Power-Aware Data Analysis in Sensor Networks

Daniel Klan #1, Katja Hose #2, Marcel Karnstedt *3, Kai-Uwe Sattler #4

#*Department of Computer Science & Automation*
Ilmenau University of Technology, Germany
1`daniel.klan@tu-ilmenau.de`
2`katja.hose@tu-ilmenau.de`
4`kus@tu-ilmenau.de`

**Digital Enterprise Research Institute*
National University of Ireland, Galway
3`marcel.karnstedt@deri.org`

Abstract— **Sensor networks have evolved to a powerful infrastructure component for event monitoring in many application scenarios. In addition to simple filter and aggregation operations, an important task in processing sensor data is data mining – the identification of relevant information and patterns. Limited capabilities of sensor nodes in terms of storage and processing capacity, battery lifetime, and communication demand a power-efficient, preferably sensor-local processing. In this paper, we present AnduIN, a system for developing, deploying, and running in-network data mining tasks. The system consists of a data stream processing engine, a library of operators for sensor-local processing, a box-and-arrow editor for specifying data mining tasks and deployment, a GUI providing the user with current information about the network and running queries, and an alerter notifying the user if a better query execution plan is available. At the demonstration site, we plan to show our system in action using burst detection as example application.**

I. Introduction

In the past couple of years, sensor networks have been in the focus of research and industry. Sensor networks consist of hundreds or even thousands of battery-powered, wirelessly connected sensor nodes. Due to wireless connections and autonomous organization, sensor networks have become popular for covering very large areas and applications on short time observations. In these cases, it is usually too expensive to build a wired infrastructure. A typical scenario is monitoring environmental changes, e.g., in traffic systems and buildings.

Limitations of sensor node hardware reveal a lot of interesting challenges. One of the main issues in this context is the reduction of the nodes' power consumption in order to increase network lifetime. The main concepts to reduce power consumption are to exploit the economical sleep mode and to minimize the expensive radio communication, because this is the most dominating cost factor in sensor networks [1]. A very promising approach to reduce radio communication is in-network processing of data and queries. This means, instead of sending each sampled data item directly to a base station, data reducing operations, such as filters and aggregations, are processed by the nodes when propagating their readings along a logical routing tree towards the base station.

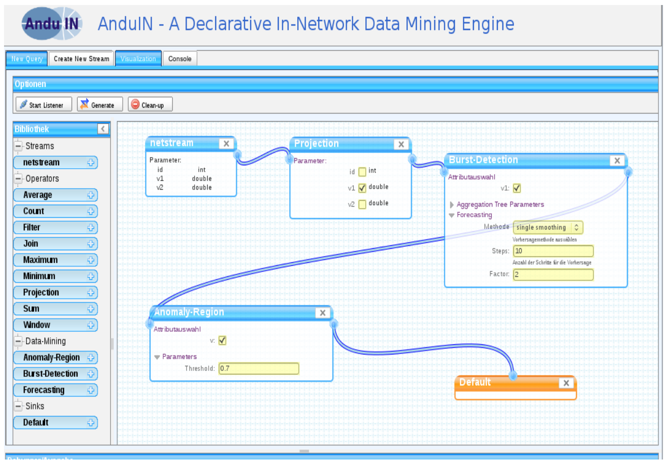
TinyDB [2] and Cougar [3] are examples for in-network stream processors (INSPs) allowing users to formulate queries using an SQL-like syntax. They support queries referencing basic operators such as filters, joins, and aggregations. However, users are often interested in more complex queries involving data mining operations such as clustering, frequent pattern mining, and burst detection. In order to overcome this problem, INSPs can be combined with data stream management systems (DSMSs) [4]. An INSP aggregates sensor data, sends aggregated data to the DSMS, which then processes complex operations on the resulting data streams.

Optimizing these two components in separate results in suboptimal solutions, because existing DSMS approaches simply do not consider in-network query processing as an option for optimization. For instance, computing burst detection in-network directly on the sensor nodes could be much more efficient. Adapting parameters for in-network burst detection based on DSMS feedback is a promising option to dynamically adjust to the current state of the network. Nevertheless, this is an issue to which much attention has not yet been paid.

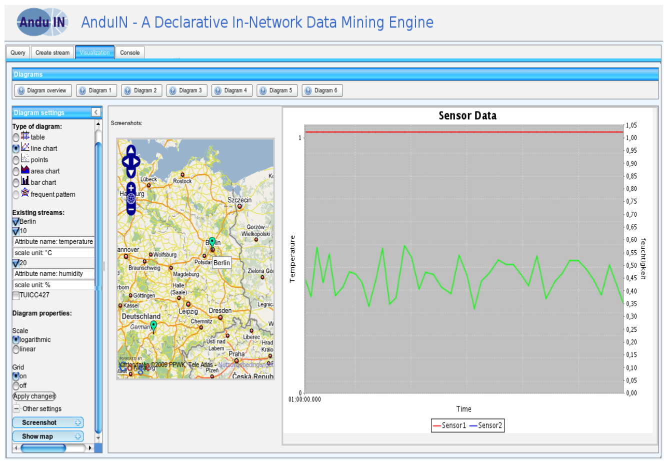
On a lower level, network deployment is another interesting aspect of optimization, which can contribute to increasing a sensor's lifetime. However, defining an optimal routing tree to perform aggregations efficiently is an NP-hard problem [5] so that in general heuristics are used to construct suboptimal routing/aggregation trees. Apart from an automatic construction, a manual deployment of logical routing trees is more reasonable in some applications. Modern building observation is such an example. Assume each room contains a number of sensor nodes monitoring aspects such as movement, temperature, humidity, etc. In general, natural hierarchies (e.g., rooms, wings, floors) are not mapped to the physical network layer when applying automatic solutions. For some operations, such as the anomaly region detection presented in [6], logical layers conforming to these natural hierarchies are necessary. In these cases, the developer should be able to develop and deploy the logical network layer manually.

To address these problems, we have developed AnduIN, a system for power-efficient in-network data mining. In particular, the main contributions of our work are:

- *Data Mining* – computation of complex data mining operations partially or completely in-network
- *Query Optimization* – cost-based power-efficient query



(a) AnduIN's GUI Component - Query Part



(b) AnduIN's GUI Component - Visualization Part

optimization and query decomposition in consideration of logical network layers

- *Network Deployment* – the option of manual creation of logical network layers in a user-friendly manner
- *Usability* – an innovative box-and-arrow GUI to define queries and logical network layers so that knowledge of a specific query language is no longer a mandatory requirement
- *Alerter* – a graphical alerter visualizing feedback from the sensor network and raising alerts if a better plan with respect to a running query and current sensor feedback is available

II. System Overview

AnduIN consists of four main components: the sensor network, a central data stream management system, in-network stream processors, and a graphical user interface. The sensor network component is based on ScatterWeb nodes [7] running the Contiki operating system [8]. For data stream processing, we implemented a library supporting a set of standard operations (e.g., filters, projections, aggregations) as well as a set of complex data mining operators. Due to limited sensor memory size, we cannot integrate the whole library into one image running on the nodes. Thus, a long-running query in AnduIN results in an individual image, which has to be disseminated within the sensor network before the query can be processed.

To illustrate the general steps of query processing and optimization in AnduIN, let us consider an example query. Assume a deployed sensor network delivers a stream S consisting of tuples containing a timestamp, the current temperature, and the humidity. Further assume we want to detect anomaly regions based on adaptive burst detection, as considered in [6]. The standard approach towards query processing is to formulate a CQL [9] query. As CQL does not support nested queries, we first have to define a view and then query this view. The following two CQL statements specify the complete data mining task:

```
create stream  S_BURST as
select temp, timestamp
from S [ burst-detection(w => 100,
          threshold =>'Holt Winters') ];
select temp, timestamp
from S_BURST [ anomaly-region(t => 10)];
```

This query detects bursts over a sliding window of size 100 and the burst threshold is adapted by Holt-Winters forecasting. The output of the burst detection is used as input for region detection, i.e., the result of the overall query consists of regions with abnormal behavior, with t denoting a threshold for anomaly regions.

Instead of having to specify queries directly in CQL, AnduIN's graphical GUI component, which was inspired by Yahoo! Pipes (*http://pipes.yahoo.com/*), allows for a much more convenient alternative to formulate queries. The GUI follows the popular box-and-arrow paradigm, which enables users to create data flow processes by connecting operators through click, drag, and drop functionalities. When the user finishes editing a query, the graphical query representation is transformed into one or multiple CQL queries. The graphical representation corresponding to the above query is illustrated in Figure 1(b).

The GUI component also offers the opportunity to deploy sensor network topologies. The user can choose from a number of registered sensor nodes and graphically connect them to form a logical routing tree. In analogy to queries, the developed tree is transformed into a CQL query and sent to the stream engine. Afterwards, the user can choose the newly created sensor network hierarchy as a source from the library to formulate queries.

Similar to traditional database management systems, AnduIN's central core engine consists of a rewriter generating a logical query plan based on the successfully parsed CQL query [9], a rule-based logical optimizer reordering operators within the logical query plan, and a physical optimizer translating logical query plans into physical execution plans. After having parsed the query, we obtain a query plan describing

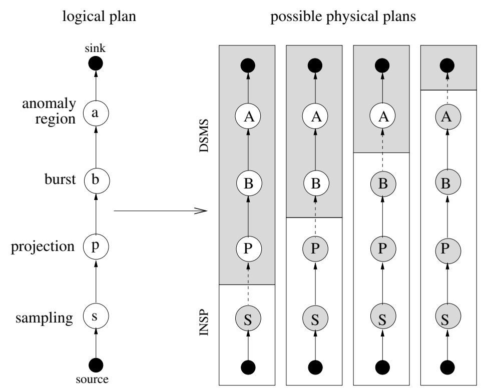


Fig. 1. Example Query Plan Optimization

all logical operations that need to be performed in order to answer the query. After logical optimization, we obtain a plan describing both mining tasks: in-network query processing and data stream processing. The leftmost plan of Figure 1 shows the logical plan corresponding to the query introduced above. A physical plan consists of two parts: one concerning the DSMS and the other concerning the INSPs. The physical plans on the right hand side illustrate all possible physical plans. The dotted line represents the data exchange between both parts. The optimizer chooses the plan minimizing power consumption. The challenge is to find the decomposition that minimizes power consumption within the sensor network.

III. Query Optimization

In order to find the optimal plan from the complex space of possible query plans, we use a rule-based optimization heuristic in conjunction with a cost-based transformation. The replacement of a logical operator is an operator representing a specific implementation, indicating, for example, a local or an in-network implementation. AnduIN's optimizer estimates power consumption for physical plans with different in-network parts. Based on a sensor network with m nodes, the remainder of this section discusses several approaches for distributed computation and introduces the corresponding cost estimations.

A. Central Processing

Applying a centralized approach, nodes sample periodically and send the data to a central server (for instance, a DSMS like AnduIN's core engine) that processes these data. The costs for sampling depend on the used sampling rate r_s, the microcontroller unit (MCU) of the sensor nodes, wake up costs c_{wake}, and the costs c_{sample} for taking one physical sensor measurement. We assume message costs of c_{msg}, comprising the header information and the sampled sensor data. We assume tha message costs for sending and retrieving are equal – thus, one message results in $2 \cdot c_{msg}$. Additionally, we assume that the sensor nodes are organized in a balanced tree, where h denotes the average number of hops from a source node to the central instance in the tree. This results in the following estimated costs for central processing model:

$$\begin{aligned} c_{centr} &= m \cdot r_s \cdot [c_{wake} + c_{sample} \\ &\quad +2 \cdot h \cdot c_{msg} + (h-1) \cdot c_{wake}] \end{aligned}$$

As sensor networks are usually built ad-hoc, we have to find a way to estimate the average number of hops h. An appropriate approach might use a feedback message mechanism and work as follows: a broadcast message is sent to all nodes within the sensor network. When broadcasting messages, hops are counted. Finally, the nodes reply with the number of hops that were necessary to contact them. For simplification, we assume no buffering of messages at intermediate nodes – this means, e.g., that a receiving node has to be waked up for every single message.

B. In-Network Processing

In the second approach, each node processes its data locally immediately after the sampling step. We denote the costs for processing an operator j on a sensor as $c_{cpu}(j)$. Consequently, the sequential processing of p operators results in additional costs of $\sum_{j=1}^{p} c_{cpu}(j)$. But, processing data in-network also results in a reduction of message volume and thus in a reduction of energy consumption, which has to be considered in the cost model. Thus, for each operator we need to consider its selectivity σ_j, $0 \leq \sigma_j \leq 1$. A selectivity of 1 indicates an operator whose output contains the same number of tuples as its input. For instance, a projection operator forwards the same amount of data that it receives. In order to reduce the number of messages significantly, special interest lies on the in-network processing of operators with a selectivity of $\sigma \ll 1$. The costs for in-network processing are:

$$\begin{aligned} c_{innet} &= m \cdot r_s \cdot \Bigg[c_{wake} + c_{sample} + \sum_{j=1}^{p} c_{cpu}(j) \cdot \sigma_{j-1} \\ &\quad + (2 \cdot h \cdot c_{msg} + (h-1) \cdot c_{wake}) \cdot \prod_{j=1}^{p} \sigma_j \Bigg] \end{aligned}$$

As processing an operator j results in constant costs $c_{cpu}(j)$ for all sensors, we can measure this overhead once for each operator. In contrast, the selectivity σ_j also depends on the data distribution ($\sigma_0 = 1$). To estimate selectivity, we can either use query feedback obtained from previous queries or default values.

C. Neighborhood-Based In-Network Processing

In dependence on the mining task, it is possible that not all operations can be processed by the sampling sensors. A solution is to establish a hierarchy of sensors so that, e.g., one node per cluster (neighborhood) exclusively computes operations for its cluster (neighborhood). Thus, we have to distinguish between two kinds of nodes: those performing sampling and basic operations and those performing additional operations for a cluster (neighborhood). A typical example scenario is the monitoring of buildings. Rooms contain a number of nodes that sample and preprocess data. Afterwards, one dedicated node – the *leader* node – in the room further processes the data, e.g., to detect outlier regions [6]. The costs for this approach are estimated as follows:

$$\begin{aligned}
c_{neigh} \;=\; & r_s \cdot \Bigg[m \cdot (c_{wake} + c_{sample} + \sum_{j=1}^{p} \sigma_{loc}^{j-1} \cdot c_{loc}^{cpu}(j)) \\
& + (m - m_l) \cdot (2 \cdot c_{msg} + c_{wake}) \cdot \prod_{j=1}^{p} \sigma_{loc}^{j} \\
& + m_l \cdot \sum_{k=1}^{q} \sigma_{neigh}^{k-1} \cdot c_{neigh}^{cpu}(k) \\
& + \sum_{k=1}^{q} \sigma_{neigh}^{k} \cdot m_l^k \cdot f^k \cdot (2 \cdot c_{msg} + c_{wake}) \cdot L^k \\
& + m_l \cdot (2(h-1)\hat{c}_{msg} + (h-2)c_{wake}) \cdot \prod_{k=1}^{q} \sigma_{neigh}^{k} \Bigg]
\end{aligned}$$

c_{loc}^{cpu} denotes the costs for the operations processed locally and c_{neigh}^{cpu} the costs for operations processing data from a neighborhood. m_l denotes the number of leader nodes. Note that operations performed by the leaders can further reduce the size of messages. $\hat{c}_{msg}$ denotes the costs for sending and receiving such a compressed message. This can be estimated with ease, because size and type of the output of an operator are known. Line 4 of the above equation corresponds to the overall costs for the inter-process communication. c_{msg}^{k} denotes the message costs for operator k. m_l^k denotes the average number of nodes involved in the inter-process communication, f^k is the average fanout of an involved node, and L^k denotes the average number of hops necessary for forwarding the message.

Using the presented cost model, we are able to evaluate all physical query plan variants. The physical optimizer chooses the plan that minimizes the estimated overall network costs, i.e., the overall energy consumption.

IV. Overview of the Demo

At the demonstration site, we plan to show our system in action. As example applications, we consider the problems of distributed outlier region detection and burst detection. We will bring real wireless sensors , which will be distributed in the demonstration room. Sensors are equipped with cameras, temperature, and humidity sensors. Detecting bursts means that we can illustrate bursts with respect to movements in the demonstration room, e.g., the sudden increase of movements correlating with the beginning of the coffee break. Apart from environmental monitoring, we plan to show the following aspects at the demonstration.

Development & Deployment: Interested attendees can generate their own queries using AnduIN's GUI They can choose from a set of standard operators (e.g., filter and aggregation) and complex data mining operators (e.g., frequent pattern mining and burst detection). After AnduIN has analyzed, optimized, and decomposed the query, the in-network part is deployed on the sensor nodes. In addition, the demonstration will show manual network deployment: attendees can manually design network topologies, which can afterwards be used to run queries.

In-Network Processing: After a query has been issued, AnduIN optimizes it so that, in dependence on the query operators, queries are partially or even fully computed within the sensor network. Sensors can detect bursts locally and exchange this information with their neighbors in order to infer geographic regions where the bursts occurred [6]. The detected bursty regions are announced to the gateway (a notebook running AnduIN's core engine) as a data stream that is further processed (Figure 1(b)). In the demonstration, we show how these bursts can be visualized dynamically with a web-based GUI. Moreover, this GUI can also illustrate the benefits of optimizing queries using our cost model.

Alerting & Adaption: The overall estimated costs of a query mainly depend on the selectivity of the distributed operators. In particular, in case the data distribution changes, the selectivity of some operators might change and thus another in-network plan might be more power-efficient than the running plan. However, deploying a new plan will also result in additional costs that have to be considered. AnduIN provides an alerter that monitors node statistics and notifies the user (e.g., using the web GUI) if there is an execution plan (another one than the running plan) that significantly improves power efficiency. In this context, significant means that the power reduction of the new plan compared to the running plan exceeds a predefined threshold. Using this recommendation, the decision to deploy the new query plan is up to the user. To show this feature, we plan to simulate data streams with changing characteristics so that the effects can be observed live at the demonstration site and illustrated by the GUI.

Acknowledgment

This work was in part supported by the BMBF under grant 03WKBD2B and is partly supported by the Science Foundation Ireland under Grant No. 08/SRC/I1407 (Clique: Graph & Network Analysis Cluster).

References

[1] D. Culler, D. Estrin, and M. Srivastava, “Overview of sensor networks,” *IEEE Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[2] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TinyDB: an acquisitional query processing system for sensor networks,” *ACM TDS*, vol. 30, no. 1, pp. 122–173, 2005.

[3] Y. Yao and J. E. Gehrke, “The cougar approach to in-network query processing in sensor networks,” *ACM SIGMOD Record*, vol. 31, no. 2, pp. 9–18, 2002.

[4] O. Cooper, A. Edakkunni, M. Franklin, W. Hong, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, and E. Wu, “HiFi: A unified architecture for high fan-in systems,” in *In VLDB*. Demo, 2004, pp. 1357–1360.

[5] B. Krishnamachari, D. Estrin, and S. B. Wicker, “The impact of data aggregation in wireless sensor networks,” in *ICDCSW '02*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 575–578.

[6] C. Franke, M. Karnstedt, D. Klan, M. Gertz, K.-U. Sattler, and W. Kattanek, “In-Network Detection of Anomaly Regions in Sensor Networks with Obstacles,” in *BTW 2009*, 2009.

[7] J. Schiller, A. Liers, H. Ritter, and R. Winter, “Scatterweb - low power sensor nodes and energy aware routing,” in *HICSS*. IEEE Computer Society, 2005, pp. 1–9.

[8] A. Dunkels, B. Groenvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *LCN '04*, 2004, pp. 455–462.

[9] A. Arasu, S. Babu, and J. Widom, “The CQL Continuous Query Language: Semantic Foundations and Query Execution,” University of Stanford, Tech. Rep., 2003.