Resource-aware Distributed Online Data Mining for Wireless Sensor Networks

Nhan Duc Phung[1], Mohamed Medhat Gaber[2] and Uwe Roehm[1]

[1] University of Sydney, School of Information Technologies
SIT Building J12, NSW 2006, Australia
{dphu9727,roehm}@it.usyd.edu.au
[2] CSIRO ICT Centre, Tasmania, Hobart, TAS 7001
Mohamed.Gaber@csiro.au

Abstract. Online data mining in wireless sensor networks is concerned with the problem of extracting knowledge from a large continuous amount of data streams with an in-network processing mode. Unlike other types of networks, the limited computational resources require the mining algorithms to be highly efficient and compact. We propose a distributed resource-aware online data mining framework for wireless sensor networks which can be used to enable existing mining techniques to be applied to sensor network environments. We have applied the framework to develop and implement a distributed resource adaptive online clustering algorithm on the novel Sun Microsystem™ Small Programmable Object Technology Sun SPOT platform. We have evaluated the performance of the algorithm on the actual sensor nodes. Experimental results show that the clustering algorithm can improve significantly in resource utilization while maintaining acceptable accuracy level.

Keywords: distributed clustering, resource adaptivity, data mining, sensor networks

1 Introduction

Online data mining in wireless sensor networks has attracted research attention in recent years. This is because deployments of large-scaled distributed sensor networks are now possible owning to hardware advances and increasing software support. Online data mining, also called *data stream mining* is concerned with extracting patterns from continuous data streams such as those generated by sensor networks. Because of the massive amount of data and the speed of which the data are generated, many data mining applications in sensor networks require in-network processing such as aggregation to reduce sample size and the communication overhead.

Adding up to the challenges are the extremely limited size of memory, available energy and processing power of the sensor nodes. These factors imply that traditional data mining techniques in order to be used in sensor network need to be highly energy efficient and compact. One of the methods is to improve the resource utilization via enabling *resource-awareness* for the mining techniques. With resource-awareness, the mining algorithm can automatically adjust its configuration in real time according to resource

availability levels. This can prolong network lifetime and it can also improve the mining techniques performance under resource-scare scenarios. Whilst there is research work on resource adaptivity in wireless sensor network, none of them provide a generic mechanism to enable resource-awareness for data mining in sensor networks.

In this paper, we propose a distributed resource-aware online data mining framework for wireless sensor network which can be applied for many mining techniques that requires constantly monitoring, aggregation of data and in-network processing. We apply the framework to implement a distributed resource-aware online clustering algorithm, which we termed DERA-Cluster, on an actual sensor platform – the Sun SPOT. We have implemented and evaluated the algorithm on the actual sensor networks. Experimental results show that our clustering algorithm with resource-awareness greatly improves resource utilization while being able to maintain acceptable accuracy.

This paper is organised as follows. Section 2 reviews the related work in this field and Section 3 briefly discusses the background of the resource-aware framework. In Section 4, we introduce our DERA cluster algorithm, and we discuss implementation issues in Section 5. Section 6 evaluates the validity of this approach in terms of resource-awareness and accuracy. Section 7 concludes this paper.

2 Related Work

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 streams. 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 [1] was the first threshold-based algorithm that uses an adjustable threshold to allow large datasets to fit into memory. Recently, it has been adopted in new algorithms such as CluStream [2] and LWC [3], 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. [2] as *microclustering*.

The second family of algorithms is frequent itemset mining which concerns with finding sets of items occurring together frequently. Giannella et al. [4] 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. [5] have discussed methods to measure the quality of data stream mining algorithms. In [5], 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. [6] 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.

An overview of recent research and application on distributed data mining can be found in [7]. Bandyopadhyay et al. described a K-Means-like technique for clustering homogeneously distributed data streams in a peer-to-peer environments like sensor networks [8].

3 Background

The resource-aware framework is a theoretical generic approach to provide resource-awareness for data stream mining first proposed by Gaber and Yu [9]. 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 [9] 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 while changing the randomization factor is an example of an APG setting. The resource-aware framework consists of three main components:

1. 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 load, communication buffers or bandwidth.
2. The data mining algorithm processes data in real-time.
3. The algorithm granularity settings that is responsible for adjusting the mining algorithm parameters according to resource availability.

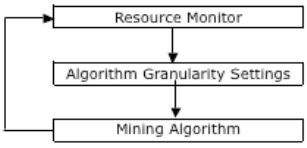


Fig. 1. The resource-aware framework by Gaber and Yu [9].

Gaber and Yu [9] have implemented a resource-aware clustering algorithm in Matlab, 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.

In our previous work, we have developed a generic resource-aware framework for wireless sensor networks. The framework has been used to implement a resource-aware clustering algorithm, which we termed Extended Resource-aware Cluster or ERA-Cluster. We have implemented and tested the framework in an actual sensor node. The sensor platform 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 is written entirely in Java and can be deployed and run from the node. Details about the non-distributed ERA-Cluster algorithm can be found in [10].

This paper presents the complete distributed resource-aware framework for wireless sensor networks. By distributed, we mean a hierarchical structure, in which each node can do some data processing such as clustering but the results will be integrated at a parent node which in turn sends to other higher level parents or to base station to answer some queries or for further offline data mining. Firstly, we will discuss the issues coming up within the design of the framework, our solution as well as other alternatives. Secondly, we describe our specific implementation on the Sun SPOT platform.

4 Distributed Resource-aware Online Data Clustering

In the following, we describe our approach to distributed resource-aware data clustering in sensor networks, termed DERA-Cluster. We start by defining the problem we want to solve. After that, we describe our clustering algorithm and how it can adapt to computational resource availabilities. In particular, we focus on how to react to low battery resources in a distributed way in order to meet the lifetime goal with maximal result accuracy. We describe our solutions with respect to the feasibility of the development platform as well other possible alternatives.

4.1 Problem definition

We consider a system of a hierarchical or peer-to-peer wireless sensor network that comprises hundreds of nodes. Each node monitors the environments and does clustering over these collected online data. We propose DERA-Cluster, a distributed resource-aware online clustering algorithm, which can adapt to computational resource availabilities. In a distributed computational model, the main goal is that given a user-specified running time

and a task such as data clustering, the aim is that our network is able to complete the preset runtime and produce as accurate results as possible. The other objective is to minimize the accuracy loss in case few nodes die or stop working due to low availability of resources such as running out of battery, full of memory, and/or full of CPU utilization. Our approach is to migrate current results from a near-dead node to another 'best' neighbour. This gives rise to three main questions:

- Which neighbour to migrate to?
- When to migrate?
- How to migrate (and merge these clustered data)?

In general, the issues are divided into three aspects: migration of data, predicting dynamic thresholds and wireless sensor networking issues.

4.2 The core algorithm

The core of DERA-Cluster is based on our previous work in [10] where we developed a resource-adaptive online clustering algorithm called Extended Resource-aware Cluster or ERA-Cluster. Via ERA-Cluster, we wanted to show a typical AGS scheme – the way the algorithm adjusting to resource availability. *To the best of our knowledge, ERA-Cluster is the first resource-aware algorithm that runs on a sensor node with limited resource availability*. ERA-Cluster is an online threshold-based clustering algorithm, which can be used to reduce or summarize streaming data into microclusters. We allow mechanisms to control the accuracy of the algorithms.

In this paper, we extend this work to DERA-Cluster, a fully distributed clustering approach. The core algorithm runs locally on each node where it subscribes to the resource monitor to receive resource events, and adapts to changes in battery level, remaining memory and CPU utilization similar to what we introduced in [10]. Beside this local adaptation, we introduce a new distributed strategy: If the battery level drops below a minimal threshold, a node will migrate its microclusters to a suitable neighbour, where they will be merge with the existing microclusters. In the following, we present the details of our approaches to migrate microclusters – *when*, *where*, and *how* – in DERA-Cluster.

4.3 Using linear extrapolation model to estimate dynamic migration threshold

A node has to dynamically estimate if the node is able to complete the runtime at each timeframe. If not possible then it will migrate its current result to the best neighbour. In order to answer the question when to migrate, we use a simple linear regression model to dynamically and iteratively estimate three thresholds in descending order:

1. the *adaptive threshold*,
2. the *best-neighbour-finding threshold*, and
3. the *migrating threshold*.

The *adaptive threshold* is the one that triggers the resource adaptation process. This is described in details in [10]. However, there are cases which resource adaptation cannot improve much the situation. In that case, we choose to migrate its existing results before it dies. The second threshold is called *best-neighbour-finding threshold*. As its name

suggests, when resources drop below this threshold, the node starts to broadcast request to its neighbours. Information in the replies is the remaining resource levels. The link quality can also be estimated from the replies. From this information, a 'best' neighbour will be marked. Finally, when resources reach the *migrating threshold*, which typically just enough energy for it to send its data before it dies, this node will migrate its data to the selected neighbour. One simple approach to know when to adapt to resource availability or to migrate data is to use some *predefined* threshold. For instance, when the battery level reaches 70%, a node can start to adapt to resource availabilities; when battery reaches 30%, it starts to query for best neighbour and when the battery reaches 10%, it migrates results. This approach is simplest and also easiest to implement. Under some cases such as where all nodes do the same operation and the resources are consumed steadily, perhaps this is the best approach. However, we are also interested in developed a more dynamic scheme whereas user does not need to specify these predefined threshold but the node dynamically estimate these thresholds. We choose to use a simple linear extrapolation model to estimate whether a node is able to complete its specified runtime. It is the only suitable regression model because non-linear regression model are complicated to implement and cost a significant amount of energy and computational resources.

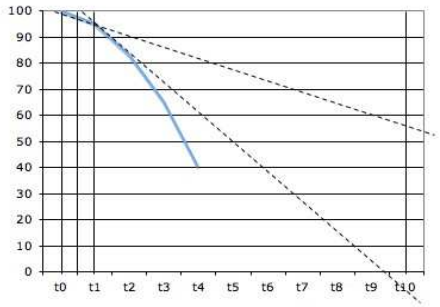


Fig. 2. Linear extrapolation model.

Fig. 2 shows our linear extrapolation model. Suppose, a node is programmed to run for 10 minutes, which is marked t10. At each time frame t0, t1, t2 the node checks its availability resources but it keeps only the most recent time frame resource record. The y-axis shows the battery level. Soon after started running, at time t0, the node measures its battery level. At time t1, it re-measures the battery and calculates the line equation through t0 and t1 which is used to check if it can complete 10 minutes runtime. In this case, it does so the node continues run normally. At time t2, the node re-measures the battery level. In this case, battery drops significantly and it detects that it cannot reach 10 minutes runtime. Thus, it starts the resource adaptation process. Later, if the node detects that resource adaptation cannot improve the situation, it starts to query for best neighbour. Finally, it will migrate result when battery level reaches the minimum amount necessary for sending data. Currently, this minimum battery level is pre-defined for the sake of

simplicity. Given a certain sensor platform, we can measure or estimate level by experimenting with the node.

4.4 Selecting 'best' neighbour

To find the 'best' neighbour to migrate data to, a node can broadcast a query to all of its neighbors asking for their current computational resources level. From the replies, it can also detect the link quality. Most platforms allow this feature. In our approach, we use such broadcasting and then built a two dimensional matrix to represent this information and we have a weighting scheme and a formula to determine the 'best' neighbour. For example, remaining battery is given the best priority; second comes link quality and remaining memory; last is CPU utilization.

4.5 Migrating data.

Data is sent in byte array format, not string, to minimal the amount of transferred data. As current SPOT' API does not support the *serializable* mechanism directly, we need to create our own mechanism to marshal/unmarshal objects to byte array. Basically, we define an *IPersistence* interface which contains the persist() and resurrect() methods. The Cluster class, which represents microcluster, extends this interface and implements these two methods defining how its attributes are actually persisted and revived. We also create a class called VectorHelper to serialize the Vector class, which contains collection of microclusters. Upon migration, ERA-Cluster persists all of its current microclusters to byte array then delegate to the Communicator class to send this data. Communicator is responsible for fragmenting this data into multiple datagram, adding appropriate header and flags before sending off the datagrams. At destination, the data is received and assembled by the Server class.

4.6 Merging data at destination.

At the destination node, the new arrival clustered data will be merged with the existing data on the node. The merging method depends on the mining algorithms. For our DERA-Cluster, the algorithm to merge the data is as follows:

```
FOR EACH new microcluster
   find the minimum distance min_dist to all existing microclusters
   IF min_dist > cluster_creation_ threshold
         keep this new microcluster
   ELSE
         merge this new microcluster with the microcluster with min_dist.
```

Fig. 3. DERA-Cluster's merging algorithm.

The merging formula can be a simple calculation of average mark with weights. Given two microclusters ($a_1, a_2 ... a_N$) with K number of records and ($b_1, b_2 \ldots b_N$) with L number of records. Each new attribute of the new microcluster is given by (1):

$$attribute_i = \frac{a_i \times K + b_i \times L}{K + L} \tag{1}$$

The number of records of the new cluster is $K + L$.

4.7 Networking issues

Firstly, most of current sensor node platform supports two basic type of communication: the packet-based or datagram-based communication and the streaming communication. The communication is, however, also one of the most significant factors that consume energy of the node. Thus, in general case, we choose datagram-based communication because it cost much less energy compared to streaming communication. The unreliability factor can be taken into account during implementation.

Secondly, when a node is querying for 'best' neighbour, broadcasting will be used as sensor networks may not necessarily have a robust routing system implemented. When broadcasting, we should assume we only get replies from 'direct' neighbours or the neighbours within the range of the sensor node. One issue that should also be noticed here is that the network follows a hierarchical structure, thus, one might consider the case that a child node always migrate to the parent node whenever it runs out of resources. That is a much simpler model and easier to implement. However, it is not always the best solution as it may lead to a bottleneck at the parent node. Migrating-to-parents can be used in a heterogeneous network in which parent nodes are of different kind than child nodes and have more resources. However, with a network that uses similar nodes, migrating to parent nodes is not the optimum solution.

5 Implementation of the Distributed Resource-Aware Framework

This section discusses issues we faced during the design and implementation of our distributed resource-aware framework for online data mining on the Sun SPOT platform.

5.1 Architectural design of the resource-aware framework

We use a couple of software design patterns to make the framework generic, extensible and maintainable and easy to implement on any platforms. Design patterns are classified in the well-known 'Gang-of-Four' book [11].

Firstly, we use the publish/subscribe pattern to decouple the resource monitor and the adaptive mining 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 Fig. 4, we have implemented our **ResourceMonitor** extends 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.

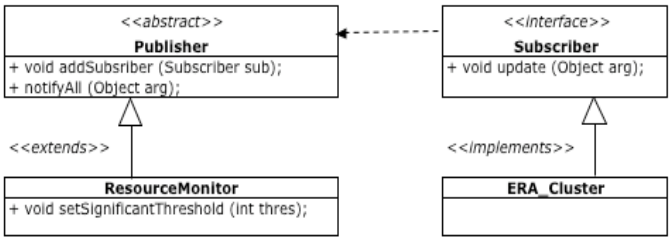


Fig. 4. Publish-Subscribe pattern of resource-awareness framework.

Secondly, we have implemented an abstract *factory* pattern for the data stream generator. We have a **Sensor** class to generate actual data stream sensed from the environment. Currently, data are light, temperature and 3D acceleration values, x, y and z. However for experimental purposes, there is the need for some synthetic data generator that gives us control on the evaluation parameters. For this purpose, we implemented a **RandomDS** class, which generates random data suitable to test our clustering algorithm.

In order to uncouple the data stream generator with the clustering algorithm, we use the factory design pattern. Following this pattern, we create a generic class called **DSGenerator** from which both **Sensor** and **RandomDS** extend. **DSGeneratorFactory** is responsible for creating **DSGenerator**. The implementation of **DSGenerator** is encapsulated and unknown to outsiders. Therefore, we can alternate between **Sensor** and **RandomDS** without changing the rest of the code. Fig. 5 shows the class diagram of the factory pattern.

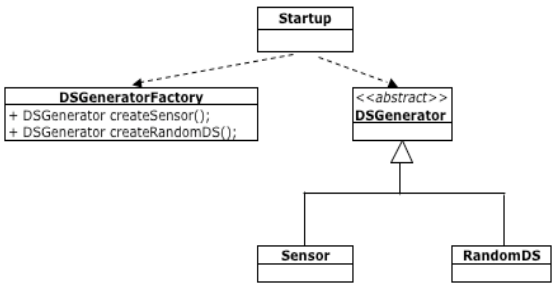


Fig. 5. Factory pattern.

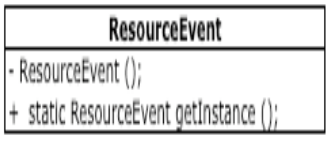


Fig. 6. Singleton pattern.

Thirdly, instead of creating a new resource event object for each update, we choose to have only one static singleton object of the **ResourceEvent** class. This can minimize the consumption of limited virtual memory of the node. Following this pattern (Figure 6), the constructor of each class is marked *private* instead of the normally *public* keyword. This means outside classes cannot arbitrarily create new object of this class. We then have a public and static method named *getInstance*() to return an object of this class. This method will return the existing object if there is already one or create a new object. Besides the resource event, some entities such as the battery simulation class and CPU utilization class are desired to be unique throughout the scope of the application. Therefore, we also apply the singleton pattern to these classes.

5.2 Resource Monitor

The responsibility of the resource monitor is to periodically examine remaining battery, memory and CPU utilization and 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.

The 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 quickly. 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 [12]. 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 facilitate experimental setup.

5.3 High level architectural diagram

Fig. 7 illustrates the high level architecture of the system. The ERA-Cluster and the resource monitor block are existing components from our previous project. We add the Communicator to facilitate the sending of datagram and the ServerDeamon is responsible for listening for incoming request and act accordingly. These building blocks make up the basis for the system.

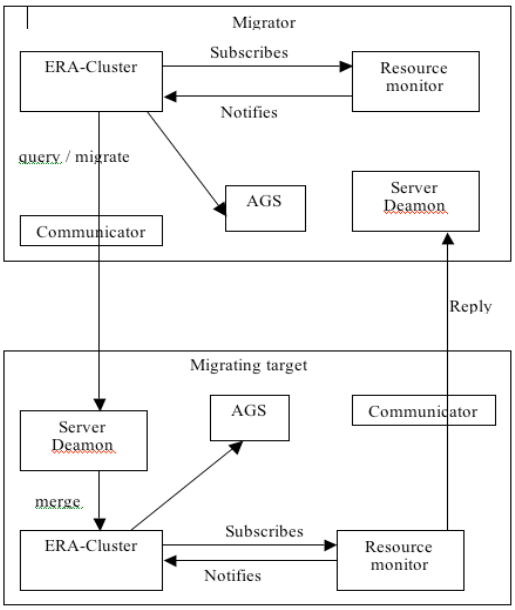


Fig. 7. Distributed Resource-aware Framework

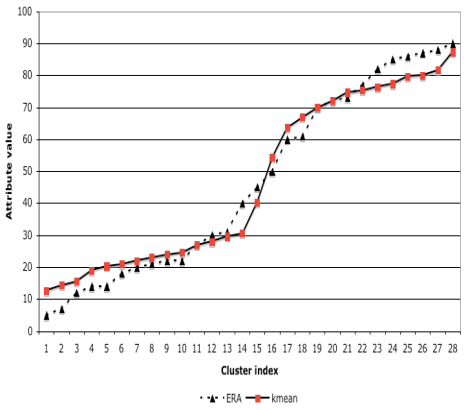
6 Experimental Evaluation

We conducted a small experimental evaluation of our resource-aware framework. We focus on proving two issues: The first is the resource adaptiveness of the framework. In other words, how effective the mining algorithms adapt to resource changes. This issue is to examine by comparing the resources – memory, battery, CPU utilization consumption pattern of the mining algorithm with against without the resource adaptiveness. The second issue of the evaluation deals with proving that the accuracy of the mining algorithm is acceptable even with its parameters adjusted to resource levels. This can be done by using another well known algorithm as a benchmark. For example, we compare the accuracy of our DERA-Cluster with the Weka's [13] simple K-Means clustering. Results show that DERA-Cluster's accuracy is comparable to Weka K-Means under normal operation (with resource adaptiveness). Under high CPU load, the accuracy will

be reduced. However, the overall accuracy is still acceptable. These experiments and results are detailed in [10].

For the distributed case, we aim to show that the accuracy of the migrated results at the destination node is acceptable. Similar to the previous approach, we use Weka K-Means clustering algorithm as a benchmark. The rationale behind using K-Means as a benchmark are also discussed in [10].

We use synthetic data for the experiments. Values are drawn from an uniform random integer of the range 0 to 100. We use a network comprising of two nodes for the experiments. We run node 1 for 10 seconds then migrate its clustered result to node 2. This migrated result is then merged with existing clustered result by the algorithm. We investigate the accuracy of this merged result. The original synthetic data set used up to that moment of node 1 and node 2 are combined. We then run K-Means 3 times over this synthetic data with k = n, n is the number of microclusters of the merged result. We sort all of the results of the merged result and three K-Means according to ascending order of mean value of the microclusters. We then plot the mean value of DERA-Cluster against the average mean value of K-Means. Figures 8 and 9 shows the results of this experiment.



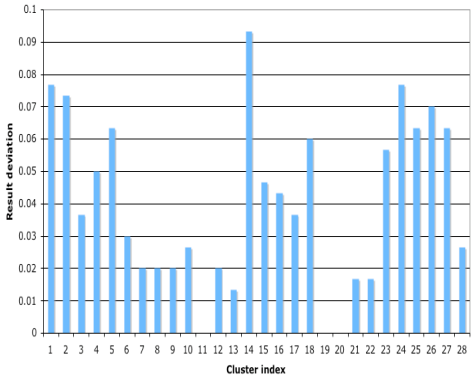


Fig.8. Accuracy of merged result compared to K-Means. **Fig.9.** Result deviation of merged result and K-Means.

From Figure 8, we can see a closed match between the two results despite some deviations. To justify these deviations, we calculate the result deviation of the merged result with K-Means, which is the absolute value of the deviation between the merge result and K-Means average over the maximum range of the mean value. Figure 9 shows that these deviations are small with the maximum accuracy loss are less than 10% while The average result deviation was less than 0.05. In other words, average accuracy loss is less than 5%.

7 Conclusions

This paper has presented a distributed resource-aware framework which can be used to enable resource adaptiveness for selective mining algorithms to be used in wireless sensor networks, in particular the Sun SPOT environment. The design of the framework is detailed and our approach to migrate results when nodes run out of battery are described.

Using the framework, we have implemented a distributed resource-aware online clustering algorithm termed DERA-Cluster. We have evaluated the accuracy of the migrated and merged results at the target node. Experimental results show that the loss of accuracy is acceptable. Possibilities for further work include: a) Evaluate the performance of the frame work on actual sensor data and with a network of multiple nodes. b) Implementation and study of the framework in another sensor platform such as Berkeley's Mote for comparison. c) Using the resource-aware framework to implement other online mining algorithms.

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

References

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

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

[3] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review", SIGMOD Record 34(2): 18-26, 2005.

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

[5] C. Franke, M. Karnstedt, and K.-U. Sattler, "Mining Data Streams under Dynamically Changing Resource Constraints", in *KDML 2006*

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

[7] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed Data Mining in Peer-to-Peer Networks", *IEEE Internet Computing*, vol. 10, pp. 18-26, 2006.

[8] S. Banyopadhyay, C. Giannella, U. Maulik, H. Kargupta, S. Datta, and K. Liu, "Clustering distributed data streams in peer-to-peer environments", *Information Science*, vol. 176, 2006.

[9] 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 ACM SAC* 2006.

[10] D. N. Phung, M. M. Gaber, and U. Roehm, "Resource-aware Online Data Mining in Wireless Sensor Networks", in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*. Honolulu, USA, 2007.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1993.

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

[13] I. H. Witten and E. Frank, Data Mining: Practical machine learning tools and techniques, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.