Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

Resource-aware Online Data Mining in Wireless Sensor Networks

Nhan Duc Phung, Mohamed Medhat Gaber and Uwe Röhm
University of Sydney, School of Information Technologies
Sydney, NSW 2006, Australia
dphu9727@mail.usyd.edu.au, {mgaber, roehm}@it.usyd.edu.au

***Abstract* - Data processing in wireless sensor networks often relies on high-speed data stream input, but at the same time is inherently constrained by limited resource availability. Thus, energy efficiency and good resource management are vital for in-network processing techniques. We propose enabling resource-awareness for in-network processing algorithms by means of a resource monitoring component and designed a corresponding framework. As proof of concept, we implement an online clustering algorithm, which uses the resource monitor to adapt to resource availability, on the Sun SPOT sensor nodes from Sun Microsystem^{TM}. We refer to this adaptive clustering algorithm as Extended Resource-Aware Cluster (ERA-Cluster). Finally, we report on the outcome of several experiments to evaluate the validity of our approach in terms of resource adaptiveness and accuracy of the ERA-Cluster. Results show that ERA-Cluster can effectively adapt to resource availability while maintaining acceptable level of accuracy.**

I. Introduction

Wireless sensor networks have come into prominent because of the advance of Micro-Electro-Mechanical Systems (MEMS) technology and its promising potential. It is a confluence between many fields including signal processing, protocols and networks, embedded system, distributed database management and distributed algorithms and computation. Its wide range of applications include habitat monitoring [1], structural monitoring [2] and many others. However, wireless sensor networks still face many challenges mostly because of its inherently problematic nature. Wireless sensor networks often have adhoc deployment, unattened operation in untethered environment and are susceptible to changes in physical resources, location and many other unpredicted factors. Owing to these factors, traditional techniques are often found to be incapable or poor-performed when applied directly to wireless sensor networks. As a result, many research have been carried on in many fields such as routing, protocols, data gathering, architecture, prolonging network lifetime or data processing, which is the concern of this paper.

Unlike in any other environment, processing techniques in wireless sensor networks often work with continuously high-speed input rate, typically higher than processing power. They are often referred to as *online processing techniques* or *online mining algorithms*. Furthermore, they often have very limited resources such as battery or memory because of the size of the sensor node and its untethered operation. The main challenges include how to maintain a desired throughput rate while maintaining an acceptable level of accuracy. In this paper, we propose a resource adaptation framework, which borrows from the field mining data stream, to enable resource-awareness for data processing algorithms in wireless sensor networks. Our goals are to adapt online processing techniques to resource availability to (1) improve resource consumption patterns and (2) their throughput under scarce-of-resource condition.

This paper is organised as follows. Section 2 reviews the related works in this field. Section 3 briefly discusses the background of the resource-aware framework. Section 4 presents the design and implementation of the resource monitor and the design of our adaptive online clustering algorithm that uses the resource monitor to enable resource-awareness. Section 5 evaluates the validity of this approach in terms of resource-awareness and accuracy. Section 6 concludes this paper.

II. Related Works

In this paper, we discuss an approach to adapt mining data stream techniques to resource availability. Online data stream mining has attracted more and more research attention in recent years. Gaber et al. [3] have done an in-depth survey of mining data stream. There are several existing approaches to adapt data stream techniques to changes in resource constraints.

The first approach is the threshold-based approach for clustering algorithms. BIRCH [4] was the first threshold-based algorithm that uses an adjustable threshold to allow large datasets to fit into memory. Recently, it has been

1-4244-0705-2/07/$20.00 ©2007 IEEE

adopted in new algorithms such as CluStream [5] and LWC [6], which adds more features and/or modifies its structures to be able to adapt to streaming environments. Online stream clustering also has been termed by Aggarwal et al. [5] as *microclustering*.

The second family of algorithms is frequent item set mining which concerns with finding sets of items occurring together frequently. Giannella et al. [7] have proposed a method to extend the traditional FP tree for finding frequent item sets to mine streaming data in a time-sensitive way. Franke et al. [8] have discussed methods to measure the quality of data stream mining algorithms. In [8], they have used these measurements to analyze and enhance a frequent itemset mining technique. The enhanced technique can estimate the quality of output depending on the current resource situation (mainly available memory) as well as allocate resources needed for guaranteeing user-specified quality requirements.

Teng el al. [9] have proposed the RAM-DS algorithm, which uses a wavelet-based approach to control the resource requirements. The algorithm is used to mine temporal patterns and is be used in conjunction with a regression-based stream mining algorithm proposed by the authors.

III. Background

The resource-aware framework is a theoretical generic approach to provide resource-awareness for data stream mining first proposed by Gaber and Yu [10]. It promotes a holistic approach that jointly considers adjusting the settings of the mining algorithm input, output and/or processing endpoints according to resource availability. Gaber and Yu [10] have coined the algorithm input settings as Algorithm Input Granularity *AIG*, the algorithm output setting as Algorithm Output Granularity AOG and the processing settings as Algorithm Processing Granularity *APG*. In general, they are referred to as the Algorithm Granularity Settings or *AGS*.

The AIG represents the process of changing the data rates that feed into the algorithm such as sampling rates or data structure. The AOG represents the process of changing the output size of an algorithm such as the number of clusters formed by a clustering algorithm. The APG represents the process of changing the algorithm parameters to consume less processing power. Changing the randomization factor is an example of an APG setting.

The resource-aware framework consists of three main components:

- A resource monitoring component that periodically monitors the availability of various resources. The implementation of the resource monitoring component is platform dependant and the resources to be monitored can also vary. Common resources are battery charge, remaining memory, CPU utilization, communication buffers or bandwidth.
- The data mining algorithm that processes data in real-time.
- The algorithm granularity setting that is responsible for adjusting the mining algorithm parameters according to resource availability.

Gaber and Yu [10] have also developed a resource-aware clustering algorithm, called RA-Cluster, which uses the resource monitoring component to adapt to resource availability. RA-Cluster adjusts its microcluster creation radius threshold according to remaining memory, sampling rate according to remaining battery and the randomization factor according to CPU utilization. By increasing the radius threshold, RA-Cluster discourages the formation of new microclusters, thus, reduces memory consumption. This is done in combination with the removal of outliers and inactive microclusters to free more memory. The randomization factor affects a strategy called randomized assignment. The randomized assignment means that when determining a new data point, only a random number of existing microclusters are examined instead of all microclusters. The higher the randomization factor is, the less number of microclusters are examined. RA-Cluster uses adaptor threshold bounds to adjust the trade off between the resource adaptation and accuracy loss of the algorithms.

Previously, Gaber and Yu [10] have only implemented and tested the framework and RA-Cluster in Matlab 7. In this research, we have extended and employed this approach to develop and test the framework in an actual sensor node. *To the best of our knowledge, this is the first adaptive data mining algorithm that runs on a sensor node with limited resource availability*. The sensor node is the novel Sun Small Object Programmable Technologies sensor node from Sun Microsystems, a.k.a. Sun SPOT. Sun SPOT uses the Squawk Virtual Machine, which is a high performance JVM written mostly in Java and designed specifically for resource-constrained devices. Applications for the Sun SPOT node are written entirely in Java to be deployed and run from the node. Current version of Sun SPOT can detect light, temperature and 3D acceleration.

IV. Design and Implementation

This section discusses the design and implementation of the resource-aware framework and our clustering algorithm. Our implementation employs the publish-subscribe pattern for the resource-aware framework. Following this pattern, the resource monitor is the publisher of resource events, which contains updates of different resource availability. The data mining algorithms that require receiving resource events should subscribe to the resource monitor to get notified. We have also extended the RA-Cluster algorithm to be able to work on the actual sensor environment. We have termed the new algorithm, Extended Resource-aware Cluster (ERA-Cluster). The following subsections discuss each component in more details.

A. Resource Monitor

The responsibility of the resource monitor is to periodically examine remaining battery, memory and CPU utilization and publishes a 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. It is also the initial scheme used by Gaber and Yu [10] in his original framework. 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, many of the updates will be wasted as there is no need to adjust the algorithm settings. This becomes a more important issue when we consider using the resource monitor for the whole sensor network for distributed processing techniques because of the communication cost. 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, especially with remote monitoring. Using either approach, there is only one resource event object follows the *singleton* pattern. This is to minimize unnecessary usage of limited memory of the sensor node.

Current implementation of the resource monitor allows monitoring of battery charge, free memory and CPU utilization. For the memory, we use the available API provided by Sun SPOT as memory can be consumed quite fast. However, we create two simulations for the battery and the CPU utilization to facilitate the manipulation of resource availability, thus, make it easier to experiment with resource adaptation and accuracy of the algorithm. The battery simulation employs a credit point system, which is used by Younis and Fahmy in [11]. With this approach, each activity of the sensor node is assigned an amount of points and the maximum battery capacity is defined. Activities such as sleep mode, send/receive radio signal, sensing data and computational processing are defined. During operation, the battery charge is decreased gradually according to the sensor activities. With the CPU simulation, we use a simple queuing model that has a fixed queue length and tasks with random generated service time. The CPU utilization is computed as the percentage of total service time of existing tasks in the queue over maximum load. Both simulations have methods to set the resource to a specified level to do experiments.

B. ERA-Cluster

ERA-Cluster is the first resource-aware clustering algorithm written in Java for wireless sensor networks. It is extended from RA-Cluster to work in the Sun SPOT node. Similar to RA-Cluster, it is an online threshold-based clustering algorithm and can be used to reduce data generated by the sensor to be processed offline later. Current implementation of ERA-Cluster can adapt to change in battery level, remaining memory and CPU utilization.

Current Squawk VM of Sun SPOT does not support floating-point number representation to avoid the complexity of floating-point representation, cost of memory and computation. Hence, we need to use Manhattan distance in the clustering algorithm instead of the normally used Euclidean distance. In addition, we have also used a work-around way to represent variables that need floating point accuracy.

ERA-Cluster creates microclusters during its operation. In order to preserve memory, we choose a short representation for microclusters which only consists of the *mean* of the clusters and the *size* which is the number of records added to the clusters. When a new record is added to the microcluster, its mean will be recalculated and its size will be incremented. The record is not kept. In addition, each microcluster has an *inactivePeriod* which is the time since the last record has been added until current time. It is used to detect *inactive microcluster*s which are clusters that have become obsolete. Figure 1 shows the pseudo-code of ERA-Cluster algorithm.

```
Repeat
Get a new record
Assign new record to existing microclusters
        If (randomization_factor == 100%)
                Examine the all existing microclusters.
                Find min_dist and min_cluster.
        Else
                Examine a random number of existing
                microclusters (based on randomization_factor).
                Find min_dist and min_cluster.
        If (min_dist < radius_threshold)
                Insert record to min_cluster.
                Update min_dist microcluster.
                Update inactive period for all microclusters
                 except min_cluster.
                Set create_new_cluster flag = false.
        Else
                Set create_new_cluster flag = true.
                Update inactive period for all microclusters.
If (create_new_cluster flag == true)
        Create new microcluster.
        Update new microcluster.
Until finish
```

Figure 1 ERA-Cluster algorithm

In the algorithm, the randomization factor is the percentage of existing microclusters to be examined during arrival of new record.

C. Algorithm Granularity Settings

The algorithm granularity setting is the way that the algorithm adapts to resource availabilities. By changing the algorithm input, output and/or processing endpoints, we can decrease or increase the consumption of remaining battery, memory or CPU utilization. Figure 2 shows the pseudo-code of ERA-Cluster algorithm granularity settings:

```
If (NoFMem > RTMem)
        radius_threshold = radiusLB
Else
        Reclaim outliers and inactive microclusters.
        Recalculate radius_threshold.
If (NoFCPU < RTCPU)
        randomization_factor = 100%;
Else
        Recalculate randomization_factor
If (NoFBatt > RTBatt)
        sampling_interval = sampling_interval_lower_bound.
Else
        Recalculate new sampling_interval
```

Figure 2 ERA-Cluster algorithm granularity setting

When the remaining memory reduces to a *criticality threshold*, the adaptation will be triggered. The adaptation first starts with the detection and removal of outliers and inactive microclusters. Then, the *radius threshold* of microcluster formation will be recalculated based on the remaining battery level. Memory adaptation is followed by CPU adaptation. We use a *randomized assignment* approach, which we only examine a random subset of the existing microclusters when determining a new record. In our implementation, the randomization factor is the percentage of existing microclusters to be examined and is calculated directly based on the CPU utilization. By reducing the number of microclusters examined, we can reduce the CPU consumption of ERA-Cluster. Certainly, this also reduces the accuracy of the clustering algorithm as the microcluster having the minimum distance to the new record might not be selected - a s*uboptimal* problem. However, even if the nearest microcluster is not selected, a reasonable closed one will be. Thus, as we shall see in the evaluation section, the suboptimal effect tends to be minimal. Finally, the battery adaptation is performed. The main factor that consumes significant energy is the receiving or emitting of data streams or radio signal. Therefore, after the battery reaches the criticality threshold, we decrease the sampling interval of ERA-Cluster according to the remaining battery.

We can see that during the adaptation process, the accuracy of the algorithm is reduced because of the randomized assignment or prolonged sampling interval. Thus, we have mechanisms to maintain and adjust the loss of accuracy. Figure 3 illustrates the adaptation of randomization factor against CPU utilization. In this case, the *criticality threshold* of CPU utilization is set to 40%. If the CPU utilization is less than 40%, the randomization factor is always 100%, which means we examine all existing microclusters in determining a new record. From 40% to 100% CPU utilization, we spread the randomization factor equally between 100% and 50%. Here, 50% called *lower bound* or the *adaptor threshold bound* of the randomization factor.

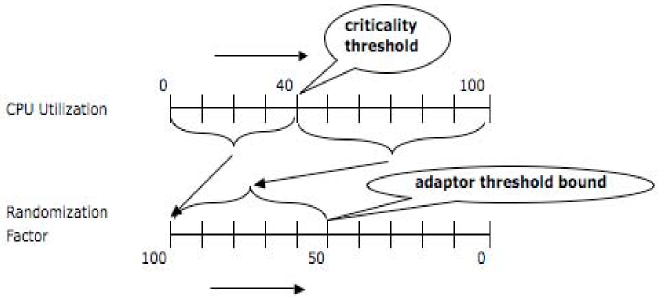

Figure 3 Adaptation of randomization factor against CPU utilization

Given the symbols described in Table 1, we formalize the adaptation strategies of each parameter as follows:

Variable	Meaning
lb	Lower bound of the parameter.
ub	Upper bound of the parameter.
memory	Remaining memory in percentage.
X_crit_threshold	Criticality threshold of resource X in percentage.
battery	Remaining battery in percentage.
cpu	Current CPU utilization in percentage.

TABLE 1 SYMBOLS FOR ADAPTATION FORMULAS

The microcluster creation radius threshold is calculated as:

$$radius = ub - memory \times \frac{ub - lb}{mem\_crit\_threshold} \tag{1}$$

The sampling interval (SI) is calculated as:

$$SI = ub - battery \times \frac{ub - lb}{batt\_crit\_threshold} \tag{2}$$

The randomization factor (RF) is calculated as:

$$RF = \frac{10000 - cpu\_crit\_threshold \times lb - (100 - lb) \times cpu}{100 - cpu\_crit\_threshold} \tag{3}$$

Having presented the design and implementation of the resource-aware framework for the Sun SPOT, the next section describes the evaluation we used to validate the framework.

V. Evaluation

To assess the effectiveness and performance of the ERA-Cluster, we have created an evaluation framework that provides insights into the following aspects of the system:

- Resource-awareness: Is the algorithm able to adapt to changes in resources to improve its performance as well as resource consumption patterns?
- Accuracy: Is the accuracy while adjusting the algorithm granularity settings acceptable?

A. Resource-aware assessment

The goal of the resource-awareness assessment is to prove the benefits of having resource-awareness for the data processing in wireless sensor networks, which in this particular case is the ERA-Cluster algorithm. These benefits include the ability to adapt to changes in resources availability to have better resource management as well as improving the throughput of the algorithm under resource-constrained cases.

The assessment of memory, battery and CPU is done over synthetic data sets. The data rate is set to be faster than the processing rate of the algorithm to reflect the nature of streaming data. The radius threshold of the microcluster creation also set to be small compared to the maximum value of the Manhattan distance formula in order to create many microclusters in a short period. We run the same synthetic data set on the clustering algorithm with resource-awareness enabled and disabled. We sample the interested characteristics of the algorithm with and without resource-awareness at each time frame.

Figures 4 shows the changes of remaining memory level over time. We set the memory criticality level at 80%. As it can be seen from the graph, with RA, the memory reduction follows a much stepper patterns than without RA. With resource-awareness, after the memory reaches the criticality threshold, it fluctuates around the threshold and does not go lower. Figure 5 helps explains how the memory is kept at 80% level. It shows the number of clusters formed over time when run the similar test cases but this time we sample the number of clusters created. Without RA, the number of clusters increases dramatically over a period of 200 seconds. It reaches a peak of about 350 microclusters at the last time frame. With RA the number of clusters fluctuates around 125 microclusters and does not increase any more. This is because after the memory goes beyond the criticality threshold, ERA-Cluster increases the radius threshold to discourage the creation of new microclusters as well as detects and removes outliers and inactive microclusters to free more memory. Figure 6 shows the reactions of radius threshold against memory level.

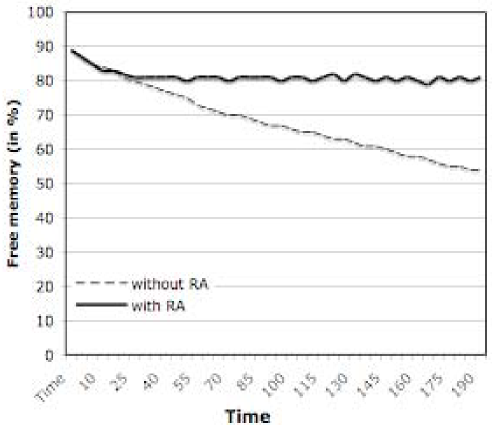

Figure 4 Remaining memory over time

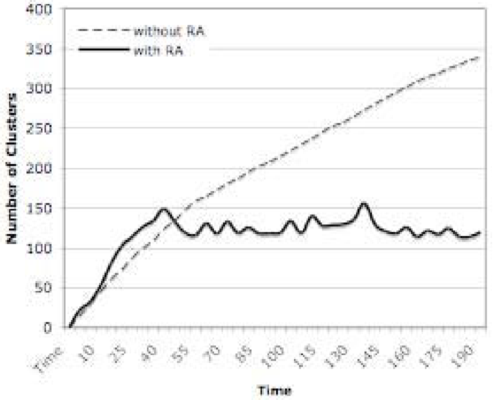

Figure 5 Number of clusters formed over time

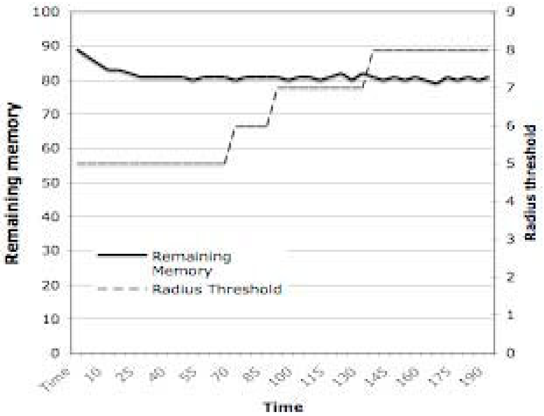

Figure 6 Radius threshold versus remaining memory over time

Figure 7 shows the reaction of sampling interval against battery over time. The criticality threshold of battery is 90%. We set the battery simulator to decrease battery level at periodic time frame. After, the battery reduces below 90%, the sampling interval starts to increase proportionally.

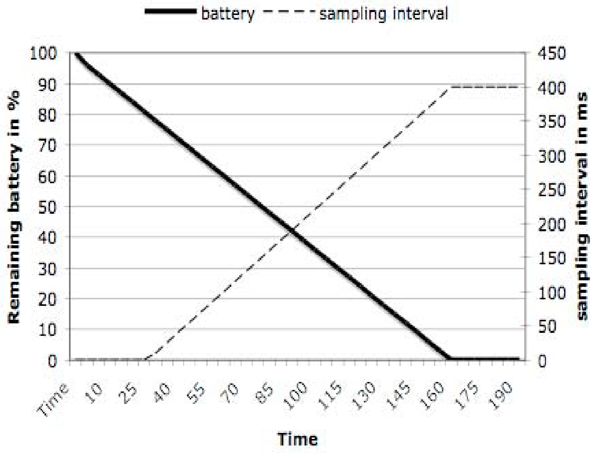

Figure 7 Sampling rate versus remaining battery over time.

Figure 8 hows the adaptation of randomization factor against CPU utilization. Again, the CPU utilization is generated randomly by the CPU simulator. However, we set it to maximum limit at some period to see how the algorithm reacts. The criticality threshold of CPU utilization is set at 40%. The graph is fluctuating because we only sample CPU utilization at fixed time frame not for a period of time. This is because we are only interested in seeing how the randomization factor is calculated based on the CPU utilization. As it can be seen from the graph, when the CPU is below 40%, the CPU utilization is always 100%. This is most clear at time frame number 41 - 43. When the CPU is 100%, the randomization factor reaches 10% and does not go beyond this threshold. This is because 10% is the lower bound threshold of the randomization factor.

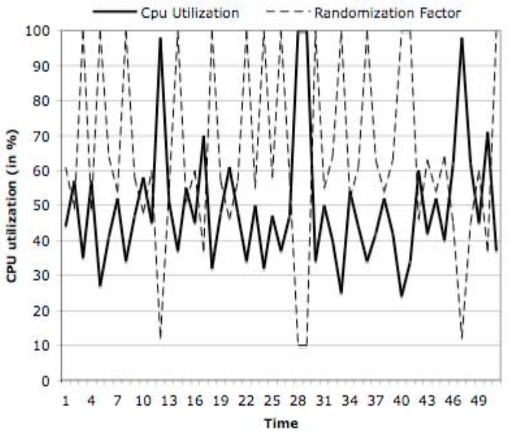

Figure 8 Randomization factor versus CPU utilization

The above results provide an evidence of the adaptation of the algorithm over time corresponding to resource availability. It also shows a significant improvement in the memory consumption patterns with the resource-awareness. However, there are tradeoffs between the adaptation and accuracy of the algorithm. This is investigated in the next subsection.

B. Accuracy assessment

ERA-Cluster has mechanisms to control its accuracy including adjusting the adaptor threshold bounds, the resource criticality thresholds and removal of outliers and inactive clusters. By adjusting these threshold values, users can adjust the loss of accuracy of the algorithm. Firstly, this section starts with a description of the methodology that we use to benchmark the accuracy of ERA-Cluster. Secondly, it presents results of accuracy tests under normal operation scenarios. Finally, it presents the loss of accuracy tests under resource-stress scenarios.

The major challenge with the accuracy assessment is how we are going to benchmark the accuracy of the ERA-Cluster. Clustering is classified as unsupervised learning. Unlike supervised learning, there is no *priori* output, no training data and testing data. Benchmarking the accuracy of unsupervised learning is often much more complicated than supervised learning. We have use another clustering algorithm as a benchmark and compare the accuracy of ERA-Cluster with this algorithm. Among other algorithm, kmeans is chosen because it has been used in resource-constrained environments such as astronomical applications due to its low complexity. Examples include clustering earth science data in a NASA project using kmeans [12] and mission planning on-board Mars rovers using kmeans [13]. In these projects, it has been pointed out that the use of kmeans is due to its low complexity and the scarce of computational resources for such missions. To avoid biased implementation, we have chosen the kmeans algorithm provided by the Weka software package. Weka [14] is a free data mining software package that contains many different data mining algorithms both supervised and unsupervised types.

Firstly, we aim to show that under normal operation, including random resource adaptation, the accuracy of ERA-Cluster is competitive to kmeans in Weka over the same synthetic data set. We use 1-attribute record dataset for this test to simplify the task of representing the results. The attribute is a uniform random integer of the range 0 to 100. We run the algorithm over a dataset of 660 records, to create a number of microclusters, say *n*. We then run kmeans 3 times over the same synthetic data with k = n to create the same number of clusters. We sort all of the results (ERA-Cluster and 3 kmeans according to ascending order of mean value of the microcluster. We then plot the mean value of ERA-Cluster against the average mean value of kmeans. Figures 9 and 10 show the results of this experiment with different dataset sizes.

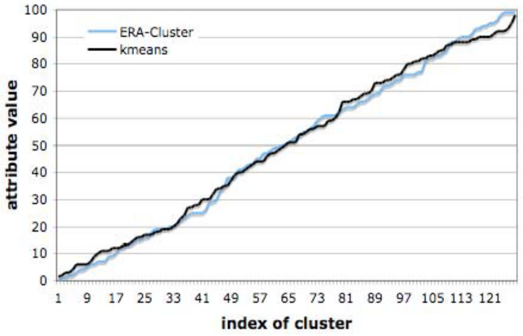

Figure 9 Accuracy of ERA-Cluster vs. kmeans (660 records)

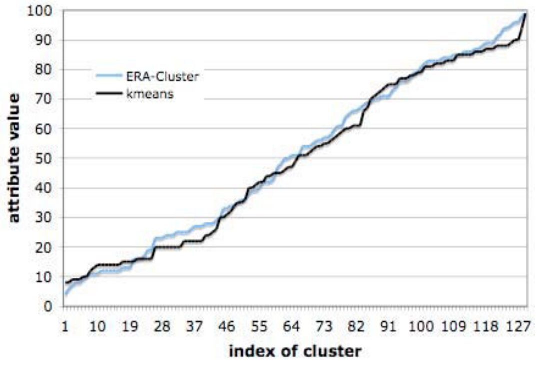

Figure 10 Accuracy of ERA-Cluster vs. kmeans (2000 records)

We also measure the average result deviation of each cluster between kmeans and ERA-Cluster for the two above tests. This can tell us about the accuracy loss of ERA-Cluster. For each cluster, the result deviation is calculated as the sum of the normalized deviation along each dimension (i.e. each attribute) averaged over the number of dimensions, where the normalized deviation along a dimension is obtained by dividing the difference of the cluster mean values (between kmeans and ERA-Cluster) by the spread of the true cluster along that dimension (i.e. the maximum value of the ERA-Cluster distance formula). As we have only one dimension (one-attributed record), the result deviation is just the normalized deviation of this attribute. Then, the average result deviation of one cluster is calculated as the sum of the result deviations of all clusters average over the number of clusters. The results are shown in Figure 11. It shows that the result deviation is low for both tests on the 660-record data set and the over 2000-record data set.

Results of the previous experiments show that under normal circumstances, including resource adaptation during operation, the accuracy of ERA-Cluster is competitive to that of kmeans. Having seen the accuracy of ERA-Cluster under normal operation, we are also interested in knowing how worse the accuracy degrades under resource stress scenarios.

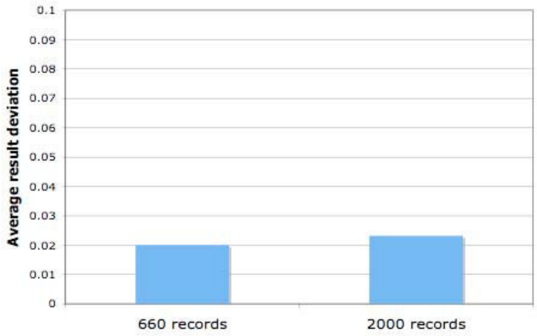

Figure 11 Average results deviations of ERA-Cluster

The goal of these experiments is to show the loss of accuracy under different resource stress scenarios. For example, for a criticality CPU utilization threshold of 40%, how worse the accuracy degrades if the resource-aware clustering algorithm is constantly under the CPU utilization of 50% or 100% compared to the case when CPU utilization is always less than 40%. Similar to the approach used in the previous section, we measure the loss of accuracy by the average result deviation of one cluster.

For this test, we set the adaptor threshold bound (the lower bound) of the randomization factor to be 10% and keep the sampling interval and radius threshold always constant. The size of the synthetic data set is about 1500 records. For the ideal case, we run the ERA-Cluster with less than 40% CPU utilization, which is the criticality threshold we set for CPU utilization. This means that there is no randomized assignment occurs. For the second case, after half of run time, we set the CPU utilization to be constantly at 50%. This means when determining a new data point, only 85% of existing microclusters are examined randomly. For the third case, after half of run time, we set the CPU utilization to be constantly at 100%, which corresponds to randomized assignment at 10%. We repeat these cases three times, each time with a different data set.

The results of these cases from ERA-Cluster and the original data set are all be used as input to the kmeans of Weka with k=10 to create ten clusters. We sort all the clusters according to ascending order of their mean value. After that, we calculate the mean values for the cpu < 40%, cpu = 50% and cpu = 100% by averaging each case over results of three runs. The result deviation of each cluster is the normalized deviation of each cluster's mean. Finally, the average result deviation of one cluster is calculated by the sum of all result deviations averaged over the number of clusters. The results are shown in Figure 12. As we can see from the graph, the average result deviation of a microcluster increases with the increase of CPU utilization. However, overall result deviations are still small.

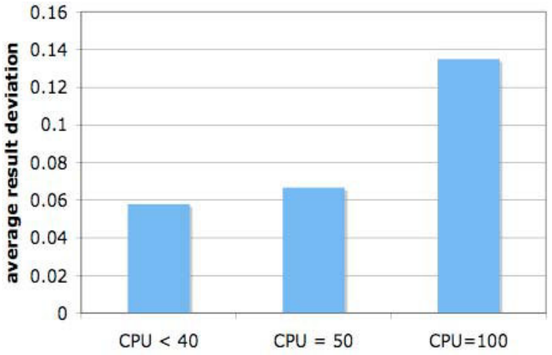

Figure 12 Average result deviation under CPU-stress

VI. Conclusions

The primary goal of our work is to enable resource-awareness for data processing in wireless sensor networks by using a resource monitoring component. The main contributions of this paper include the design of a resource-aware framework for the Sun SPOT sensor node, the development of the resource monitor and the development of our ERA-Cluster algorithm.

The proposed adaptive clustering algorithm was evaluated with regard to accuracy and resource-awareness. The results show that ERA-Cluster can effectively adapt to resource availability and that it can improve resource consumption patterns.

Based on the successful result of this research, possibilities for future work are identified as follows:

- Replacing the battery and/or CPU utilization simulation by the real API and investigate the improvement in consumption patterns of these resources using the resource-aware framework.
- Extending the resource monitor to be able to monitor a remote node or the whole network and the resource-aware framework to be able to support distributed processing techniques.
- Investigating the performance and resource consumption patterns of other processing techniques and several parallel processing techniques on the same sensor nodes.

References

[1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring.," *Proc. of the ACM International Workshop on Wireless Sensor Networks and Applications, WSNA 2002*. Atlanta, Sep. 2002.

[2] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A Wireless Sensor Network for Structural Monitoring," in *ACM Conference on Embedded Networked Sensor Systems(Sensys04)*, November 2004.

[3] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review.," *SIGMOD Rec.*, vol. 34, pp. 18-26, 2005.

[4] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases.," *SIGMOD Rec.*, vol. 25 (2), June 2006.

[5] C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A Framework for Clustering Evolving Data Streams," in *Proc. of VLDB 2004*, 2003.

[6] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "A Cost-Efficient Model for Ubiquitous Data Stream Mining," *Proc. of IPMU 2004.*, 2004.

[7] C. Giannella, J. Han, E. Robertson, and C. Liu, "Mining Frequent Itemsets over Arbitrary Time Intervals in Data Streams," Technical report, Indiana University, 2003.

[8] C. Franke, M. Karnstedt, and K.-U. Sattler, "Mining Data Streams under Dynamicly Changing Resource Constraints," in *KDML 2006: Knowledge Discovery, Data Mining, and Machine Learning*. Hildesheim, Germany, 2006.

[9] W.-G. Teng, M.-S. Chen, and P. S. Yu, "Resource-aware mining with variable granularities in data streams," *In SIAM SDM 2004*, 2004.

[10] M. M. Gaber and P. S. Yu, "A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering," in *Proceedings of the ACM SAC '06*. Dijon, France: ACM Press, 2006.

[11] O. Younis and S. Fahmy, "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks.," *IEEE Transactions on Mobile Computing*, vol. 3 (4), pp. 366-379, Oct-Dec 2004.

[12] M. Steinbach, P. Tan, V. Kumar, S. Klooster, C. Potter, and A. Torregrosa, "Clustering Earth Science Data: Goals, Issues and Results," in *KDD 2001 Workshop on Mining Scientific Dataset*.

[13] T. Estlin, R. Castano, B. Anderson, D. Gaines, F. Fisher, and M. Judd, "Learning and Planning for Mars Rover Science," in *IJCAI 2003, Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating*. Acapulco, Mexico, August 2003.

[14] I. H. Witten and E. Frank, *Data mining: practical machine learning tools and techniques with Java implementations*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2000.