Enabling Resource-Awareness for In-network Data Processing in Wireless Sensor Networks

Uwe Röhm University of Sydney, School of Information Technologies, Sydney, NSW 2006, Australia, roehm@it.usyd.edu.au

Mohamed Medhat Gaber* CSIRO ICT Centre, Hobart, TAS, Australia, Mohamed.Gaber@csiro.au

Quincy Tse University of Sydney, School of Information Technologies, Sydney, NSW 2006, Australia, qtse4594@it.usyd.edu.au

Abstract

The next-generation of wireless sensor platforms allows for more advanced in-network data processing. The central challenge remains energy and communication efficiency. This paper presents a resource-awareness framework for wireless sensor networks that allows in-network data processing to adapt to changing resource levels such as battery power or available memory. We have implemented the proposed framework as part of a query processing system for the Sun SPOT sensor network platform. As a case study, we have applied the framework to the query processor's on-line data clustering algorithm, making it resource-aware. In an experimental study, we demonstrate how communication costs can be significantly reduced by de-coupling clustering and data communication. The results also show the effectiveness of the resource-aware clustering algorithm: It can keep a constant memory footprint for only a marginal acceptable error in result accuracy.

1 Introduction

With the advent of smart sensor devices, wireless sensor networks (WSNs) are an emerging application field (Zhao & Guibas 2004, Culler & Hong 2004). Many applications need continuous monitoring of phenomena in the physical world, for example in environmental and habitat monitoring (Szewczyk, Osterweil, Polastre, Hamilton, Mainwaring & Estrin 2004), traffic control (Chen, Chen & Tu 2005), and health monitoring (Chen, Agrawal, Cochinwala & Rosenbluth 2004). While previous generations of sensor devices were very limited in their processing capabilities (Hill, Horton, Kling & Krishnamurthy 2004), the latest hardware developments provide considerable CPU, memory, and multi-tasking capabilities (Sun Microsystems n.d., Crossbow Technology n.d.). This development makes more demanding in-network data processing feasible. But at the same time, because of the battery-powered nature of the sensor devices energy efficiency remains the central optimisation goal.

In this paper, we propose a two-stage approach: Firstly, we strive to minimise data communication by moving data processing algorithms into the sensor networks, and by de-coupling data processing and data communication. For example, instead of sending individual sensor readings back to the base station, we envision data reduction techniques such as a clustering algorithm running on the sensor nodes. Secondly, we adapt those algorithms adaptive to changing computational resource levels such as available memory or the battery level. For this purpose, we have designed a resource-awareness framework for in-network data processing in wireless sensor networks. Our goal is to develop an infrastructure for *adaptive in-network data processing* in wireless sensor networks and to explore the trade-off between processing accuracy and energy efficiency.

Complex in-network data processing is made possible by recent hardware developments in WSNs. For example, the new Sun^{TM} Small Programmable Object Technology (SPOT) sensor network platform, developed at Sun Microsystems Laboratories (Simon, Cifuentes, Cleal, Daniels & White 2006, Sun Microsystems n.d.), has a 32bit ARM Risc processor, an 11 channel 2.4GHz radio, and approximately 100 times more memory than a state-of-the-art platform such as Berkeley Motes (Hill et al. 2004). The platform is programmed in Java^{TM} and has a sensor board for I/O and an 802.15.4 radio for wireless communication. The Sun SPOT system runs "Squawk VM" that is a small Java Micro Edition (ME) virtual machine (VM). The VM executes wireless sensor network applications "on the bare metal", i.e., directly on the CPU without any underlying OS, saving overhead and improving performance. With more memory and a faster CPU, alternative design decision can be made to minimise energy-costly communication by pushing data processing such as clustering into the network.

We have designed and implemented a distributed query engine for wireless sensor networks, called SSDQP, which runs on the new Sun SPOT platform. In this paper, we discuss the integration of our resource-awareness framework into SSDQP, and its application to a resource-aware clustering algorithm, ERA-cluster (Phung, Gaber & Röhm 2007). Our main contributions are as follows:

- We describe the implementation of resource-aware, in-network data clustering in a WSN query processor.
- We introduce asynchronous data clustering by

*This work was done while the author was working in the School of IT at the University of Sydney.

employing the multi-query support of our SSDQP system for de-coupling clustering and communication. We show how this can significantly reduce communication costs as compared to plain data collection queries (in our case by factor 7).

- We conducted an experimental evaluation of our techniques in a small Sun SPOT WSN network. Our results show the effectiveness of the resource-aware clustering algorithm: It can keep a constant memory footprint for only a moderate lost in result accuracy.

The paper is organised as follows: In Section 2, we give an overview of WSN query processing and our SSDQP system. We present our resource-awareness framework in Section 3. In Section 4, we present the application of our generic framework into SSDQP and making its CLUSTER operator resource-aware. Section 5 concludes this paper.

2 Background and Related Work

2.1 WSN Query Processing

Distributed query processing in wireless sensor networks has been an active research area over the last few years. TinyDB (Madden, Franklin, Hellerstein & Hong 2005) and Cougar (Yao & Gehrke 2002) represent the first generation of query processing systems in wireless sensor networks. The objective is to abstract from low-level tasks such as sensing, multi-hop data transmission in an ad-hoc network, and data merging and aggregation. The main challenges is to prolong the network lifetime by minimising communication as senor nodes are only battery-powered.

State-of-the-art WSN query processing systems provide SQL-like query languages that are able to collect, filter, and display data from sensor networks (Madden et al. 2005, Intanagonwiwat, Govindan, Estrin, Heidemann & Silva 2003, Yao & Gehrke 2002). Sensor data is viewed as a single virtual table. Results from every sensor find their way to the user through a routing protocol. WSN query processing consists of three phases: (1) the query preparation phase parses and optimises a query at the user's PC, (2) the broadcasting phase injects the sensing and collecting task into the sensor network, and (3) the data collecting phase makes results flowing up and out of the network to the PC where the results are displayed and eventually further processed.

WSN query processing systems have introduced the concept of query lifetime specifications. The user can specify how long the query should be processed. Pushing computation is used in two forms: partial aggregation and packet merging. In partial aggregation, distributive query operators are used in-network. Intermediate results are then passed to the root to integrate the results. On the other hand, packet merging is used to reduce the communication overhead produced from sending multiple packet headers. Query optimisation is done locally at the central site. Once the query is optimised, its execution plan is disseminated into the network by flooding through the routing tree.

Latest research prototypes such as SwissQM (Müller, Alonso & Kossmann 2007) or SSDQP (Scholz, Gaber, Dawborn, Khoury & Tse 2007) further abstract from the underlying sensor hardware by basing the WSN query processors on virtual machines (VM) that run on the sensor nodes. SwissQM proposes an own VM with sensor-query-processing specific extensions, into which queries are compiled. In contrast, SSDQP is running on the Squawk Java VM of Sun SPOT nodes following a more traditional approach of execution-plan compilation of queries and focusing on multi-query execution and resource-awareness.

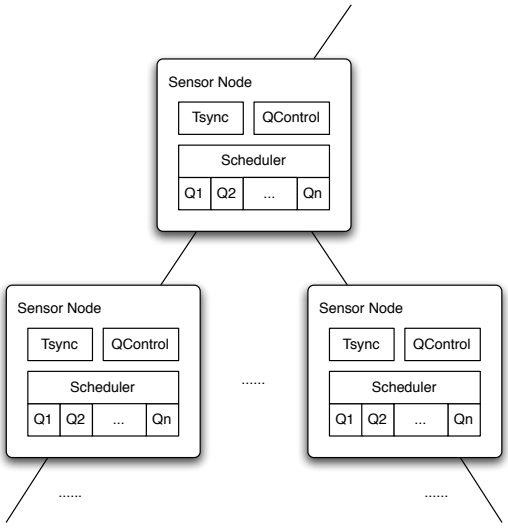

Figure 1: SSDQP Semantic Routing Tree Fragment.

2.2 Sun SPOT Distributed Query Processor

The Sun SPOT Distributed Query Processor (SSDQP) consists of two components (Scholz et al. 2007): (1) the *query engine* that is executed on the Sun SPOTs and (2) the *control system* on the user's PC that is connected to the base station.

The query engine is time triggered. The whole functionality of the query engine is implemented as a set of tasks that have a start time for execution. The task scheduler of the query engine executes a task if the start time of the task has been reached. The task scheduler maintains the active tasks in a time queue. Furthermore, tasks can be periodically executed with a fixed time period and the number of repetitions is parameterisable. Tasks can be added and removed from the time queue of the task scheduler. The start time of a task is "global" in the network such that sensing and communication can be done in a synchronised fashion. The query engine has a specific time synchronisation task that keeps the clocks of the Sun SPOTs in the network in sync, similar to the TPSN algorithm (Ganeriwal, Kumar & Srivastava 2003). In Fig. 1 a fragment of a semantic routing tree is shown. A node in the semantic routing tree is a Sun SPOT device that runs the query engine. The query engine has the task scheduler that executes query tasks Q1 to Qn. The query engines also has a query control that creates new queries and deletes old queries. The time synchronisation module is responsible of establishing a global time in the ad-hoc network. It is important to note that a SQL query is translated into distributed queries for nodes in the semantic routing tree and executed in a distributed fashion on the nodes.

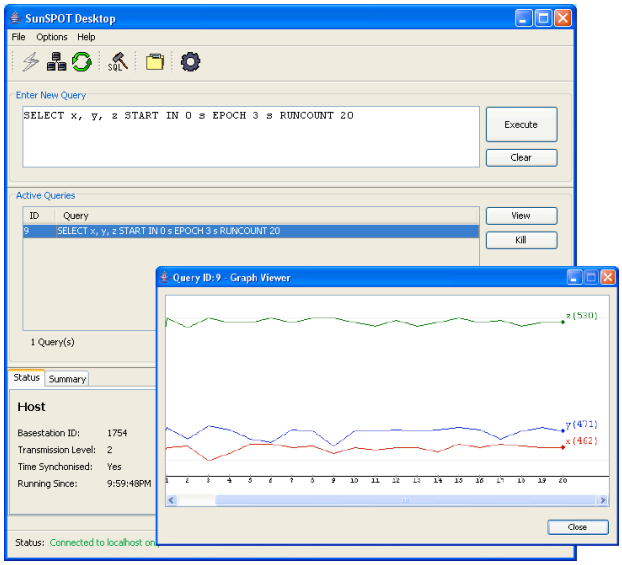

Figure 2: Screenshot of SSDQP.

Query tasks execute a physical query plan; they are composed of relational query operators that operate on relational tables. Since all sensor readings of the SUN SPOTs are integer values, the system does only support integer attributes in the relational tables. The query engine supports all the fundamental query operators including selection, projection, join and aggregation. In addition to these basic functionalities, there are

- the *sense operator* that reads the values of the sensors and creates a result row with the sensor readings,
- the *forward operator* that takes the input relation and forwards it to the parent node in the routing tree,
- the *merge operator* that receives the relations from the children in the routing tree, merges the relations, and returns the merged relation.
- the *cluster operator* that takes the current result values as input and clusters them according to a given distance threshold.

The query operations are represented as expression trees in the query engine. A specific encoding of the expression tree is used to construct expression trees from a message that is sent from the control system to a Sun SPOT node. To minimise the size of messages (and therefore energy), a basic lossless compression method is used.

The control system runs on the user's PC that is connected to the base station. It provides a graphical user interface for query input and result visualisation as shown in Figure 2. The control system

- receives and parses SQL queries, optimises the queries, and translates them into physical query plans, which are deployed in the network,
- collects the data from deployed queries,
- manages deployed queries, (i.e., status of deployed queries, termination of queries, etc.),
- provides the global time to all nodes in the network,
- displays and depicts results of queries.

The system is fully written in Java following a true object-oriented software engineering practice. This gives our system the advantage of the ease of system maintenance and extension.

2.3 In-network Data Clustering

SSDQP provides an online in-network data clustering algorithm (Röhm, Scholz & Gaber 2007) that clusters the query's projection attributes according to a given distance threshold:

```
SELECT temp, light
  INTO buffer  CLUSTER 5
  FROM sensors
 EPOCH 10 secs RUNCOUNT 6*24
```

Figure 3: Example Clustering-Query.

The CLUSTER operator uses a distance threshold to assign a new data record to an existing cluster. When a new measurement is to be clustered, the algorithm searches through the set of all clusters, calculating the Manhattan distance between the mean of each cluster and the new measurement. The use of Manhattan distance allows the algorithm to apply in a multi-dimensional case, making the algorithm much more generic. If the measured values are within the specified threshold radius from a cluster, then the measurement is added to that cluster; if more than one cluster is found, the record is assigned to the nearest cluster. The center is updated according to the weighted average of the new record and the existing cluster. If no such cluster is found, then a new cluster is created with its center assigned to the attribute values of the new record and its weight is set to one.

In (Phung et al. 2007), we have presented an extended resource-aware data stream clustering algorithm, called ERA-Cluster. It extends RA-Cluster (Gaber & Yu 2006) to work on sensor nodes with limited computational resources. Note that ERA-cluster from (Phung et al. 2007) was not integrated into SSDQP or its resource framework so far, while in (Röhm et al. 2007), we focused on the design issues for integrating data stream clustering into SSDQP. This current work builds on this by introducing the generic resource-awareness framework and describing its integration with SSDQP and its clustering capabilities.

ERA-cluster uses an adaptive distance threshold to assign a new data record to an existing cluster. ERA-cluster can adapt to different resource availability using suitable adaptation strategies. It can adapt to available memory using a distance threshold value that encourages or discourages the creation of new clusters. The battery level is addressed by using sampling of data streams in order to allow the sensor node to stop sensing or receiving streaming data to conserve the available energy. It can also adapt to CPU load using randomization of clusters to be examined to allow a new data record to be added to an existing cluster. As we will see in Section 4, the integration into SSDQP will necessitate some modifications

of ERA-cluster's adaptation strategies in particular with regard to the sampling rate.

3 Resource-awareness Framework

3.1 Problem Definition

We consider a system of a hierarchical wireless sensor network which comprises of hundreds of nodes. Each node monitors the environments and does some processing over these collected online data. The objective of this work is to design a resource-awareness framework that enables SSDQP tasks

- to adapt to available resources (energy, memory, CPU);
- to minimise resource consumption, in particular power consumption;
- to keep high levels of accuracy.

The main goal is that given a user-specified running time and a task such as data clustering, the aim is that SSDQP is able to complete the preset runtime and produce as accurate results as possible. In general, the issues are divided into two aspects: predicting dynamic thresholds and wireless sensor networking issues.

3.2 Architecture

To make the framework generic, extensible and easy to implement, we employ a couple of software design patterns into the design. Design patterns are classified in the well-known 'Gang-of-Four' book (Gamma, Helm, Johnson & Vlissides 1993).

Firstly, we use the *publish/subscribe pattern* to decouple the resource monitor and the adaptive algorithms that subscribe to receive resource availability updates. By this way, we can support one or many processing techniques that subscribe to enable resource-awareness. Besides, future extension or modification can be made to the resource monitor without any change to the rest of the system. As can be seen from Figure 4, we have implemented our *ResourceMonitor* extending the *Publisher* class, which keeps a list of references to the subscribers. The algorithms that wish to receive resources updates need to implement the Subscriber interface. The *ResourceMonitor* can then use the method *notifySubscriber* (Object resourceEvent) to dispatch resource events.

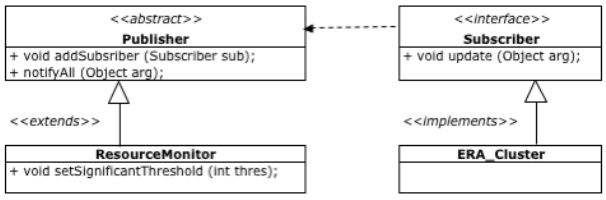

Figure 4: Publish-Subscribe pattern of framework.

3.3 Resource Monitor

The responsibility of the resource monitor is to periodically check the remaining battery, memory, and CPU utilization, and to publish the resource report, which contains status of various resource availabilities. We allow two ways of updating the resource report, periodic and aperiodic updating schemes. The periodic scheme is the traditional way of updating. This means that the resource monitor notifies the subscribed processing techniques over fixed time frames. The drawback of this approach is that if there is stability in the resource level, CPU utilization will be wasted as there is no need to adjust the algorithm settings. Thus, we have implemented an alternative method, which is the aperiodic scheme. The aperiodic scheme only notifies subscribed processing techniques when the accumulative change in resource level is greater than a significant threshold. This threshold is submitted to the resource monitor during the algorithm's subscription. For example, an algorithm can request to be notified only if there is more than 10% or 5% changes in resource level. This approach can greatly reduce processing and communication cost. To further reduce the use of the limited memory size of the node, there is only one resource event object follows the singleton pattern.

4 Case Study: Data Clustering

In the following, we present the use of the proposed resource-aware framework within our WSN query processor to enable resource-aware clustering of sensor data. We discuss, how we implemented the proposed resource-awareness framework into our SSDQP system, and how we made SSDQP's cluster operator resource-aware and asynchronous. In Section 4.3, we present some first results with our prototype system with regard to communication costs and result accuracy.

4.1 Integration into SSDQP

In order to integrate the resource-awareness framework into SSDQP, we had to extend its data model with additional resource attributes, add a new internal resource monitoring task, and then make other operators and the scheduler subscribers of this resource monitor.

Resource Attributes. We extended SSDQP with new self-reflective attributes for the on-board resources of a sensor node; they reflect the percentage of available resources. Those attributes are also exposed on the schema level, so that they can be accessed by any normal SSDQP query. In more detail, we added three new sensing operators to the query engine which retrieve the resource levels — CPUTask (CPU idle time), MemoryTask (free memory) and BatteryTask (remaining battery level). The CPU load is based on the scheduling statistics of SSDQP's own task scheduler, the available memory is retrieved from the Java runtime environment, while the battery level is retrieved through the on-board power management API.

Resource Monitor Task. We implemented the resource monitor as an SSDQP administration task that gets periodically activated by SSDQP's task scheduler. The resource monitor provides a publish-subscriber interface that allows any other part of SSDQP to become resource-aware by subscribing to specific events. Events are defined as threshold condi-

tions on single resources representing critical situations of resource availability.

Conceptually, the resource monitor is a periodic query that senses the current resource levels and updates an internal resource table. With each update, it checks the conditions of all subscriptions; if a condition has become true, the resource monitor notifies the corresponding event subscribers via the provided Java call-back methods.

Resource-aware Clustering Operator. We extended SSDQP's clustering-operator to subscribe to the resource monitor for notifications about the available memory and computational resources. Resource adaptation for the CLUSTER operator is implemented by varying the 'processing granularity' (accuracy of the aggregation operation) and the 'output granularity' (size — hence the precision — of the clusters) as described in (Phung et al. 2007).

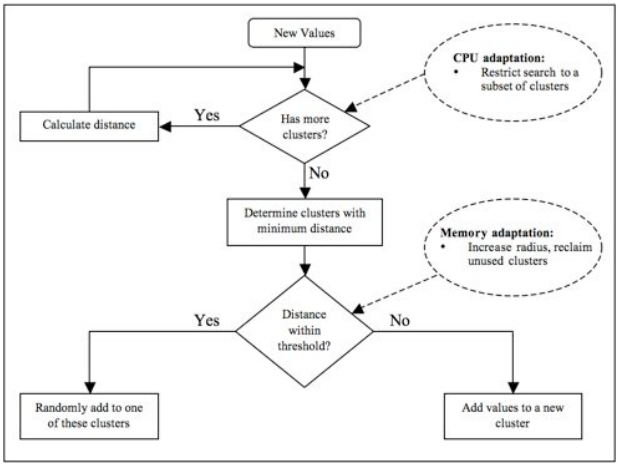

Figure 5: Adaptation to resources for clustering.

In the event of high CPU utilisation, instead of searching through the entire set of clusters, only a random subset of clusters, which is a specified proportion of the entire set, is searched. This reduces the load on the CPU, but may cause extra clusters to be formed unnecessarily.

In the event of low free memory level, the algorithm attempts to reduce its memory footprint by reducing the size of the set of clusters: First, the algorithm iterates through the entire set to locate and remove outlier clusters – those are proportional small clusters that have not been updated for some time. If no such outliers are found, it merges the oldest inactive cluster (the one that is older than a pre-defined threshold and has not been updated for the longest time) with its nearest neighbour cluster, respectively just removes it if the next cluster is too far away. Finally, the clustering threshold is also increased proportional to the memory consumption up to an upper bound, thus decreasing precision of the algorithm. When the resource levels return back to the normal range (below the threshold level), these adaptation are cancelled and the algorithm will return to operating normally.

However, in contrast to (Phung et al. 2007), the 'input granularity' (sampling period) adaptation to low battery level was not implemented in the cluster operator. This is a deliberate decision due to the fact that the cluster, conceptually being a data container, should not be able to control the sampling period of a query. Instead, the input granularity adaptation should be implemented in the scheduler.

Resource-Aware Scheduler. The scheduler is responsible for executing (query and administrative) tasks at the given start times and re-scheduling them according to the task's epoch. If it subscribes to the resource monitor to receive battery level events, it can use them to adapt query epoch dynamically. However, we left the implementation of a 'lifetime' specification similar to TinyDB (Madden et al. 2005) for future work; so far SSDQP only supports fixed length epochs.

4.2 Asynchronous Clustering

The SSDQP engine allows for several tasks being scheduled and executed concurrently. We take advantage of this for de-coupled data-processing by allowing *storage points* to keep query results locally in the node rather than sending it to the sink; similar to the database concept of materialized views, other queries can access those storage points as another local table. Note that both queries do not have to share the same activation frequency. Rather, we can use it to create a specific *clustering task* that will periodically sense and cluster the sensor readings. The second query just retrieves the current clusters from the clustering task whenever the query task awakes. The state is encapsulated within the clustering task.

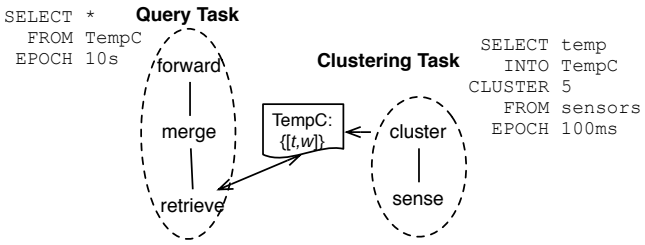

Figure 6: De-coupled Clustering Approach.

The advantage of this approach is that the data processing and communication tasks are de-coupled and can happen in different granularities. For example, the clustering task could wake every 100ms, activate one sensor and cluster the current sensor reading; while the query task would awake only every 10 seconds, retrieve the current cluster values from the clustering task and forward them to the sink.

4.3 Experiments

We conducted some experiments to demonstrate the feasibility and effectiveness of our resource-aware clustering operator in SSDQP. We focus on two primary issues: Firstly, how effective does the clustering algorithm reduce communication and adapt to resource changes? Secondly, how does the resource adaptation affect result accuracy? All experiments are conducted with the resource-aware SSDQP system and two Sun SPOT nodes. We captured the actual light and temperature readings over a period of 3000 seconds and used those values to feed the clustering algorithm in different configurations with the same values.

4.3.1 Reduction of Communication Effort

The goal of de-coupled, in-network clustering is to reduce network communication. We verify this hypothesis by comparing the communication effort needed to retrieve all temperature readings from all nodes versus the on-node clustering and periodically retrieval of them (cf. Figure 7).

```
SELECT light,temp
FROM sensors
EPOCH 1s RUNCOUNT 3000;
```

(a) Full Value Retrieval

```
SELECT light,temp
INTO tclusters CLUSTER 3
FROM sensors
EPOCH 1s RUNCOUNT 3000;

SELECT *
FROM tclusters
EPOCH 120s RUNCOUNT 25;
```

(b) De-coupled Clustering

Figure 7: Query sets for Experiment 1.

In this scenario, we are interested in the typical light and temperature values during the next 50 minutes. Without in-network data processing capabilities, we would need to retrieve all sensor readings from the sensor nodes and then process them outside the WSN. For example, we could issue a selection query as shown in Figure 7(a) with a sampling frequency of 1 Hz and a runtime of 3000 seconds. With SSDQP's in-network data clustering, SSDQP can cluster all sensor readings on the nodes into a storage point (cf. first query in Figure 7(b)), from which we then periodically retrieve the summary of the light/temperature distribution. Only this second asynchronous retrieval query causes network communication (e.g. in Figure 7(b), the clustering results are retrieved periodically in 2 minute intervals).

Note that there is a trade-off between the accuracy of the clustering in terms of cluster radius and clustering frequency, and the communication efficiency. In this paper, we focus on some reasonable values only and leave the complete evaluation of the parameter space to a later paper.

When we run those two setups, we get 3000 messages from each node for each individual sensor reading from the query in Figure 7(a) of average 16 bytes payload (not counting message headers). In contrast, the decoupled clustering will communicate less frequently, but with larger messages because each message contains all current clusters with all their centroids and frequencies. As can be seen from Table 1, this will yield a clear performance benefit for the in-network clustering as long as the average numbers of clusters is about less than half of the retrieval frequency from the clustering output.

	SELECT	CLUSTER
messages per node	3000	25
payload per node	16	24 × #clusters
effort for 2 node WSN	96000 bytes	1200 × #clusters

Table 1: Comparison of communication efforts.

In practice, this ratio is typically even better because of the message compression used in SSDQP: It can yield higher compression ratios on the cluster result messages than on the single-value messages from normal select queries, effectively reducing cluster's payload factors in Table 1 by up-to 60%. When actually running the queries of Figure 7 in SSDQP, we measured per-node communication costs of 81000 bytes for the select query and 11015 bytes for the case with de-coupled clustering – a reduction by more than 85% or factor 7.

4.3.2 Effectiveness of Resource-Awareness

Next, we are interested in how effective the resource-aware cluster operator adapts to changing resource levels. For this purpose, we executed the same clustering query (from Figure 7(b)) over the captured light and temperature readings twice – once with and once without resource-awareness enabled – and measured the actual memory consumption[1], the number of clusters, and the ERA-cluster threshold on the node in each case. Figure 8 shows the results.

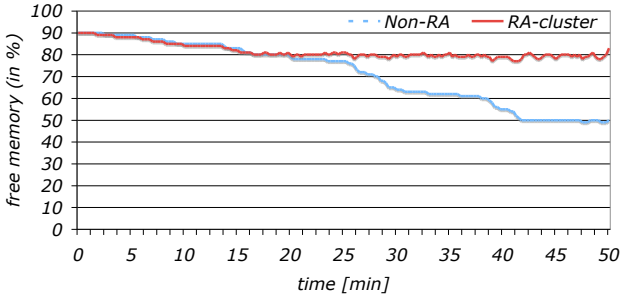

Figure 8: Memory usage vs. resource-awareness.

Without resource-awareness, the cluster operator shows a (more-or-less) linear increasing memory consumption over its runtime. However, if we let the cluster operator adapt to the available memory with a pre-defined memory threshold, its memory usage remains constant. Whenever the memory usage drops below the given threshold (in our experiment: 80%), the cluster operator starts dropping outliers and inactive clusters[2] and increases its distance threshold, effectively keeping its memory usage constant.

Figure 9 illustrates how the clustering operator internally adjusts to the memory threshold. We plotted the number of clusters over time with and without clustering (scale on the left), and also the clustering radius over time in both cases (scale on the right). Without resource awareness, the clustering operator creates more and more clusters of the sensor readings. We see a basically linear increase in number of clusters with a constant clustering radius of 3. In contrast, the resource-aware clustering operator starts dropping outliers and merging inactive clusters when its memory threshold is reached (around 15 minutes into the experiment) so that less memory is occupied by the clustering results. When this not enough, it (temporarily) increases its clustering radius so that less new clusters are created (up-to a maximum deviation from the user-defined clustering radius which however in our experiments was never reached). This can be clearly seen around $t = [25, 30]$ and $t = 40$, when some stronger changes in sensor readings would normally

[1] The memory consumption is measured with regard to available memory after startup of the SSDQP engine; it does not include the captured sensor readings which were also stored on each node during the experiments.

[2] In our experiment, we regarded a cluster as 'inactive' when there haven't been added new values for at least 10 minutes. This value was chosen based on preliminary experiments.

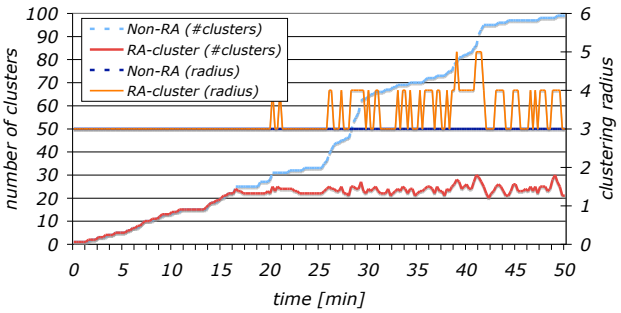

Figure 9: Clustering activity with/without resource awareness.

cause a larger increase in number of distinct clusters (as can be seen by the steeper sections of the blue dotted curve for non-resource-aware clustering). The resource-aware clustering compensates by increasing its clustering threshold (orange line) which discourages the creation of new clusters. When the memory situation improves, the clustering radius is gradually restored back to the user-specified value. Effectively, the resource-aware clustering becomes coarser in low-memory situations, but increases its accuracy again if more memory is available.

4.3.3 Accuracy of RA-Clustering

Finally, we are interested whether the accuracy of the clustering algorithm is acceptable even with its parameters adjusted to resource levels. To do so, we ran the clustering query over the captured sensor readings once without resource adaptation, and once with adaptation to memory usage (memory usage threshold set to 80%) and compare both results.

	without RA	with RA
number of clusters	99	29
avg cluster weight	30.05	60.14
final cluster radius	3	4

Table 2: Comparison of clustering accuracy.

As can be seen in Table 2, with resource-awareness, there are much less resulting clusters (about one third) than with the cluster operator not adapting to the available memory usage. This is the result of dropping outliers and merging inactive clusters during low-memory situations. On the positive side, this reduced the total communication costs by half to only 5130 bytes as compared to the 11015 bytes without resource-awareness.

Table 2 also shows that, as the resource-aware clustering algorithm adapts the number of clusters and the cluster radius, the final result is coarser than without resource-adaptation. Because we are merging old, inactive clusters to their nearest neighbours, the average cluster weight is actually higher than without resource-awareness. The effect is that clusters of old sensor readings are collapsed together, while the more recent readings are clustered with higher accuracy. The final clustering threshold was only slightly higher as specified by the user. As we have seen in Figure 9, the effect of increasing clustering radius during low-memory situations is typically only temporary.

5 Conclusions

The central challenge for WSNs is energy and communication efficiency. In this paper, we proposed a two-stage approach by pushing more data processing capabilities into the network, and by enabling algorithms to adapt to changing resource situations. We presented a resource-awareness (RA) framework for in-network data processing, and we discussed its integration into SSDQP, our distributed query processor prototype for the Sun SPOT WSN platform. In particular, we took advantage of SSDQP's multitasking to de-couple in-network data processing and communication. In our experimental study using the in-network clustering capabilities of SSDQP in a real, small WSN network, we demonstrated how decoupled clustering can significantly reduce communication costs when the retrieval frequency is chosen appropriate – we achieved a reduction by more than 85%. The results also showed that the RA framework allows to effectively adapt to changing memory resources and is able to keep a constant memory consumption. However, this comes with some moderate effects to the clustering accuracy: Newer sensor readings are clustered with higher accuracy while older readings are merged together to coarser clusters.

The behaviour of resource-awareness data processing is controlled by a combination of adaptation techniques, which all can be adjusted by their own set of parameters (as described in Section 4.1). To get the highest accuracy out of the least resource consumption, these parameters need to be balanced well. In this paper, we focused on one particular set of settings to demonstrate the general feasibility of our approach. As future work, we will study the interaction of the different adaptation strategies and their parameter space in more details. We are also working on multi-query optimisation and in-network caching.

5.1 Acknowledgements

This work is supported by the Australian Research Council (ARC) under grant no. DP0664782, and by Sun Microsystems Laboratories.

References

Chen, C.-M., Agrawal, H., Cochinwala, M. & Rosenbluth, D. (2004), Stream query processing for healthcare bio-sensor applications., *in* 'Proceedings of ICDE2004', pp. 791–794.

Chen, L., Chen, Z. & Tu, S. (2005), A realtime dynamic traffic control system based on wireless sensor network, *in* 'Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05)', pp. 258–264.

Crossbow Technology (n.d.), 'iMote2 product page'. Last visited: Feb 2007.
URL: *http://www.xbow.com/Products/ productdetails.aspx?sid=253*

Culler, D. E. & Hong, W. (2004), 'Introduction to special issue on wireless sensor networks', *CACM* **47**(6), 30–33.

Gaber, M. M. & Yu, P. S. (2006), A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering., *in* 'Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), April 23-27, Dijon, France', ACM Press, pp. 649–656.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1993), *Design patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Ganeriwal, S., Kumar, R. & Srivastava, M. B. (2003), Timing-sync protocol for sensor networks., *in* 'Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems (SenSys)', pp. 138–149.

Hill, J., Horton, M., Kling, R. & Krishnamurthy, L. (2004), 'The platforms enabling wireless sensor networks', *Communications of the ACM* **47**(6), 41–46.

Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J. & Silva, F. (2003), 'Directed diffusion for wireless sensor networking', *IEEE/ACM Trans. Netw.* **11**(1), 2–16.

Madden, S. R., Franklin, M. J., Hellerstein, J. M. & Hong, W. (2005), 'TinyDB: an acquisitional query processing system for sensor networks', *ACM TODS* **30**(1), 122–173.

Müller, R., Alonso, G. & Kossmann, D. (2007), SwissQM: Next generation data processing in sensor networks., *in* 'Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR2003), Asilomar, CA, USA, January 5-8', pp. 1–9.

Phung, N. D., Gaber, M. M. & Röhm, U. (2007), Resource-aware online data mining in wireless sensor networks., *in* 'Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007), April 1-5'.

Röhm, U., Scholz, B. & Gaber, M. M. (2007), On the integration of data stream clustering into a query processor for wireless sensor networks, *in* 'Proceedings of the First Int. Workshop on Data Intensive Sensor Networks (DISN2007), in conjunction with MDM'07, May 11'.

Scholz, B., Gaber, M. M., Dawborn, T., Khoury, R. & Tse, E. (2007), Efficient time triggered query processing in wireless sensor networks, *in* 'Proceedings of the 2007 Int. Conference on Embedded Software and Systems (ICESS)', Springer.

Simon, D., Cifuentes, C., Cleal, D., Daniels, J. & White, D. (2006), Java on the bare metal of wireless sensor devices: the Squawk java virtual machine, *in* 'Proc. of the 2nd Int. Conf. on Virtual Execution Environments (VEE)'.

Sun Microsystems (n.d.), 'SunSpotWorld website'. http://www.sunspotworld.com/.

Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A. & Estrin, D. (2004), 'Habitat monitoring with sensor networks', *CACM* **47**(6), 34–40.

Yao, Y. & Gehrke, J. (2002), 'The cougar approach to in-network query processing in sensor networks', *SIGMOD Record* **31**(3), 9–18.

Zhao, F. & Guibas, L. (2004), *Wireless Sensor Networks – An Information Processing Approach*, Morgan Kaufmann.