Lifetime Aware Resource Management for Sensor Network Using Distributed Genetic Algorithm

Qinru Qiu Qing Wu
Department of Electrical and Computer Engineering
Binghamton University
Binghamton, NY 13902
001-607-777-4918, 001-607-777-4536
{qqiu, qwu}@binghamton.edu

Daniel Burns Douglas Holzhauer
Air Force Research Laboratory, Rome Site
26 Electronic Parkway
Rome, NY 13441
001-315-330-2335, 001-315-330-4920
{Daniel.Burns, Douglas.Holzhauer}@rl.af.mil

ABSTRACT
In this work we consider lifetime-aware resource management for sensor network using distributed genetic algorithm (GA). Our goal is to allocate different detection methods to different sensor nodes in the way such that the required detection probability can be achieved while the network lifetime is maximized. The contribution of this paper is twofold. Firstly, the resource management problem is formulated as a constraint optimization problem and is solved using a distributed GA. Secondly, empirical analysis results are provided that reveals the relationship between the configuration parameters and the quality of the search. A regression model is designed to estimate the runtime of the distributed GA given the configuration parameters. The model is utilized to find energy efficient configurations of the algorithm.

Categories and Subject Descriptors
J.7 [**Computer Applications**]: Sensors and Sensor Networks

General Terms
Experimentation

Keywords
Distributed Genetic Algorithm, Sensor Network, Energy Aware Design, Resource Management

1. INTRODUCTION
Due to the fast development of information technology, the networked distributed system is gradually replacing the conventional centralized system. It is a vision of the future that large numbers of low cost smart mobile devices will be integrated into the daily life of ordinary people. Accumulated, they provide the information processing capability that is equivalent to a high performance processing station. The emerging concept of Ambient Intelligence [1] and the recent developments of sensor networks [2], and wearable computers [3] reflect such vision. A distributed system consists of multiple heterogeneous networked processing elements, which are battery-powered and work on a set of tasks collaboratively. Each processing element has limited resources, such as battery energy, communication bandwidth, etc. It is a challenging task to efficiently utilize these resources to deliver required services during the runtime in a dynamic environment.

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.

Resource management is defined as the process that assigns tasks to different processing elements, schedules their start times and decides the level of service quality, which determines the resource usage, such as the energy dissipation and communication bandwidth, to run these tasks. The execution of each task represents a positive gain when measuring or quantifying the performance of the system. It also associates a cost, which represents the resource usage. The resource management problem can be formulated as a multi-objective optimization problem, i.e. maximizing the gain while minimizing the cost. It can also be formulated as a constraint optimization problem, i.e. maximizing the gain while satisfying the cost constraint or vise versa.

In this paper we focus on the management of the energy resource in an environment monitoring sensor network that is used to monitor, model and forecast physical processes, such as environment pollution, flooding, and fire etc. The basic configuration of each node in this network consists of a microprocessor, a wireless transceiver and an array of sensors such as light detector, barometer, humidity and thermopile sensors. A set of data acquisition and signal processing applications is available on each node. They provide the tradeoffs between detection quality and resource utilization. For example, increasing the sampling rate improves the probability of detecting an abnormal event however it increases the power consumption as well.

There is usually a significant cost associated with deploying an environment monitoring system. It is desirable that the system can work for a reasonably long time after it is deployed. A common approach is to incorporate certain level of redundancy in the system. More than one node usually will be deployed to cover the same region. These nodes may be turned on alternatively to extend the network lifetime or simultaneously to increase the detection probability. If the minimum detection accuracy is given as a user constraint, the resource management problem for the system is to determine which sensor nodes should be turned on to process which data acquisition and signal processing application such that the network lifetime can be maximized while meeting the required detection accuracy. This is a well known general assignment problem which has been proven to be NP-complete [4].

Most of the traditional resource optimization algorithms are solved in a centralized, off-line approach which is not suitable for a distributed system. In this paper we study the use of distributed genetic algorithm (GA) to solve the above mentioned optimization problem, potentially using processing capabilities residing on nodes of the distributed sensor network. One of the major characteristics of the GA is that it is "embarrassingly parallel", in the sense that, its workload can easily be evenly distributed among processors, making it an appropriate choice for solving optimization problems in distributed systems. The configurations of the distributed, multi-

deme GA, such as the population size, the migration rate, and the parallelism, has a significant impact on the quality of the search [8]. Finding efficient configurations of the distributed GA is an important research topic. The contribution of this paper is twofold. Firstly, the resource management problem is formulated as a constraint optimization problem and is solved using a distributed GA. The simulation results show that the resulting task allocation scheme increases the system lifetime by 14.4% in average, comparing to heuristic approaches. Secondly, empirical analysis results are provided that reveals the relationship between the configuration parameters and the quality of the search. A regression model is presented that estimates the runtime of the distributed GA given the configuration parameters. This model is then used to find energy efficient configurations of the algorithm.

Many previous works on sensor network resource management and task allocation address network communication issues [5][6]. In these schemes the nodes are dynamically awakened to route a message. In reference [7] the resource allocation problem in a vehicle tracking system is modeled as a virtual market and solved using feedback control. This work focuses more on the tracking of a moving object rather than the collaborative detection of a static event. Therefore it cannot be applied in the environment monitoring system. Reference [9] focuses on task allocation on the gateways in a cluster-based sensor network. The problem is also formulated as a constraint optimization problem and is solved using simulated annealing, which is a centralized stochastic searching algorithm. Compared with reference [9], the resource management problem considered in this paper has a different set of constraints and objective functions and is solved using a distributed GA.

The rest of this paper is organized as follows. Section 2 introduces the sensor network architecture. Section 3 presents the distributed GA algorithm. Section 4 provides the empirical analysis of the relationship between the configuration parameters and the quality of search of GA and derives the regression model for runtime estimation. Section 5 discusses the utilization of the regression model to design energy efficient distributed GA. Sections 6 and 7 provide the experimental results and summaries, respectively.

2. SENSOR NETWORK ARCHITECTURE

We consider the sensor network that is deployed with a certain level of redundancy. The network can be partitioned into several clusters. Each cluster consists of p sensor nodes that are responsible for performing monitoring and hazard detection in the same region. Each sensor node is low cost and low quality; however combined together they provide very accurate detection. The nodes in the same cluster have direct communication with each other via wireless communication channels. The nodes in different clusters communicate with each other through gateways. In this work we assume that the clustering and routing scheme is provided. We also assume that each cluster has advanced data fusion capability so that the traffic of inter-cluster communication is low.

An array of w sensors is installed on each node. The reading from these sensors can be sampled by l different sampling frequencies. Obviously, higher sampling frequency leads to higher detection probability while consumes more energy. The sampled data from sensor i can be analyzed in x_i different ways. They provide different tradeoffs between accuracy and energy dissipation. A *detection method* (i.e. *task*) is considered as a combination of sensing function, sampling frequency and signal processing algorithm.

Each task-processor pair (i, k), $1 \le i \le n$ and $1 \le k \le p$, associates with two variables $pow_{i,k}$ and $prob_{i,k}$, which represent the power consumption and the detection probability of task i when it is running on processor k. The $prob_{i,k}$ is a function of the location and the environment of the sensor node. We assume that this function is pre-calibrated and installed on each sensor node before its deployment. The sensor node will collect the environment information and calculate the detection probability using the provided function periodically. To improve the detection probability, the node is allowed to use more than one detection method at the same time. The combined detection probability of node k can be calculated as $1 - \prod_{i \in \Delta_k} (1 - prob_{i,k})$, where Δ_k is the set of tasks that are allocated to node k. The total node power consumption can be calculated as $\sum_{i \in \Delta_k} pow_{i,k}$. The detection probability $Prob$ of a cluster with p nodes is calculated as $Prob = 1 - \prod_{1 \le k \le p} \prod_{i \in \Delta_k} (1 - prob_{i,k})$.

The goal of resource management is to find the Δ_k for each processor k so that the combined detection probability of the cluster is larger than the user defined constraint while the network lifetime is maximized. In this work, we define the network lifetime as the time from the deployment of the sensor network to the time when the first node runs out of battery energy. We assume that each sensor node is built with the smart battery Bus (SMBus) [10] which enables the system software to keep tracking of the remaining battery capacity and estimate the remaining lifetime.

3. RESOURCE MANAGEMENT USING DISTRIBUTED GA

A Genetic Algorithm (GA) is a stochastic search technique based on the mechanism of natural selection and recombination. It starts with an initial *population* of individuals, i.e. a set of randomly generated candidate solutions. The solutions are represented by *chromosomes*, which are collections of numbers or symbols that map onto parameters of the problem. Individuals are evolved from generation to generation, with *selection*, *mating*, and *mutation* operators that provide an effective combination of exploration of the global search space and pressure to converge to the global minimum. The solution quality is measured by a *fitness* function.

The Island multi-deme GA is one of the parallel GA models that are widely used [8]. In this model, the population is divided into several sub-populations and distributed on different processors. Each sub-population evolves independently for a few generations, before one or more of the best individuals of the sub-populations migrate across processors. The time between migrations is called *epoch*.

In this work the Island multi-deme GA is used to optimize the resource management for a cluster of sensor nodes. Each individual solution is a chromosome of n symbols, where n is the total number of tasks in the cluster. We assume that each task can only be selected by at most one sensor node in a cluster because multiple executions of the same task only generate redundant information. If the jth task is allocated to node x then the jth entry of the chromosome is equal to x. If the jth entry of the chromosome is -1 then this task is not allocated to any of the processors. Denote the user specified minimum detection probability as $prob_{th}$, the fitness function is:

$$fitness = \begin{cases} 0 & \text{if } Prob < Prob_{th} \\ \min\limits_{1 \le k \le p} (B_k / \sum\limits_{i \in \Delta_k} pow_{i,k}) & \text{otherwise} \end{cases} \quad (1)$$

where B_k is the remaining battery capacity of node k. The fitness of an individual is 0 if the corresponding resource management scheme cannot meet the user specified detection threshold; otherwise its

fitness is equal to the minimum remaining lifetime of the nodes. Single point crossover mating function is used in our experiment. The mutation probability is set to 1%, and involves flipping bits in integer representations of the parameters stored in chromosomes.

The GA is running on *np* processors. Each sub-population is initialized randomly and its size is denoted as *pop*. The sub-population evolves independently for *c* generations, and then 5 of the best individuals are broadcast to all other processors. The three parameters, *np*, *pop* and *c*, will be referred as the *configuration parameters* in the rest of this paper. The value of the configuration parameters has significant impact on the convergence speed of the GA and the quality of the solution. An empirical analysis is next.

4. CONFIGURATION PARAMETERS

We are interested in understanding the effect of configuration parameters on the quality of the search of the distributed GA that is previously discussed. Some work has been carried out in this area [8]. However, most of these involve the analysis of simple optimization problems such as the fully deceptive function [12]. Whether their results can be applied to our problem is unknown. Due to the extremely large search space and very complicated stochastic behavior of the GA, we found that it is difficult to perform an analytical study. Therefore, extensive experiments have been simulated and the relation between the configuration parameters and the quality of search is derived empirically.

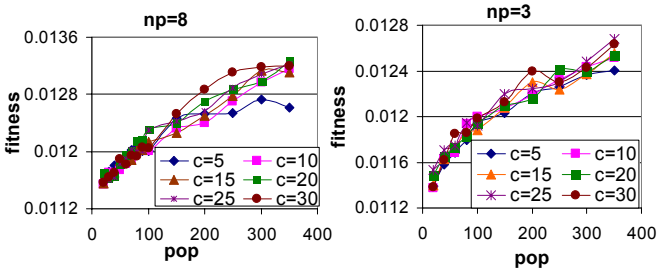
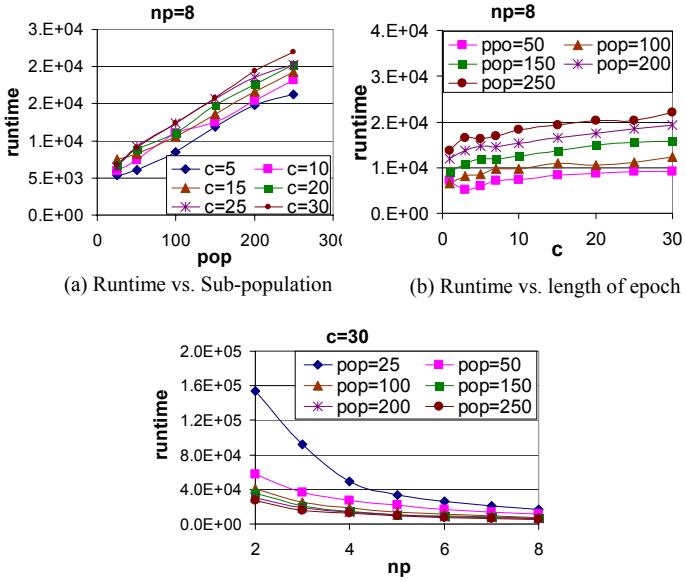


Figure 1 Normalized fitness vs. Sub-population size

Two sets of experiments have been carried out. In these experiments, we model a cluster of 10 sensor nodes. There are 100 tasks available. The GA is running on *np* sensor nodes with $np \le 10$. The detection probability and the power consumption of each task are uniformly distributed random variables whose range is 1% ~ 25% and 0.1Watt ~ 10Watt respectively. The battery of each sensor has the capacity of 5000 Ampere·hour and the V_{dd} is 1V. Because GA is a stochastic algorithm, we run each simulation 50 times and report the mean value.

The first set of experiments is designed to find out the effect of the configuration parameters on the quality of the solution. We swept the *np* from 2 to 8, the *pop* from 25 to 350, and the *c* from 1 to 35. For each configuration, the distributed GA is simulated. The GA will stop when the fitness of the best individual does not improve for 2000 generations. The relation between the *pop* and the normalized fitness of the best individual is reported. Figure 1 shows two sets of data for *np* (i.e. the number of processors) equal to 8 and 3. The results show that increasing both the *pop* and the *np* improves the quality of the solution. However, varying *c* has very little impact on it. Therefore the quality of the solution is determined by the size of the total population which is the product of the *pop* and *np*.

The second set of experiments is designed to find out the effect of the configuration parameters on the runtime of the GA. The value of *np*, *pop* and *c* are swept in the same way as the first experiment. The GA stops when the fitness of the best individual exceeds the threshold which is set to be 5 times of the expected fitness of a random individual. The number of generations that the GA has iterated is reported. Due to the iterative nature of GA, it is reasonable to assume that the runtime of each generation is approximately the same and it increases linearly as population size increases. Therefore we use the product of *pop* and the number of generations that the GA has iterated as a measure of the runtime.



(a) Runtime vs. Sub-population

(b) Runtime vs. length of epoch

(c) Runtime vs. parallelism

Figure 2 Runtime vs. configuration parameters

The relation among *pop*, *c*, *np* and the runtime are extracted from the results of second experiment. Figure 2 (a)-(c) show some of the data that we have obtained. Several observations can be made from these data. First, when the size of the sub-population increases, the runtime increases linearly. Combined with the results from the first experiment we can see that if the goal of the GA is to find the best possible solution, then a large population should be used. However, if the goal of the GA is to find a good solution in a short time, then increasing the population size will not help. Instead a small population should be used. Second, reducing the migration rate will result an almost linear increase in solution time. The slope is the same for different sub population size. Third, increasing the number of processors will reduce runtime, and this effect is more dominant when the sub-population is small.

In order to consider the combined effect of all of the three configuration parameters, we introduce a new variable called *effective population* (*Epop*). The size of the effective population increases when the size of the sub-population, the parallelism or the migration rate increases. It can be calculated as the following:

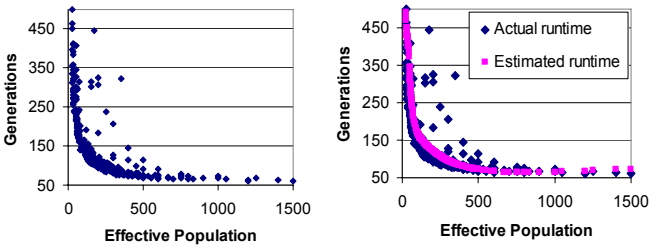
$$Epop = pop + pop \cdot (np - 1) / c \qquad (2)$$

Given the effective population, the runtime of the distributed GA can be predicted. Let *G* denote the number of generations that the GA has iterated before it finds the solution with the required fitness. Figure 3 (a) gives the relation between *Epop* and *G*. It shows that *G* is a continuous and differentiable function of *Epop*.

Based on the observation, we construct a prediction model to predict the number of generations that the GA has iterated.

$$G = a + a_0 / \sqrt{Epop} + \sum_{i=1}^{5} a_i / Epop^i \qquad (3)$$

The coefficients a, a_0 …, a_5 are obtained using regression analysis. Note that the value of the coefficients will change if the experiment setup changes. Here the experiment setup includes the threshold of fitness and the distribution function of *prob* and *power*. For each new setup, regression analysis should be performed to obtain the values of the coefficients.



(a) G vs. effective population G (b) Comparing predicted and actual G

Figure 3 Runtime vs. effective population

The above model gives quite accurate prediction of the number of generations that GA has iterated given the configuration parameters. Figure 3 (b) compares the prediction model with the simulated results. The blue dots give the G value obtained from simulation and the magenta dots give the G value obtained using the prediction model. The runtime T is measured as the product of *pop* and the number of generations $T = G \cdot pop$.

5. ENERGY EFFICIENT CONFIGURATIONS OF DISTRIBUTED GA

Sensor nodes are energy constraint systems. Any application running on the sensor node should be designed carefully to achieve high speed and energy efficiency. In this section we will discuss how to select the configuration parameters to minimize the energy dissipation of the distributed GA.

In a computing system with fixed supply voltage (V_{dd}) and clock frequency, reducing the runtime of an algorithm leads to linear reduction of the energy dissipation if the processor can be turned off after the program finishes. More energy saving is possible by using *dynamic voltage and frequency scaling* (*DVFS*), which is one of the runtime power management approaches that is supported by many processors for the state-of-the-art mobile computing platforms. It is a property of CMOS digital circuit that reducing the V_{dd} can reduce the energy dissipation quadratically but increase the circuit delay linearly [11]. In a system with DVFS capability, the program is running at the minimum supply voltage and clock frequency so that it finishes just before the deadline. Due to the convex relation between the energy and the runtime, this gives more energy saving than running the program at the nominal speed and turn off the processor after the program finishes.

Population migration among the processors is an important feature in distributed GA. The communication energy to broadcast the best individuals must be considered. Under the assumption of a fixed transmission power and a constant transmission speed, the communication energy is proportional to the size of the transmitted data. The communication energy will not be affected by DVFS.

The computing energy is a product of the runtime and the power consumption of the processor. Therefore, the runtime model proposed in Section 4 is the key for the energy estimation of the distributed GA. While increasing the migration rate, decreasing the population size and increasing the parallelism reduce the runtime of GA and consequently reduce the computing energy, frequent population migration leads to high communication energy. The configuration parameters must be selected carefully to minimize the overall system energy dissipation, which is the sum of computing energy and communication energy.

Let T_{nom} denote the process time for a single individual in each generation at nominal V_{dd} and let p_{nom} denote the power consumption of the processor at nominal V_{dd}. The energy dissipation of GA on a processor without DVFS can be calculated as:

$$E = T_{nom} \cdot p_{nom} \cdot T + G/c \cdot N \cdot E_{bit}, \qquad (4)$$

where G is the number of generations that GA has iterated, T is the runtime of the GA that is measured as the product of *pop* and G, c is the length of an epoch, N is the size of data that is broadcasted during each population migration, and E_{bit} is the energy to transmit one bit data. The first term in equation (4) is the computing energy and the second term is the communication energy. Furthermore, $T_{nom} \cdot p_{nom}$ represents the computing energy to processor one individual in each generation and $N \cdot E_{bit}$ represents the communication energy to broadcast the best individual during one migration. T_{nom}, p_{nom}, and E_{bit} are hardware related constant parameters. N is determined by the size of migrations which is also a constant value. Because we are not interested in calculating the absolute energy dissipation, we simplify equation (4) and consider a normalized energy dissipation which is calculated as the following,

$$E_{norm} = \frac{E}{T_{nom} \cdot p_{nom}} = T + \frac{G}{c} E_{mig}, \qquad (5)$$

where E_{mig} is the ratio of communication energy versus computing energy and it is calculated as $E_{mig} = \frac{N \cdot E_{bit}}{T_{nom} \cdot p_{nom}}$. As we can see the value of E_{mig} is determined by the system hardware configuration. For example, the power consumption of a Lucent ORiNOCO USB Wireless Adapter is 360mA in TX mode and 245mA in RX mode. The typical active power of an Intel XScale processor is 300mA. Assume that the data is transmitted at 1Mbit/s. If N equals to 1k bytes and T_{nom} equals to 5μs, which is the time to run 10k instructions at 200MHz clock, then E_{mig} is approximately 160.

E_{norm} is an increasing function of *pop* and a decreasing function of *np* because changing these two parameters only affects the computing energy. The only configuration parameter that affects both the computing and communication energy is c. Provided with the value of *pop*, *np*, E_{mig}, it is not difficult to find the optimal c that minimizes E_{norm} by solving the differential equation $\frac{\partial E_{norm}}{\partial c} = 0$.

Because GA is running on multiple sensor nodes, the total energy dissipation can be calculated as $E_{total} = np \cdot E_{norm}$ where *np* is the parallelism of the GA.

If the DVFS is available on the processor, then the computing energy can be scaled quadratically as the runtime decreases. The energy dissipation of GA on each processor can be calculated as the following,

$$EDVFS = T_{nom} \cdot p_{nom} \cdot T \cdot s^2 + G/c \cdot N \cdot E_{bit} \qquad (6)$$

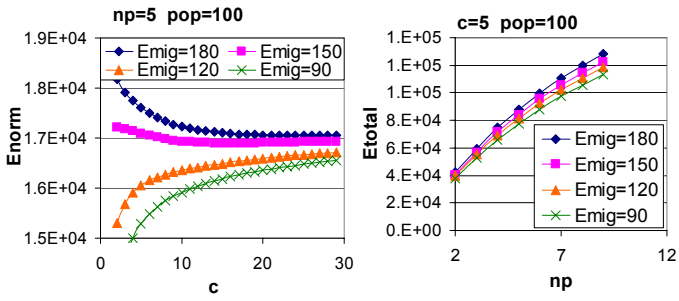
Here s is the scaling factor and it is calculated as $s = \frac{T_{nom} \cdot T}{T_{req}}$, where T_{req} is the deadline before which the GA must return a solution with the required fitness. Again, we simplify equation (6) and consider the normalized energy dissipation as the following,

$$EDVFS_{norm} = (T \cdot T_{nom} / T_{req})^3 + G/c \cdot E'_{mig}, \qquad (7)$$

E'_{mig} is calculated as $E'_{mig} = \frac{N \cdot E_{bit}}{p_{nom} \cdot T_{req}}$ which stands for the ratio of the communication energy for one migration versus the computing energy of the program if if takes exactly T_{req} time when running at the nominal V_{dd}. For the previous mentioned hardware system, which consists of Lucent ORiNOCO USB Wireless Adapter and Intel XScale processor, if the T_{req} is 1ms then E'_{mig} is approximately 0.8.

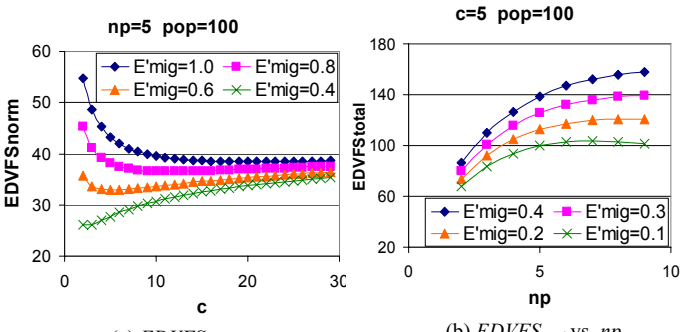
Again, the total energy dissipation can be calculated as $EDVFS_{total} = EDVFS_{norm} \cdot np$ and the optimal c that minimizes the energy dissipation can be found by solving the differential equation $\frac{\partial EDVFS_{norm}}{\partial c} = 0$.

Plug in the runtime estimation of G and T into equation (4)~(7), the energy dissipation of GA can be expressed as a function of the configuration parameters. Figure 4 (a) shows the relation between E_{norm} and c in a system without DVFS. The np and pop are set to 5 and 100 respectively. The E_{mig} varies from 90 to 180. As we can see from the figure, the energy is an increasing function of c for small E_{mig} and a decreasing function of c for large E_{mig}. Furthermore, when the E_{mig} falls into certain range, the energy is first a decreasing then an increasing function of c. In this case, we need to solve the previous mentioned differential equation to find the most energy efficient migration rate. When the parameter c gets larger, the E_{norm} under different E_{mig} approach to the same value. This is because the migration rate is so low that a small difference in the communication energy does not have a significant affect on the total energy.



(a) E_{norm} vs. c (b) E_{total} vs. np

Figure 4 Energy vs. configuration parameters without DVFS



(a) $EDVFS_{norm}$ vs. c (b) $EDVFS_{total}$ vs. np

Figure 5 Energy vs. configuration parameters with DVFS

Figure 4 (b) shows the relation between E_{total} and np in a system without DVFS. The parameters c and pop are set to 5 and 100 respectively. E_{mig} varies from 90 to 180. It is interesting to note that the total energy always increases no matter how we change the E_{mig}. This indicates that without DVFS the energy efficiency will decrease as the parallelism increases.

Figure 5 (a) shows the relation between $EDVFS_{norm}$ and c in a system with DVFS. The np is set to 5, the pop is set to 100 and the E'_{mig} varies from 1.0 to 0.4. In this figure, we see the similar trend as what has been shown for the system without DVFS. Figure 5 (b) shows the relation between the $EDVFS_{total}$ and np with E'_{mig} varies from 0.4 to 0.1. As we can see that for systems with $E'_{mig} \geq 0.3$, increasing the parallelism always increases the total energy dissipation. However, for systems with $E'_{mig} < 0.3$, increasing the parallelism will first increase then decrease the total energy. This is because increasing the parallelism reduces the overall computing energy quadratically and increases the overall communication energy linearly. Eventually the quadratic decreasing in computing energy will become dominant.

6. EXPERIMENTAL RESULTS

In order to evaluate the performance of the GA based resource management scheme, a C++ based software program is constructed to emulate the environment monitoring sensor network. The cluster consists of 10 low cost and low quality sensor nodes and 100 tasks. The battery of each sensor has the capacity of 5000 Ampere·hour. The detection probability and the power consumption of each task are randomly generated. Different distributions with different variances are tested in the experiment. Furthermore, to emulate the behavior of the real sensor network which is deployed in a dynamic environment, the detection probability of the sensors is constantly changing. Every 1000 hours, for a set of x sensor nodes, their detection probability $prob_i$, $1 \leq i \leq 100$ will be regenerated and reapplied to model the change of their environment. The x is set to be 1, 2, and 5.

The environment setup is named by a quintuplet (*distribution*, *prob variance*, *power variance*, *biased/unbiased*, *x*). The first field specifies the type of distribution that is used to generate the detection probability and power consumption of each task. It can either be uniform distribution or normal distribution. The second and third filed specifies the variance of the detection probability and the power consumption respectively. The fourth field is either biased or unbiased. When an environment setup is biased, half of the sensor nodes have lower power consumption than the others. This field is designed to model a heterogeneous network. The final field specifies the number of sensors whose detection probability changes due to changes in the environment. Table 1 column 1 gives the list of environment setups that were tested in our experiments. Note that the variance of power consumption is different for the biased and unbiased environment.

Our distributed GA algorithm which is presented in section 4 is denoted as *GA-lifetime*, since its objective is to maximize the lifetime of the sensor network. The program is distributed on 5 processors ($np = 5$). The subpopulation size is set to 100 ($pop = 100$) and the number of generations in each epoch is 5 ($c = 5$).

We designed two algorithms to compare with the GA-lifetime. The first one is also a distributed GA whose objective is to minimize the total power consumption of the cluster. Therefore it is denoted as *GA-power*. Instead of using equation (1), the GA-power uses a fitness function as the following.

$$fitness = \begin{cases} 0 & \text{if } Prob < \text{Prob}_{\text{th}} \\ 1/\sum_{i \in \Delta_k} pow_{i,k} & \text{otherwise} \end{cases}.$$

The second one is a heuristic algorithm which selects and allocates task based on the power versus detection probability ratio. For each

task, it first selects the sensor node that has the highest power vs. detection probability ratio. Then it arranges the available tasks based on the descending order of this ratio. From the beginning of the list the algorithm selects the tasks one by one and assigns them to the sensor node, which is the most power efficient, until the overall detection probability of the cluster exceeds the user defined threshold $Prob_{th}$. In our experiment, the $Prob_{th}$ is set to 99.9%. The same threshold is applied to two other programs as well. We applied the above mentioned three resource management algorithms in the sensor network emulator. The lifetime of the network is recorded. The results are provided in Table 1. The first column specifies the environment setup and the last three columns specify the network lifetime (in hours) with different resource management algorithms.

Table 1 Network lifetime under different algorithms

ENVIRONMENT SETUP	GA-LIFETIME	GA-POWER	HEURISTIC
uniform, 8.3, 7.1, biased, 1	44752.22	41930.85	40066.12
uniform, 8.3, 7.1, biased, 2	42158.9	35224.52	35028.29
uniform, 8.3, 7.1, biased, 5	42186.89	33207.64	31848.58
uniform, 8.3, 8.3, unbiased, 1	23874.2	26462.19	26776.26
uniform, 8.3, 8.3, unbiased, 2	26828.16	31983.17	32294.36
uniform, 8.3, 8.3, unbiased, 5	27656.94	29365.77	29794.95
normal, 2, 5.5, biased, 1	480000	430909.1	450000
normal, 2, 5.5, biased, 2	511200.4	436666.7	450000
normal, 2, 5.5, biased, 5	507500	404285.7	416923.1
normal, 2, 2, unbiased, 1	14531.39	14218.28	10358.73
normal, 2, 2, unbiased, 2	13892.64	10881.77	9143.943
normal, 2, 2, unbiased, 5	15708.92	16131.24	14628.62
normal, 1, 1, unbiased, 1	2788.086	2224.231	2255.613
normal, 1, 1, unbiased, 2	2759.893	2597.652	2248.155
normal, 1, 1, unbiased, 5	2857.993	2647.201	2647.037
normal, 1.5, 4.2, biased, 1	43315.71	40533.04	34495.43
normal, 1.5, 4.2, biased, 2	49774.59	52759.19	47284.2
normal, 1.5, 4.2, biased, 5	50744.86	50134.21	48893.41

Figure 6 shows the percent lifetime improvement of GA-lifetime relative to the heuristic algorithm. We can see that the GA-lifetime generally works better than the heuristic algorithm. The average lifetime improvement is 14.4%. The only case for which the heuristic algorithm works better than the GA-lifetime is when the detection probability and power consumption of the tasks are distributed uniformly and the network is unbiased. This is because, in this environment setup, the detection probability and power consumption have significant variety. Therefore, there exist some task-processor pairs that are much more power efficient than others. A similar reason can be used to explain why the GA-lifetime works relatively better in the environment setup with normal distribution.

Figure 7 shows the comparison between the GA-lifetime and the GA-power. The average lifetime improvement of GA-lifetime over GA-power is 6.5%. This indicates that merely reducing the power consumption is not a good way to improve the network lifetime. If a sensor node has more remaining battery, it should be allocated with more tasks even though it is not the most power efficient node that can be used to process these tasks. In another word, to extend the network lifetime, it is more important to evenly distribute the tasks. We also observe that the GA-power outperforms the GA-lifetime when the environment setup is uniform and unbiased. This shows us that these two algorithms are complementary to each other, and they can be applied in different situations.

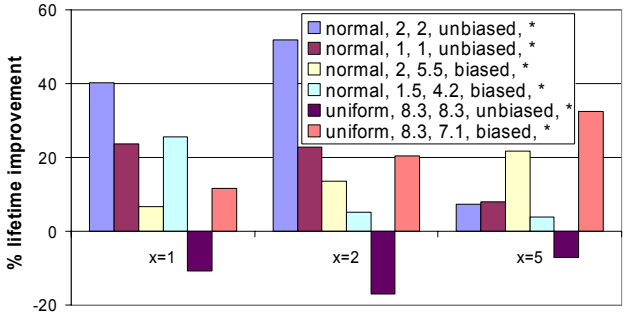


Figure 6 GA-lifetime vs. heuristic algorithm

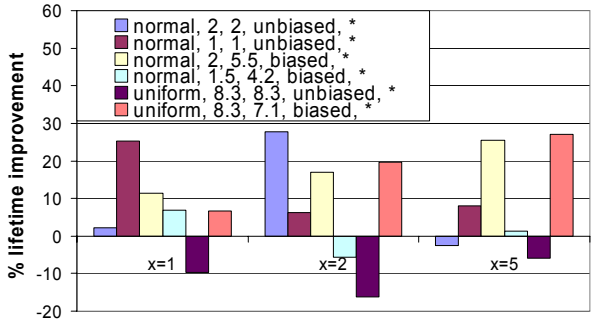


Figure 7 GA-lifetime vs. GA-power

7. CONCLUSIONS

In this paper we present a distributed GA algorithm that solves the resource management problem in a sensor network. A regression estimation model is presented that estimates the runtime of this algorithm. It is used to find the energy efficient configurations of the GA. The experimental results show that the proposed algorithm improves network lifetime by 14.4% in average.

8. REFERENCES

[1] ISTAG, "Ambient Intelligence: From Vision to Reality," Sept. 2003.

[2] I. F. Akyildiz, S. Weilian, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, Volume 40, Issue 8, pp. 102-114, Aug. 2002.

[3] E. R. Post and M. Orth, "Smart Fabric, or Wearable Computing," *Proc. First Int'l Symp. Wearable Computers*, pp. 167-168, Oct. 1997.

[4] H. Feltl and G. R. Raidl, "Evolutionary computation and optimization (ECO): An improved hybrid genetic algorithm for the generalized assignment problem," *Proceedings of the 2004 ACM symposium on Applied computing*, March 2004.

[5] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energyefficient communication protocol for wireless microsensor networks," *Proceeding of International Conference on System Sciences (HICSS)*, Jan. 2000.

[6] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Topology management for sensor networks: Exploiting latency and density," *Proceeding of International Symposium on Mobile Ad Hoc Networking and Computing*, 2002.

[7] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, Adaptive Resource Allocation for Sensor Networks," *Symposium on Networked Systems Design and Implementation*, May 2005.

[8] E. Cantu-Paz, "A Survey of Parallel Genetic Algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, Vol. 10, No. 2.

[9] M. Younis, K. Akkaya, and A. Kunjithapatham, "Optimization of task allocation in a cluster-based sensor network," *Proceedings of IEEE International Symposium on Computers and Communication*, 2003.

[10] http://smbus.org/.

[11] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. on Design Auto. of Elec. Systems*, Vol. 1, No. 1, pp. 3-56, 1996.

[12] E. Cantu-Paz, "Markov chain models of parallel genetic algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 4, Issue 3, pp 216-226, Sep. 2000.