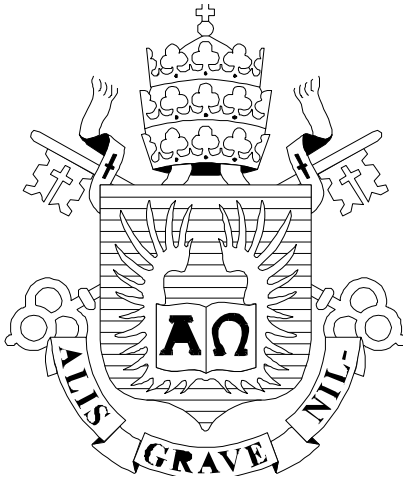


PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 17/13

On-line Detection of Collective Mobility Patterns through Distributed Complex Event Processing

Gustavo Luiz Bastos Baptista
Marcos Roriz
Rafael Vasconcelos
Bruno Olivieri
Igor Vasconcelos
Markus Endler

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 17/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dec., 2013

On-line Detection of Collective Mobility Patterns through Distributed Complex Event Processing

Gustavo Luiz Bastos Baptista, Marcos Roriz, Rafael Vasconcelos, Bruno Olivieri, Igor Vasconcelos, Markus Endler

{gbaptista, mroriz, rvasconcelos, bolivieri, ivasconcelos, endler}@inf.puc-rio.br

Abstract. Applications such as fleet management, mobile task force coordination, logistics or traffic control can largely benefit from the on-line detection of collective mobility patterns of vehicles, goods or persons. However, collective mobility pattern analysis is exponential by nature, requires the high-throughput processing of large volumes of mobile sensor data, and thus generates huge communication and processing load to a monitoring system. Considering the benefits of the event-based asynchronous processing model for on-line monitoring applications, in this paper we argue that several collective mobility patterns can be elegantly described as a composition of reusable Complex Event Processing (CEP) rules, and specifically focus on the detection of the cluster mobility pattern. We also present a DDS-based mobile middleware that supports a distributed deployment of these CEP rules for such collective mobility pattern detection. As means of evaluating our approach we show that using our middleware it is possible to detect this mobility pattern for thousands of mobile nodes, with a latency that is adequate for most monitoring applications.

Keywords: Monitoring applications; collective mobility patterns; on-line detection; Complex Event Processing; Event Processing Network; EPL rule composition

Resumo. Aplicações como gerenciamento de frotas, coordenação de forças-tarefa móveis, logística ou controle de tráfego podem ser amplamente beneficiadas pela detecção on-line de padrões de mobilidade coletivos de veículos, bens ou pessoas. Entretanto, a análise de padrões de mobilidade coletivos é exponencial por natureza, requer processamento com alta vazão de grandes volumes de dados de sensores móveis, e portanto gera enormes cargas de comunicação e processamento para um sistema de monitoramento. Considerando os benefícios do modelo de processamento assíncrono baseado em eventos para aplicações de monitoramento on-line, neste trabalho defendemos que diversos padrões de mobilidade coletivos podem ser elegantemente descritos como uma composição de regras de Complex Event Processing (CEP), e focamos especificamente na detecção do padrão de mobilidade de agrupamento (cluster). Nós apresentamos também um middleware para dispositivos móveis baseado na tecnologia DDS que suporta a implantação distribuída destas regras CEP para tal detecção de padrões móveis. Como uma forma de avaliar nossa abordagem, mostramos que utilizando nosso middleware é possível detectar este padrão de mobilidade para milhares de nós móveis, com uma latência adequada para aplicações de monitoramento on-line.

Palavras-chave: Aplicações de monitoramento; padrões de mobilidade coletiva; detecção on-line; Complex Event Processing; Event Processing Network; composição de regras EPL

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

In several distributed applications, such as fleet or mobile task force management, traffic control or surveillance in mass events, it may be important to perform on-line monitoring of mobile node movement and to be able to detect when a given set of nodes are collectively moving according to a certain pattern, such as flock-in (clustering), flock-out, parallel tracks (i.e. in same direction and with same speed) or convoy. This requirement of coordinated movement detection is motivated by reasons such as ensuring safety of the nodes (which may be vehicles, people or unmanned aerial vehicles - UAVs), ensuring availability of enough resources, optimizing global operation of the nodes, or mutual navigation support among the nodes. Unlike the offline analysis of the mobile node's trajectories [1]–[3], which may be necessary for identifying bottlenecks or just understanding the movement behavior of vehicles or crowds over a longer time span, on-line detection provides means of Situational Awareness [4], [5] required for rapid decision making about immediate reactions - by the managing staff – to promptly solve or mitigate the related problem. There are several examples of collective mobility patterns that are relevant for applications such as public transportation, logistics, traffic management, air traffic control, security and others. All these applications rely on the continuous and periodic transmission of mobile sensor data from all mobile nodes, which may be as simple as the geographic coordinates plus a timestamp, or may include also other data such as the velocity vector, the operational status of the node, or data about the node's current activity.

In road traffic control, for example, it is important to early detect some traffic bottleneck due to a total or partial obstruction of a street/road. In this case, the associated collective mobility pattern is characterized by two regions: one with a high density of vehicles moving at unusual low speed, and another region with fewer vehicles, but moving at higher speed, or accelerating. A related mobility pattern is the recurrent unexpected maneuver by nodes at a given geographic position. In this case, several but not necessarily all vehicles exhibit a driving/movement behavior not expected from normal traffic at this place, such as sudden, intense de-acceleration of short duration (braking) or change of lanes. It can indicate a temporary obstruction of part of the road, such as an animal on the road, a hole in the asphalt, or an object/box that fell from a vehicle. Other examples can be found in public transportation management, where regularity, punctuality and smooth operation are paramount. Hence, in this field it may be important to detect, as early as possible, if there is some undesired aggregation of demanding nodes (e.g. passengers at a tram/bus stop) or of the providing nodes (e.g. the trams or busses) at some places in the city. In the former case, clusters of passengers at specific stops may indicate that some tram/bus line is not satisfying the current demand of passengers, so that the number of circulating busses should be increased. In the second case, it may be an indicator that the busses are not traveling according to the pre-established schedule, or that some serious obstruction happened in in their itinerary. Also in logistics, collective mobility patterns - of goods and materials - play an important role. As logistics is concerned with the detailed planning and organization of a large complex operation, involving the scheduled flow of items (raw materials, intermediate items, and finished goods) through an organization or supply chain, aggregation and irregular flow of vehicles or objects (e.g. tracked and with RFID tags) should be detected as soon as possible.

As can be seen from all previous examples, unexpected flock-in (clustering) or out-of-schedule visits of nodes at places are recurrent collective mobility patterns in several

application domains, and early detection of these patterns is fundamental to mitigate transportation problems or minimize their impacts.

Online detection of collective movement patterns however, poses several challenges to a monitoring application. First, a high throughput of the sensor data streams (including position and other data) produced by a large set of mobile nodes has to be handled and processed (i.e., analyzed) in a timely manner. Secondly, efficient algorithms and a parallel approach have to be employed to cope with the intrinsic exponential complexity of the mutual comparisons of the position, speed or other data of pairs of mobile nodes. Thirdly, it has to be taken into account that the set of monitored mobile nodes sometimes is open and variant, i.e. new elements may join, while others may leave the set (e.g. new vehicles entering and leaving a city perimeter), so that the pattern boundaries may be unclear or difficult to detect. Moreover, mobile sensor data usually comes with a measurement error, which implies that collective movement patterns cannot be defined in terms of exact relative differences between the sensor data, but instead must be defined in terms of interval relations. For example, if L is the threshold that defines if two nodes are close to each other, and ε is the position error, then a cluster pattern should be detected as long as the mutual measured distances among a set of nodes is less-equal than $L+2\varepsilon$. And finally, the pattern detection and notifications should be reliable, precise and fast, otherwise they may become useless for an adequate and timely response by a coordinating central or by the mobile nodes themselves.

The first two challenges demand the use of a data stream processing [6] approach, where the detection logic is defined in terms of continuous queries or detection rules. Among the several available stream processing approaches and technologies, Complex Event Processing (CEP) [7] is the most promising one due to its support for describing mobility patterns in terms of complex events and rules, which in turn are defined terms of simpler events and rules. Hence CEP allows one to build a hierarchy of composite events and rules that can be reused, combined, and executed in a distributed manner in a network of Event Processing Agents (EPAs). However, in order to support a distributed CEP solution that is scalable, it must be based on a high-performance distributed communication middleware, enabling EPAs to transfer events among each other at high throughput and low latency.

In this paper we describe a middleware for mobile communications and complex event processing based on the Data-Distribution Service for Real-Time Systems (OMG DDS) [8] specification that supports on-line detection of mobility patterns in a mobile cloud architecture. Its main advantages include the ability to scale in the number of mobile connections, support a scalable and parallel execution of the CEP rules on a dynamic set of Processing Nodes in the cloud, and divide the entire sensor data stream into regional sub-flows. The middleware uses the RTI Connext DDS [9] on the core of its communication backbone.

Moreover, we focus on the cluster (flock-in) mobility pattern and show how this collective pattern can be described as a composition of complex events and rules, where some CEP rules can, in principle, be (re)-used as building blocks for other collective mobility patterns. We also present some preliminary performance results of our middleware at it is used for detecting the Cluster pattern for hundreds to thousands of simulated mobile nodes.

In the next section we discuss related work focused on online mobility pattern monitoring and in Section 3 we explain basic concepts and technologies underlying our approach. In Section 4 we then summarize the main layers and components of our middleware system and in Section 5 describe how the Cluster mobility pattern can be described by a composition of CEP rules. In Section 6 we then explain the experimental

set-up and the performance results (the latency and precision of detection) obtained when monitoring large sets of mobile nodes, and in Section 7 we finalize with concluding remarks.

2 Related Work

Collective mobility patterns have been subject of research in engineering, natural and social sciences, usually in the study of animal, vehicle traffic and human collective behavior. However, most of them either focus at offline analysis of aggregations or trajectories [10] or at the control and feedback mechanisms employed by the mobile elements to coordinate their movements in real-time. In this work, we assume that collective movement patterns emerge spontaneously, and are interested in detecting the patterns in real-time, i.e. right after they emerge, and monitoring its evolution over time. We refer to this type of detection as on-line detection of mobility patterns.

In [11] the detection of mobility patterns is also performed, and as in our work, the flock-in (Cluster) pattern is studied. However, the presented algorithm is only applicable off-line, since it requires knowledge of future movement data.

Dense [12] is a middleware platform for cluster detection based on a peer to peer overlay network. However, in contrast to our work, their focus is on reducing the frequency of location updates sent by mobile nodes, by using a series of sampling algorithms.

Mobiiscape [13] is a middleware platform for online detection of mobility patterns in a city-wide scenario. The middleware uses a language called Moving Object Monitoring Query Language (MQL) to define mobility patterns, and also use Complex Event Processing (CEP) for the pattern detection. The difference from out approach is that it focuses on pre-defined geographic regions, and it defines four primitives - enter, leave, stay and pass - to represent basic events that each MN produces while moving around. Moreover, Mobiiscape is geared towards the detection of individual, rather than collective, mobility patterns.

The work of [14] proposes an extension of a complex event processing engine with fuzzy spatial relations between events to enable the definition of qualitative spatiotemporal patterns. In particular, they address the fuzzy distance relation between two events in order to consider vagueness and uncertainty, which is not the focus of our work.

There are also commercial products that implement spatial extensions to CEP engines to enable the processing of spatiotemporal events: SpatialRules, from ObjectFX is a CEP engine for geospatial data that is compliant with OGC geospatial specifications [15] and where event processing rules have built-in geospatial and temporal operators. GCEP from SURNA INC. is an extension of Esper that allows the use of OGC Geospatial Functions for filtering and supports 12 topological functions. Finally, the RuleCore CEP Server1 is a complex event processing engine used for real-time detection of complex event patterns that enables defining rules using location information (data collected from GPS sensors). The engine allows creation of data streams of events coming from specific geographic zones. However, the detection capability of RuleCore is mainly focused on spatiotemporal patterns of individual nodes (e.g. delays and excepts in supply chains, or vehicle/driver not complying with working hours regulation), rather than collective mobility patterns.

1 wwww.rulecore.com

3 Basic Concepts and Assumptions

In this section we will present and discuss the main concepts and technologies underlying our work.

3.1 Collective Mobility Patterns and the Cluster Pattern

In this paper we will concentrate on detecting a specific collective mobility pattern, the *Cluster* pattern, since it has many applications. The cluster pattern is dynamic, transitory and non-precise by nature, and is essentially determined by an unusual density of mobile nodes at some geographic point or region, which in turn may be previously defined, or not. In the first case, the goal is to check for a cluster formation at certain points/regions of interests, such as gateways, street sections, city neighborhoods, etc. In this work, however, we will focus on identifying clusters anywhere, by comparing the locations of mobile nodes relative to one another, since this is clearly a more complex problem.

The definitions of properties that are relevant to be considered in the detection of clusters, such as the density to be considered as unusual, among other factors, are dependent to application requirements. For example, an application may be interested either in detecting: only the occurrence and location of clusters (independent of their size and number of nodes); the approximate perimeter (or size) of a cluster; the current number of nodes in each detected cluster; or the identities of the nodes that are members of the detected clusters. In regard to the evolution of the cluster patterns, applications may be interested to trace how the density, perimeter or location of the clusters are changing over time.

For the sake of simplicity, but still quite useful for many applications, in this work we concentrate in detecting the occurrence of clusters and theirs locations. Hence, we define a detectable cluster pattern as follows:

Definition: Given a threshold distance D, a minimum number of clustered nodes CN and a time t, we say that a cluster of MNs was formed at time t **iff** during some interval [t, t+T], with $T > \Delta$, there exists a set S of mobile nodes, where $|S| > CN$, such that $EucledianDist(pos_i + \varepsilon , pos_j + \varepsilon) \leq D$, and $i, j \in S$.

Note that D and CN are parameters provided the application and CN/D is the unusual density that characterizes a cluster. Moreover, Δ is the minimum cluster existence time.

3.2 System Model and Assumptions

The system is composed of mobile nodes (MN) with wireless connection, and stationary nodes. The stationary nodes have Internet connectivity and are reliable (do not fail).

We assume that each MN is able to sense its current position, *pos*, with a maximum error E, can probe some other relevant sensors, and is also capable of computing its approximate speed vector v. Using its wireless connection it transfers a message (carrying all collected mobile sensor data with a timestamp) to some stationary machine once every ρ time units. Delivery of messages through the wireless medium is reliable, and in normal circumstances, the transmission delay is at most δ. If the MN's wireless communication link is experiencing connectivity problems, than the MN is notified of a disconnection after $k * \delta$ time units, where $k * \delta < \rho$. The sensor probing time by the MN is negligible.

Among the stationary nodes, Processing Nodes (PN) are in charge of executing the mobility pattern detection logic of the monitoring system, while Gateways (GW) are entirely dedicated to handling all connectivity issues with a subset of all MNs.

3.3 Complex Event Processing

Complex Event Processing (CEP) [7] is an event-based technology that provides an asynchronous processing model for the online detection of situations of interest. A CEP system receives streams of *raw events* generated by different sources (e.g. sensors), continuously processes these event streams and generates *derived events*, which are sent to event consumers (e.g. online monitoring applications) interested in receiving notifications about the occurrence of detected situations.

These situations are described by *CEP rules*, which are set-up in one or more *CEP engines* to continuously process event streams. *CEP rules* are Event-Condition-Action (ECA) rules that use operators (e.g. logical, quantifying, counting, temporal, causal, spatial and sequencing operators) that are applied on received events, seeking for correlations among them, generating *complex*, or *composite*, *events* that summarize the combination of constituent *elementary* events (i.e. events that are not composed by other events). Raw events are usually elementary, while derived events can be either elementary or complex. A data dissemination service transports the events from data sources to the CEP system, and the derived events generated by the CEP system to all interested event consumers [16].

Most CEP systems have the concept of Event Processing Agents (EPAs), which are software modules that implement part of the entire event processing logic between event producers and event consumers, encapsulating some operators and CEP rules. The type of an EPA is defined by the behavior of the CEP rules it implements, such as filtering, transformation (e.g. translation, aggregation, splitting, composition) or specific event pattern detection. An Event Processing Network (EPN) is a network of interconnected EPAs that implement the global processing logic for pattern detection through event processing [17]. In an EPN the EPAs are conceptually connected to each other (i.e. output events from one EPA are forwarded and further processed by other EPAs) without regard to the particular kind of underlying communication mechanism for event dissemination.

CEP systems, are an evolution of publish/subscribe systems [18]. Here, events are pieces of information at different levels of abstraction, with inherent semantics and coming from different sources. Comparing CEP with content-based publish/subscribe, the former provides the capacity of specifying relationships not only over event attributes, but also relationships between different types of events (and their attributes) [16]. In addition, it allows taking into account the history of already received events, and most importantly, allowing subscribers to express interest in composite events. The expressiveness of CEP also exceeds the capabilities of simple Data Stream Management Systems (DSMS), since it provides the ability to detect complex patterns of conjunctions and disjunctions, sequencing, ordering, and other types of relationships among events.

The event composition capability of CEP is very useful for enhancing the abstraction level of the detection of mobility patterns, describing rules that generate events with different levels of abstraction, which can be hierarchically organized to represent complex patterns composed of simpler ones.

3.4 Distributed Complex Event Processing (DCEP)

One important aspect with significant impact on the scalability of the processing capacity of an EPN is the deployment model, which may be centralized or distributed [16]. In a DCEP system, as shown in Figure 1, the event streams are processed by EPAs deployed at different nodes interconnected by a communication infrastructure. Distributed event processing architectures can either be deployed in a computer cluster, where nodes are tightly coupled by a fast and reliable network and belong to a same administrative domain, or in an overlay network of nodes dispersed in a Wide Area Network (WAN). Clustered systems benefit from parallel event processing, but demand high network bandwidth usage between the cluster nodes and the remote producers and consumers of events. Networked architectures, on the other hand, focus on minimizing network bandwidth usage, by deploying EPAs closer to event producers and consumers. In this work, we assume a clustered approach, since we take advantage of a middleware infrastructure specially suited to provide scalability on handling a large number of mobile nodes sending data streams to a single computer cluster, but with different points of attachment (see Section 4.1). The placement decision for the EPAs of a particular application can balance different criteria, such as geographical location of event sources and sinks, network throughput and reliability, functionality, data segmentation, load-balancing, etc [16].

Different academic and commercial distributed event-processing systems are available. The majority of them use a clustered deployment, e.g. [19]–[21], while a few implement a networked solution [22], [23].

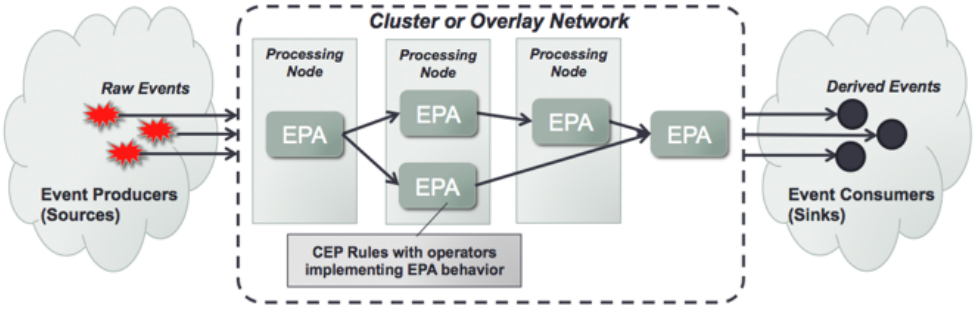


Figure 1 - Typical characterization of a DCEP system, with EPAs deployed on different processing nodes of a cluster or overlay network.

3.5 DCEP Systems' Processing and Communication Layers

As show in Figure 2, a DCEP system is typically organized into different layers. The Event Processing Layer is responsible for managing all event-processing entities and mechanisms, such as the CEP engines, deployment of EPAs (a single Processing Node can have many EPAs locally deployed), CEP rules, etc, while a Communication Layer implements a service that transports events from data producers to the CEP system, internally between EPAs, and from the CEP system to the event consumers. Figure 2 shows nodes of a network running instances of a DCEP system, with the Event Processing Layer with some deployed EPAs and a Communication Layer used by the system to propagate events.

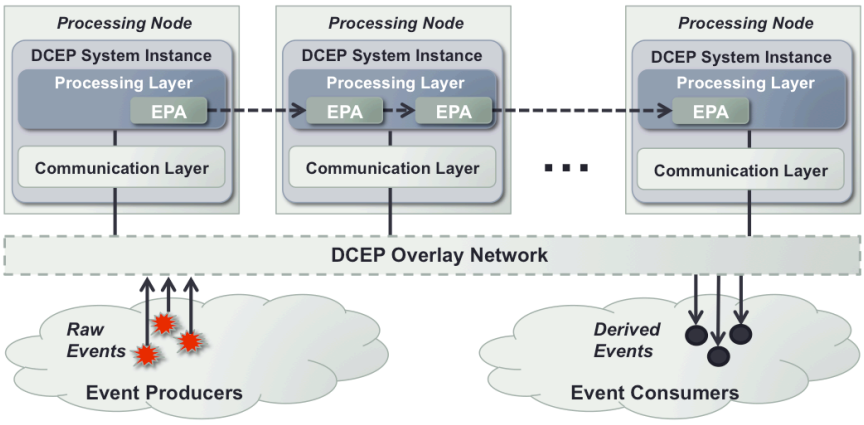


Figure 2 - A DCEP system organized into a Processing Layer and a Communication Layer.

3.6 Data Distribution Service for Real-Time Systems (DDS)

The vast majority of existing DCEP system use a push-based (i.e. asynchronous) communication for its event dissemination layer. The most recent and state-of-the-art evolution of distributed pub/sub is the Data Distribution Service for Real-Time Systems (DDS) [8], which brings along many advantages in performance, scalability, availability and quality of service mechanisms. The DDS standard defines a fully distributed peer-to-peer (i.e. broker-less) real-time data-centric publish/subscribe (DCPS) communication model. It provides high performance communication - comparisons such as [24] show the performance superiority of DDS over other pub/sub platforms (e.g. CORBA Notification Service, JMS and SOAP) - scalability and availability, and supports the specification of Quality of Service (QoS) contracts between data producers and consumers, and also mechanisms for dealing with real-time aspects (e.g. priority channels and specific QoS policies). It allows interoperability across different DDS implementations, programming languages and platforms, as well as automatic discovery of DDS publishers/subscribers.

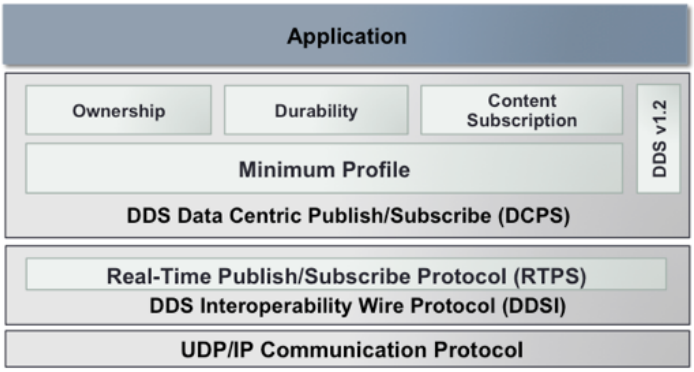


Figure 3 - DDS specification architecture.

Figure 3 shows the DDS specification architecture. In this model, a Minimum Profile provides the concept of DDS Topics, which are logical entities defined to compose a distributed relational data model, also known as Global Shared Data Space. The DCPS Model also provides a Durability Service for data persistence, an Ownership Service for defining ownership of different sources of data, and Content Subscription, which allows content-based subscriptions for data.

DDS Topics are first class entities for information transfer, which application peers can publish or subscribe to, and can be regarded as distributed relational database tables. Topics are defined by the application programmer, which writes an IDL file describing the names, data types, and the keys that identify the instances (that can be seen as da-

tabase rows) of each Topic, which are called samples. A compiler takes the IDL as input, and generates code for communication stubs in a desired programming language. By using the generated stubs, applications can join a DDS Domain, and publish and subscribe to data in this Domain. The DDS Domain, which contains all shared data, is fully distributed over the peer-to-peer network formed by participating network nodes, without any intermediate broker or centralized management entity. DDS applications produce and consume data to this Global Shared Data Space without any direct knowledge of the parties involved in the production and consumption of data (i.e. naming and location transparency).

The DDS specification also provides a large set of QoS features and enforcement mechanisms. Every producer and consumer of data defines its QoS requirements or capabilities through QoS contracts that are matched by a protocol, called Requested-versus-Offered (RxO) protocol, which checks QoS matching before any establishment of communication. These QoS parameters address many aspects, such as communication reliability, latency or transport priority, data persistence, and behavior of node discovery mechanisms, among many others.

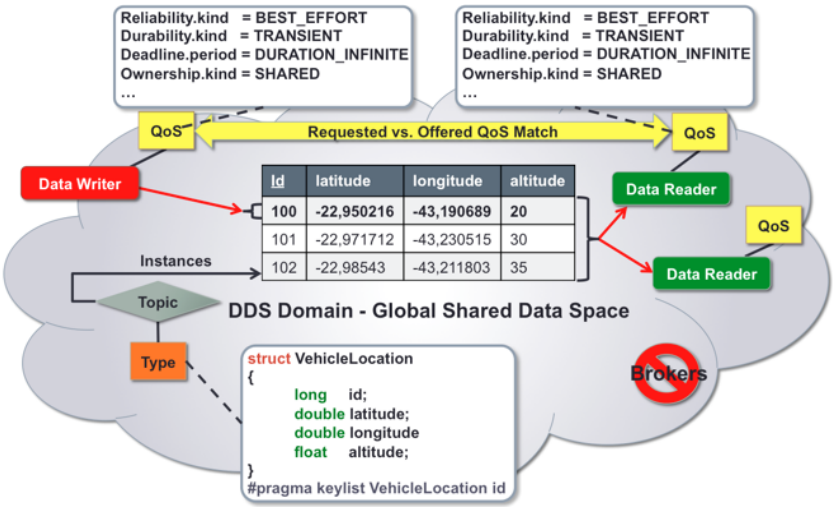


Figure 4 - Illustrative example of a DDS Domain with a DDS Topic and its instances, Data Reader, Data Writer, and QoS matching.

Figure 4 shows an illustrative example of the main entities involved in a DDS communication: a DDS Domain containing a Topic with the name VehicleLocation, which has an associated data type description. A DataReader specifies required QoS policies to be met by any publisher of the Topic to be read. A DataWriter specifies the QoS policies offered to any subscriber of data from the Topic. When the DataWriter inserts or updates a Topic instance (i.e. a row in the "table" of the Topic), all DataReaders registered to that Topic and with matching QoS policies will participate in the data transmission and will receive the published data. Different existing DDS implementations are available. This work uses the RTI Connext DDS [9] implementation.

Considering the advantages in system's performance, scalability and availability, as also observed by other work such as [25], DDS is a data dissemination technology with a great potential to benefit DCEP systems. However, to the best of our knowledge, so far no existing known DCEP implementation uses DDS for data dissemination. Furthermore, this work presents an infrastructure that extends the DDS infrastructure to support the connection of mobile nodes, using a simple protocol that bridges their connection from the Internet with the core DDS domain. Section 4 explains the system architecture, presenting how the DCEP system uses the DDS specification and how mobile devices connection with this system is performed with scalability.

4 System Architecture

Our infrastructure for mobile communications and distributed complex event processing is composed of two main layers: at the bottom, a communication layer provides reliable and scalable communication to and from mobile nodes, as well as the load-aware distribution of sensor data streams to Processing Nodes (PN) at the fixed network. At the upper layer, and using this base communication layer, an event processing layer provides means of deploying event processing networks, i.e. EPAs executing CEP rules, on a set of distributed PNs. The two middleware layers are presented in more detail in the following.

4.1 Scalable Data Distribution Layer

At the bottom layer, the Scalable Data Distribution Layer (SDDL) [26] is a communication middleware that connects stationary nodes in a wired core network (DDS domain) to mobile nodes with an wireless internet connection (see Figure 5). Some of the stationary nodes are data processing nodes, others are gateways for communication with the mobile nodes, and yet others are monitoring and control stations operated by humans. The latter ones can display the mobile nodes' current position (or other relevant sensor data of each node), and allow the operator to receive event notifications about situations related to one or several mobile nodes. SDDL employs two communication protocols: DDS's (Data Distribution Service) Real Time Publish/Subscribe (RTPS) for the wired communication within the SDDL core, and the Mobile Reliable UDP (MR-UDP) [27] for the inbound and outbound communication between the core network and the mobile nodes. DDS is described in detail in Section 3.6 . The Mobile Reliable UDP (MR-UDP) protocol is the basis for the Gateway-mobile node interaction. It implements TCP-like functionality on top of UDP and has been customized to handle intermittent connectivity, Firewall/NAT traversal and robustness to changes of IP addresses and network interfaces. Each message, in either direction, requires an acknowledgement that, if not received, causes each transmission to be retried several times before a connection is considered broken.

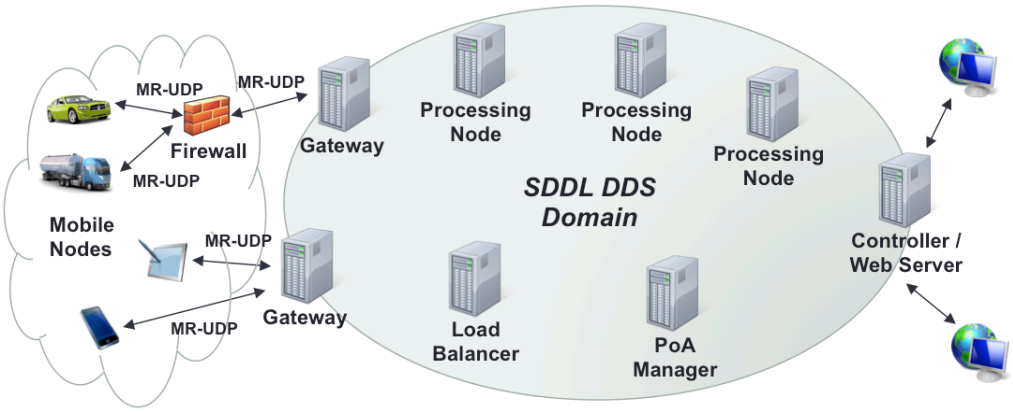


Figure 5 - SDDL middleware overview.

In addition, as part of the SDDL core, three types of nodes play an important role in regard to event processing of mobile sensor data streams:

The Gateway (GW) defines a unique point of attachment (PoA) for connection of the mobile nodes to the services of the SDDL core. The Gateway is responsible for managing a separate MR-UDP connection with each MN, transforming any application-specific message or sensor data from the MN into a DDS message, and in the opposite direction, converting DDS messages to MR-UDP messages and delivering them reliably to the corresponding MN(s). Being the handler of connections to the mobile nodes

(MNs), the Gateway is also responsible for notifying other SDDL core services when a new MN connects to the system or when some MNs disconnect from it. This information is used by other SDDL services, such as caching of messages addressed to temporary offline mobile nodes, for later delivery.

The PoA-Manager is responsible for two tasks: to periodically distribute a list of Points of Attachment (PoA-List) to the MNs and to eventually request that some MNs switch to a new Gateway/PoA. The PoA-List is always a subset of all available Gateways in SDDL, where the first element points to the preferred Gateway/PoA, then to the second choice, and so forth. By having an updated PoA-List, an MN may always switch its Gateway if it detects a weak connection or a disconnection with the current Gateway. Moreover, by distributing different PoA-Lists to different groups of mobile nodes, the PoA-Manager is able to balance the load among the Gateways as well as announce to the mobile nodes when a new Gateway is added to or an existing Gateway is removed (or failed) from the SDDL core.

The Load Balancer (LB) is a node, which continuously monitors the current load of the Processing Nodes in the SDDL core, and assigns Data Processing Slices, i.e. portions of the global mobile data sensor stream, to each of the PNs. This data volume slicing can be done in regard to any attribute of the message produced by the MNs, such as the UUID of the sender or its geographic coordinates. By this, it is possible to divide the total data processing load among any number of PNs. Every data item of the data stream must have assigned a single Data Slice in order to be processed by some PN. In SDDL this mapping of data items to slices is performed at the Gateways, while the Load Balancer transparently sets the subscription filters for each PN so that it only receives the data items that pertain to the corresponding Data Slices assigned to it [28].

The Universal DDS Interface (UDI) [26] is a library that fully abstracts the DDS implementation, promoting reusability and facilitating the development of the architecture components. Its main goal is to encapsulate the specificities of DDS implementation, and to provide common abstractions for use of different DDS products. As well as simplifying the set-up and configuration of DDS entities. The UDI supports the creation of DDS Topics (and Content Filtered Topics), Domain Participants, Publishers, Subscribers, Data Readers and Data Writers. In this work, we used the Real-Time Innovation's Connext DDS [9] implementation.

4.2 Distributed Complex Event Processing Layer (DCEPL)

The Distributed Complex Event Processing Layer (DCEPL) is the middleware component that supports the deployment and execution of distributed complex event processing networks for the detection of situations of interest, and in particular, for the detection of collective mobility patterns. Figure 6 depicts the main elements of the architecture, following the typical organization of a DCEP system with event sources (the mobile nodes), a set of processing nodes and event consumers.

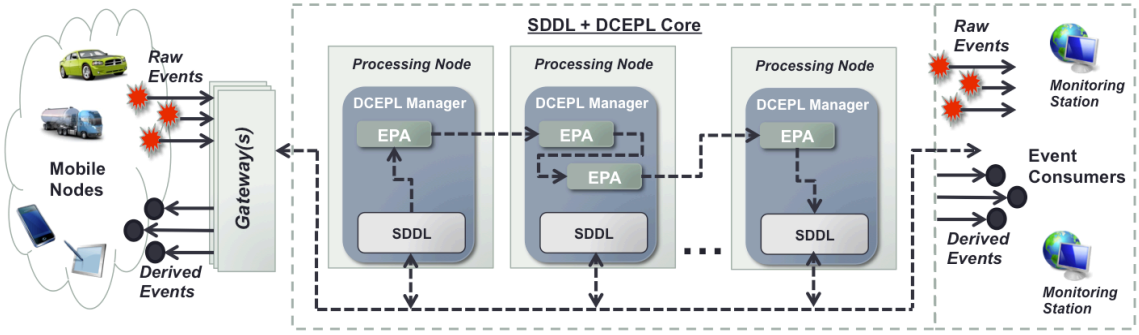


Figure 6 - DCEPL overview.

The DCEPL Manager implements the event processing mechanisms, which supports the definition and deployment of EPAs at different processing nodes, so as to compose distributed EPNs.

Each DCEPL Manager instance contains a local CEP engine, which may have several deployed EPAs. The deployed EPAs process events received from mobile nodes and from other EPAs and generate derived events that are consumed by other EPAs (e.g. running locally or at other nodes) and by event consumers (i.e. subscribed monitoring applications or services).

Its important to make clear that all communication follows the publish/subscribe paradigm, i.e. all event producers and consumers, such as mobile nodes, EPAs and applications only publish or subscribe to data anonymously, with time, space and reference decoupling.

The DCEPL system uses the SDDL middleware, explained in Section 4.1 , as its communication layer. SDDL provides reliable and scalable communication to and from mobile nodes, which are outside the SDDL Core. Mobile nodes connect to Gateways, producing the raw events that are fed into the system, and also receiving any derived event generated from within the SDDL Core.

In order to take advantage of SDDL's load balancing mechanisms for mobile sensor data processing, instances of the DCEPL system are deployed on SDDL Processing Nodes. Each DCEPL Manager instance uses SDDL for its local EPAs to receive elementary events from MNs, and to exchange elementary and composite events among each other within the SDDL Core. Each EPA contains a SDDL/UDI Subscriber for subscribing to events and a SDDL/UDI Publisher for publishing derived events resulting from their CEP processing. For example, an EPA can subscribe to slices of sensor data produced by mobile nodes or subscribe to derived events produced by other EPAs.

As already mentioned, each DCEPL instance running on a processing node allows the instantiation of any desired number of EPAs. An EPA has a set of rules, where each rule is composed of sequential Event Processing Language (EPL) statements describing the CEP rule processing logic. EPL was chosen since it is very similar to SQL, and is currently one of the most popular event processing languages.

In order to facilitate application development and support flexible deployment, all entities to be instantiated by each DCEPL instance are described in a *Configuration Repository* accessible by each processing node. By defining entities in this metadata repository, the application developers and system administrators can deploy EPNs into the system in a straightforward way, without requiring knowledge of any programming language other than EPL. The configuration repository also functions as a library of reusable rules, which can be loaded by any processing node.

Among the entities defined in the configuration repository are: a description of the event types referenced in EPL rules (and exchanged by among EPAs), EPL statements describing the EPAs' logic, external functions implemented in Java and called by these statements, execution parameters, such as arguments to rules, and deployment parameters, such as the mapping of EPAs onto processing nodes and references to relational databases. The databases can be used as knowledge bases for rules, or else, for storing events generated by the system, for example for auditing and generation of reports. The modularity of the DCEPL design allows it to be adapted to use different CEP engine implementations that uses the Event Processing Language (EPL) as the means for defining CEP rules. In the current version of DCEPL we used the Esper [20] engine.

5 Detection of the Cluster Pattern – A Case Study

The cluster pattern is detected when a sufficiently large set of MNs gather and become close to each other. Hence, the main approach for detecting the cluster pattern is as follows:

The most basic information for the detection of mobility patterns, i.e. the raw events needed for the detection of these patterns, is the position information periodically sent by each mobile node (MN). We call each of these raw events *a MNLocationReport*, which contains the mobile node's current position, i.e. its geographical coordinates and measurement accuracy.

MNLocationReport events are captured into an EPN that processes them in order to detect other events with higher levels of abstraction. Figure 9 gives an overview of the EPN, with the constituent EPAs and the events detected by each one, leading to the detection of the Cluster pattern. The following subsections explain in more detail the correlation logic in each of the EPAs..

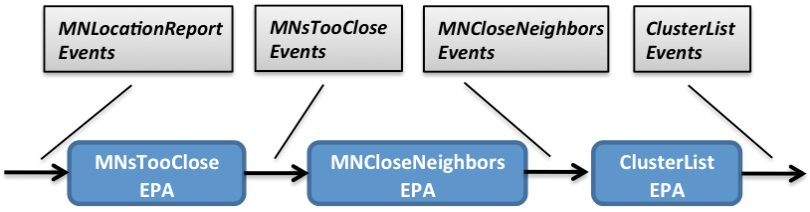


Figure 7 - EPN with the chain of EPAs to detect cluster events.

Since the cluster pattern looks for mobile nodes that are close to each other, the SDDL architecture uses location filters, defined in terms of latitude and longitude intervals, to separate the *MNLocationReports* into sub-regions before inserting them into the processing network (as illustrated by Figure 8). This split is done by the Gateways using the data-slice mechanism of the load-balancing DPSLB solution [28] (see Section 4.1). It suffices to check the distance between all pairs of MNs within the same sub-region of the total area of interest, since nodes that are distant from one another will never be aggregated into a cluster. By using this approach, a significant reduction of the processing complexity in the EPN is achieved, since agents subscribed to each slice only handle location updates of a given sub-region.

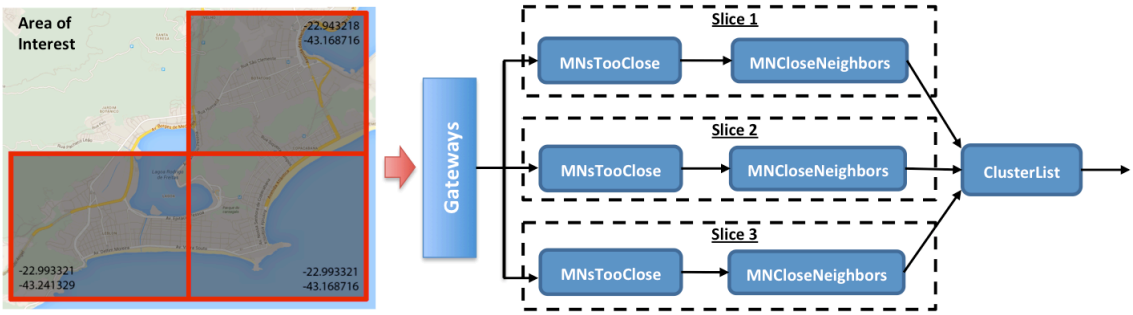


Figure 8 - Overview of the event processing network.

5.1 MNsTooClose

The first step to detect a cluster is to detect pairs of MNs that are within a distance D of each other. The MNsTooClose EPA at each different slice processes the mobile nodes location updates (i.e. *MNLocationReport* events) correspondent to that slice, combining all pairs of location updates that are within distance $D + 2\varepsilon$ from each other, where ε is

the measurement accuracy. Each pair of location update is aggregated into a complex event denoted *MNsTooClose*.

To do that, each EPA operates in a time-batch scheme, where it retrieves all location updates in a given time period (e.g. all *MNLocationReport* events from timestamp *T* to timestamp *T - 30* seconds). The agent correlates this set of location update events with the set itself, producing a Cartesian product of all possible pairs of location updates. This result is then filtered by the distance function, that checks if each pair is close to each other , as shown in Code 1. If the coordinates are within a minimum Eucledian distance (considering de accuracy of both coordinates), then the EPA produces an *MNsTooClose* event containing the pair of location updates.

```
INSERT INTO MNsTooClose
SELECT A, B
FROM   MNLocationReport.win:time_batch(30 sec) as A,
       MNLocationReport.win:time_batch(30 sec) as B
WHERE  A.id <> B.id AND
       distance(A.latitude, A.longitude, B.latitude, B.longitude) < D + 2ε
```

Code 1 – Rule to identify MNs that are near one another.

5.2 MNCloseNeighbors

The MNsTooClose events pinpoint two mobile nodes that are close to each other. However, the cluster pattern depends on detecting a minimum number of such close nodes *CN*, here denoted neighbors. Thus, a fundamental requirement is to know the number of *MNsTooClose* events each node has. Therefore, all *MNsTooClose* events for each node are collapsed into a single event that expresses a nodes' location update and its number of close neighbors, called *MNCloseNeighbors*, as shown in Code 2. Since we are interested on clusters, a sub-query is executed to filter nodes that have at least a minimum number of neighbors.

```
INSERT INTO MNCloseNeighbors
SELECT      Close.A, count(*)
FROM        MNsTooClose.std:unique(A.id,B.id).win:time(30 sec) as Close
GROUP BY    Close.A
HAVING      count(*) > CN
```

Code 2 – Filtering rule to extract nodes that have at least CN neighbors.

5.3 ClusterList

A *ClusterList* event is a list of all clusters detected at a given moment, i.e. a list of locations where a sufficiently large set of MNs are gathered close to each other. In order to generate such list, the *ClusterList EPA* receives *MNCloseNeighbors* events, which provide information about mobile nodes with a significant number of close neighbors. When a new *MNCloseNeighbors* event arrives at the *ClusterList EPA*, the location update arrived inside this event is considered as a candidate for a cluster location, called cluster-head. With this candidate, the rule searches in the existing cluster list for cluster-heads in the vicinity of the candidate's location. If no match is found, a new cluster is created and the candidate is inserted as its cluster-head. If a match is found, and since several MNs close to one another may share a large number of neighbors, it is necessary to collapse these candidates into a unique cluster. The cluster head with the largest number of neighbors becomes the cluster-head of the new collapsed cluster. Also, it is important to identify the cluster-head candidates that are positioned far from each other, since they are genuine members of different and disjoint clusters. Candidates that are far from each other (*i.e.* not within $D + 2\varepsilon$), are added to the cluster list and considered as cluster-heads of independent clusters.

```
ON    MNCloseNeighbors AS candidate
MERGE MNClusterList    AS cl
WHERE distance(candidate.latitude, candidate.longitude,
                    cl.latitude, cl.longitude) < D + 2ε
WHEN NOT MATCHED
     THEN INSERT INTO EventMNClusterList
          SELECT      candidate.mnLocationReport, candidate.numNeighbors
WHEN MATCHED AND      candidate.numNeighbors > cl.numNeighbors
     THEN UPDATE SET  cl.mnLocationReport = candidate.mnLocationReport,
                      cl.numNeighbors     = candidate.numNeighbors
```

Code 3 – Rule to extract unique cluster-heads.

6 Performance Evaluation

In order to evaluate our approach, we ran several experiments using many simulated MNs, that together produce a continuous stream of GPS sensor data to be processed by the EPN deployed in our middleware, with the goal of detecting the Cluster mobility pattern. In order to simulate the cluster mobility pattern of nodes, we defined Attractor Points at random coordinates in a map region. Square areas, called *AttractorPointSquare* (APS), of 400m² where defined around these points. Then MNs where forced to be located, and moving randomly, within or close to these APSs for a certain interval of time (*APS Permanence Time*). This *APS Permanence Time* is chosen randomly between a minimum and a maximum number of location update periods ρ. After its *APS Permanence Time* is over, a simulated MN is released from this APS to a random place in the map region, where it resumes its random walking movement. As for the density that characterizes a cluster, it is assumed that the monitoring application is interested in detecting clusters that have at least 20 simulated MNs gathered in such an APS, *i.e.*, one MN per 2,000 square meters. Although this collective instantaneous flock-in movement behavior is rather artificial, it allows us to measure the latency of the detection using our middleware and to evaluate how this latency behaves for various simulation parameters.

6.1 Configuration and simulation parameters

The experiments were executed with following system configuration and simulation parameters:

Name	Description	Set of Values
#MNs	Total number of simulated MNs	500, 1000, 2500, 5000, 7500
PermT Max	Maximum APS Permanence Time	Uniform dist. 1 to 10.
#APS	Total number of simultaneous clusters	1
#PN	Total number of Processing Nodes	1, 2, 3
#DPS	Total number of Slices of the DPSLB	1, 2, 3

Table 1. Simulation parameters.

The primary choice of simulating a single cluster was to stress the system architecture and discover the limits of the designed rules. By dividing the nodes into different clusters we would also divide the work task, since the number of operations and comparisons executed by the CEP rules would be divided.

6.2 Experimental setup

The experimental set-up was as follows: A single gateway, up to three PNs and three MN-simulation applications. All these machines were directly connected to a Gigabit Ethernet switch.

Name	Description
Gateway	Mac OS X Intel i7 2.3 GHz, 16 GB DDR3 RAM Oracle Java 1.6
Processing Node	CentOS 5 Intel i5 2.3 GHz, 8 GB DDR3 RAM Oracle Java 1.6
MN-Simulator	Windows 7 Intel i7 2.3 GHz, 4 GB DDR3 RAM Oracle Java 1.6

Table 2. Machines configuration.

The MN simulation program uses a thread pool to execute a specified number of mobile nodes, each one sending location updates with a periodicity of 15 seconds. The nodes connect to the SDDL Gateway using the MR-UDP protocol. Although it is not feasible to make all simulated mobile nodes use the real cellular network for sending data, the connection through the Gateway and the use of the MR-UDP protocol seeks to simulate the real-world scenario where mobile nodes use 2G/3G connections to send their location updates. A Wi-Fi connection was not used for the tests since it would make all simulated MNs to compete to access of the same wireless 802.11 network, causing many medium access collisions, contrary to the real-world situation where each MN would have its own 2G/3G connection.

The simulation was executed in a delimited scope of the south region of the city of Rio de Janeiro, as illustrated previously by Figure 8. The regions delimited by three rectangle of this map represents directly the slice that each processing node subscribes. A processing node receive only the location updates of its subscribed slice. The simulation emulates clusters of 500, 1000, 2500, 5000, and 7500 mobiles nodes on this map. With a single EPA, the entire 7500 nodes move to a single rectangle, while with two EPAs they move to two regions (3750 each), and with three EPAs they move to three regions (2500 each).

6.3 Performance Results

The exponential complexity of the problem reflected directly in the experiments results. As expected, the results had an exponential curve, i.e., a slight increase in the number of nodes had a huge increase in the detection latency, as shown by the graph of Figure 9 (in logarithm scale). For example, the simulation of 2500 nodes executed in average 6.250.000 comparisons, whilst 5000 nodes required in average 25.000.000 comparisons. The proposed distributed solution mitigate the detection period. By using a distribution configuration, in one instance, the detection time was reduced from 30 minutes to 20 seconds. In a higher scale, using only one EPA, for 7500 mobile nodes, the simulation took more than 1 hour (3600 seconds) and did not end. Using a distributed EPA configuration the detection period was reduced to an average of 45 seconds.

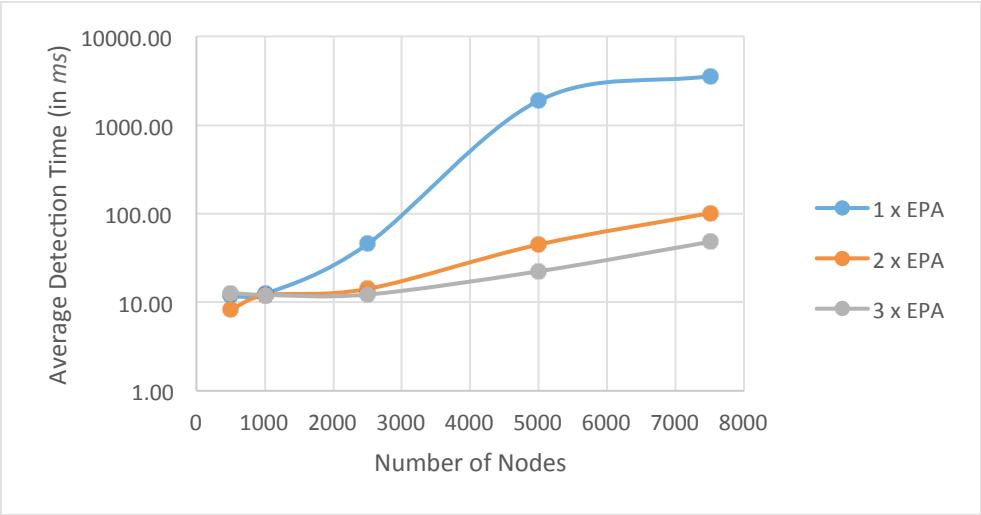


Figure 9 - Average time required for detecting a given cluster of nodes with one, two and three EPAs.

With 500 and 1000 nodes the results showed that the central solution performed better than the distributed one, as illustrated by the graphs of Figure 10 (in logarithm scale). Particularly, because the distribution cost was higher than the number of simulated nodes. Nevertheless, the distributed processing nodes configuration had a similar detection time than the centralized one.

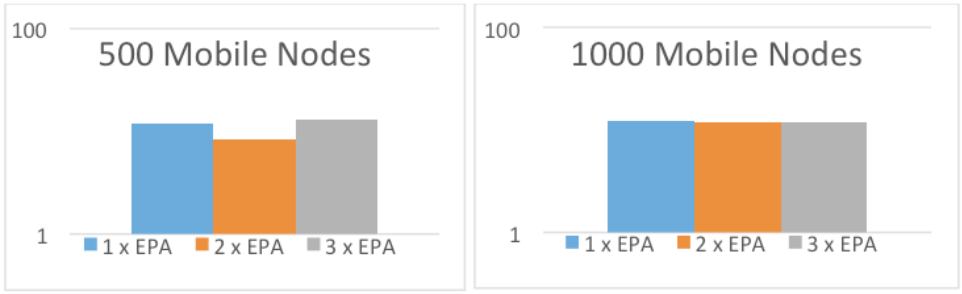
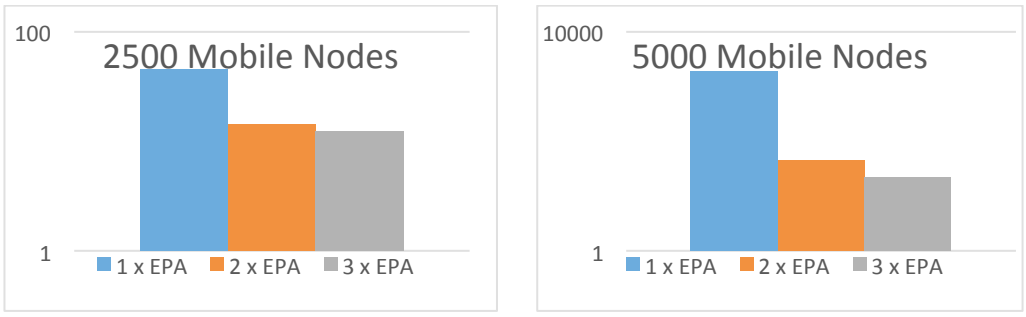


Figure 10 - Detection latency for 500 and 1000 Mobile Nodes.

For higher number of mobiles nodes, such as 2500, 5000 and 7500, the results clearly indicate the bottleneck of the single machine solution, as illustrated by the graphs of Figure 11 (in logarithm scale). The distributed solution was better in both instances, with gains up to 99% over the centralized solution. For instance, to detect a cluster of 5000 nodes, the central EPA took 1888.507 seconds (around 31 minutes), while the distributed EPA took 22.360 seconds.



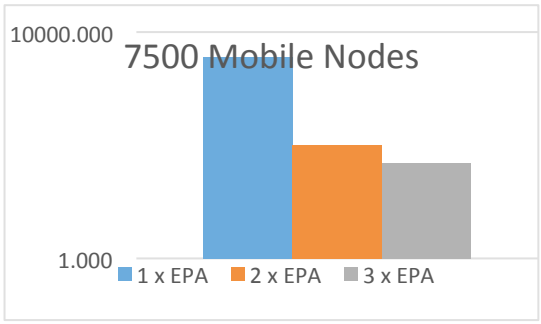


Figure 11 - Detection latency for 2500, 5000 and 7500 Mobile Nodes.

As expected, the results show that with the addition of EPAs the detection time deeply decreases. However, surprisingly, the reduction rate where better than the predicted when adding additional EPAs, as shown by Figure 14. Given the distribution complexity, additional EPAs add a higher configuration cost, thus for lower numbers, such as 500 and 1000 mobiles nodes, the distribution cost of three EPAs exceed the process required time of these nodes in two EPAs. However, for large numbers, such as 5000 and 7500, an additional distribution EPA clearly mitigate the detection time required.

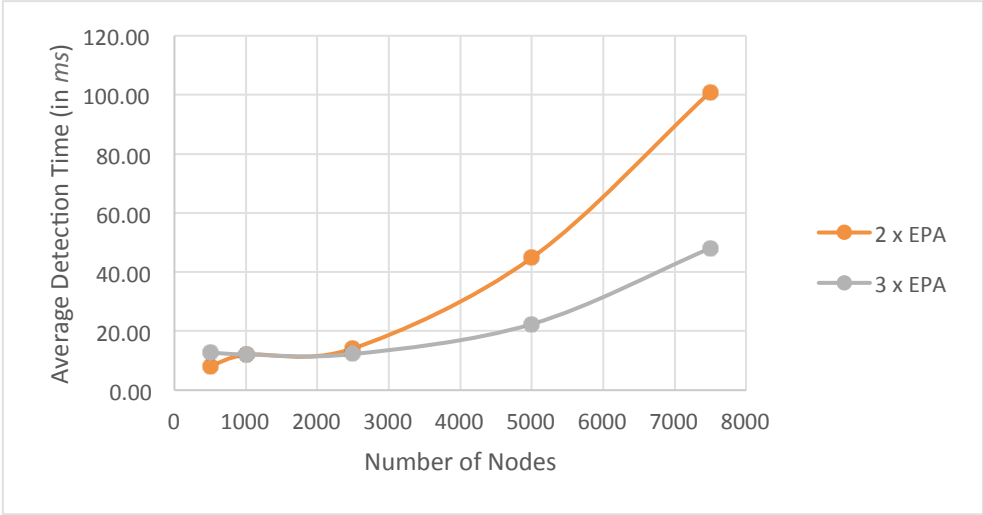


Figure 12 - Average time required (y for detecting a given cluster of nodes with two and three EPAs.

7 Conclusion

Online detection of collective mobility patterns can be useful for many application areas, and so far has not been extensively studied. To the best of our knowledge, our work is the first attempt to describe a specific collective mobility pattern – the cluster pattern – in terms of a set of CEP rules, and to implement a mobile communication and distributed event processing middleware that supports the distributed deployment of EPAs, with the goal of detecting such patterns in a scalable way, and with low latency.

Although it is early to conjecture which sorts of collective mobility patterns can be described and efficiently detected using CEP, we believe that our preliminary results are encouraging, and may open a new thread of research.

As the next steps, we plan to improve the support for CEP rule programming and distributed deployment of EPAs in our middleware, experiment with CEP rules in order to optimize the Cluster pattern detection, and investigate which, and how, other mobility patterns can be effectively represented as CEP rules.

8 References

[1] H. Jeung, M. L. Yiu, and X. Zhou, “Discovery of convoys in trajectory databases,” *Proc. VLDB …*, 2008.

[2] M. R. Vieira, P. Bakalov, and V. J. Tsotras, “On-line discovery of flock patterns in spatio-temporal data,” *Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst. - GIS ’09*, no. c, p. 286, 2009.

[3] M. R. Vieira and V. J. Tsotras, “Complex motion pattern queries for trajectories,” *2011 IEEE 27th Int. Conf. Data Eng. Work.*, pp. 280–283, 2011.

[4] M. R. Endsley, “Toward a theory of situation awareness in dynamic systems,” *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 37, no. 1, pp. 32–64, 1995.

[5] E. Feibush, N. Gagvani, and D. Williams, “Visualization for situational awareness,” *Comput. Graph. Appl. IEEE*, vol. 20, no. 5, pp. 38–45, 2000.

[6] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: A new class of data management applications,” 2002, pp. 215–226.

[7] D. C. Luckham, *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001.

[8] OMG, “Data-Distribution Service for Real-Time Systems (DDS).” 2006.

[9] RTI, “RTI Connext DDS.” [Online]. Available: www.rti.com.

[10] P. Laube, M. Kreveld, and S. Imfeld, “Finding REMO — Detecting Relative Motion Patterns in Geospatial Lifelines,” in *Developments in Spatial Data Handling*, Springer Berlin Heidelberg, 2005, pp. 201–215.

[11] M. R. Vieira, P. Bakalov, and V. J. Tsotras, “On-line discovery of flock patterns in spatio-temporal data,” in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009, pp. 286–295.

[12] I. Boutsis, V. Kalogeraki, and D. Gunopulos, “Efficient Event Detection by Exploiting Crowds,” in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, 2013, pp. 123–134.

[13] B. Kim, S. Lee, Y. Lee, I. Hwang, Y. Rhee, and J. Song, “Mobiiscape: Middleware support for scalable mobility pattern monitoring of moving objects in a large-scale city,” *J. Syst. Softw.*, vol. 84, no. 11, pp. 1852–1870, Nov. 2011.

[14] B. M. F. Barouni, “An Extended Complex Event Processing Engine to Qualitatively Determine Spatiotemporal Patterns,” in *Proceedings of Global Geospatial Conference*, 2012.

[15] S. Panzer, “ObjectFX Wins US Geospatial-Intelligence Foundation’s Industry Achievement Award,” 2010. .

[16] G. Cugola and A. Margara, “Processing Flows of Information: From Data Stream to Complex Event Processing,” *ACM Comput. Surv.*, vol. 44, no. 3, pp. 1–62, Jun. 2012.

[17] O. Etzion and P. Niblett, *Event processing in action*. Manning Publications Co., 2010.

[18] P. T. H. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[19] SAP SYBASE, “SAP Sybase Event Stream Processor CEP.” [Online]. Available: http://www.sybase.com/products/financialservicessolutions/complex-event-processing.

[20] EsperTech, “Esper.” [Online]. Available: http://esper.codehaus.org/.

[21] TIBCO StreamBase, “TIBCO StreamBase.” [Online]. Available: http://www.streambase.com.

[22] TIBCO Business Events, “TIBCO Business Events.” [Online]. Available: http://www.tibco.com/products/event-processing/complex-event-processing/default.jsp.

[23] E. Fidler, H. A. Jacobsen, G. Li, and S. Mankovski, “The PADRES Distributed Publish/Subscribe System,” in *8th International Conference on Feature Interactions in Telecommunications and Software Systems*, 2005.

[24] M. Xiong, J. Parsons, and J. Edmondson, “Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems,” 2011.

[25] R. Baldoni, S. Bonomi, G. Lodi, M. Platania, and L. Querzoni, “Data Dissemination Supporting Complex Event Pattern Detection ∗,” no. Dd4lcci, pp. 1–25, 2010.

[26] L. David, R. Vasconcelos, L. Alves, R. André, and M. Endler, “A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes,” *J. Internet Serv. Appl.*, vol. 4, no. 1, p. 16, 2013.

[27] L. David, M. Roriz, and M. Endler, “MR-UDP: Yet another Reliable User Datagram Protocol, now for Mobile Nodes,” *Monogr. em Ciência da Comput. Pontifícia Univ. Católica do Rio Janeiro*, 2013.

[28] R. O. Vasconcelos and M. Endler, “Autonomous Load Balancing of Data Stream Processing and Mobile Communications in Scalable Data Distribution Systems,” *... Intell. Syst.*, 2013.