Designing Sensor-Based Event Processing Infrastructures: A Tradeoff Analysis

Agnès Voisard* and Holger Ziekow †

TR-09-004

June, 2009

Abstract

Systems for distributed event processing have recently gained increasing attention in a broad range of application domains. This raises the demand for methods to adapt the system design to application-specific needs. Our approach considers (i) tradeoffs regarding the hardware infrastructure and (ii) tradeoffs in the software design. The presented model supports design decisions in dependence of application-specific event properties and design goals.

* Fraunhofer ISST and Freie Universität Berlin, Steinplatz 2, 10623 Berlin, Germany
† Institute of Information Systems, Humboldt-Universität zu Berlin, Spandauer Straße 1, 10178 Berlin, Germany

I. Introduction

The advent of technologies such as Radio Frequency Identifier (RFID) and sensor networks has opened new opportunities to monitor real world phenomena in real-time. Applications exist in a broad range of different domains, from business operations monitoring to tsunami detection or building surveillance. A specific class of event processing systems dealing with data gathered from distributed data sources are *E*arly *W*arning *S*ystems (EWS). These include for instance the Tsunami Early Warning System presented in [6], a seismic EWS for nuclear power plants described in [13] and a wildfire warning system presented in [3], to name but a few. The devices used to take measurements range from seismic stations in the first cases to embedded sensor nodes in the latter, employing wireless connectivity either through satellite or radio communication towards a central processing entity. As pointed out in [7], it is necessary to evaluate distributed event detection in networks featuring scarce resources in order to provide maximal system utility.

Different implementation options exist for addressing the different design goals. One major design decision is if and how (to what extent) to distribute event processing. However, the options for distribution are constrained by the available hardware. The challenge is to align the software architecture and the underlying hardware infrastructure to the targeted application setting. Designers must take monetary cost as well the application performance into account.

In this paper we address the combined tradeoff of choosing hardware components and software architectures. We provide metrics that quantify influential factors of the optimization goals and make the tradeoffs explicit. We furthermore show how different classes of event queries impact the application design and provide application designers with support for decision making.

Close to our work, [11] studies operator placement in hierarchies of (frozen) devices with different power capacity. However, the authors follow a more systematic approach than ours and our focus is more on design decision for human users (infrastructure designers). [4] presents a design tool for sensor-based applications in the context of *W*ireless *S*ensor *N*etworks (WSNs) only. In [12] the focus is on the cost of shifting event processing in wireless environments exclusively also. Especially in the domain of embedded networked sensors where resource constraints force an application programmer to consider system behavior before the actual deployment, a number of publications illustrate tradeoffs between system parameters. The approach described in [10] stores events inside a sensor network instead of streaming them towards a base station. The authors provide an estimation for the communication costs the network is exposed to for different types of storage and varying numbers of events and queries for events. They suggest a mechanism for load balancing among the nodes and thus illustrate the tradeoff between push- and pull-based data acquisition in sensor networks. A tradeoff between computation and communication has been identified in [8], proving communication to be the most energy intense operation on a node. A common technique to avoid energy wastage is to aggregate data before sending it, thus favoring local processing on the nodes wherever possible. The problem tackled in this paper is more general than the approaches above as (i) it considers any type of infrastructure, being wireless or not, and (ii) it regards the infrastructure as being variable.

The remainder of the paper is structured as follows. Section II provides background information. It presents the basic terminology for event handling and introduces a taxonomy that classifies events of interest. Section III describes the fundamental architectural options for event-based systems that serve as a reference for our analysis. Section IV presents our metrics to quantify optimization goals and shows how to apply the metrics to evaluate a system design. Section V summarizes the implications of the identified event classes on design options and optimization goals. Finally, Section VI concludes the paper.

II. Main Concepts

This section presents the main concepts needed to discuss the possible infrastructures for event handling. Some terminology can be found in [9]. We than introduce a classification of events occurring in such systems that may have an impact on the configuration of event-based systems.

A. Event-related concepts

An event is "something that happens", i.e., the state transition of an entity of interest at a certain point in time. A new RFID read, a new temperature value, or a storm coming into an area are examples of events. These examples illustrate the fact that event-based systems may consider events at many levels of abstraction, from the application level (e.g., an environmental application considering storms) down to the sensor level (e.g., a new temperature value on a particular thermometer).

We distinguish primitive events from complex (sometimes called composite) events. Complex events are derived from other (primitive or complex) events. They are specified in expressions (referred to as *event queries*) using operators such as conjunction or disjunction (e.g., temperature on Sensor S1 greater than 32F and temperature on Sensor S2 greater than 36F). Different applications consider different complex events. These can be more or less elaborate, for instance depending on whether they take time constraints into account or not (e.g., temperature on Sensor S1 greater than 32F and temperature on Sensor S2 greater than 36F within a five-minute interval). Event queries are expressed using languages such as SQL extended to time-related issues (e.g., [5]) or newly defined specification languages (e.g., [2]).

In the following, we abstract from the implementation and discuss generic concepts of event processing. When an event takes place we refer to its *occurrence*. With an occurrence is associated a time, the unit of which is expressed using a time referential. The latter itself depends on the application (it can be expressed in minutes, hours, days, and so on). When an event enters the system we refer to it as *raw event*. It is for

instance a measurement or data sample. (In the literature, a raw event is sometimes denoted a data item). A raw event can either be processed locally on a sensor - for instance to eliminate the noise - or not. In both cases we use the general term "event" unless we need to address explicitely raw events. Note that an event can occur without being "seen" (recognized) by the system (e.g., if a sensor is defective).

When an event is recognized by the system we refer to its *detection*. There may be a time lapse between event occurrence and event detection. Note that the process of event detection occurs on different levels of abstraction. On a low level, detection is simply capturing a sensor measurement. On a high level, detection is the match of an expression (the result of an event query) that specifies a complex event. When the system needs to evaluate the relevance of an event the latter is filtered. A *filter* evaluates a condition on an event. That is, the system finds out whether the event matches some conditions (e.g., "average temperature of Area A greater than 32F") in order to react to the event, often by sending a notification[1]. When a relevant event is filtered - i.e., satisfies a criteria - it is selected, which means that it can be used as such or as part of complex event detection. Throughout this paper we refer to operations with more than one input event (possibly coming from the same source) as complex operations. An example of complex operations is the average computation of temperature values over a certain time span.

Event processing refers to conducting operations for detecting events of interest that are grouped in collections of events defined by a query. For instance, the events "temperature dropped below 32F". Distributed event processing considers events coming from different sources.

In a list of selected detected events, an event can be used (consumed) instantaneously in a query and "forgotten" (discarded) or kept for future (re)use. In this case, we refer to history-sensitive events. They are kept in a history usually called trace. Such a history may take significant space, which is a criterion not to be neglected in distributed event processing. More information regarding event selection and detection can be found in [14], [1].

B. Taxonomy of Events

We now examine the two major dimensions of event handling that strongly impact resource consumption in a chosen configuration: (i) The first distinction is drawn between local and distributed event detection. This distinction is straightforward as it considers the number of devices which are involved within the process of event detection. While in a local detection an event may be detected by a single device, the data of several sensor have to be aggregated for a distributed event. This may be the case either when the sensing range of one sensor cannot cover a large enough area or when the accuracy of a single sensor is not sufficient to guarantee the demanded precision; (ii) the second dimension concerns the time-scale involved in the detection of an event, namely the amount of previously sampled data that has to be taken into account. In case an event can be immediately derived upon the arrival of a data sample, we refer to this as an instantaneous event. History-sensitive events rely on previously sampled data items. Consequently, samples must be kept in a buffer or synopsis data structure. Local acquisition of data samples on a sensor node affects the storage needed on each device. Both spatial and temporal event complexity are commonly subsumed under the concept of composite events, but have to be addressed separately in this context due to their differing impact on the investigated costs. According to these two dimensions of event handling, four different event collection classes are derived:

- Local, instantaneous event detection [Class EC_1]
  This class of event collections features the most basic event detection possible, where only one node is involved and no historical information has to be kept. An example of such an event collection is when a sensor value drops below 32F.
- Distributed, instantaneous event detection [Class EC_2]
  In this class, event collections encompass events that span over a spatial area, thus involve the participation of several nodes for detection, but memory does not have to be provided. A representative example is the detection of 32F in a certain area (e.g., a hill) or the average temperature value of an area becoming greater 35F.
- Local, history-sensitive event detection [Class EC_3]
  This class holds collections of events that require the temporal context; thus data storage is mandatory. An example is the detection of temperature 32F for the last five minutes.
- Distributed, history-sensitive event detection [Class EC_4]
  Event collections in this class encompass events that span over a spatial area and have a temporal context that has to be stored. An example is the detection of 32F for the last five minutes in a certain area.

TABLE I
CLASSES OF EVENT COLLECTIONS

	local	**distributed**
instantaneous	EC_1	EC_2
history-sensitive	EC_3	EC_4

Events from collection of these different classes naturally have a different impact on the costs for storage, network load, and event detection delay. Table I summarizes the event classes presented above.

III. ARCHITECTURAL OPTIONS

In order to illustrate the scope and application area of our approach and give an idea of the impact it may have on planning future system architectures, three complementary

[1]The active database community often refers to "Event - Condition - Action (ECA)" rules. That is, *events* occur and if they follow some *condition* some *action* is taken.

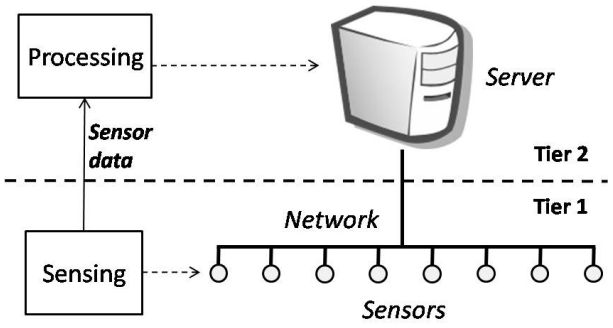

Fig. 1. Configuration I

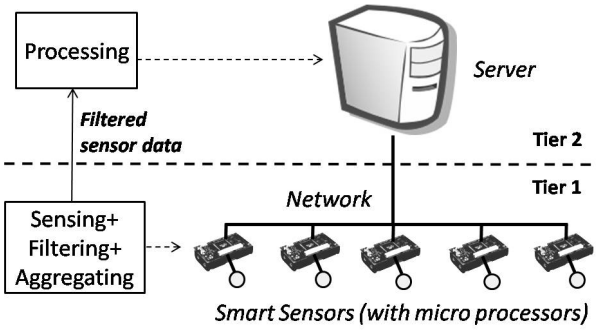

Fig. 2. Configuration II

system designs targeting the same application serve as a basis throughout this paper. We target systems that process information from a number of distributed sensing sources. The system design can vary in (i) the number of hardware tiers and (ii) the capabilities of devices on each tier. The three fundamental configurations that represent basic architectural patterns and that we consider here are: (I) a system design based on cheap sensors without processing capabilities, (II) a system design that uses smart sensors, and (III) a system design with a hierarchy of smart devices. Although we focus our discussion on these three configurations our method extends to variations of these design options analogously.

A. Configuration I: Centralized Event Processing

Configuration I, a classical centralized system design, features lightweight sensors capable of taking all the measurements needed. This configuration is visualized in Figure 1. The system design has two hardware tiers. The lower tier (Tier 1) comprises simple sensor devices. For Configuration I we explicitly consider cheap and simple sensors that provide no functionality beyond measuring a physical dimension. The upper tier (Tier 2) consists of a powerful processing unit. Typically this is a server with rich resources that is dedicated to the processing tasks. A system design such as this one can be found for instance in some manufacturing plants, typically in the process industry. For example, a manufacturer may collect data from machine sensors in its plant and trigger alerts if aberrations occur in the production. The sensors sample their environment regularly and forward them to a central server which does all the event processing. That is, it applies filters and complex operations on the input events in order to detect events that are of interest in the given application. Note that different application environments use different network technologies for communication between sensors and the server. For example, the fieldbus technology is a known option in manufacturing. Another example is ad-hoc sensor networks that use wireless hop-by-hop communication for transmitting sensor data.

B. Configuration II - Distributed Filtering and Aggregation

Configuration II features smart sensors equipped with a micro controller and a powerful central processing unit. This configuration is visualized in Figure 2. The system design has two hardware tiers. The lower tier (Tier 1) comprises smart sensor devices. In contrast to the sensors considered in Configuration I, these sensors are capable of conducting simple processing tasks. The upper tier (Tier 2) consists of a powerful processing unit with rich resources. Like in Configuration I this is typically a server that is dedicated to the processing tasks. A system design such as this one can be found, for instance, in RFID-based inventory monitoring. There, RFID readers embedded in smart shelves regularly detect tagged objects in their proximity and report changes to the monitoring application. In this example the smart RFID readers filter out redundant reads. That is, they report only if they observed a new object or if a previously observed object disappeared. Hence it is possible to exclude repetitive observations of the same object from reports to the central server. The server receives the filtered events from the readers and may apply more complex operations on them. For instance, the server may run a query that integrates all readers and checks if overall inventory drops below a threshold.

C. Configuration III - Distributed Event Detection

Configuration III features smart sensors and a hierarchy of processing units. This configuration is visualized in Figure 3. For the sake of simplicity, we assume in the following the existence of three hardware tiers in the system design. However, more tiers are possible. The lowest tier (Tier 1) comprises smart sensor devices like in configuration 2. The middle tier (Tier 2) includes devices with processing power and network connectivity to several sensors. The processing units in Tier 2 can be for instance dedicated servers in a wired network (often referred to as edge servers) or selected sensor nodes in a wireless sensor networks (often referred to as master nodes). The highest tier (Tier 3) comprises a powerful server, like in Configuration I and II.

A system design such as this one can be found for instance in sensor network applications. Here, selected nodes (master nodes) collect and process events from nearby sensors. Consider for instance an example from avalanche detection. This application uses smart sensor nodes that are distributed across a mountain side to measure snow conditions. The sensors organize themselves in local clusters and select a master node for their clusters. The master nodes are represented by Tier

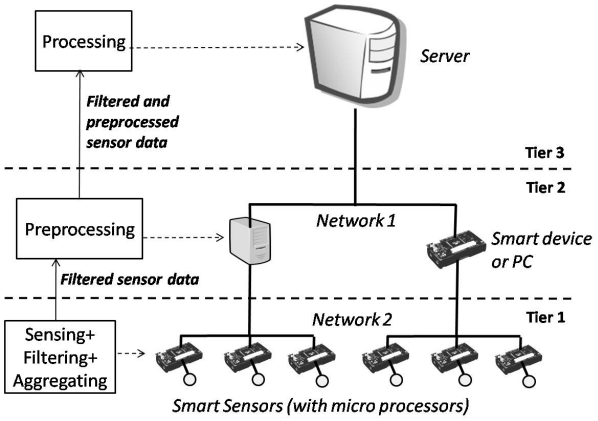

Fig. 3. Configuration III

2 in our sample architecture. (Other applications, e.g., in manufacturing, use dedicated computing devices in Tier 2).

Each smart sensor applies local filters to detect events of potential interest. Then sensors forward filtered events to the master node of the cluster. The master then applies complex event processing operators on the filtered input events. That is, the master node checks if the accumulated events from nodes in the cluster exceed a critical threshold and forwards corresponding events to the server in Tier 3. The central server then processes events from all clusters to check whether or not it should issue an avalanche warning.

Note that the described configurations can use different networks for connecting the different hardware tiers. For instance some installations in manufacturing use a fieldbus network for communication between sensors on Tier 1 and *P*rogrammable *L*ogic *C*ontrollers (PLCs) on Tier 2. The PLCs on Tier 2 can use Ethernet to communicate with the server in Tier 3. In other setups, all tiers can use the same network for communication. For instance sensor nodes on Tier 1 communicate via a wireless network to the master nodes on Tier 2 and the master nodes use the same network for communicating with the central server on Tier 3.

IV. Evaluating Candidate System Configurations

Application designers need to choose the most suitable system configuration given application-specific requirements, bearing in mind the cost of the infrastructure. It is necessary to quantify relevant properties of a candidate system configuration for making a well-founded decision. In the discussion we refer to these properties as cost parameters that should be minimized in choosing the system configuration. Note that use the term *cost* in a generic sense that captures monetary cost as well as resource utilization and performance parameters. In our work we focus on the four main cost parameters which strongly influence the decision on which configuration to favor in systems with multiple event sources. These are *network load*, *delay* in the event detection, *storage* utilization for processing, and *monetary hardware cost* for the hardware infrastructure. When looking at costs for event processing, the basic entity for which a certain cost is calculated is the collection E_i that holds the events of interest.

- The generated *network load*, denoted by $C_{network}(E_i)$, reflects the overall network load for detecting events in E_i in a given configuration. Note that this includes the detected events in E_i as well as any other events transfered through the network for detecting event in E_i in a given configuration.
- The *delay* of event detection is denoted by $C_{delay}(E_i)$. It defines the time it takes to transfer the needed information to a base station (central server). This notion of delay represents a lower boundary for any system responses to the events. Applications in many domains have strict constraints for these requirement, e.g. systems that must react in real-time.
- The *storage* needed in order to reliably detect and process events is indicated by $C_{storage}(E_i)$. This cost is an important factor to determine the resource settings of sensor nodes and therefore their physical dimension, but can be disregarded on the base station.

All these factors have an impact on the monetary hardware cost $C_{monetary}$ of the system. The challenge is to find a cost efficient solution that is suitable for a given application. In the following we discuss cost parameters for detecting events from the previously introduced classes using the different system configurations and resulting monetary cost for the hardware infrastructure.

A. Costs of Centralized Event Processing (Configuration I)

This architectural setup relies on a central entity that consumes raw events streams and conducts all operations of event processing. We consider the reporting rate of data items to be equal to the sampling rate

1) Network Load: All sensor nodes in Tier 1 forward their data samples to the central server in Tier 2 according to the given sampling rate r of sensors within an epoch e. The cost function $C_{network}(E_i)$ in terms of packets transmitted for recognizing events does not rely on the introduced event classes; as in this configuration, all samples are forwarded. Thus, the network of detecting events collections E_i of any event class can therefore be described as

$$C_{network}(E_i) \quad = \sum_{\forall N_j \in N} r * c_{N_j,BS} \tag{1}$$

where N is the set of sensors on Tier 1 and $c_{N_j,BS}$ is the network load caused by transferring one sending unit from node N_j to the base station BS. Note that $c_{N_j,BS}$ depends on network specific factors, such as the topology and the used protocols.

Dependent on the topology, $c_{N_j,BS}$ may return different values for different locations of N_j in the network. In networks where messages travel multiple hops, $c_{N_j,BS}$ depends on the network distance (number of hops) between N_j and BS. For a

simplified estimation of the network load one may work with an average value for $c_{N_j,BS}$.

2) Delay of Event Detection: If each node simply forwards its samples, the delay of the event detection is given by the time $T_{send}(N_j, BS)$ that the network needs to forward the samples correlated to the event and the time the base station needs to process the incoming samples $T_{p,BS}(E_i)$. Hence the cost function $C_{delay}(E_i)$ for calculating the delay is

$$C_{delay}(E_i) = T_{send}(N_j, BS) + T_{p,BS}(E_i) \tag{2}$$

T_{send} heavily depends on the used network technology, i.e. speed of the data link. Like $c_{N_j,BS}$, $T_{send}(N_j, BS)$ may depend on the location of N_j because the number of network hops is reflected in the delay. For events from the collections in the classes E_1, E_3 one may work with the average time to approximate $T_{send}(N_j, BS)$ in estimating the delay. This is feasible, because detection of E_1, E_3 must only consider one event source at a time. For detecting events in collections from the classes E_2, E_4 one needs data from different sources. The time $T_{send}(N_j, BS)$ therefore depends on the source with the longest network distance. For estimating $T_{send}(N_j, BS)$ one should therefore use the maximum value as approximation.

3) Data Storage: Once forwarded to the base station, it is assumed that secondary storage is used to log the data. As it is not a cost-intensive factor in such a system, a very large amount of data (as opposed to the memory size deployed to sensor nodes) can be kept. This enables aposteriori analysis and evaluation of the gathered sensor data at a later point in time. Since data storage cost can therefore be considered to be negligible in a centralized setting, the influence of the temporal dependency of events does not manifest itself in additional overhead in terms of storage, but solely in a necessity for additional processing. Within this configuration storage is not a critical resource; thus its corresponding cost $C_{storage}(E_i)$ is neglected.

B. Costs of Distributed Event Preprocessing (Configuration II)

Distributed preprocessing involves processing of filters and aggregates on the sensor nodes themselves. That is, instead of directly sending data items to a central entity, sensor nodes can locally apply operators on their measurements. Configuration II enables this local preprocessing of events at the sensor nodes (i.e. Tier 1). In the following, $f_{local}(E_i, N_j)$ denotes the frequency of the local occurrence of an event in a class E_i that is detected by an operator at node N_j within a given life span. For comparability reasons, we assume an event notification to fit into one sending unit.

1) Network Load: The network costs in the distributed setting depend on whether an event is local or distributed, but does not depend on the time span that is needed to recognize an event. Thus the costs for detecting an event of a collection in class EC_1 are the same as for an event of an collection in class EC_3. The costs for detecting an event in a collection of class EC_2 are the same as for an event of a collection from class EC_4. A sensor node sends a locally detected event (which can be composite) to Tier 2; hence the cost in terms of packets transmitted for detecting a local event from a collection $E_i \in \{EC_1, EC_3\}$ are:

$$C_{network}(E_i) = \sum_{\forall N_j \in N} f_{local}(E_i, N_j) * C_{N_j,BS} \tag{3}$$

where N is the set of sensors on Tier 1 and $C_{N_j,BS}$ are the costs for sending a packet from node N_j to the base station BS, comparable to the centralized setting. For $E_i \in \{EC_2, EC_4\}$ the network load is the same as in Configuration I. Like in Configuration I, $c_{N_j,BS}$ may depend on the network position of N_j, if messages travel via multiple hops. This is particularly the case if the smart sensors in Tier 1 are deployed as a wireless ad-hoc sensor network. In these networks, sensor nodes pass on messages from their neighbors to forward them toward the base station. Thus, messages from far distant nodes N_j travel more hops, resulting in a higher value for $C_{N_j,BS}$. For a simplified estimation of the network load one may work with an average value for $c_{N_j,BS}$.

2) Delay of Event Detection: The cost function to specify the delay of reporting a local event of a collection in class EC_1 is the sum of the time $T_{p,local}(E_i)$ needed to process the data on the sensor node to detect this event and the time $T_{send}(N_j, BS)$ for sending the event notification to the base station.

$$C_{delay}(E_1) = T_{p,local}(E_i) + T_{send}(N_j, BS) \tag{4}$$

Note that like in Configuration I, $T_{send}(N_j, BS)$ is network dependent and for multi-hop communication depends an the position of N_j.

Events involving the evaluation of history information on a node contribute to the overall complexity of the event detection scheme. This increase in complexity can be observed as a longer processing time depending on the size of history information considered per event $T_{P,mem}(E_i)$, as well as an increased demand for data storage. Therefore, the delay for an event from a collection in class EC_3 can be expressed as:

$$C_{delay}(E_i) = C_{delay}(E_1) + T_{p,mem}(E_i) \tag{5}$$

Configuration II processes events of collections in the classes EC_2 and EC_4 centrally. Thus, the delay for detecting events in these event classes is the same as for Configuration I.

3) Data Storage: With the advent of flash memory, integration of storage space of several gigabytes (for the time being) for sensor nodes has become reasonable. Thus, logging of certain data and events on the sensor nodes has become possible. Nevertheless, the memory size of sensor nodes is still very limited, especially when requested to store continuous streams of raw data. If raw data are stored on the sensor nodes, then logging of events, as a means of data compression, is limited by the size of the memory on the sensors. A posteriori analysis of data can only be done for the recent past, thus automatically limits the capability of history-sensitive event detection to a storage-dependent time frame. This is to be considered when designing the system in order to be able to keep all necessary historical data.

C. Costs of Distributed Event Detection (Configuration III)

Distributed event detection involves in-network processing of complex events. This is in contrast to Configuration II, where only simple filters over event attributes and aggregates over the local measurements are processed decentrally. Tier 2 in the three-tier architecture of Configuration III enables decentralized processing of complex events with multiple - possibly spatially distributed - input sources as well. In the following, $f_{spatial}(E_i, N_m)$ denotes the occurrence of a spatially distributed event in collections E_i at the master node N_m (i.e. device on Tier 2 in Configuration II).

1) Network Load: Distributed event detection relies on message exchange of several nodes. Given a predefined clustering with a determined master node N_m, the simplest algorithm to achieve the notion of a spatial event is for a sensor node transmitting the local event to notify its master node. The set of sensor nodes belonging to the cluster with master node N_m is referred to as S_{Nm}. In case a predefined threshold number of local event occurrences are reported by the participating nodes within a time frame, i.e. an epoch, the master sends out an event to be transferred to the base station. With M being the set of all master nodes (devices on Tier 2), the cost the collaborative detection of events in collections $E_i \in \{EC_2, EC_4\}$ imposes can therefore be described as:

$$\begin{aligned} C_{network}(E_i) &= \sum_{\forall Nm \in M} f_{spatial}(E_i, N_m) * C_{Nm,BS} \\ &+ \sum_{\forall N_m \in M} \sum_{\forall N_j, \in S_{Nm}} f_{local}(input(E_i), N_j) \\ &* C_{N_j,Nm} \end{aligned} \tag{6}$$

where $input(E_i)$ is an collection of events needed for the later detection of events in E_i, the costs for sending a packet from senor N_j on Tier 1 to the corresponding processing unit N_m on Tier 2 is C_{N_J,N_m}, and $C_{N_m,BS}$ is the costs for sending a packet from a processing unit N_m on Tier 2 to the central server BS on Tier 3.

Note that C_{N_m,B_S} may weigh differently than C_{N_j,N_m}. This is because communication from Nm to BS may go via a different network than communication from N_j to Nm. Consider for example a candidate setup with battery-powered smart sensor nodes on Tier 1, PCs as base stations in Tier 2, and a server in Tier 3. The smart sensors in Tier 1 may use wireless communication for transferring event messages to a nearby base station in Tier 2. The base stations in Tier 2 may use Ethernet for communication with the server in Tier 3. In this scenario, network load between Tier 1 and Tier 2 is the major concern. This is because the wireless communication uses up the battery of the sensors and the bandwidth is lower then in the Ethernet. We can account for this in the estimation of $C_{network}(E_i)$ in the definitions of C_{N_j,N_m} and $C_{N_m,BS}$.

For $E_i \in \{EC_1, EC_3\}$ the network load is the same as in Configuration II.

2) Delay of event detection: Configuration III processes events of collection in the classes EC_1, EC_3 locally on the sensor nodes in Tier 1. Thus the delay for detecting events from these collections is the same as in Configuration II. The processing and thus the delay differs for distributed events. Distributed event detection relies on the cooperation of several nodes. The respective cost function $C_{delay}(E_2)$ for the delay reflects this in terms of additional time needed. Local node processing time, time spent for sending the event from the location of local event occurrence N_s to a master node N_m and from a master to the base station as well as the maximum time fraction a master node waits for incoming local event notifications including the processing time $T_{p,spatial}(N_m, E_i)$ sum up to the overall costs in delay:

$$\begin{aligned} C_{delay}(E_2) &= T_{p,local}(N_s, E_2) + T_{send}(N_s, N_m) \\ &\quad + T_{p,spatial}(N_m, E_2) + T_{send}(N_m, BS) \end{aligned} \tag{7}$$

where E_2 is an event from collection in the class EC_2.

Note that $T_{send}(N_s, N_m)$ may differ from $T_{send}(N_m, BS)$. This is because different networks - with possibly different latencies - may facilitate the communication from Nm to BS and Ns to Nm. Also, one must model the potential impact of the locations of N_s and N_m separately for each network. Similar to Configuration II, one must also consider that detecting events from collections in the class EC_2 requires input from multiple sources. Thus using the maximum value for T_{send} is suitable for an approximate estimation of the delay.

Events involving the evaluation of history information on a node contribute to the overall complexity of the event detection scheme. This increase in complexity can be observed as a longer processing time depending on the size of history information considered per event $T_{P,mem}(E_i)$, as well as an increased demand for data storage addressed in the next subsection. Therefore, the delay for events in a collection from class EC_4 can be expressed as:

$$C_{delay}(E_i) = C_{delay}(E_2) + T_{p,mem}(E_i) \tag{8}$$

3) Data Storage: The concerns for memory consumption in Configuration III are similar to those in Configuration II. A difference is in the required memory for detecting complex events. Some operators for complex event processing must accumulate events from different sources in a buffer. An example is calculating the median from different sources. Configuration II processes such operators on the central server, where memory is not a major concern. Configuration III processes these operators in the middle tier (Tier 2). It heavily depends on the application domain what devices are available for this tier and if memory is a limited resource. It can for instance be suitable to use PCs or dedicated sensor nodes for processing in Tier 2. The domain specific tradeoff between different hardware options is captured in the following subsection.

D. Hardware Cost

Existing approaches for optimizing the deployment of event queries assume a given hardware infrastructure to be in place. However, this is not the case when application designers build systems from scratch. In such cases, the hardware infrastructure itself becomes a variable in the optimization problem of finding the most suitable configuration. Application designers can chose from different options for some hardware components or for the whole hardware infrastructure. The most common design goal is to keep the hardware infrastructure cheap. However, low-cost hardware poses limitations to the options for the software design. Finding the optimal solution is consequently a tradeoff between options for the hardware and the software deployment. For each tier, designers must chose from the available devices on the market, considering monetary cost and application requirements. In the following we discuss how different system designs impact the cost structure of the solution.

Each configuration for an application corresponds to a mapping $conf \subseteq Op \times Ht$ of event processing operators $op \in Op$ to the different hardware tiers $t_i \in Ht = \{t_1, t_2, t_3\}$. This mapping defines the hardware requirements for the devices in each tier. Namely, operators need a certain amount of memory and CPU power. The memory requirements for an event processing operator op are defined by the size of the synopsis data structure that must be kept for processing op. The required CPU power is defined by the tolerable delay for processing operator op on the respective device. One upper bound for the tolerable delay is defined by maximum delay that the application allows for event detection. Another bound is defined by the rate at which input events arrive at op. The target device must process events at least at this rate to avoid buffer overflows.

Formally, the requirements to run an operator op are given by the function $ReqMem(op) \mapsto mem$ that defines the required memory and $ReqCPU(op) \mapsto cpu$ that defines the required processing power. For each hardware tier N designers can choose from a set of available device types $DevTypes_N$ (e.g. available devices on the market). A device type is defined by the tuple $(mem, cpu, cost)$ where mem is the available memory, cpu the CPU power, and $cost$ the monetary cost of the device.

For a configuration using tiers 1-N, with $\#t_N$ being the number of devices on tier N, and the set $DevTypes_N$ being the set of possible device types for populating tier N, the cost of the configuration is defined by:

$$\begin{aligned} &TotalDeviceCost(conf) = \\ &\sum_{i=1}^{N} \#t_i \cdot min(\{cost | (mem, cpu, cost) \in DevTypes_i \\ &\qquad \wedge mem \geq \sum_{(op,N) \in conf} ReqMem(op) \\ &\qquad \wedge cpu \geq \sum_{(op,N) \in conf} ReqCPU(op)\}) \end{aligned} \quad (9)$$

In addition to the processing devices, one must consider the monetary cost for the employed network infrastructure. The different configurations pose different constraints to the network. That is, each configuration results in different requirements for bandwidth and network delay. Furthermore, a certain network infrastructure constrains the available device types $DevTypes_N$ by defining the required network interface. This results in a mutual dependency of the monetary network cost and the monetary device cost. To determine the total hardware cost $C_{monetary}$, one must calculate $TotalDeviceCost(conf)$ under the constraints of different network choices. The value for $C_{monetary}$ is the cost of the configuration where the sum of $TotalDeviceCost(conf)$ and the monetary cost for the network infrastructure are minimal.

V. Tradeoffs in Choosing Configurations

The benefit of our cost estimation model is to support the choice for a specific configuration for event processing. Yet, the introduced cost parameters are interdependent and optimizing them can result in conflicting goals. It is therefore important that application designers understand the mutual dependencies to assign suitable weights to the different cost parameters. In this section we discuss these tradeoffs and their impact on the application. We conclude this section by providing an overview of how the identified classes of event queries relate to system configurations and what tradeoffs result.

A. Data Rate vs. Event Rate

High data rates are usually needed when the occurrence of critical events (e.g. a value rising above a critical threshold) has to be detected with a minimal delay, or the temporal complexity of events demand for a high data granularity. On the other hand, the event rates are bounded by the data rate while the event rate does not influence the data rate. Note that from a data perspective, events can be seen as a semantical compression of data streams, so the increase of the event rate in consequence of an increase of the data rate will be orders of magnitude smaller.

A higher data rate will directly impact the network load within the centralized setting. Increasing the data rate will therefore lead to a factorial increase of the network costs of an event. Within the distributed, in-network configuration, a higher data rate does not influence the network costs whatsoever (unless it correlates with the event rate).

Whenever the event rate within the application scenario is raised (which is basically determined by the application domain), the distributed, in-network configuration has to accept higher network load. As long as the capabilities of the base station are not exceeded, a centralized approach is oblivious to a variance of the event rate.

B. Time-Critical Event Detection vs. Event Complexity

Early Warning systems demand for event reporting within a predefined, usual very tight time interval. When evaluating whether this interval can be met with a chosen configuration, two parameters dominate this analysis: (1) the time needed for sending a message and (2) the time needed for processing, independent of the considered event class or choice of configuration. In the centralized approach, network congestion can increase the network delay. This effect may bring decentralized solutions into favor. However, it is important to note that devices on lower hardware tiers usually have only little processing power. An example is smart sensor nodes that may populate Tier 1 in Configuration II or Tier 1+2 in configuration III. For highly complex events, the processing time for event detection on lower tiers dominates the delay. The centralized approach may be favored, if the resulting processing delay is not tolerable.

C. Energy Consumption vs. Temporal Event Complexity

When deploying battery sensor nodes, the available amount of energy per node directly influences network and thus application lifetime. In such an environment, network load becomes a very critical factor. Since communication is the most energy intense operation a node can execute, algorithms generally prefer processing over communication whenever applicable.

In a distributed setting, a high temporal event complexity demands for high storage capabilities on sensor nodes. Energy-wise these are to favor over a centralized configuration as long as the energy spent on network load exceeds the corresponding costs in terms of energy for local event processing. Higher temporal event complexities (accounted for in number of samples) lead to a greater energy consumption for event processing in a distributed approach. However, due to the inherent data compression nature of events, this will highly cut the costs in network load.

D. Relation between Event Query Classes, Cost Factors and Configurations

As pointed out above, application designers must trade off network load, delay in event detection, and monetary cost of the candidate configurations. Each of these factors has a application specific weight in the decision process. Given these weights and the formulas presented in Section IV, application designers can choose the most suitable configuration for the targeted class of event queries. The decision can be treated as the optimization problem of minimizing the weighted sum of $C_{network}$, C_{delay}, and $C_{monetary}$. For different event classes and different weights of the cost factors, this brings different configurations into favor.

The weights on each cost factor depend on the application domain. The factor $C_{network}$ usually has a high weight in applications with autarkic, battery-powered devices, e.g. sensor networks. This is because high network load results in heavy usage of the limited energy resources and shortens the application lifetime. The weight on $C_{network}$ is typically low for setups with wired networks. However, designers may use a high weight if the event detection application must share the network resources with other applications and the available bandwidth is limited.

The weight on the cost factor C_{delay} depends on the time constraints of the application. Applications that trigger actuators in response to detected events can have real-time requirements and therefore put a high weight on C_{delay}. An example for such applications is the detection of hazardous situations and the corresponding triggering of an alarm. Other applications can tolerate some delay in the event detection and put a low weight on C_{delay}. For example, some manufacturers monitor the current performance of production stations, but can often tolerate several seconds delay.

With regard to $C_{monetary}$, the aim is generally to keep this factor low. However, this may come at the cost of higher values for $C_{network}$ and C_{delay}. The weights on these factors must be set to reflect this tradeoff. In addition, one can define constraints on each factor in terms of absolute bounds. For instance, real-time requirements can pose an absolute limit to the factor C_{delay}, or the total available bandwidth may constraint $C_{network}$.

Table II gives an overview of the relation between the application specific relevance of cost factors, event query classes, and system configurations. Note that some configurations are not suitable for all classes of event queries. For the remaining cases the table shows the most relevant factors at a glance and indicates which configurations are likely to be suitable.

VI. Conclusion

Many current application fields such as disaster management or manufacturing rely on event processing. Event sources may be basic sensors with no processing capabilities or elaborate sensors with sophisticated functionalities for event processing or pre-processing, leading to different options for the underlying infrastructure.

In this paper we studied three major types of infrastructures in this context, namely (i) centralized event processing, (ii) distributing filtering and aggregation, and (iii) distributed event detection. We identified key parameters to consider such as the network load, the delay of event detection, and the storage needed to store events when the processing requires historical data. Finally, we presented the combined tradeoff of choosing hardware components and software architectures. Our analysis

TABLE II
CASE-SPECIFIC ADVANTAGES OF THE DIFFERENT CONFIGURATIONS

	$Conf_1$	$Conf_2$	$Conf_3$
EC_1 (Local, instantaneous Events)	- If cost for sensor CPU are high - If low delay in event detection is needed	- If low network load is required - If cost for sufficiently fast central server are high	–
EC_2 (Local, history-sensitive Events)	- If cost for sensor CPU are high - If cost for sensor memory are high - If low delay in event detection is needed	–	- If low network load is required - If cost for sufficiently fast central server are high
EC_3 (Local, history-sensitive Events)	- If cost for sensor CPU are high - If low delay in event detection is needed	- If low network load is needed - If cost for sufficiently fast central server are high	–
EC_4 (Distributed, history-sensitive Events)	- If cost for sensor CPU are high - If cost for sensor memory are high - If low delay in event detection is needed	–	- If low network load is required - If cost for sufficiently fast central server are high

provides guidance for application designers in choosing the most suitable system configuration for a particular application.

Acknowledgments. We wish to thank Katharina Hahn and Kirsten Terfloth for useful feedback. Part of this work beneficiated from collaboration with them. We also wish to thank Christopher Switzer for help on the last version.

REFERENCES

[1] J. Carlson and B. Lisper. An event detection algebra for reactive systems. In *Proc. of the 4th ACM Intl. Conf. on Embedded Software (EMSOFT)*, New York, NY, USA, 2004. ACM.

[2] S. Chakravarthy and D. Mishra. SNOOP: An Expressive Event Specification Language for Active Databases. *Journal of Data and Knowl. Engineering*, 14(1), 1994.

[3] D. Doolin and N. Sitar. Wireless Sensors for Wildfire Monitoring. In *Proceedings of the SPIE Symposium on Smart Structures & Materials/ NDE 2005*, March 2005.

[4] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, and F. Reiss. Design considerations for high fan-in systems: The hifi approach. In *Proc. of the Int. Conf. on Innovative Data Systems Research (CIDR)*, 2005.

[5] S. Gatziu and K. R. Dittrich. SAMOS: An Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):23–26, 1992.

[6] GTZ-Potsdam. German Indonesian Tsunami Early Warning System. http://www.gitews.org/, May 2007.

[7] J. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, and Michael Zink. An End-User-Responsive Sensor Network Architecture for Hazardous Weather Detection, Prediction and Response. In *Proceedings of the Asian Internet Engineering Conference (AINTEC 2006)*, pages 1–15, November 2006.

[8] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor For Sensor Networks. In *Proceedings of the ACM SIGMOD*, June 2003.

[9] G. Muehl, L. Fiege, and P. R. Pietzuch. *Distributed Event-based Systems*. Springer Verlag, 2006.

[10] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT, a geographic hash table. *Mobile Networks and Applications (MONET)*, 8(4):427–442, 2003. Special Issue on Wireless Sensor Networks.

[11] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proceedings of ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS)*, New York, NY, USA, 2005. ACM.

[12] K. Terfloth, K. Hahn, and A. Voisard. On the cost of shifting event processing within wireless environments. In *Proceedings of the Dagstuhl Event Processing Workshop*, 2007.

[13] M. Wieland, L. Griesser, and C. Kuendig. Seismic early warning system for a nuclear power plant. In *Proceedings of the 12th World Conference on Earthquake Engineering (WCEE 2000)*, February 2000.

[14] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, page 392, New York, N.-Y., USA, 1999. IEEE Computer Society.