Exploring Load-Balance to Dispatch Mobile Sensors in Wireless Sensor Networks

You-Chiun Wang, Wen-Chih Peng, Min-Hsien Chang, and Yu-Chee Tseng
Department of Computer Science, National Chiao Tung University, Hsinchu, 30010, Taiwan
Email:{wangyc, wcpeng, changms, yctseng}@cs.nctu.edu.tw

***Abstract*— In this paper, a hybrid sensor network consisting of static and mobile sensors is considered, where static sensors are used to detect events, and mobile sensors can move to event locations to conduct more advanced analysis. By exploring the load balance concept, we propose a CentralSD algorithm to efficiently dispatch mobile sensors. Our algorithm is general in that the numbers of mobile sensors and events can be arbitrary. When mobile sensors are more than event locations, we transform the dispatch problem to a maximum-matching problem in a weighted bipartite graph. When there are fewer mobile sensors than event locations, we propose an efficient clustering scheme to group event locations so that the maximum-matching approach can still be applied. To reduce message cost, we also develop a distributed GridSD algorithm. Simulation results are presented to verify the effectiveness of the proposed algorithms.**

***Keywords*—**Mobile sensor dispatching, sensor data management, wireless sensor network.

I. Introduction

Recent advances in micro-sensing MEMS and wireless communication technologies have promoted the development of *wireless sensor networks* (*WSNs*). A WSN has many attractive characteristics including context-aware capability and fast ad-hoc networking configuration, so that it can be widely used in various applications such as surveillance and environment monitoring. However, sensor nodes are usually simple and may provide rough descriptions of events. For example, in a military application, pressure sensors may be deployed to check if any enemy passes. However, these sensors can only report something passing but cannot describe what passes through them. In this case, we may prefer using more powerful sensors like cameras to recognize the passing object. However, it is too expensive to mount a camera on each node due to the large number of sensors deployed. Alternatively, a better way is to utilize *mobile sensors* equipped with powerful sensing capabilities and dispatch these mobile sensors to visit event locations [1], [2].

In this paper, a hybrid WSN consisting of static and mobile sensors is considered, where static sensors are deployed to detect events and mobile sensors equipped with more resources such as sensing capability and computation power are dispatched to event locations to conduct more advanced analysis. Since mobile sensors are operated by small batteries, one important issue is to conserve their energies. In general, for a mobile sensor, the moving energy cost is larger than that of sensing and computing operations. Thus, we investigate how to efficiently dispatch mobile sensors to visit event locations with the purpose of maximizing the *system lifetime*, which is defined as the time period until some event locations cannot be reached by any mobile sensor.

Given a set of event locations round by round, one intuitive solution is to minimize the total moving energy of mobile sensors in each round. However, such an iteratively-optimized method may lead some mobile sensors early to exhaust their energies, thereby shortening the system lifetime. Consider an example in Fig. 1, where two mobile sensors s_1 and s_2 are at locations a and b, respectively. Both s_1 and s_2 have an initial energy of 400 units. We consider only the energy consumption of movements in mobile sensors. Assume that two events occur at locations c and d (respectively, a and b) in each odd (respectively, even) round. Fig. 1(b) shows the execution of the above iteratively-optimized method. To minimize the total moving energy, s_1 and s_2 are assigned to move between the pair of locations (a, c) and (b, d), respectively, resulting in a minimum cost of 95 units in each round. After seven rounds, s_2 almost runs out of its energy and stays at location d. In the eighth round, no mobile sensor has enough energy to reach the event location b so that the system lifetime is seven rounds. Actually, to dispatch mobile sensors more efficiently, we should not only reduce the total moving energy but also *balance* the loads of mobile sensors. Fig. 1(c) shows that by exploring load balance concept, s_1 and s_2 are assigned to move between the pair of locations (a, d) and (b, c), respectively. Although spending more energy (i.e., 100 units) in each round, balancing loads of mobile sensors in dispatching can extend the lifetime to eight rounds. It can be seen in Fig. 1 that simply optimizing the solution in each one-round dispatch could shorten the system lifetime because the early-exhausted mobile sensors will burden other still alive ones, which in turn drastically drains energy of alive mobile sensors.

Consequently, by exploring the load balance concept, we propose an algorithm *CentralSD (Centralized Sensor Dispatching)* to dispatch mobile sensors to maximize the system lifetime. Assume that a server is to collect the locations of mobile sensors and events. In CentralSD, we schedule mobile sensors to visit event locations to reduce the total moving distance. In addition, we balance the moving distance of each mobile sensor so that they can have similar energy costs to visit event locations and thus the system lifetime can be extended. Furthermore, in CentralSD, the numbers of mobile sensors and event locations are arbitrary and thus two cases are considered. Explicitly, when the number of event

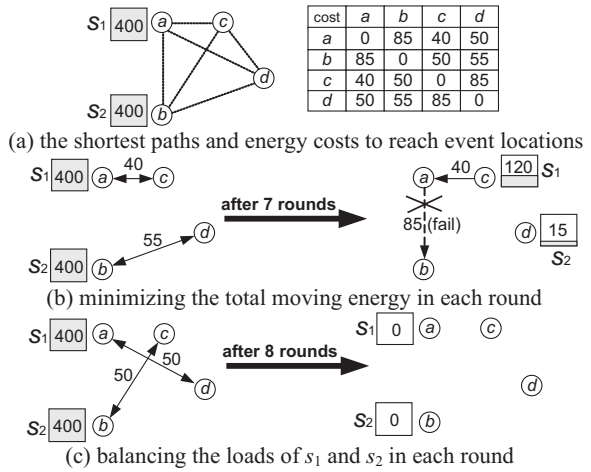


Fig. 1. Comparison of different dispatch methods.

locations is no larger than that of mobile sensors, we transform the dispatch problem to a maximum-matching problem in a weighted bipartite graph, where the vertex set contains mobile sensors and event locations, and the edge set contains the edges from each mobile sensor to each event location. However, instead of finding the matching with a minimum edge weight [3], we use a *preference list* and a *bound* to select the matching. Specifically, the load-balance can be achieved in that the preference list helps assign an event location with a suitable mobile sensor, while the bound is to avoid selecting edges with extreme weights. When event locations are more than mobile sensors, we propose an efficient clustering scheme to group locations into clusters, where the number of clusters is the same as that of mobile sensors. In this way, we can adopt the above matching approach to dispatch each mobile sensor to a cluster of event locations. Then, mobile sensors can use the traveling-salesman approximation algorithm [3] to reach all event locations within clusters. Nevertheless, CentralSD requires a server to collect information of mobile sensors and events, which incurs a considerable amount of message transmissions. To solve this problem, we develop an algorithm *GridSD (Grid-based architecture for Sensor Dispatching)* to dispatch mobile sensors in a distributed manner. In GridSD, a grid structure is maintained and the loads of collecting information and message exchanging are distributed into grids, thereby reducing both message transmissions and computation complexity.

A large amount of research [4]–[6] has elaborated on the issue of using mobile sensors to enhance the sensing coverage of a WSN. The authors in [7] consider how to move more sensors close to the locations of events predicted, while maintaining complete coverage of the sensing field. Assuming that the object's trajectory can be predicted and discusses how to maneuver mobile sensors to acquire data from the object in real-time, the authors in [8] use mobile sensors to track a moving object. In [9], static sensors detecting events will navigate nearby mobile sensors to move to their locations. The mobile sensor with shorter moving distance and more energy will be invited by the static sensors. In [10], static sensors estimate the coverage holes close to them and use the hole size to compete for mobile sensors. The mobile sensor will select the largest one and move to fill the hole. Prior works mainly utilize mobile sensors for coverage holes and object tracking, but not to the general dispatch problem formulated in this paper, let alone exploring load-balancing in dispatching mobile sensors and developing two algorithms to deal with the general dispatch problem. These features distinguish this paper from others.

II. The Sensor Dispatch Problem

In this paper, a hybrid WSN consisting of static and mobile sensors is considered. Static sensors form a connected network and fully cover the area of interest to continuously monitor the environment. When events are detected, there is a set of n mobile sensors $S = \{s_1, s_2, \ldots, s_n\}$ to be dispatched to the event locations (as reported by static sensors) to provide higher quality of sensing results. By existing localization techniques [11], sensors are aware of their locations.

The problem of dispatching mobile sensors is modeled as follows. We consider that there is a set of event locations $L = \{l_1, l_2, \ldots, l_m\}$, each to be visited by a mobile sensor. We allow an arbitrary relationship of m and n. The goal is to determine a *dispatch schedule* Φ_i for each mobile sensor s_i such that each location in L is visited exactly once by one mobile sensor. Each schedule Φ_i is denoted by a sequence of event locations, and the jth location to be visited is written as $\Phi_i[j]$. Let e_i be s_i's energy and $c(\Phi_i)$ be the energy required to complete s_i's schedule, $c(\Phi_i) = \Delta_{\text{move}} \times (d(s_i, \Phi_i[1]) + \sum_{j=1}^{|\Phi_i|-1} d(\Phi_i[j], \Phi_i[j+1]))$, where Δ_{move} is the energy required to move a sensor one-unit distance, $d(s_i, \Phi_i[1])$ is the distance from s_i's current position to $\Phi_i[1]$, and $d(\Phi_i[j], \Phi_i[j+1])$ is the distance between $\Phi_i[j]$ and $\Phi_i[j+1]$. Clearly, a schedule of a mobile sensor should satisfy $e_i \geq c(\Phi_i)$.

For performance reason, the objective function of the dispatch problem is to minimize the total energy cost incurred by movements, i.e., $\min \sum_{s_i \in S} c(\Phi_i)$. To balance the energy consumption of mobile sensors, we should also attempt to reduce the standard deviations of energy consumption among mobile sensors.

Note that the above modeling is only concerned about one round of sensors' dispatch schedules. In general, multiple rounds of dispatch schedules need to be determined, where each round contains those events being detected over a fixed amount of time, and the goal is to extend mobile sensors' lifetimes to the maximum number of rounds. The length of a round depends on users' real-time constraint. Since event locations are unexpected, we only focus on the optimization of one round in this paper.

III. Mobile Sensor Dispatching Algorithms

We first propose a centralized solution, where there is a central server that collects the sets L and S and computes sensors' schedules. Then, a distributed solution is developed.

Without loss of generality, we remove those mobile sensors in S that do not have enough energy to reach any location in L.

A. CentralSD: A Centralized Dispatch Algorithm

When $|S| \geq |L|$, we transform the dispatch problem to a maximum-matching problem in a weighted bipartite graph. When $|S| < |L|$, we partition L into $|S|$ clusters so that each mobile sensor only needs to visit one cluster of event locations. Then a maximum-matching approach can be applied again.

1) Case of $|S| \geq |L|$: We first construct a weighted bipartite graph $G = (S \cup L, S \times L)$ such that the vertex set contains all mobile sensors and all event locations, and the edge set contains the edge (s_i, l_j) from each $s_i \in S$ to each $l_j \in L$. The weight of (s_i, l_j) is defined as $w(s_i, l_j) = c(s_i, l_j)$, where the energy cost $c(s_i, l_j) = \Delta_{\text{move}} \times d(s_i, l_j)$ to move s_i to each location $l_j \in L$. With G, our goal is to find a matching P such that (1) the number of edges in P is the largest, (2) the total edge weight of P is minimized, and (3) the standard deviation of edge weights in P is minimized.

To find P, we associate a *preference list* $Plist(s_i)$ to each s_i, which ranks each location $l_j \in L$ by its weight $w(s_i, l_j)$ in an increasing order. When edge weights are equal, events' IDs are used to break the tie. Similarly, for each l_j, we associate it with a preference list $Plist(l_j)$, which ranks each $s_i \in S$ by its weight $w(s_i, l_j)$ in an increasing order.

To reduce the standard deviation of edge weights in P, we use a *bound* B_{l_j} for each $l_j \in L$ to restrict the candidate mobile sensors that l_j can match. Specifically, l_j can consider a mobile sensor s_i only if $w(s_i, l_j) \leq B_{l_j}$. The initial value of each B_{l_j} is set as $\frac{1}{m}\sum_{j=1}^{m} \min_{\forall(s_i,l_j)}\{w(s_i, l_j)\}$. For each $l_j \in L$, we find a sensor s_i in $Plist(l_j)$ such that $w(s_i, l_j)$ is minimized and $w(s_i, l_j) \leq B_{l_j}$ to match with. If no any s_i is available, we continue extending B_{l_j} with an increasing level Δ_B until one sensor is discovered for l_j. Note that the value of Δ_B should be carefully designed so that the weight of each pair will not increase sharply while the number of operations to extend the bound can be reduced. In particular, for each event, we should derive the distance interval of the farthest and the nearest mobile sensors. Those mobile sensors staying in the distance interval should be taken into consideration for dispatching. Furthermore, when more mobile sensors are available, an event can easily find a mobile sensor in its neighborhood. Otherwise, one should use a larger level to increase the possibility of finding available mobile sensors. Therefore, the increasing level Δ_B is formulated as

$$\frac{\delta}{mn} \times \Big(\sum_{j=1}^{m} \max_{\forall(s_i,l_j)}\{w(s_i, l_j)\} - \sum_{j=1}^{m} \min_{\forall(s_i,l_j)}\{w(s_i, l_j)\}\Big),$$

where δ is an adjustable coefficient.

As an unmatched location l_j expands its bound B_{l_j}, more candidates are included in its $Plist(l_j)$. If the first unvisited candidate s_i in $Plist(l_j)$ is also unmatched, the pair (s_i, l_j) is added into P. Otherwise, s_i must be matched with another location l_o. With the bounds B_{l_j} and B_{l_o}, we can determine to which location s_i should be dispatched. This is referred to as a competition between l_j and l_o. In particular, s_i should be matched to l_j if one of the following cases is satisfied:

- $B_{l_j} > B_{l_o}$. Since enlarging the bound will increase the risk of including an edge with an extreme weight into P, we will prefer matching s_i with l_j to avoid expanding the larger bound B_{l_j}.
- $B_{l_j} = B_{l_o}$ and l_j is prior to l_o in $Plist(s_i)$. As s_i prefers l_j, we match s_i with l_j to reduce the total weight of P.
- $B_{l_j} = B_{l_o}$ and s_i is the last candidate in $Plist(l_j)$ but not in $Plist(l_o)$. Since l_j cannot have another candidate to pick in $Plist(l_j)$, s_i should be matched with l_j.

Once s_i is matched with l_j, the pair (s_i, l_o) should be replaced by the new pair (s_i, l_j) in P, and l_o should search for another mobile sensor to match with. It is possible that l_o competes with other locations for mobile sensors.

Consider an illustrative example in Fig. 2, where δ is set to 2. The initial bound is $\frac{101+77+92}{3} = 90$ and the increasing level $\Delta_B = \frac{2\times((213+234+229)-(101+77+92))}{3\times4} = 67.7$. We start with the event location l_1. Since there is no available mobile sensor in $Plist(l_1)$ with the initial bound, B_{l_1} is expanded by Δ_B. Accordingly, B_{l_1} is updated to 90+67.7=157.7. As a result, we have three candidates s_1, s_2, and s_3. Since l_1 finds that the first unvisited candidate s_1 is unmatched, we add (s_1, l_1) to the matching P. Following the same operation, the pair (s_3, l_2) is determined shown in Fig. 2(b). However, after expanding B_{l_3}, l_3 finds that the first candidate s_1 has been matched with l_1. Thus, l_3 competes with l_1 for s_1. It can be verified that case 2 is satisfied (i.e., $B_{l_3} = B_{l_1} = 157.7$ and l_3 is prior to l_1 in $Plist(s_1)$). Consequently, (s_1, l_1) is replaced by (s_1, l_3) in Fig. 2(c). Following the same procedure, l_1 obtains s_3 from l_2 and then l_2 has to find an unmatched mobile sensor s_4 to pair with. Fig. 2(e) shows the final result.

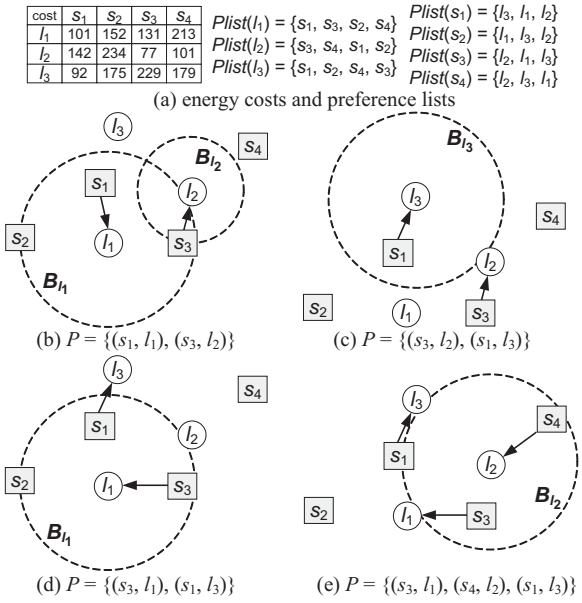


Fig. 2. An example to show how to find the matching P.

2) Case of $|S| < |L|$: When the number of event locations is larger than that of mobile sensors, we can group those event locations whose distance are close to each other into a cluster. This can be achieved by K-means [12]. Consequently, by the previous matching scheme, each mobile sensor can be dispatched to one cluster and then travels the event locations within the assigned cluster. To facilitate the presentation of this paper, we briefly describe how K-means works. In K-means, event locations are randomly partitioned into n non-empty clusters. For each cluster, we determine the mean from the event locations assigned to the same cluster. Then, each event location should join the cluster whose mean is the closest one to it. After all event locations decide their corresponding clusters, we should re-calculate the mean for each cluster. The above procedure is repeated until no event relocation is needed.

To evaluate the energy cost of the clustering result, the cost $\phi(\hat{c}_k)$ of each cluster $\hat{c}_k$ is formulated as the total edge weight of the minimum spanning tree in that cluster, where the weight of an edge (l_i, l_j) is the distance between two event locations l_i and l_j. For example, in Fig. 3(a), $\phi(A) = 50$, $\phi(B) = 12$, $\phi(C) = 15$, and $\phi(D) = 68$. Unfortunately, K-means may not minimize the total cost of the clusters derived, especially when some sparse event locations are far away from others. In this case, K-means groups these sparse locations into the same cluster, thereby increasing the total cost of clusters. Consider an example in Fig. 3(a), where four clusters are determined by K-means. Since both clusters A and D consist of sparse event locations (i.e., l_1 and l_{10}), the total cost of clusters is thus increased. By properly splitting and merging some clusters, the result of K-means is adjusted to reduce the total cost of clusters. Intuitively, those clusters containing sparse event locations should be split. However, in order not to change the number of clusters, we should merge two clusters when splitting a large one. In particular, let $w_{\max}^{\text{intra}}$ be the maximum edge weight among edges in all clusters and $w_{\min}^{\text{inter}}$ be the minimum inter-cluster distance, where the distance between two clusters $\hat{c}_a$ and $\hat{c}_b$ is the distance of the two closest locations $l_i \in \hat{c}_a$ and $l_j \in \hat{c}_b$. If $w_{\max}^{\text{intra}} > w_{\min}^{\text{inter}}$, we can split the cluster with the edge whose weight is $w_{\max}^{\text{intra}}$ (by removing that edge) and then merge two clusters whose distance is $w_{\min}^{\text{inter}}$ (by connecting them with the shortest edge). We can repeat the above procedure until $w_{\max}^{\text{intra}} \leq w_{\min}^{\text{inter}}$. In this way, we can avoid scenarios that some clusters have too large costs, thus reducing the total cost of clusters. Fig. 3 illustrates an example. In Fig. 3(a), $w_{\max}^{\text{intra}}$ is 50 (in cluster D) and $w_{\min}^{\text{inter}}$ is 15 (between clusters A and B). We thus split cluster D into two clusters D_1 and D_2, and then merge clusters A and B into the same one, as shown in Fig. 3(b). Similarly, we can further split cluster A and then merge clusters C and D_1 to reduce the total cost of clusters. The final result is shown in Fig. 3(c).

After grouping event locations into n clusters $\hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n\}$, we can use the matching scheme to dispatch mobile sensors to these clusters. To assign edge weights of the graph $G = (S \cup \hat{C}, S \times \hat{C})$, the energy cost is re-formulated as $c(s_i, \hat{c}_k) = \Delta_{\text{move}} \times (d(s_i, l_j) + \phi(\hat{c}_k))$, $\forall s_i \in S$ and $\hat{c}_k \in \hat{C}$,

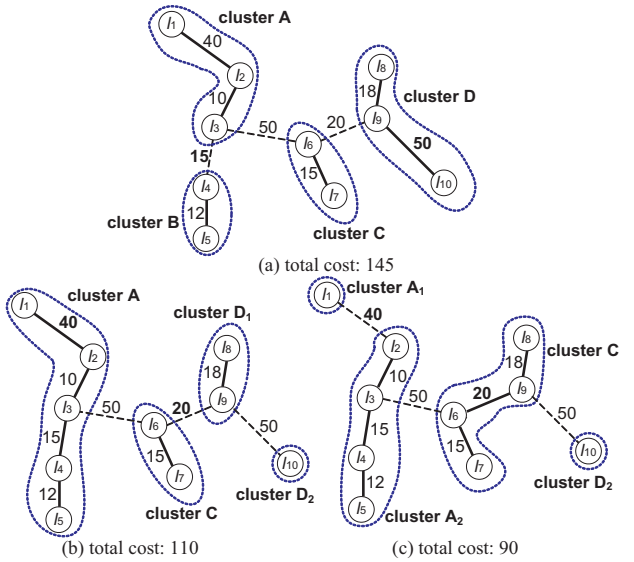


Fig. 3. An example to cluster event locations.

where $l_j \in \hat{c}_k$ is the closest event location to s_i. Specifically, the total energy consumption for s_i to visit $\hat{c}_k$ includes the energy to move to the closest event location l_j in $\hat{c}_k$ and the energy to reach all event locations in $\hat{c}_k$. When s_i is dispatched to a cluster $\hat{c}_k$, it first moves to the closest event location l_j in $\hat{c}_k$ and then exploits the solution of *traveling salesman problem* [3] to reach other event locations with a minimum cost.

B. GridSD: A Distributed Dispatch Algorithm

Note that CentralSD needs a central server to collect the information of mobile sensors and events, which results in a large amount of message transmissions. In this section, we propose a distributed algorithm GridSD that explores a grid-based architecture to reduce the messages incurred.

As shown in Fig. 4, GridSD divides the sensing field into grids. For each grid, we select a *grid head* to collect the information of mobile sensors and event locations within its territory. Specifically, each mobile sensor informs its location and remaining energy to its corresponding grid head. When detecting events, static sensors report to their grid heads. On obtaining such information, a grid head performs CentralSD to dispatch mobile sensors to the events occurred in its grid. However, if there is no mobile sensor in the grid, the grid head will search available mobile sensors in other grids.

To reduce the number of message transmissions when a grid head searches for mobile sensors in other grids, we propose a modified approach of the *grid-quorum* [5]. Specifically, each grid head sends *advertisement* (ADV) messages containing the number of mobile sensors in its grid to the same column of grids. In this way, each grid head has the information of mobile sensors in other grids located in the same column. When a grid head wants to search mobile sensors in other grids, it sends a *request* (REQ) message to the grid head in the same row. Due to the grid structure, there must be a grid head receiving both ADV and REQ messages. Consider an example in Fig. 4,

where the grid head in $(0,0)$ sends an ADV message to inform the grids $(0,1)$, $(0,2)$, and $(0,3)$ that two mobile sensors are available in grid $(0,0)$. Since there is no mobile sensor in grid $(1,2)$ and its corresponding column, its grid head sends an REQ message to the grids $(0,2)$, $(2,2)$, and $(3,2)$ to search mobile sensors. Clearly, the grid head in $(0,2)$ will receive both ADV and REQ messages and thus can reply the available mobile sensors in grid $(0,0)$ to the grid head of $(1,2)$.

Unlike grid-quorum, we exploit a *search length* to limit the number of grids to be queried in searching mobile sensors. In particular, each REQ message is associated with two integers α and M_{grid}, where α is the search length and M_{grid} is the number of mobile sensors found so far. Initially, $\alpha > 0$ and $M_{\text{grid}} = 0$ for each REQ message. When receiving the REQ message, a grid head increases M_{grid} by the number of mobile sensors in the column. If $\alpha > 1$, the grid head decreases α by one and propagates the REQ message to the next grid in the same row. However, if $\alpha = 1$ and the value of M_{grid} is still zero, which means that there is no mobile sensor found yet, the grid head sends the REQ message with $\alpha = 1$ to the next grid until at least one mobile sensor can be found. Fig. 4 illustrates the above scenario, where the grid head in $(1,2)$ sends an REQ message with $\alpha = 1$. On receiving the REQ message, the grid head in $(0,2)$ increases M_{grid} by two and decreases α by one. Since α becomes zero, the REQ message will not be propagated to the left-hand side. When the grid head in $(2,2)$ gets the REQ message, since $\alpha = 1$ and M_{grid} is still zero, the grid head in $(2,2)$ propagates the REQ message with $\alpha = 1$ to grid $(3,2)$ for searching mobile sensors. With the search length, GridSD can reduce not only the message complexity but also the competition of mobile sensors from grid heads. Once obtaining the information of mobile sensors and events, a grid head is able to perform CentralSD locally.

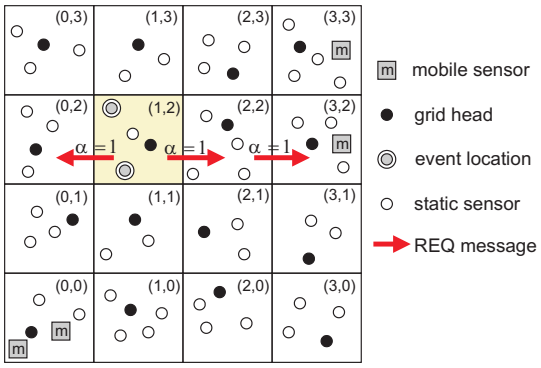


Fig. 4. An example to show how GridSD works.

IV. Experimental Results

In this section, we evaluate the performances of our algorithms by simulations. We set up a sensing field as a 450 m $\times$ 300 m rectangle, in which there are 400 static sensors and several mobile sensors randomly deployed. Each mobile sensor has an initial energy of 3960 J (joule) reserved for movement and the moving energy cost per meter is set to 8.27 J.

The first experiment evaluates the system lifetime of different algorithms when dispatching 50 mobile sensors. In each round, 10 to 15 static sensors are randomly selected as event locations. Mobile sensors then move to these locations based on the dispatch algorithm and stay at their last-visiting locations to wait for the next dispatch schedule. When mobile sensors are fewer than event locations, the proposed clustering scheme is adopted to group event locations. We observe the ratio of alive mobile sensors in each round. The system lifetime is referred as the round when all mobile sensors exhaust their energies. We compare CentralSD and GridSD against the iteratively-optimized method discussed in Section I.

Fig. 5 shows the system lifetimes of different methods. Although dispatching mobile sensors with the minimal energy cost in each round, the iteratively-optimized algorithm has the shortest system lifetime. This is because it does not balance the loads of mobile sensors, draining of the energy of some mobile sensors. The situation becomes worse as the exhausted mobile sensors burden the remaining alive ones with heavy loads. Our proposed algorithms have a longer lifetime since they not only reduce the total moving energy but also balance the loads of mobile sensors. Note that CentralSD outperforms GridSD since it has the global knowledge of the network.

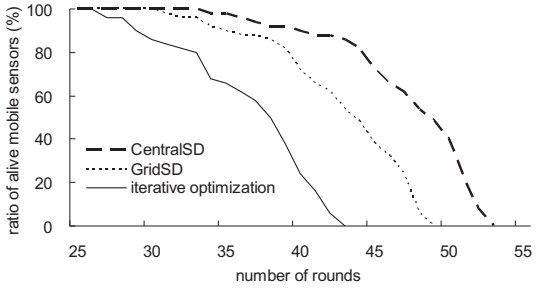
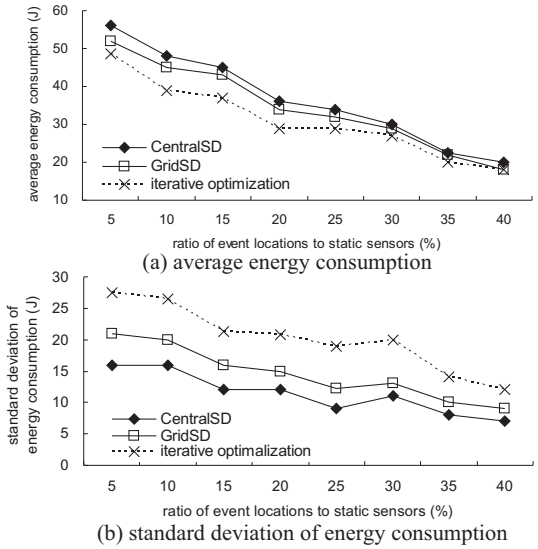


Fig. 5. Comparison on system lifetime.

We further evaluate these algorithms in terms of the moving energy and the load-balance metric (i.e., the standard deviation) among mobile sensors. The event locations are randomly selected from 5% to 40% of static sensors. To fairly compare the standard deviation, we set the number of mobile sensors as equal to that of event locations, so that each mobile sensor is dispatched to exactly one location.

Fig. 6(a) illustrates the average energy consumption. Since the iteratively-optimized algorithm always finds the optimal solution, it has the smallest average of energy consumption. By adopting the preference lists, the averages of our algorithms are slightly higher than that of the optimal solution. Note that in GridSD, with search lengths, grid heads with event locations are able to search those mobile sensors nearby, thereby having a smaller average compared with CentralSD. Fig. 6(b) shows the standard deviation of energy consumption. We can observe that the standard deviation of the iteratively-optimized algorithm is almost twice than that of CentralSD, indicating that the former results in seriously unbalance loads among mobile sensors. Note that GridSD has a larger standard

deviation compared with CentralSD since each grid head only has partial information of mobile sensors.



(a) average energy consumption

(b) standard deviation of energy consumption

Fig. 6. Comparisons on energy consumption.

Although CentralSD outperforms GridSD in terms of system lifetime, it incurs a large amount of message transmissions. Fig. 7 shows the number of packet delivery of CentralSD and GridSD. We can observe that the number of message transmissions in CentralSD grows fast as the event locations increase, while that in GridSD remains constant since the loads of message exchange are distributed among grid heads.

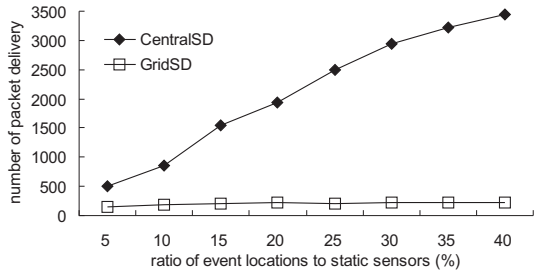


Fig. 7. Comparison on number of packet delivery.

When the number of event locations are larger than that of mobile sensors, we group event locations into clusters and dispatch mobile sensors to each cluster. Fig. 8 shows the effect of our clustering scheme on the energy consumption. As can be seen in Fig. 8, when the clustering scheme is adopted, mobile sensors can have a lower energy consumption because they do not need to travel around event locations far from each other.

V. Conclusions

In this paper, a general problem of dispatching mobile sensors is formulated. By exploring the concept of load balance, we proposed CentralSD to efficiently dispatch mobile sensors. A clustering scheme is developed to group event

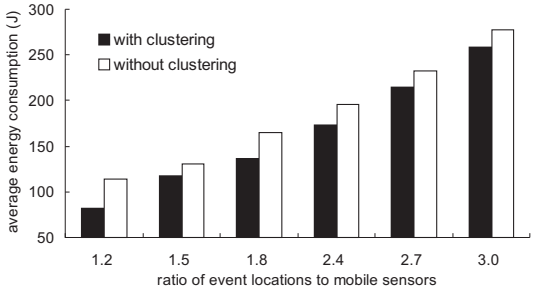


Fig. 8. The effect of clustering on energy consumption.

locations when the number of events is larger than that of mobile sensors. To reduce message transmissions, we proposed a distributed algorithm GridSD. Simulation results show that the proposed algorithms can have a longer system lifetime compared with the iteratively-optimized algorithm.

Acknowledgement

Y. C. Tseng's research is co-sponsored by Taiwan MoE ATU Program, by NSC grants 93-2752-E-007-001-PAE, 96-2623-7-009-002-ET, 95-2221-E-009-058-MY3, 95-2221-E-009-060-MY3, 95-2219-E-009-007, 95-2218-E-009-209, and 94-2219-E-007-009, by Realtek Semiconductor Corp., by MOEA under grant number 94-EC-17-A-04-S1-044, by ITRI, Taiwan, by Microsoft Corp., and by Intel Corp.

W. C. Peng is supported in part by Taiwan MoE ATU Program and by the National Science Council, Project No. NSC 95-2211-E-009-61-MY3 and NSC 95-2221-E-009-026, Taiwan, Republic of China.

References

[1] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: a robotic wireless and sensor network testbed," in *IEEE INFOCOM*, 2006.

[2] Y. C. Tseng, Y. C. Wang, and K. Y. Cheng, "An integrated mobile surveillance and wireless sensor (iMouse) system and its detection delay analysis," in *ACM Int'l Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2005, pp. 178–181.

[3] J. R. Evans and E. Minieka, *Optimization Algorithms for Networks and Graphs, 2nd ed.* Marcel Dekker Inc., 1992.

[4] N. Heo and P. K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Trans. on Syst., Man and Cybern. A*, vol. 35, no. 1, pp. 78–92, 2005.

[5] G. Wang, G. Cao, T. L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *IEEE INFOCOM*, 2005, pp. 2302–2312.

[6] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *IEEE INFOCOM*, 2003, pp. 1293–1303.

[7] Z. Butler and D. Rus, "Event-based motion control for mobile-sensor networks," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 34–42, 2003.

[8] M. D. Naish, E. A. Croft, and B. Benhabib, "Dynamic dispatching of coordinated sensors," in *IEEE Int'l Conf. on Systems, Man, and Cybernetics*, 2000, pp. 3318–3323.

[9] A. Verma, H. Sawant, and J. Tan, "Selection and navigation of mobile sensor nodes using a sensor network," in *IEEE Int'l Conf. on Pervasive Computing and Communications*, 2005, pp. 41–50.

[10] G. Wang, G. Cao, and T. L. Porta, "A bidding protocol for deploying mobile sensors," in *IEEE Int'l Conf. on Network Protocols*, 2003, pp. 315–324.

[11] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices," *IEEE Personal Commun. Mag.*, vol. 7, no. 5, pp. 28–34, 2000.

[12] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, D. D. Cerra, Ed. Academic Press, 2001.