Deploying In-Network Data Analysis Techniques in Sensor Networks

George Valkanas, Alexis Kotsifakos, Dimitrios Gunopulos
Dept. of Informatics & Telecommunications
University of Athens, Greece
{gvalk,ak,dg}@di.uoa.gr

Ixent Galpin, Alasdair J G Gray, Alvaro A. A. Fernandes, Norman W. Paton
School of Computer Science,
University of Manchester, United Kingdom
{ixent,a.gray,alvaro,norm}@cs.man.ac.uk

***Abstract*—Sensor Networks have received considerable attention recently, as they provide manifold benefits. Not only are they a means for data acquisition and monitoring of unexplored or inaccessible areas, they are also a low-cost alternative for sensing the environment, which greatly aids to better understand our surroundings. A major motivation in either occasion is to acknowledge endangering situations and take action(s) accordingly. To this end, we would like to enable data mining or analysis techniques on top or, even better, *within* such networks, due to the prohibitive cost of communication in this setting. In this work, we demonstrate running data mining algorithms on a set of sensors, which are of low-processing power. In addition to showcasing the execution of data analysis algorithms on resource-constrained hardware, our demo is intended to show how to take advantage of the properties of each algorithm to make better use of the sensors and their capabilities. We support the execution and monitoring of these algorithms with a graphical user interface (GUI).**

I. Introduction

Sensor Networks (SN) have received considerable attention recently, as the benefits from using them are manifold. Not only are they a means for data acquisition and monitoring of unexplored or inaccessible areas, they are also a low-cost alternative for sensing the environment, which greatly aids to better understand our surroundings or inaccessible areas.

Sensor network techniques have to effectively deal with the constraints inherent in the domain:

- the limited power available to the sensor nodes,
- the cost of wireless communication,
- the processing and storage limitations of the nodes.

To address these problems, exchanged messages must be minimized as much as possible, especially since communication is more resource consuming than processing [16]. However, we should not simply place strong constraints on communication, rather we should adopt and employ *efficient* communication algorithms and protocols to provide the desired functionality efficiently [4], [6] and with acceptable quality. Building upon the communication trees in which SN are organized is a first step towards this direction [5]. Figure 1 shows such a tree, where arcs between nodes form the shortest path from the network sink (node `0`) to every other. Moreover, implemented algorithms are not only required to make efficient memory usage, but should also have a small memory footprint. As such, it would be good to have an a-priori indication of the memory requirements of the algorithm.

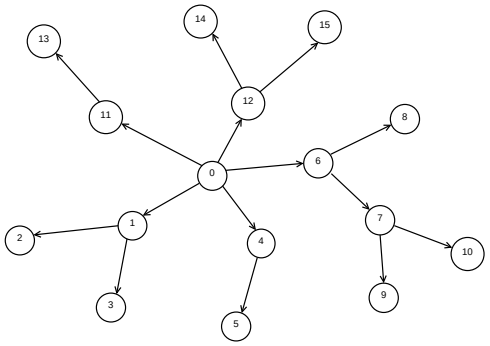

Fig. 1. A Sensor Network example

Part of the current research in the field involves the development of SN query processors (SNQPs) that run on top of SNs to manipulate them more efficiently. Their goal is to abstract the entire network as if the user was requesting data from a centralized data store. The techniques used to achieve this is to simplify deployment of executables, handle node communication, assign processing tasks to nodes (operator placement) etc. To the degree that these goals are fullfilled, a better abstraction is provided. Examples in this direction for data collection include `TinyDB` [12], TAG [11], Cougar [15] and more recent works, e.g. SNEE [7], [8] and *AnduIN* [10].

Users of such engines may pose queries in a declarative manner, using SQL-extensions , enhanced to suit the streaming environment [1], [3]. In addition to the above goals, the engine tries to optimize the query *wrt* processing time, but also takes into account node availability and power consumption. These are, of course, additional constraints imposed by the nature of the sensor network setting.

Nevertheless, a primary motivation for using sensors is to acknowledge endangering situations and take action(s) accordingly. Sensors could be used to either report all sensed data, or inform the user of *interesting* events only. Whatever may be the case, we want to enable data mining or analysis techniques on top or, even better, *within* such networks, for the above reasons. Data analysis algorithms provide improved functionality to the users; they can also prove extremely useful for the network itself, prolonging its life expectancy. For instance, identifying whether a message should be sent or not can greatly assist in

increasing network longevity. An example of such a case is when a sensor *acknowledges* that no significant change in its surrounding has occurred.

A. Our contribution

In this work, we demonstrate running data mining algorithms within a sensor network. In particular, we have implemented a linear regression classifier and the *D3* algorithm [13] on a set of sensors. In addition to showcasing the execution of data analysis algorithms on resource-constrained hardware, our demo is intended to show how to take advantage of the properties of each algorithm to make better use of the sensors and their capabilities.

We focus on algorithms that are *aggregate-based*, i.e. they use aggregation functions on sensed data to make decisions. Such aggregation functions are *sum, count, avg*, etc. The reason is threefold:

- They provide sufficiently useful results, despite their computational simplicity.
- They can be efficiently computed, maintained and approximated.
- To demonstrate that there is not a single communication scheme that best fits all such techniques, despite using the same set of operations.

In our demo, we illustrate the behavior of several communication schemes and how each one affects the algorithm's performance. We feel that our work could be useful to provide insight on best practices in implementing in-network processing of data analysis techniques. We support monitoring these algorithms with a graphical user interface (GUI), where the results of the executions will also be displayed.

II. Data Analysis Algorithms

There are several data analysis algorithms. Therefore, it is practically impossible to showcase them all. On the contrary, our interest is in demonstrating how different techniques best suit different algorithms. We have, thus, implemented a *Linear regression* classifier and the *D3* outlier detection algorithm, which require different manipulation to behave efficiently.

A. Linear regression

Linear regression (LR) is one of the most well-known classification algorithms. The algorithm assumes that there is a linear correlation between the free variables with the dependent one. Therefore, the goal of such a classifier is to find the correlation parameters minimizing the error among all observations. An example LR classifier is displayed in Fig. 2, depicting a hypothetical relation between *temperature* and *pressure* data.

We plan to show different communication protocols for this classifier, which heavily impact the network's efficiency and life expectancy. The various approaches we plan to show include:

i) A **centralized** approach, where all data will be sent to the sink node.

ii) A **decentralized** approach, where nodes perform local computations.

iii) An efficient communication protocol between nodes, that greatly minimizes the number of exchanged messages while remaining efficient and robust. The protocol utilizes the communication tree and acts in a decentralized manner. Sibling nodes, i.e. nodes at the same level of the hierarchy tree, communicate so that the parent node saves energy.

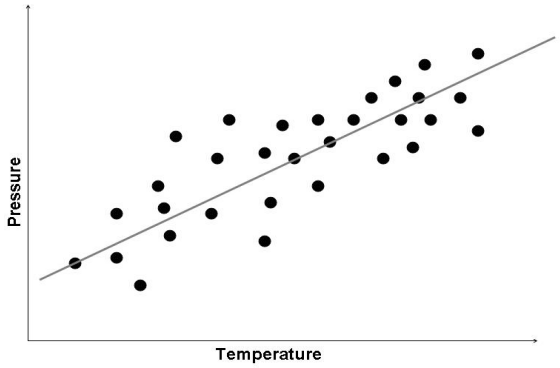

Fig. 2. Linear regression classifier

B. D3

D3 [13] is an outlier-detection algorithm. Such algorithms help identify system failures, or indicate an **event** e.g. a sudden rise in temperature could be indication of fire. Outliers are usually defined as obersvations which are significantly different from the norm. For instance, outliers in a 2D data set are shown in Fig. 3, being the sparsely distributed points outside the circled areas.

D3 captures the density of points dispersed over an area using *kernel density estimators*. For each reading acquired from the environment, the algorithm identifies whether it is an outlier or not. To do so, it computes the number of points that are expected to exist in the point's vicinity. If that number is quite low, then the point is labeled as an outlier and is reported to the parent node.

D3 has some attributes that make it attractive and well-suited for implementation within sensor networks:

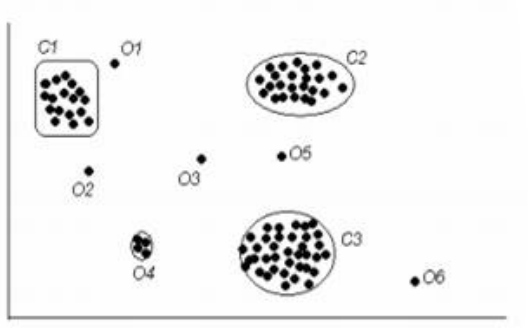

Fig. 3. Example of outliers in 2D

- **It runs in a distributed fashion**: Each node runs the same piece of software and makes decisions based on local observations alone. Communication is, thus, limited to the extent that outlier values need to be propagated to parent nodes.
- **Sensors *could* be organized in a hierarchy**, though this is not mandatory. In fact, several hierarchy levels could be introduced, which can be useful to identify outliers in multiple granularities.
- **Efficient computation**: The algorithm relies on statistical measures which can be easily computed and efficiently maintained, such as `sum, count, standard deviation` etc. It also uses the Epanechnikov kernel, which has an integral with closed form, thus can be computed in constant time. Furthermore, those measures can be approximated even more efficiently using well-known techniques for streaming environments [2].

A sample usage of the D3 algorithm, expressed in a declarative query language such as `SNEEql`, is shown in Fig. 4. In this example, we are interested in finding outliers from a flood stream, for which we know the temperature value of incoming tuples. In short, data analysis models are expressed in `SNEEql`, and manipulated by SNEE internally, as if they were ordinary extents. Having created a D3 outlier detection model, we can check whether each current tuple is an outlier by specifying an explicit threshold.

```
SELECT RSTREAM f
FROM flood[NOW] f, d3 od
WHERE f.temperature = od.temperature AND
od.probability < 0.15;
```

Fig. 4. Example query of D3 outlier detection.

In Fig. 5, we show the basic form of the algorithm to compute D3. The `parentProcess` function is shown for completeness; as we do not assume any type of hierarchy in our demo, we will not be using it.

III. Hardware

The demonstration may use either a set of Crossbow Mica2 MPR410 sensors or a set of SunSPOTs. The former have a 433 MHz processor and 128KB of Program Flash Memory and 4KB EEPROM. Communications are handled by a CC1000 FSK modulated radio. As all Mica2 models, MPR410 are equipped with an Atmega128L microcontroller. Mica2 nodes provide a variety of sensoring devices, such as light, temperature, acceleration etc. They use 1.5V AA batteries for their power supply.

Nodes are programmed using `TinyOS` 2.x version. The programs are written in nesC [9] and executables are precompiled to save time. To better handle the motes and understand exchanged messages, only one of the two programs will be running at any given time. This is consistent with the scenario where nodes are deployed to do a particular task and are then collected and reprogrammed for another task.

Algorithm D3 (Distributed Deviation Detection)
Let W^w and W^b be the sliding windows of leaf and parent nodes;
Let R^w and R^b be the samples on W^w and W^b;
Let σ^w and σ^b be the standard deviations on W^w and W^b;
Let f be the fraction of the sample propagated from a child to its parent;

1. procedure **D3()**
2. assign one leaf node to each one of the input streams;
3. configure all parent nodes in a hierarchy on top of leaf nodes;
4. initiate ParentProcess() for each parent node;
5. initiate LeafProcess() for each leaf node;
6. return;

7. procedure **LeafProcess()**
8. when a new value S(i) arrives
9. update R^w, σ^w;
10. if (S(i) included in R^w)
11. send S(i) to parent with probability f;
12. if (IsOutlier(R^w, σ^w, S(i)))
13. report S(i) as an outlier;
14. send S(i) to parent;
15. return;

16. procedure **ParentProcess()**
17. when a new message from a child node arrives
18. if (message is new outlier **P**)
19. if (IsOutlier(R^b, σ^b, S(i))
20. report **P** as an outlier;
21. send P to parent;
22. if (message is new value from child l)
23. update R^b and σ^b
24. if (the new value is included in R^b)
25. send new value to parent with probability f;
26. return;

27. procedure **IsOutlier**(sample R, stddev σ, point **P**)
28. use R and σ to estimate N(**P**, r);
29. if (N(**P**, r) $<$ t)
30. mark P as an outlier;
31. return;

Fig. 5. Outline of D3 algorithm

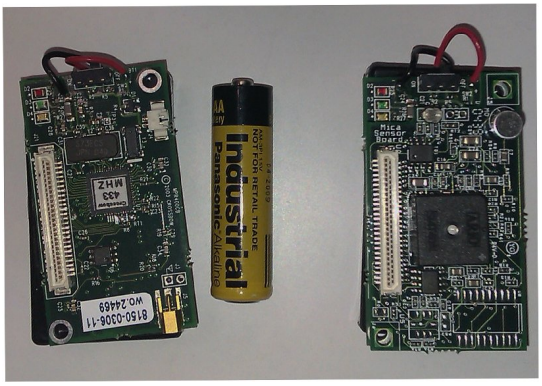

Fig. 6. Two mica2 sensors, part of the demonstration hardware

Alternatively, we can use SunSPOTs, which are equipped with thermometer, photometer (light sensor) and accelerometer sensors. These devices run on a 400MHz processor (therefore, similar to the Mica2) and communicate via a CC2420 radio, implementing the 802.15.4 protocol. They are charged from a 3.7V Li-Ion battery and are programmed using J2ME, conforming to CLDC 1.1.

The graphical user interface, along with the monitoring application, runs on an ordinary laptop where the basestation (i.e. the sink node) will also be connected.

IV. Demonstration

A. Basic Configuration

The demonstration includes running the above algorithms with various communication configurations on a set of sensors and providing their results on the GUI. We deploy our sensors in the demo room after we have uploaded the `TinyOS` image with the algorithm we wish to run (either LR or D3). Sensors begin communicating and exchanging appropriate messages. The sensors will be sensing their surroundings periodically and take action accordingly: for the LR scenario the classifier will be built and demonstrated to the user, whereas for the *D3* case, the user will be informed of detected outliers. The user will be able to see how messages are exchanged between the nodes and how the overall network behaves. An additional part of the demo will be to display part of SNEE's functionality, whereby received tuples from a networked source are output to the user.

B. Linear Regression demo

For the linear regression case, we are interested in demonstating how the network performs using the various communication schemes, which we already briefly described. Node communication will be intercepted by the basestation, which will be sniffing the air medium for exchanged messages. This is to show how each technique behaves and how custom-made communication protocols are more suited to particular algorithms. Communication between nodes will be presented to the user through the GUI, also displaying measures such as number of messages exchanged and message payload.

Evaluating the communication protocols can be both on-line, as the algorithm executes and the user sees the exchanged messages, but also off-line, through stored communicationg logs. This allows for an overall comparative evaluation of the protocols' performance.

C. D3 demo

For the D3 case, as we want to demonstrate the algorithm itself, each node will be sending its computed kernel function, which will be graphically displayed to the user. Every once in a while, a random attribute value will replace an actual tuple reading. This is to simulate outliers, i.e. values which are outside of the norm of the actual readings. Random values are injected in unpredefined intervals, to show that we do not know when to expect that an outlier will be observed. If the reading is marked as an outlier, the input tuple is also propagated to the basestation and the readings (the sensed values) are written to the output. The node that actually sent the outlier blinks on the screen as well, notifying the user of the location where the outlier observation occurred. Another example is to light a lighter near the sensor, which would instantly increase the temperature and mark the input variable as an outlier.

Acknowledgment

This work has been supported by the SemSorGrid4Env (FP7-223913) European Commission project.

References

[1] A. Arasu, S. Babu, and J. Widom. *The CQL continuous query language: semantic foundations and query execution*. The VLDB Journal 15, 2 (June 2006), pp. 121-142.

[2] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. *Maintaining variance and k-medians over data stream windows*. In Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '03). New York, USA, pp. 234-243.

[3] C.Y. Brenninkmeijer, I. Galpin, A.A.A. Fernandes, and N.W. Paton. 2008. *A Semantics for a Query Language over Sensors, Streams and Relations*. In Proc. of the 25th British national conference on Databases: Sharing Data, Information and Knowledge (BNCOD '08).

[4] G. Chatzimilioudis, H. Hakkoymaz, N. Mamoulis, D. Gunopulos: *Operator Placement for Snapshot Multi-predicate Queries in Wireless Sensor Networks*. Mobile Data Management 2009: pp. 21-30

[5] G. Chatzimilioudis, D. Zeinalipour-Yazti, and D. Gunopulos. 2010. *Minimum-hot-spot query trees for wireless sensor networks*. In Proc. of the 9th ACM Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '10). ACM, New York, NY, USA, pp. 33-40

[6] G. Chatzimilioudis, N. Mamoulis, D. Gunopulos: *A Distributed Technique for Dynamic Operator Placement in Wireless Sensor Networks*. Mobile Data Management 2010: pp 167-176

[7] I. Galpin, C. Y.A. Brenninkmeijer, F. Jabeen, A. A.A. Fernandes, and N.W. Paton. *Comprehensive Optimization of Declarative Sensor Network Queries*. In SSDBM 2009, pp. 339-360.

[8] I. Galpin, C. Y.A. Brenninkmeijer, A. J.G. Gray, F. Jabeen, Alvaro A.A. Fernandes, N. W. Paton. *SNEE: a query processor for wireless sensor networks*. Distributed and Parallel Databases, 2011: 31 85

[9] D. Gay, P. Levis, R.von Behren, M. Welsh, E. Brewer, and D. Culler *The nesC Language: A Holistic Approach to Networked Embedded Systems*, In Proc. of Programming Language Design and Implementation (PLDI) 2003, June 2003

[10] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K.-U. Sattler, *Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIN*. In Distributed and Parallel Databases 29(1-2): 151-183 (2011)

[11] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. *TAG: a Tiny AGgregation service for ad-hoc sensor networks*. SIGOPS Oper. Syst. Rev. 36, SI (December 2002)

[12] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. *TinyDB: an acquisitional query processing system for sensor networks*. ACM Trans. Database Syst. 30, 1 (March 2005), pp. 122-173.

[13] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. 2006. *Online outlier detection in sensor data using non-parametric models*. In Proc. of the 32nd Int. Conf. on Very large data bases (VLDB '06), VLDB Endowment pp. 187-198.

[14] H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, and C. Zaniolo *SMM: a Data Stream Management System for Knowledge Discovery*, In ICDE 2011, Hannover, April 11-16, 2011.

[15] Y. Yao and J. Gehrke, *The Cougar Approach to In-Network Query Processing in Sensor Networks*, SIGMOD Rec. 31, 3 (September 2002), pp. 9-18

[16] G. J. Pottie and W. J. Kaiser. *Wireless integrated network sensors*. Commun. ACM, 43(5):5158, 2000.