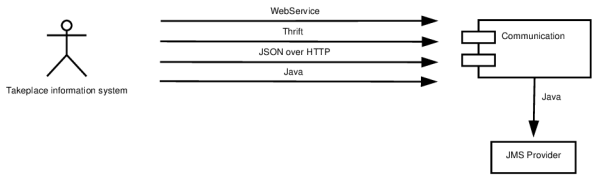


Fig. 3. Protocols

Adding new communication protocol is fairly easy. It means modifying Communication module, which resides in directory *src/notx-communication*. When adding new protocol, it is necessary to implement all NotX methods. Each method implementation usually just creates standard JMS message and puts it into MQ. Then only modification of *CommunicationMain* class will make sure that after launch of communication module the interface into NotX will be functional.

3.2 Data storage

NotX uses data store for:

– users and theirs' tags
– statistics
– fail-over

Each user has his contacts stored in data store. This way NotX is able to send notifications by any engine for this particular user.

Statistics are saved mainly to charge users of paid services and performance tinkering.

Data storage is also used as a fail-over mechanism. Whenever a notification message is not sent successfully it is saved into data store and can be viewed via web interface with exception that caused the failure. It's possible to send specific failed notifications back to the message queue to retry the sending.

The data store itself is implemented using Cassandra NoSQL database. Decision to choose this storage type was led by need for multi-platform and highly scalable data store.

4 Development process

The development process of NotX was driven by SCRUM methodology. This agile process introduced by Schwaber and Shuterland [9] suits development of the NotX best because its requirements was from start more about searching of possibilities instead of launching repeatable processes.

SCRUM itself is being used with 2 week sprints (sprint is one iteration in SCRUM). Each sprint starts with sprint planning, where spring goal is presented (major functionality, or tangible goal that is to be produced by this sprint) and product backlog items for this sprint are presented (product backlog item is high level business requirement). SCRUM itself doesn't give many hints how to specify backlog items, but there are publications addressing this issue e. g. [10] which introduce *user stories* into SCRUM.

Then development proceeds and at the end of the sprint *sprint review* takes place where output of an iteration is presented. In NotX settings the sprint review and sprint planning took place same day, usually at Wednesday.

Product backlog as well as sprint backlog are kept in Open Office spreadsheet. This low tech approach always yields less administration and more focus on actual work. From backlog it can be derived how much work was spent on specific product backlog item each day and how well estimated the task was.

To our knowledge there are not major modifications of SCRUM methodology for web development (also in [11] there wasnt found any consistent difference in decision making for web projects). There are however some subtle differences when developing system as NotX in general (not just with agile practices):

– External tools spike first
– Sprint review should contain technological details
– Sprints to refactor code has to be more explicitly specified
– More focus on automatized integration testing

Its essential that each external tools like TTS system or SMS gateway that are used to carry out notifications are spiked first before adding any product backlog items that are dependent on this TTS. We recommend to have sprint in which external tools are examined. Such sprint helps at planning next sprints because developers of NotX can help product owner to prioritize and estimate product backlog items that will include external tool usage.

Sprint review should include technological details because product owner represents technologically experienced users (developers of information system).

Sprints to refactor code has to be specified very explicitly with carefully formulated sprint goal. Sprint goals of these sprints shouldnt be vague or not measurable like: create more readable code. But there should be measurable goals e. g.:

- write automated test that will fire up in-memory database and performs CRUD operations
- rewrite logic of configuration loading and present this new design using class diagram and sequence diagram at sprint review

Focus on integration testing comes from the fact that NotX itself uses several external systems and lot of logic is simple orchestration between JMS provider and external engines. This makes unit testing less effective.

All points above can be addressed with SCRUM by managing content of product backlog and sprint reviews.

5 Further work

In future, we plan to extend NotX to be publicly available service to be used by any IS developer. Technically it is possible right now because NotX supports lot of protocols for communication. There are, however, some missing functionalities like IS developer registration or billing reports.

Last important way to add more functionality to NotX is extending its communication module. There are many possibilities:

- Facebook engine - engine that notifies directly into accounts wall or private message
- Twitter engine - sends the notification to twitter
- FTP/SCP engine - puts the notification on FTP server or via SCP on some server
- IRC engine
- Skype engine - calling by skype. We didn't tested feasibility of this option yet.

6 Conclusion

In this paper we reported state of NotX - Service with capability of sending notifications via various engines. NotX gives value added to information system

developers by taking burden of setting up infrastructure to send SMS, voice and email notifications. Additionally NotX helps with contacts management as it stores the contact information about users and doesnt reveal those contacts to IS developer.

NotX reduces time to integrate interesting functionality for any new information system with low development time and bring out of the box governance capabilities like fail-over, statistics and large scale notification sending in various languages.

Acknowledgment

The authors would like to thank Pallo Grešša for refining architecture of NotX and also to Lukáš Rychnovský for ideas from CEP and experience with building large scale distributed application that he shared.

References

1. Kyuchang Kang, Jeunwoo Lee and Hoon Choi, "Instant Notification Service for Ubiquitous Personal Care in Healthcare Application" in *International Conference on Convergence Information Technology* 2007 pp. 1500-1503
2. Apache Maven Project
http://maven.apache.org/
3. Apache Tomcat
http://tomcat.apache.org/
4. Schmandt, C. and Marmasse, N. and Marti, S. and Sawhney, N. and Wheeler, S. "Everywhere Messaging" in *IBM Syst. J.*, vol. 39, issue 3-4, July 2000, p. 660-670
5. J. Jeng and Y. Drissi, "PENS: A Predictive Event Notification System for e-Commerce Environment," in *The Twenty-Fourth Annual International Computer Software and Applications Conference*, October 2000.
6. Chi Po Cheong; Chatwin, C.; Young, R.; , "An SOA-based diseases notification system," in *Information, Communications and Signal Processing, 2009. ICICS 2009. 7th International Conference* on , vol., no., pp.1-4, 8-10 Dec. 2009 doi: 10.1109/ICICS.2009.5397519
7. Mohamed, Nader Al-Jaroodi, Jameela Jawhar, Imad A generic notification system for Internet information in *Information Reuse and Integration, 2008. IRI 2008. IEEE International*
8. A. Sadat , G. Sorwar , M. U. Chowdhury, "Session Initiation Protocol (SIP) based Event Notification System Architecture for Telemedicine Applications," in *1st IEEE/ACIS International Workshop on Component- Based Software Engineering, Software Architecture and Reuse (ICISCOMSAR' 06)*, pp. 214-218, July 2006.
9. Ken Schwaber, Mike Beedle Agile *Software Development with Scrum* Prentice Hall, 2001
10. Mike Cohn *User Stories Applied For Agile Software Development* Addison-Wesley, 2010 ISBN:0-321-20568-5
11. Carmen Zannier and Frank Maurer Foundations of Agile Decision Making from Agile Mentors and Developers in *Extreme Programming and Agile Processes in Software Engineering*, June 2006, LNCS 4044, p. 11-20

A.5.6 Co-authored IDC 2013

Paper [58]

Distributed Event-driven Model for Intelligent Monitoring of Cloud Datacenters

Daniel Tovarňák, Filip Nguyen and Tomáš Pitner

Abstract When monitoring cloud infrastructure, the monitoring data related to a particular resource or entity are typically produced by multiple distributed producers spread across many individual computing nodes. In order to determine the state and behavior of a particular resource all the relevant data must be collected, processed, and evaluated without overloading the computing resources and flooding the network. Such a task is becoming harder with the ever growing volume, velocity, and variability of monitoring data produced by modern cloud datacenters. In this paper we propose a general distributed event-driven monitoring model enabling multiple simultaneous consumers a real-time collection, processing, and analysis of monitoring data related to the behavior and state of many distributed entities.

1 Introduction

With the emergence of distributed computing paradigms (e.g. grid) the importance of monitoring steadily grew over the past two decades and with the advent of *cloud computing* it rapidly continues to do so. When monitoring a distributed infrastructure such as grid or cloud, the monitoring data related to a particular entity/resource (e.g. message queue, Hadoop job, and database) are typically produced by multiple distributed producers spread across many individual computing nodes.

A two-thousand node Hadoop cluster (open-source implementation of MapReduce) configured for normal operation generates around 20 gigabytes of application-level monitoring data per hour [3]. However, there are reports of monitoring data rates up to 1 megabyte per second per node [4]. In order to determine the state

Daniel Tovarňák, Filip Nguyen and Tomáš Pitner
Masaryk University, Faculty of Informatics
Botanická 68a, 60200 Brno, Czech Republic
e-mail: xtovarn@fi.muni.cz, e-mail: xnguyen@fi.muni.cz, e-mail: tomp@fi.muni.cz

and behavior of a resource all the relevant data must be collected, processed, and evaluated without overloading the computing resources and flooding the network.

In our research we are particularly interested in behavior monitoring, i.e. collection and analysis of data related to the actions and changes of state of the monitored resources (e.g. web service crash) as opposed to the monitoring of measurable state (e.g. disk usage). The goal of state monitoring is to determine if the *state* of some resource deviates from normal. Our goal, on the other hand, is to detect *behavior* deviations and their patterns.

The volume, velocity, and variability of behavior-related monitoring data (e.g. logs) produced by modern cloud datacenters multiply and there is a need for new approaches and improvements in monitoring architectures that generate, collect, and process the data. Also the lack of multi-tenant monitoring support and extremely limited access to provider-controlled monitoring information prohibits cloud customers to adequately determine the status of resources of their interest [12]. As the portfolio of monitoring applications widens the requirements for monitoring architecture capabilities grow accordingly. Many applications require huge amounts of monitoring data to be delivered in *real-time* in order to be used for intelligent *online* processing and evaluation.

The goal of this paper is to propose a novel distributed event-driven monitoring model that will enable multiple simultaneous consumers to collect, process, and analyze monitoring data related to the behavior and state of many distributed entities in real-time. The rest of this paper is organized as follows. In Section 2 we present basic terms and concepts used in this paper. Section 3 deals with the proposed distributed event-driven monitoring model. Section 4 concludes the paper.

2 Background

In this section we introduce basic terms, concepts, and principles that our model is founded on. Based on these principles we incrementally design basic cloud monitoring model that we build upon later.

We define **monitoring** as a *continuous and systematic collection, analysis, and evaluation of data related to the state and behavior of monitored entity.* Note that in the case of distributed computing environment (such as cloud) its state and behavior is determined by the state and behavior of its respective constituents (components). *State* of monitored entity is a measure of its behavior at a discrete point in time and it is represented by a set of state variables contained within a state vector [6]. *Behavior* is an action or an internal state change of the monitored entity represented by a corresponding event [6]. Computer logs (system logs, console logs, or simply *logs*) are widely recognized as one of the few mechanisms available for gaining visibility into the behavior of monitored resource [9] regardless if its operating system, web server or proprietary application. Therefore in our work we consider logs (log events) to be the primary source of behavior-related information.

To properly describe the respective phases of *monitoring process* we adhere to its revised definition originally introduced by Mansouri and Sloman in [7]. Terms used to denote entities participating in the process are based on the terminology presented in Grid Monitoring Architecture [11] and later revised by Zanikolas and Sakellariou in [13]. The monitoring process is composed of the following stages: *Generation* of raw monitoring data by sensors; *Production*, i.e. exposure of the data via predefined interface; *Distribution* of the data from producer to consumer; its *Consumption* and *Processing*.

The production, distribution, and consumption stages are inherently related – the distribution depends on the fashion the monitoring data are produced, and consequently, the consumption is dependent on the way the data are distributed. Therefore, the three stages will be collectively referred to as **monitoring data collection**. Also, there is a difference between online and offline monitoring data processing and analysis. An online algorithm processes each input in turn without detailed knowledge of future inputs; in contrast an offline algorithm is given the entire sequence of inputs in advance [1]. Note that the use of online algorithms for processing and analysis do not necessarily (yet more likely) lead to real-time monitoring, and vice-versa, the use of offline processing algorithms do not necessarily prohibit it.

In our work we consider physical and virtual machines to be the primary producers of monitoring data via the means of their operating systems (without any difference between host and guest OS). For simplicity's sake we do not consider virtualization using bare-metal hypervisor.

Meng [8] observed that in general, monitoring can be realized in different ways in the terms of distribution of monitoring process across the participating components.*Centralized data collection and processing*, i.e. there is only one single consumer; *Selective data collection and processing*, i.e. the consumer wishes to consume only a subset of the produced data; *Distributed data collection and processing*, i.e. data are collected and processed in a fully decentralized manner.

In general, the communication and subsequent monitoring data transfer can be initiated both by consumer (i.e. *pull model*) and producer (i.e. *push model*) [10]. In the pull model the monitoring data are (usually periodically) requested by consumer from the producer. On the other hand, in the push model the data are transferred (pushed) to the consumer as soon as they are generated and ready to be sent (e.g. pre-processed and stored).

3 Distributed Event-driven Monitoring Model

In this section we further extend the basic cloud monitoring model with multi-tenancy support whilst leveraging principles of Complex Event Processing to allow for real-time monitoring of large-scale cloud datacenters. The resulting model allows multiple simultaneous consumers to collect, process, and analyze events related to the behavior of many distributed entities. State-related events are also supported to allow for state-behavior correlations.

To achieve data reduction the data collection is primarily based on publish-subscribe interaction pattern for the consumers to specify which monitoring data they are interested in. More importantly, the subscription model allows for definition of complex patterns and aggregations following a *pattern-based* publish-subscribe schema. To reasonably process a tremendous amounts of monitoring data generated by cloud datacenters distributed collection and processing is primarily considered. From the data evaluation point of view the goal is to allow consumers to collect events from many distributed sources, define complex subscriptions and consequently analyze and evaluate the incoming highly-aggregated events in their own way. The monitoring architecture following this model is intended to be the part of provider's cloud infrastructure.

Historically, traditional DBMSs (including *active databases*) oriented on *data sets* were not designed for rapid and continuous updates of individual data items required by online monitoring discussed in this paper and performed very poorly in such scenarios. As pointed out by Babcock et al. [2], to overcome these limitations a new class of data management applications emerged: Data Stream Management Systems (DSMSs) oriented on evaluating *continuous queries* over *data streams*. According to Babcock, data streams differ from the conventional relational data model in several ways: (1) the data elements in the stream arrive online; (2) the system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams; (3) data streams are potentially unbounded in size; (4) once an element from a data stream has been processed it is discarded or archived – it cannot be retrieved easily unless it is explicitly stored in memory, which is typically small relative to the size of the data streams.

Full-fledged DSMSs typically allow for quite expressive queries supporting many standard operations (e.g. averages, sums, counts) and also windows (e.g. sliding window, and pane window) to specify the particular portion of the incoming data stream. As pointed out in [5] whilst considerably capable, DSMSs are still focused on the traditional relational data and produce continuously updated query results (e.g. as output data stream). Detection of complex patterns of elements involving sequences and ordering relations is usually out of the scope of DSMSs.

Complex Event Processing (CEP) in general follows the same goals and principles as DSMSs, yet as it is apparent from the term, it is focused on the processing of a very specific type of data elements – events (event is an occurence within a particular domain – computing infrastructure monitoring in our case).

In our previous work [12] we introduced the concept of event-driven producer of monitoring data using extensible schema-based format. We argued that unified representation of monitoring information in the form of events increases data correlation capabilities, makes processing easier, and avoid the complexity of monitoring architecture. Together with standard delivery channel it is an important step towards extensible and interoperable multi-cloud (inter-cloud) monitoring.

In our model, CEP can be perceived as an extension of pattern-based publish-subscribe schema enabling consumers to subscribe for complex (composite) events based on expressive queries, e.g. using sequence patterns, temporal constraints, windows, filters and aggregations. Typically, the complex events can be re-introduced

to the data stream for further processing (i.e. creating new complex events) which we consider to be very powerful. Example 1 represents an example of simple monitoring subscription in SQL-like declarative Event Processing Language used by Esper[1] CEP engine. Subscription S1 subscribes for complex events that can indicate possible password cracking attack using dictionary approach.

```
select hostname, username, success, count(*) as attempts
from LoginEvent.win:time(30 sec)
where attempts > 1000, success=false
group by hostname, username
```

Example 1 Subscription S1 using EPL

The flow of monitoring data (see Figure 1) in our model can be described as follows: The *sensors* generate raw monitoring data related to a specific entity (e.g. Hadoop job, SSH daemon, and CPU). Based on this data, *producers* register and instantiate *simple events* (i.e. an occurrence related to one or more *entities*) with clearly defined structure. *Consumers* then create *subscriptions* to instrument *processing agents* to perform one or more processing functions. Such a function takes stream of *events* as input and outputs a stream of *complex events*. When applicable, a single *subscription* is partitioned into several simpler *subscriptions* and distributed among several *processing agents* (and *producers* as well). The examples of common processing functions include: filtering, sequence detection, aggregation, and anomaly detection. The *consumers* then receive the *complex events* they previously subscribed for. In order to achieve *multi-tenancy* [12], *consumers* can be restricted to subscribe for *events* related to a particular entity.

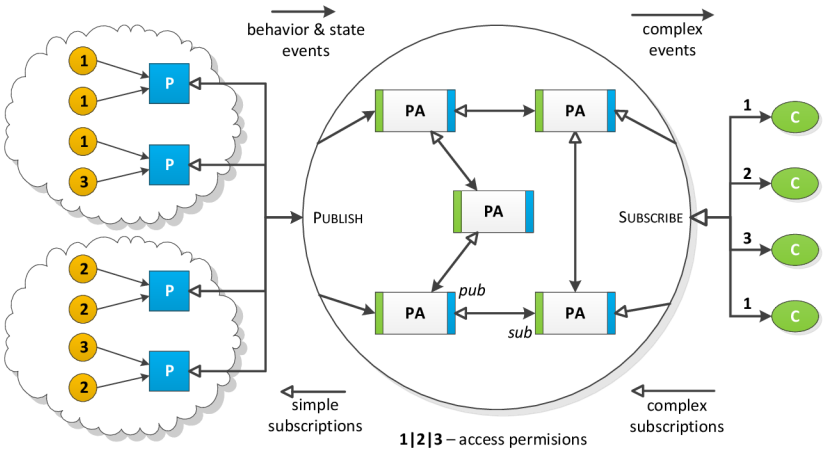


Fig. 1 Overview of Distributed Event-driven Monitoring Model in (Multi-)Cloud scenario

[1] http://esper.codehaus.org/

4 Conclusions and Future Work

In this paper we have presented a distributed event-driven model for intelligent cloud monitoring. An architecture following this model will allow multiple simultaneous consumers to collect, process, and analyze events related to the behavior of many distributed entities in real-time. We expect improvements in many areas when compared to traditional monitoring models based on offline algorithms. In future we plan to implement prototype of monitoring architecture following proposed model which will be then experimentally evaluated in the terms of intrusiveness, network overhead, and throughput with respect to the number of producers, consumers, volume, velocity, the variability of monitoring events, and the complexity and number of queries it is capable of dealing with. Multiple approaches for processing agents' topology, routing, query rewriting, and event distribution will be considered and evaluated.

References

1. ATALLAH, M. *Algorithms and Theory of Computation Handbook, 2 Volume Set.* CRC, 1998.
2. BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., AND WIDOM, J. Models and issues in data stream systems. In *Principles of database systems* (New York, NY, USA, 2002), ACM.
3. BOULON, J., KONWINSKI, A., QI, R., RABKIN, A., YANG, E., AND YANG, M. Chukwa, a large-scale monitoring system. In *Proceedings of CCA* (2008).
4. CREŢU-CIOCÂRLIE, G. F., BUDIU, M., AND GOLDSZMIDT, M. Hunting for problems with artemis. In *Proceedings of the First USENIX conference on Analysis of system logs* (Berkeley, CA, USA, 2008), WASL'08, USENIX Association, pp. 2–2.
5. CUGOLA, G., AND MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv. 44*, 3 (June 2012).
6. MANSOURI-SAMANI, M. *Monitoring of distributed systems.* PhD thesis, Imperial College London (University of London), 1995.
7. MANSOURI-SAMANI, M., AND SLOMAN, M. Monitoring distributed systems. *Network, IEEE 7*, 6 (nov. 1993).
8. MENG, S. *Monitoring-as-a-service in the cloud.* PhD thesis, Georgia Institute of Technology, 2012.
9. OLINER, A., AND STEARLEY, J. What supercomputers say: A study of five system logs. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on* (june 2007), pp. 575 –584.
10. ROSENBLUM, D. S., AND WOLF, A. L. A design framework for internet-scale event observation and notification. *SIGSOFT Softw. Eng. Notes 22*, 6 (Nov. 1997), 344–360.
11. TIERNEY, B., AYDT, R., GUNTER, D., SMITH, W., AND SWANY, M. A grid monitoring architecture. *Global Grid Forum* (2002), 1–13.
12. TOVARŇÁK, D., AND PITNER, T. Towards Multi-Tenant and Interoperable Monitoring of Virtual Machines in Cloud. *SYNASC 2012, MICAS Workshop* (September 2012).
13. ZANIKOLAS, S., AND SAKELLARIOU, R. A taxonomy of grid monitoring systems. *Future Generation Computer Systems 21*, 1 (2005), 163 – 188.