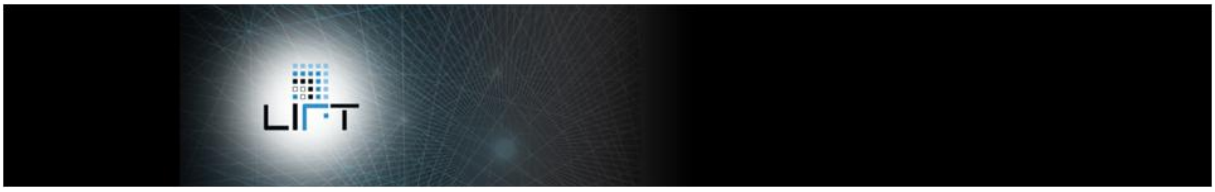


ICT, FET Open

LIFT ICT-FP7-255951

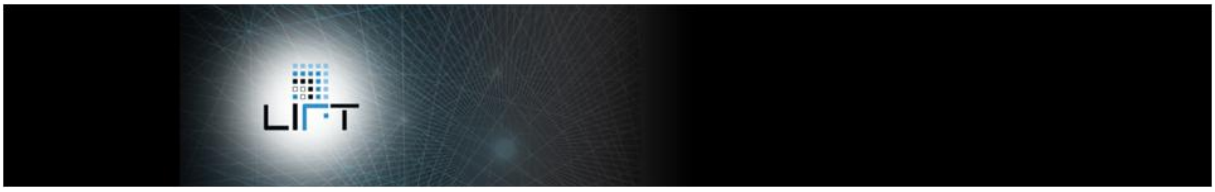
Using Local Inference in Massively Distributed Systems

Collaborative Project

D 3.2

Distributed Protocols and Data Management

Contractual Date of Delivery:	31.03.2013
Actual Date of Delivery:	08.05.2013
Author(s):	Antonis Deligiannakis (TUC), Moshe Gabel (Technion), Michael Kamp (FHG), Daniel Keren (HU), Michael Mock (FHG), Roberto Trasarti (CNR)
Institution:	TUC
Workpackage:	WP 3
Security:	PU
Nature:	R
Total number of pages:	15



Project coordinator name: Michael May

Project coordinator organisation name:
Fraunhofer Institute for Intelligent Analysis
and Information Systems (IAIS)
Schloss Birlinghoven, 53754 Sankt Augustin, Germany
URL: http://www.iais.fraunhofer.de

Revision: 5

Abstract:

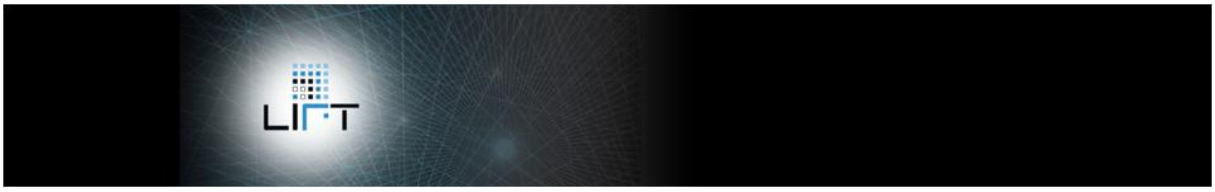
This document is the LIFT deliverable of WP3 for months M25 – M30 of the third review period (01.10.2012 – 30.09.2013). The document contains guidelines how the algorithms and techniques developed in WP1 and 2 can efficiently be implemented using the LIFT reference architecture described previously in Deliverable 3.1.

Revision history

Administration Status		
Project acronym: LIFT		**ID:** ICT-FP7-255951
Document identifier:	D 3.2 Distributed Protocols and Data Management (01.10.2012 – 31.03.2013)	
Leading Partner:	TUC	
Report version:	5	
Report preparation date:	26.03.2013	
Classification:	PU	
Nature:	REPORT	
Author(s) and contributors:	Antonis Deligiannakis (TUC), Moshe Gabel (Technion), Michael Kamp (FHG), Daniel Keren (HU), Michael Mock (FHG), Roberto Trasarti (CNR)	
Status:	-	Plan
	-	Draft
	-	Working
	-	Final
	x	Submitted

Copyright

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951



LIFT Deliverable 3.2

Distributed Protocols and Data Management

A. Deligiannakis, M. Gabel, M. Kamp, D. Keren, M. Mock, R. Trasarti

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

Contents

Figures

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

Introduction

The goal of this deliverable (D3.2) is to put the abstract LIFT reference architecture described in Deliverable D3.1 into the context of specific environments and infrastructures and to provide useful guidelines about the potential alternatives to consider when adopting LIFT technology in various applications. This will exemplify how LIFT technology can be instantiated in different environments and also provide extensions to the LIFT protocol description if needed by the environment.

In Deliverable D3.1 we first reviewed existing approaches and standards for distributed data management and discussed issues that our architecture should handle. We then presented the LIFT reference architecture and described how different network organizations (i.e., distributed vs. hierarchical) can be easily handled within the architecture. Moreover, we documented the requirements of our application scenarios and demonstrated how these requirements are incorporated within our reference architecture. We then provided a full and concrete example (including pseudocode) of how this reference architecture can be used in one of our application scenarios. Meanwhile the LIFT architecture has been fully implemented in a further application scenario, namely *Monitoring Movement Behavior Using Stationary Scanners*, using embedded Bluetooth scanners as sensors and a standard web service as a coordinator [4].

In this deliverable we first seek to provide (Section 2) particular guidelines for the efficient implementation of the techniques developed in WPs 1 and 2. We then move one step further and provide in Section 3 an overview on how to instantiate LIFT in state-of-the Art Big Data infrastructure environments.

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

Guidelines and Alternatives for Efficient Implementation

We now present several guidelines for implementing LIFT techniques in real applications. In our discussion we aim to clarify the alternative choices that are available when deploying LIFT technologies and seek to describe which alternative is more preferable in each case.

In our discussion hereafter we provide guidelines for the following topics:

1. Determination of Safe Zones
2. Alternatives for balancing upon local violations
3. Dealing with dynamic network topologies

These topics are discussed in the subsequent subsections.

Determination of Safe Zones

The original geometric monitoring (GM) approach of [1] provided a generic technique for monitoring complex, non-linear functions computed over data vectors monitored at distributed nodes. On one hand, the GM approach is both applicable for any monitored function and can significantly reduce the required communication, compared to a central collection of data. On the other hand, in LIFT we demonstrated that the SZ approach can improve upon the GM approach in two very important ways:

1. Safe zones can drastically further reduce the required communication (often by orders of magnitude) when compared to GM monitoring. The difference is more profound in higher-dimensional monitoring problems (i.e., when the dimensionality of the maintained local statistics vectors is increased).
2. Checking whether a local violation has occurred is typically much more efficient in the SZ approach, since the only test that is required is whether a drift or local vector lies within a convex region. On the contrary, the GM approach requires computing the maximum/minimum of a complex function within a spherical/ellipsoid subset of the input space, a process that could potentially be time consuming. Allowing for efficient local violation tests makes the SZ approach able to handle data streams with high input rates, while also providing significant energy savings (due to

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951

reduced computational requirements) in resource-constrained network nodes (i.e., sensor nodes with limited battery and CPU capabilities).

Based on the above facts, as well as the theoretical findings of LIFT, anyone seeking to utilize LIFT techniques should aim at using the SZ approach, instead of geometric monitoring.

Given that the SZ approach can assign safe zones of different shape and size at each node, an important question is how to determine the shape/dimensionality of the used safe zone at each case. We distinguish two cases:

1. **Informed users** of LIFT internals: The convex hull of the safe zones assigned to the distributed nodes should be such, so that their Minkowski average resembles (in shape) the shape of the admissible region (region where the function remains under the desired threshold). What this means is that knowledge of the shape of the admissible region can guide the selection of proper safe zones. For example, a spherical admissible region points towards using spherical safe zones at the nodes. Similarly, triangular (often encountered in ratio queries) or box shaped admissible regions point towards using similar shapes (triangles or boxes) at the individual nodes, with the used safe zones having the same dimensionality as the one of the admissible region. Similarly, admissible regions that resemble parabolas (often encountered in quadratic functions) can be well approximated by using polyhedral at the different nodes.
2. **Uninformed users** of LIFT internals: If the user of the LIFT technologies is not familiar with the characteristics (i.e., shape of the admissible region) of the monitored function, then a simple guideline is to utilize polytopes with a minimum number of vertices being twice the number of dimensions of the local statistics vectors. Thus, in 4-dimensional local statistic vectors, the optimization algorithm for computing polytopes with (at least) 8 vertices should be invoked. The optimization algorithm of the safe-zone computation is flexible enough to generate a safe zone of correct shape, with the only drawback (compared to the informed user) being the larger computational requirements and the (potentially) not optimal selection of the number of used vertices.

A final, but very important, decision that greatly impacts the running time of the safe zone calculation algorithm is the issue of *hierarchical clustering* in large networks. The findings of LIFT is that our algorithms that seek to compute the safe zones of different nodes by hierarchically clustering (based on the distribution of their data vectors) the network nodes are very efficient, highly

scalable, and can reduce the computation time (compared to a straightforward non-hierarchical implementation) by orders of magnitude. We thus recommend to anyone using the LIFT technology to follow the hierarchical clustering approach, especially in cases of large networks.

Alternatives for Balancing upon Local Violations

While both the geometric approach and (mainly) the safe zone approach manage to drastically reduce the amount of communication required by the network nodes, there are cases when local violations do occur. An important decision that severely impacts the efficiency (i.e., reduction in number of transmitted messages) of the LIFT approach involves the balancing process that is followed when local violations occur.

A significant observation is that the majority of local violations do not result in a global violation (threshold crossing) and can, thus, be resolved using the data of other network nodes. The fewer the nodes that are contacted, the lower the communication overhead is. On the other hand, the larger the number of rounds that are required for resolving a local violation, the larger the required latency is for determining whether a threshold crossing has indeed occurred. The available choices for balancing thus pose a tradeoff between communication overhead and detection latency. In particular, a general set of alternatives is the following:

1. Global synchronization: This approach results in the lowest latency, but is by far the worst in terms of communication overhead. Most of the times, contacting just a few nodes is sufficient for resolving local violations. Thus, the global synchronization technique that requests data from all nodes is grossly inefficient in terms of bandwidth consumption.
2. Iterative communication: This approach iteratively contacts a new node and requests its data until the local violation has been resolved. This approach has the drawback that it may require a number of communication rounds, in the worst case of a real threshold crossing, that is linear to the number of network nodes.
3. Iteratively communication of progressively larger groups: In this approach the number of nodes contacted at each communication round is doubled, making the worst case number of communication rounds logarithmic in the size of the network. This approach offers a very good tradeoff between communication reduction and latency. It is thus the approach that should be followed in most cases.

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951

A second very important decision that anyone utilizing LIFT technology has to make is, after deciding that an iterative communication process will be used to resolve local violations, involves the selection of *which* nodes will be contacted first in a violation resolution attempt. In LIFT we proposed constructing a hierarchy of the nodes. This hierarchy is build based on how often a node can balance violations of other nodes that lie nearby in the hierarchy. Thus, upon a local violation, the iterative communication of progressively larger groups is followed, but the members of each group have been carefully selected in order to minimize the expected number of communication rounds required to resolve the violation. This is the recommendation that we wish also wish to make to anyone adopting the LIFT technology.

Dealing with Dynamic Network Topologies

In many cases, the network topology may change due to the addition or removal of nodes. One may be tempted in this case to tackle such events by performing a synchronization process, which will also serve as a new initialization procedure for the network. Such a process requires calculating and assigning (transmitting) new safe zones to the nodes, since the Minkowski average of the assigned safe zones should be contained within the admissible region.

While the above procedure ensures correctness, it is largely inefficient as the safe zones of a large network of nodes are not expected to be influenced severely by the addition or removal of a single (or a few) node(s). Please also note that nodes that are added to the network may contain no information (i.e., pdf) regarding the distribution of their local vectors, thus limiting any initial informed decisions for assigning safe zones to the new nodes. Note, of course, that in the case of a subsequent threshold crossing, these newly added nodes will have acquired useful statistics concerning their local data vectors, thus allowing a subsequent informed decision for them.

Our recommendation is to attempt to skip the (expensive) communication from the nodes to the coordinator and the recalculation of the all the safe zones.

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

Insertion of a node. The newly added node can be assigned a safe zone that is equal to the largest convex subset of the admissible region that also contains the Minkowski average of the existing nodes (computed over the previous assignment of safe zones to the nodes). The resulting Minkowski average is guaranteed to lie within the same convex subset of the admissible region. This process requires no communication from/to the existing nodes, other than the newly added node(s). Moreover, its computational cost is minimal. Thus, addition on nodes can be handled in a simple manner.

Removal of a node. This case can be resolved without a synchronization process in the case when the Minkowski average of the remaining nodes continues to lie within the admissible region. While this is not guaranteed, it is very likely for networks with large number of nodes. To understand this, let $\boldsymbol{S_{remaining}}$ denote the Minkowski average of the remaining nodes and let $\boldsymbol{S_{departed}}$ denote the safe zone that the removed node used to hold. Based on the safe zone assignment, the weighted Minkowski average of $\boldsymbol{S_{remaining}}$ and $\boldsymbol{S_{departed}}$ lies within the admissible region. Assuming that *n* nodes existed in the network before the node removal, the weight of $\boldsymbol{S_{remaining}}$ will be 1-1/n. This implies that $\boldsymbol{S_{remaining}}$ by large determines the result of its weighted average with $\boldsymbol{S_{departed}}$ and that $\boldsymbol{S_{remaining}}$ is very likely to also be contained inside the admissible region. Considering the hierarchy constructed during the hierarchical clustering process for assigning safe-zones at nodes, this can be verified by recomputing the Minkowski average in the leaf-to-root path (of the hierarchy) originating at the removed node. This process requires a logarithmic number of computations.

Summarizing our observations, a global communication is avoided in the case of node insertion, while it is very likely to be avoided in cases of node removals.

An important final note is that the aforementioned observations hold only in the (likely) case that the monitored function does not contain in its formula the number of network nodes and, thus, the admissible region itself is not altered when the number of nodes in the network is modified.

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

LIFT in Big Data Environments

Overview

Big Data environments typically address scalability of storage and computing facilities in Cloud and Cluster infrastructure up to hundreds and thousands of computing nodes, working together on a common task on a large amount of data, which is either stored in the cluster or streamed into the cluster. Software components of Big Data environments can be roughly divided into three categories (see Figure 1):

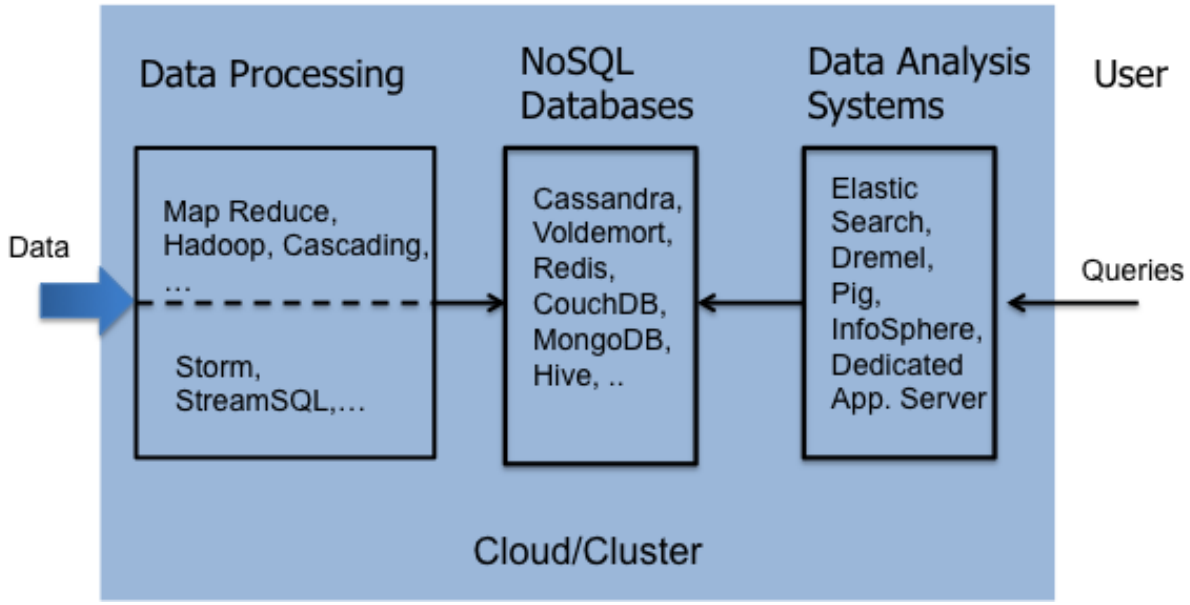


Figure 1: Overview on Big Data Environments

Figure 1 the general structure of a typical Big Data environment. On the one side, data is coming into the system, either in form of real-time streams or by some kind of explicit data import. On the other side, users emit queries to get useful information from the system. At the heart of a typical Big Data environment, there is a scalable, massively distributed and fault-tolerant database system, which, in contrast to traditional databases and Data Warehouses, provides a rather restricted query interface, mostly in form of a so-called key-value store. In order to overcome the restrictions of the reduced

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951

query interface, there are two complementary approaches: providing data analysis systems that bridge the gap between the semantically high-level user queries and the low-level database system, and incorporating data analysis already in the processing of the data that happens before the data (and analysis results) are stored in the database. Depending on the nature of the incoming data, the processing is either done on data stored in the cluster in a batch layer (typical examples are Hadoop and Cascading) or the processing is performed on streaming data with streaming systems like Storm [2] or StreamSQL. The LIFT methodology clearly contributes to the data processing part of a Big Data environment, and, with its massive reduction of communication (and thus, data to be stored), brings Big Data environments even to a larger scale.

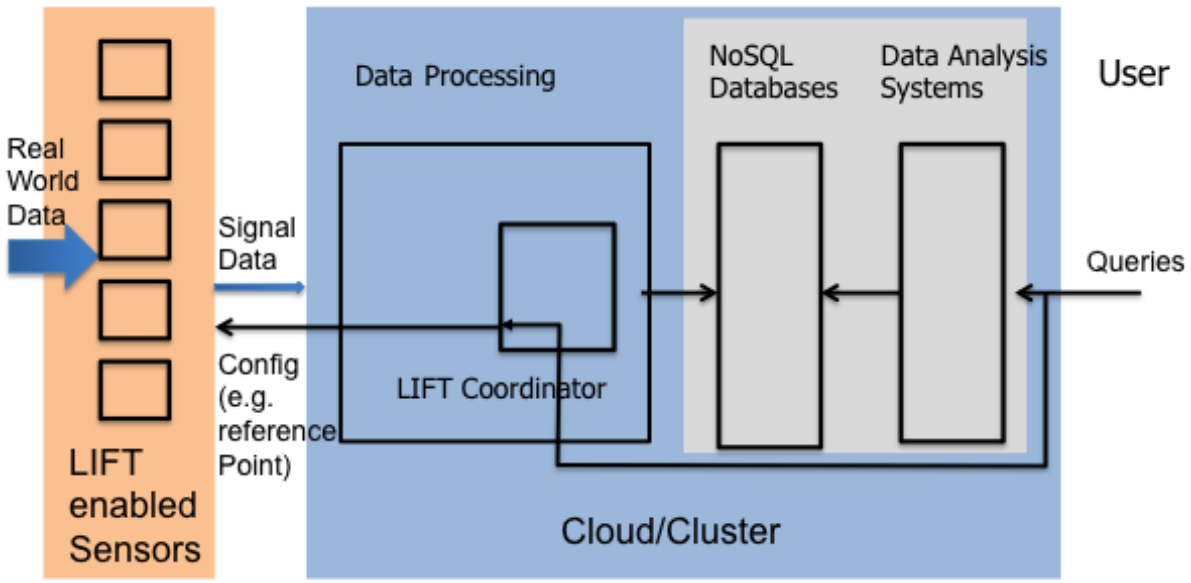


Figure 2: LIFT in a Big Data Environment

As shown in Figure 2, the major effect of LIFT communication reduction is achieved when the sensors become part of the Big Data system by instantiating the local monitoring part of LIFT queries in the sensor (i.e. the implementation of the interfaces described in section 4.1.3 of reference architecture deliverable D3.1). Using LIFT technology, a user query will be partitioned over the sensors such that only a massively reduced amount of communication, for instance safe zone violations, will pass through and go into the cluster. Depending on the application, the amount of data to be communicated can be reduced so much

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

that further Big Data technologies like NoSql databases become obsolete. Note however, that the LIFT architecture requires to establish a "back-channel" from the coordinator to the sensors (see 4.1.1 of reference architecture deliverable D3.1) for the distribution of configuration parameters such as the reference point needed to compute the sensor's local safe-zone. In case of having sensors that cannot be equipped with the LIFT local monitoring facilities, e.g. because they are not programmable or are under supervision of a different organizational domain, LIFT local monitoring should be instantiated in the data path as close as possible to the sensor. Even in the extreme case that LIFT local monitoring is established as very first step of the processing part inside the cluster, it can still achieve a massive reduction in amount of data to be processed and stored.

Concrete Example

In this section, we describe the instantiation of LIFT technology and a LIFT application scenario in a Big Data environment. We refer to the LIFT mobility application scenario for visit pattern monitoring which was already described in term of the LIFT reference architecture in deliverable D3.1. The basic structure of the application is described in Figure 3 and **Fehler! Verweisquelle konnte nicht gefunden werden.**Figure 4 (please refer to deliverable D3.1 and D4.2 for more details).

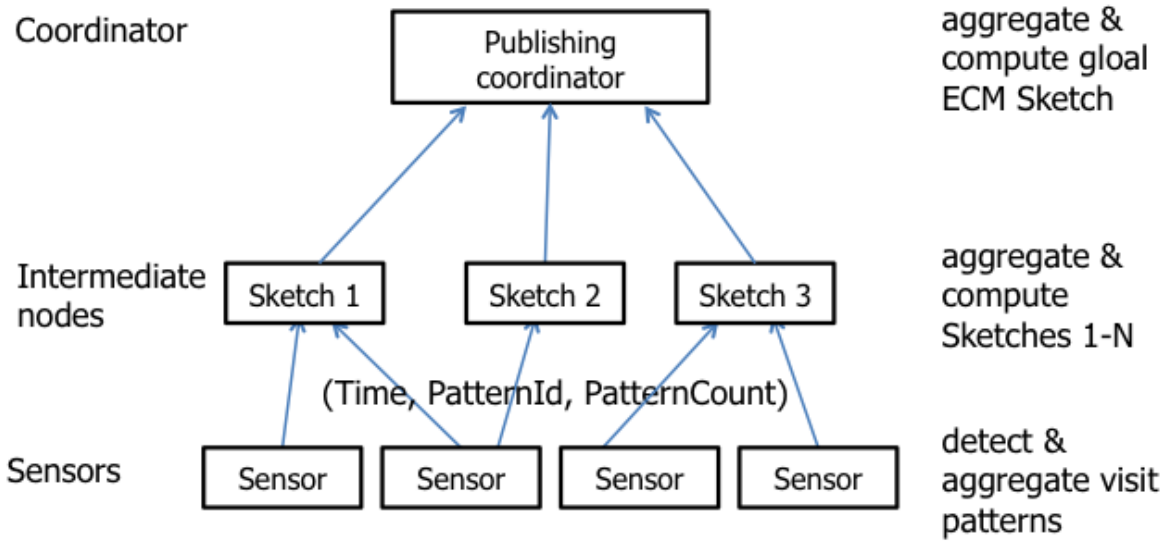


Figure 3: Visit pattern counting with ECM-Sketches – overview

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951

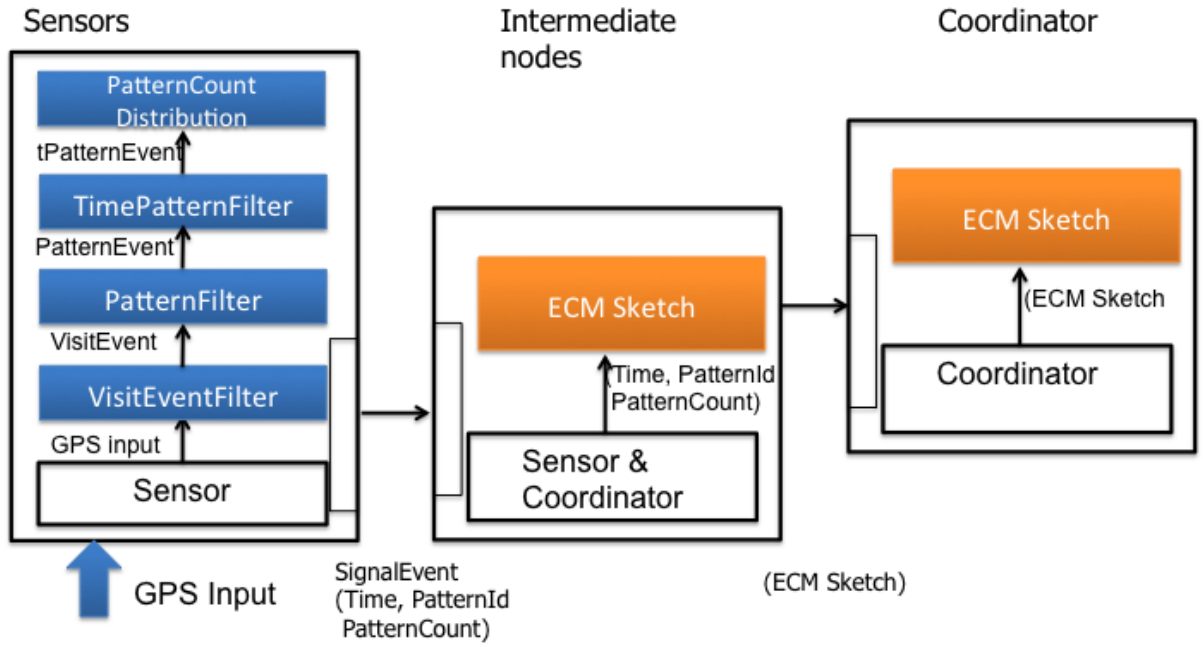


Figure 4: Visit pattern counting with ECM-Sketches – reference architecture

As can be seen from Figure 4, the structure this LIFT example can be regarded as series of filters and components whose execution is driven by a streamed data flow, each step being triggered when data to be processed arrives and each step reducing and aggregating the data. This structure maps directly to the stream processing part of a Big Data Environment. In a stream processing system like Storm, a so-called "topology" is defined to connect processing nodes with data streams. Each node can emit one or multiple data-streams consisting of data tuples, where a data tuple can be seen as the instantiation of the "event" data structure defined in section 4.1.2 of the LIFT reference architecture deliverable D3.1, i.e., a data tuple contains named data values. When defining a topology, each node is connected to one or multiple input streams, which are emitted from one or more other nodes. When submitted to the cluster, the topology nodes are automatically replicated for parallel execution and commutation channels are established and monitored by the Storm runtime system. On a node or permanent communication failure, the topology instance is automatically reconfigured. Storm distinguishes between two node types: Spouts and Bolts. Spouts are input nodes generating the stream data for the topology and Bolts are processing nodes. Whenever a data tuple arrives at a Bolt on an input

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951

stream, it is processed by the Bolt's processing function and, possibly, data tuples are generated on the Bolt's output streams.

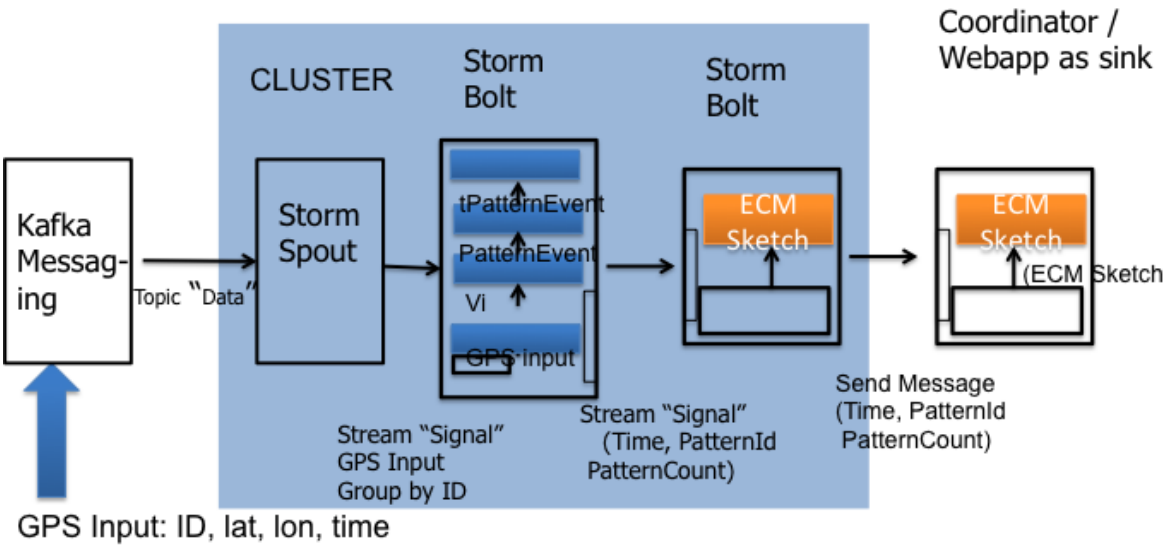


Figure 5: Visit pattern monitoring with Storm

As can be seen in Figure 5, the data flow of Figure 4 can easily be modeled in Storm using a single data stream named "signal" in this example. The data to be processed is stored in Kafka [3], which is a replicated publisher/subscriber based messaging system. The Storm Spout acting as data input to the stream subscribes to GPS data stored under the topic "data", retrieving message giving the GPS position of a tracked person identified by ID at a certain point of time (monotonically increasing in time with the number of messages). Kafka gets the message either from the external world or from its internal, persistent message store. The Storm Spout forwards the GPS message as data tuple to the first processing Bolt on a stream named "Signal" using a grouping method provided by Storm that ensures that GPS data tuples with the same ID are always forwarded to the same instance of the processing Bolt. The processing Bolt processes the LIFT filter hierarchy of the pattern monitoring application, and whenever a pattern match is detected, emits a corresponding output tuple on its output stream name "Signal". Note that the internal filter steps could also be instantiated as separate Bolts, for instance, for enforcing different degrees of parallelization for the different filter steps. The next Bolt in the topology of Figure 5 represents the intermediate nodes of Figure 3, and the top level coordinator

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

ECM sketch is maintained outside of the Storm Topology, acting as sink for the aggregated data and providing the query facility to the user. Overall, using Kafka's persistent message store for executing the LIFT application in a Storm stream processing cluster as in Figure 5 allows to emulate thousands of real data GPS tracks efficiently in parallel for the real-world evaluation of ECM-sketch accuracy and coordinator processing load in the pattern monitoring application. Cascading/Hadoop can be used similarly by just switching from Kafka's message based input method to Hadoop's distributed file system.

Extending Big Data Technologies for LIFT

Section 3.2 has shown, that data flow driven Big Data processing technology directly can be used to model the filter update and signaling communication flow of the LIFT reference architecture. However, for realizing a complete LIFT system, the "config" communication channel from Figure 2 must also be modeled, for instance for distributing a new reference point after a safe zone violation. Figure 6.

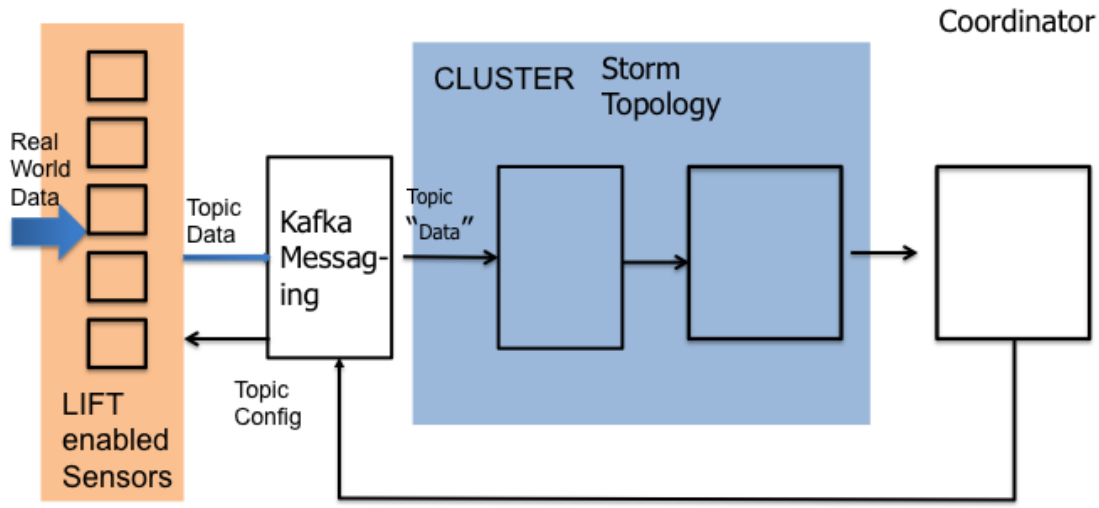


Figure 6: Distributing reference points with Kafka

As shown in Figure 6, Kafka's anonymous publisher/subscriber communication mechanism can be used to implement the back channel needed from the coordinator to the sensors. That would be achieved by letting all sensors subscribe to a topic "config", corresponding to the "config" message of the LIFT

Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

reference architecture. For establishing a back channel inside the cluster when using Storm, a named stream (e.g. “config”) can be used.



Project funded by the European Community
under the Information and Communication
Technologies Programme
Contract ICT-FP7-255951

References

[1] I. Sharfman, A. Schuster, D. Keren. "A Geometric Approach to Monitoring Threshold Functions Over Distributed Data Streams". In proc. of ACM SIGMOD, 2006.

[2] Storm: https://github.com/nathanmarz/storm

[3] Kafka: http://kafka.apache.org/

[4] A. Burmeister. "Implementierung und Evaluierung eines Sketch-basierten, verteilten Monitoring Systems mit eingebetteten Bluetooth-Scannern", Bachelor Thesis, Otto-von-Guericke Universität Magdeburg, supervised by M. Mock, 2013.

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community under the Information and Communication Technologies Programme
Contract ICT-FP7-255951