Using Failure Models for Controlling Data Availability in Wireless Sensor Networks

Riccardo Crepaldi, Mirko Montanari, Indranil Gupta, Robin H. Kravets
{rcrepal2,mmontan2,indy,rhk}@illinois.edu
University of Illinois at Urbana–Champaign

***Abstract*— This paper presents Pirrus, a replica management system that addresses the problem of providing data availability on a wireless sensor network. Pirrus uses probabilistic failure models (e.g., derived from environmental conditions and estimation of available energy) to adaptively create and maintain a number of replicas of the data. Replica management is formulated as an energy optimization problem, then solved with a greedy heuristic that only uses information gathered from neighbors. Intuitively, Pirrus trades off the energy saved by limiting the number of replicas when the network health is good to extend the lifetime when more replicas are needed. Our simulation results show how the solution provided by Pirrus achieves good performance with a sustainable computational cost. Compared to the performance of a fixed number of replicas, Pirrus extends the network lifetime by more than 20%.**

I. Introduction

Traditionally, replica management systems have been designed to operate over relatively reliable machines, where the probability of failure of one machine is low enough to allow a small fixed number of replicas (e.g., two or three) to provide very high data availability. In such systems, the probability of failure of each machine or device is assumed to be constant in time. For example, the reliability of hard drives is provided by the manufacturer as a Mean Time Between Failures (MTBF) and is considered to be constant. However, this is only an approximation of the real failure behavior.

In sensor networks, these assumptions hold even less, since nodes can fail for many different reasons, such as depletion of the available battery energy, hardware defects in the node and harsh environmental conditions in which the node operates. Moreover, this failure rate is not constant: at the beginning of the network life-cycle, batteries are charged and nodes have just been tested, so the failure probability of a node is low. As the network continues, nodes consume energy and are more likely to run out of energy. Additionally, the presence of particularly harsh environmental conditions, such as very high or very low temperatures, can affect the failure rate of nodes.

To cope with this risk, one possible solution is to periodically send data to a base station with more reliable hardware, which can safely store data avoiding loss. However, the high energy cost of communication makes this solution unpractical. Instead, a solution where data is locally stored on the sensor nodes and collected using queries or data mules would provide better energy use and increase the network lifetime. However, the possibility of node failure puts data at risk, even when data retrieval is a reliable operation. This is because when a node loses its ability to communicate, either through a fault or battery depletion, the data it stores is lost. Therefore, replicating data on multiple nodes represents a method for reducing the possibility of data loss. Due to the limited resources in sensor networks, the main challenge is defining the number and placement of the replicas to support high data availability while maintaining long network lifetime.

Data replication has been proposed to prevent such data loss in distributed environments. Traditional replication approaches use a fixed number of replicas defined *a-priori* during network deployment [1]. Given the dynamic failure potential of the nodes in the network, these static solutions are either too expensive or cannot provide enough availability during the network lifetime. To cope with the variability in the network, replica management should be adaptive and adjust the number and location of replicas to provide the same data availability.

The contribution of our research is threefold. First, we formalize the problem of providing energy-efficient replica placement for data availability as an energy optimization problem. Given that the optimal solution requires global knowledge of network state, we next define a heuristic approach that only requires local information. Finally, we adapt the heuristic approach to design a novel distributed protocol, called Pirrus, which adopts a greedy approach and provides near-optimal results under varying conditions.

The key idea in Pirrus is that each node only uses local information about their n-hop neighbors to solve the problem of space availability. This local approach scales to large networks without suffering from excessive overhead or computational complexity. Since the environment has a strong influence on node reliability, Pirrus is designed to be flexible and allows users to define appropriate probabilistic failure models. In comparison to static approaches that do not consider the reliability of the nodes in the network, Pirrus adapts to changing network conditions. When data availability is a priority, results show that Pirrus is able to extend network lifetime compared to other approaches.

The rest of the paper presents the replica placement problem and describes our approach to solve it. Section II presents an overview of the problem of adaptive data replication. Section III formalizes the replica placement problem. Section IV describes the Pirrus protocol and its replica placement strategy. Section V presents an evaluation of Pirrus in comparison to other approaches.

II. Motivation and related work

Distributed databases are well-studied in wireless sensor networks. Systems such as COUGAR [2] and TinyDB [3]

provide a database-like view of a sensor network enabling data storage on nodes and retrieval of only the data that the user or the application is interested in. However, wireless nodes are unreliable devices. Factors such as hardware defects, energy depletion and considerations about the environment where the network is deployed affect the reliability of nodes. In a distributed database, node failures cause an irreversible loss of data. The goal of data storage systems is to provide reliable storage over an unreliable network. The reliability level of the storage is a parameter of the problem and is quantified as the probability p_{req} that at least one copy of the data is currently surviving.

The high unreliability of sensor nodes is likely to make the probability of successfully accessing a node and its data lower than the requested p_{req} over time. Replication can be used to provide data availability. Due to energy limitations, in the context of sensor networks the biggest challenge is the placement and management of replicas.

The factors that affect nodes' reliability can be captured by a probabilistic model of failure to estimate the reliability of a node. Failure models based on temperature, humidity and other working conditions have been defined for industrial equipment [4]. Even if, to the extent of our knowledge, none of these models have been defined for sensor nodes, similar reliability engineering techniques could be applied to sensor networks. Probabilistic reliability models could be obtained from theoretical considerations about the system or from experimental data.

Using these models, a probability of failure $p_j(t)$ can be defined for each node. These failure probabilities are assumed to be independent (i.e., the variations in the failure probability of a node do not affect the failure probability of the rest of the network). This probability, that can change over time, is abbreviated as p_j in the rest of this work.

Considering both availability requirements and the failure model, replica placement needs to satisfy the following condition: $\prod_j^K p_j < 1 - p_{req}$. If we relax the assumption of independence the approach is still valid, but the equation has to be adjusted.

Creating, maintaining and deleting replicas incur communication costs, which depend on replica location (i.e., the distance between the replica and the original data source). A replication strategy, to be efficient, needs to consider these costs when choosing replica locations.

Replica placement strategies have previously been proposed in sensor networks. A method for automatically caching frequently accessed data in intermediate nodes during routing was proposed by Yin and Cao [5]. The proposed approach allows nodes in the path to the final destination to answer queries using cached information, if it is recent. Tang et al. [6] propose a distributed algorithm for the optimal allocation of replicas in an ad-hoc network. This work analyzes replica placement as an optimization problem: each node needs to access some data contained in other nodes. Data closer to the requesting nodes is less expensive to obtain. Thus, the problem becomes finding the optimal placement of replicas to minimize the access cost and satisfy the constraints on the space available in each node.

Furthermore, the concept of Quality of Service for replication is introduced in Q-NiGHT [1]. This work enhances the Geographic Hash Table storage [7] by adding the concept of explicit QoS guarantees for each class of data. The QoS level is provided by the programmer as the number of required replicas. This approach requires to set a fixed number of replicas at the beginning of the network lifetime and does not adapt to varying conditions.

However, goals of these strategies are providing load balancing, energy savings and efficient data access. None of the previous work in this area optimizes replica placement to provide a constant level of data availability.

III. Replica Placement

The problem of replica placement has many angles that make its formulation a challenge. The goal of replica placement is to find a suitable set of nodes on which data replicas can be placed such that the availability requirements are met, while minimizing the energy spent by the system. Communication is one of the most expensive operations in WSNs, hence the optimization goal can be expressed as the minimization of the number of messages used to manage replicas.

This section introduces our modeling of the problem as the minimization of the number of messages exchanged. The formulation addresses the cost of creating, maintaining and removing replicas. In general, each node could have different costs for replica management. For the purpose of simplicity, the formulation considers only three different costs: creating, maintaining and removing a replica. However, our approach is able to handle different costs for each node.

In our system model, each node stores a set of environmental samples taken during a predefined period T. After an initial set up time, the size of the collected data does not change: new data replaces the oldest samples contained in the set. This model represents a system where environmental samples are collected continuously, but only the most recent ones need to be kept for analysis.

To obtain a model of the replica placement problem in this context, it is necessary to analyze the trade-off involved in replica creation and maintenance: each replica increases data availability, but at the same time consumes storage memory and requires periodic communication to be kept up-to-date. To capture the complexity of this trade-off, we present a model of the general replica placement problem. This first formulation assumes complete and omniscient knowledge of the network, which we will later relax in Section IV.

Replica placement can be modeled as a binary optimization problem. The number of messages exchanged for replica maintenance is the value to minimize, subject to constraints over data availability and storage space available on the nodes. The binary decision variables represent the presence or absence of a data replica on a particular node. Formally, if we consider a system with N nodes, the binary variables can be represented in a matrix X of size $N \times N$ where each variable x_{ij} is equal to 1 if j stores a replica for node i:

In the system model that we consider, each node stores a data set of size w_i, where i refers to the node that generated

and maintains that replica. Each node j has a limited amount of memory W_j in which it can store replicas and data.

The communication cost of creating a new replica can be expressed as the number of messages that are needed to send the data to the new replica location. For a single-hop network, this number is expressed as the constant k_a. When more than a single hop is needed to reach the new location, the total number of messages is proportional to the number of hops, indicated with $d(i,j)$, between the source node i and the destination node j.

Also, the problem needs to consider maintenance and deletion cost of replicas, since this operation also requires the exchange of messages between the peers. Let $\hat{X}$ be the matrix representing the current replica placement, whose elements $\hat{x}_{ij}$ are defined as for the matrix X. Let E and A be two matrices of size $N \times N$. The elements of E, e_{ij}, are binary variables representing the deletion of replica i from node j, while the elements of A, a_{ij}, represent the addition of a replica.

We also define two additional constants: k_m, the cost of maintaining an existing replica, and k_e, the cost of sending an invalidation command for an existing replica. These two costs are defined as the number of messages required to perform the replica management operations in each period T.

The problem can now be expressed as an optimization problem: the function to minimize is the total number of messages exchanged in the network, subject to the constraints of availability of each data and storage space on each node (shown in the same order in the equations). The problem can be expressed as following:

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N} (k_m \cdot (\hat{x}_{ij} - e_{ij}) + k_a \cdot a_{ij} + k_e \cdot e_{ij}) \cdot d(i,j) \quad (1)$$

$$\begin{array}{ll}
\sum_{j=1}^{N} ((\hat{x}_{ij} - e_{ij} + a_{ij}) \cdot \log p_j) < \log\left(1 - p_{req}^{(i)}\right) & \forall i \\
\sum_{i=1}^{N} (\hat{x}_{ij} - e_{ij} + a_{ij}) \cdot w_i < W_j & \forall j \\
a_{ij} \leq 1 - \hat{x}_{ij} & \forall i,j \\
e_{ij} \leq \hat{x}_{ij} & \forall i,j \\
a_{ij} + e_{ij} \leq 1 & \forall i,j \\
a_{ii} = 0, e_{ii} = 0, x_{ii} = 1 & \forall i.
\end{array}$$

The correct formulation of the optimization problem requires the definition of additional constraints between $a_{ij}, \hat{x}_{ij}$ and e_{ij} to restrict the feasible solutions of the problem only to valid solutions for our model. First (line 4), a replica can be added only if it was not present in the previous solution (i.e., it is not possible to add a replica twice on the same node). Similarly, it is not possible to remove replicas that were not present on the node (line 5). Another constraint has been added to avoid solutions that contain a replica that is both added and removed within the same optimization (line 6). Finally, it is not possible to add or eliminate replicas on the data source node.

To go toward the definition of a distributed solution, this global optimization problem can be partially rewritten as a set of individual minimization problems, one for each node i. In this local formulation, each node optimizes the number of messages that it sends to create and maintain its own replicas. Binary variables similar to the ones used in the global problem are defined: $\hat{x}_j$ represents the presence of a replica of the data on node j, while e_j and a_j represent the elimination and addition of a replica on the node.

These optimization problems are not completely independent: every time a replica is allocated on a node j, the space available to store other replicas is decreased. This fact is expressed by considering a set of variables $\hat{x}_j^{(k)}, e_j^{(k)}, a_j^{(k)}$ that represent the allocation and removal of replicas on node j by the other optimization problems.

The problem for a node i can now be expressed as follows:

$$\min \sum_{j=1}^{N} (k_m \cdot (\hat{x}_j - e_j) + k_a \cdot a_j + k_e \cdot e_j) \cdot d(i,j) \quad (2)$$

$$\begin{array}{ll}
\sum_{j=1}^{N} (\hat{x}_j - e_j + a_j) \log p_j < \log(1 - p_{req,i}) & \\
(\hat{x}_j - e_j + a_j) w_j < W_j - \sum_{k=1}^{N} (\hat{x}_j^{(k)} - e_j^{(k)} + a_j^{(k)}) w_k & \forall j \\
a_j \leq 1 - \hat{x}_j & \forall j \\
e_j \leq \hat{x}_j & \forall j \\
a_j + e_j \leq 1 & \forall j \\
a_i = 0, e_i = 0, x_i = 1 & .
\end{array}$$

In this formulation, the optimization function (line 1) is simplified to take into account only messages sent by a specific node. The availability constraint (line 2) maintains its previous structure. The storage space (line 3) constraints needs to consider less than W_j as space available for replica allocation: it needs to take into account the solutions to the other sub-problems (represented by $\hat{x}_j^{(k)}$, $e_j^{(k)}$, $a_j^{(k)}$) that allocated replicas on j. The rest of the constraints have the same purpose as the respective constraints in (1).

Optimal placement of replicas requires knowledge of the current state of the system. The available capacity for replicas on each destination node, and hence the optimal replica placement for each of the single sub-problems, depends on the solution of all other sub-problems in the network. Additionally, each node needs to know the failure probability of all other nodes in the system. The distributed nature of a sensor network and its dynamic conditions make the problem of maintaining an up-to-date view of the entire system at each of the nodes too expensive in terms of communication, and so not scalable. A local approximated solution to this problem can be defined using only local information.

Local replica placement

To obtain a distributed and scalable solution, it is necessary to limit the amount of information that each node requires to compute replica placement. Our first observation is that the cost of creating and maintaining replicas is proportional to their distance from the source node. If the failure probability is geographically uniform (i.e., there are no areas of the network that are less failure-prone than others), a set of replicas placed on local nodes is less expensive than a set of replicas created far from the data source. When the failure probability is not uniform, a set of replicas placed on nodes far from the data source could represent a better solution to the problem. However, disseminating up-to-date information

about failure probability of distant nodes results in significant overhead in sensor networks. Investigating a solution where only information collected from neighbor nodes is used to compute a solution represents an interesting approach to limit the network overhead and, at the same time, to provide enough information to compute a feasible replica placement.

Another problem of optimal replica placement is its computational complexity. Even if the optimal solution can be computed by solving the separate sub-problems, the interaction between them does not change the overall complexity: to obtain a scalable solution, it is necessary to limit the interaction between nodes. An efficient heuristic replica placement approach can consider the available space in each node j to be only the space not previously allocated to other replicas. This solution will not provide an optimal replica placement, however it greatly reduces the communication between nodes.

The formulation of this heuristic is similar to the formulation of the optimal problem (2). However, the constraint over the available space on each node considers only $W_{rem,j}$, the space not assigned to other replicas on node j. The constraint can be rewritten as $(\hat{x}_j + a_j - e_j)w_j < W_{rem,j}$ for all values of j.

The computation of this heuristic placement requires solving a binary optimization problem. However, running a binary optimization algorithm in a wireless node is too computationally expensive. For this reason, the next section introduces a greedy solution implementable on a sensor node.

IV. PIRRUS

Pirrus is a distributed approach for replica placement. In this approach, replicas are placed according to the variable probability of failure of nodes. The system continuously monitors the state of the replicas and reacts to failures or to decreasing availability. This approach requires a small cost for replica maintenance when the network is reliable, while providing availability even in adverse conditions. Pirrus is designed to be a framework where the user can embed a model of failure formulated for the specific deployment. The use of a more precise model results in a more effective protocol behavior.

The new storage space constraint defined in the local replica placement problem specifies a set of nodes where enough space for the placement of a new replica is available. Considering only the nodes that satisfy this constraint, the replica placement problem becomes similar to a knapsack problem [8]: minimization of the number of messages subject to the constraint of availability requirements (representing the capacity of the knapsack problem).

For this reason, it is possible to use a well-known greedy strategy to provide a feasible solution to the replica placement problem. When expressed according to our problem formulation, the greedy strategy involves ordering the nodes with enough storage space according to the ratio between the gain that placing a replica on node j would provide, and its cost. The gain is expressed as the coefficient in the availability constraint, $\log p_j$, while the cost is expressed as the number of hops $d(i,j)$, thus $\frac{\log p_j}{d(i,j)}$.

Pirrus is defined by the following set of actions:

1) Collect $< W_{rem,j}, p_j >$ about nodes within h hops
2) Sort in descending order a list L of nodes with enough storage space according to the above greedy strategy.
3) Select the first q nodes from L such that:
   $\prod_{j=1}^{q} p_{L(j)} < 1 - p_{req}$
   where $p_{L(j)}$ is the probability of failure associated with the j-th element of the list.
4) Send requests for the allocation of replicas to the selected nodes
5) After receiving the allocation requests, allocate the replica on node j. If multiple requests are received, accept only the first one received.
6) If the allocation request has been dropped, choose a different node and start again from step 1.

The distribution of information about $< W_{rem,j}, p_j >$ in each node can be implemented through the use of periodic broadcast heartbeat messages. These messages are forwarded for a limited number of hops d_{max}. When new replicas need to be created, data transfer is performed using a Selective Repeat ARQ technique. Unresponsive nodes are removed from the list of nodes considered by the system.

V. EVALUATION

The main goal of our evaluation is to demonstrate the efficiency of the greedy approach of Pirrus in solving the replica placement problem. We investigated the difference in terms of energy cost between Pirrus and the local replica placement, obtaining similar performance. We also performed simulations in MATLAB varying the network size. The total cost of the solution grows linearly with the number of nodes, providing evidence of the scalability of our Pirrus [1].

Furthermore, we analyze the performance of Pirrus, focusing on its ability to extend the lifetime of the network, compared to other non-adaptive solutions implementable on WSNs. We consider the network lifetime to be expired when at least one node is not able to satisfy the availability requests. For this analysis we focused on the evolution of the number of replicas managed at each moment and the relative energy cost. The results highlight how the adaptive behavior of Pirrus can save energy in the initial phase of network lifetime, and use this energy later when the network becomes less reliable.

We implemented Pirrus in *ns-2*. We compared its greedy solution with other strategies that can be used to provide data availability: sending data to a base station connected to a reliable storage or providing and maintaining a fixed number of replicas for the whole network lifetime.

The network used in the simulations is composed of 250 nodes placed on a regular grid of size 5200 meters. The channel access is performed using a simple CSMA scheme, obtained from the 802.11 MAC implementation. The RTS/CTS mechanism is disabled, the data rate is 400 Kbits/s and the maximum packet size is 500 bytes.The network uses the AODV reactive routing protocol.

The nodes consume 0.0035 mW in idle, 0.06 mW in transmission and 0.03 mW in reception. The amount of energy stored in the battery in the beginning is a random variable uniformly distributed between 1875 J and 2500 J. The simulation collected data for a week of simulated time.

[1]Results are not included for space constraint.

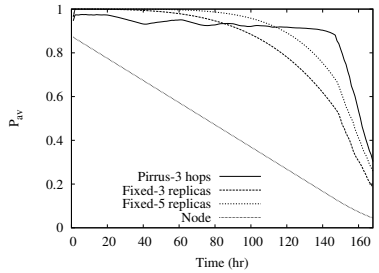

Fig. 1. Average data availability over time

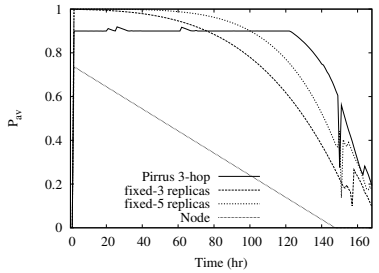

Fig. 2. Minimum value of data availability for each hour.

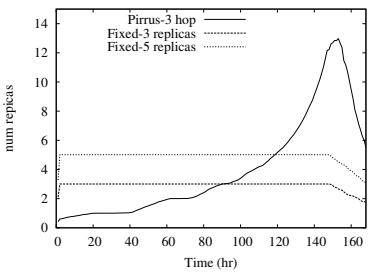

Fig. 3. Average number of replicas over time.

Each replica in the system has a size of 80 Kbytes, storing enough samples for three days of information. Each replica update has a size of 1 Kbyte. An update is sent hourly, while heartbeats are broadcast every 300 seconds.

We implemented a very simple failure model to run our simulations. The node availability is proportional to the remaining energy of the battery, capturing the unreliability of the estimation of this value and of the energy consumption of the application running on the nodes. To address all aspects that could affect the nodes' availability, a more complex model has to be defined. This is not the goal of this paper. However a new model can be embedded in Pirrus to enhance its ability to provide data availability.

In the first set of simulations, we compared the data availability provided by Pirrus with the data availability provided by a system that creates a fixed number of replicas. The replica placement strategy that we compare to is presented in [1], and creates a fixed number of replicas on the nodes located nearby the nodes that originate the data to replicate.

In these simulations, the fixed replica system selects the replicas among neighbors within a distance of three hops, starting from the closest ones that have available storage. As in Pirrus, this system uses heartbeat messages to verify that nodes on which replicas are allocated are still alive.

Our simulations compare the average data availability provided by the various approaches (see Figure 1). Additionally, we also analyze the minimum value for the availability in the network at each time tick (Figure 2). The results highlight how, for both the metrics, Pirrus outperforms the non-adaptive solutions. In particular, our definition of lifetime considers the network dead when data availability drops below a threshold (fixed to 90% in these simulations) for at least one dataset. Looking at the graph presenting the minimum value of availability, it is clear that Pirrus extends the network lifetime almost 50% compared to the 3-replica solution, and more than 20% with respect to the 5-replica solution.

To investigate the source of this improvement, we investigate how the number of replicas changes with the evolving conditions of the network (Figure 3). At the beginning of the network lifetime, node availability is high. Most nodes can satisfy the availability requests without creating replicas. Only a few nodes require some replicas and this results in an average number of replicas that is less than one. However, from the very first moment, the other approaches create a fixed number of replicas that they maintain as long as the energy left in the batteries is enough to provide connectivity. This results in availability that is higher than requested, for both the average and the minimum values (see again Figure 1 or Figure 2). As time progresses, the average node availability starts decreasing and so does data availability. Pirrus reacts by increasing the number of replicas, keeping the minimum data availability constant. The system fails after 120 hours when the minimum value of availability goes below the threshold value. This happens 20 hours after the 5-replica approach and 40 hours after the 3-replica one.

Creating and maintaining a fixed number of replicas costs energy from the beginning. The energy spent grows linearly with time. Pirrus instead creates replicas only when needed, requiring less communication and reducing the energy spent. This savings can be used later when more replicas are required to provide availability.

VI. Conclusions

The approach presented in this paper offers many directions for further research. The failure estimation method that we used in our simulations takes into account only a single cause of failure: the available energy. More complex estimation techniques can be used to take into account environmental conditions or information about the failure of neighbor nodes. Additionally a combination of Pirrus with a system designed to provide efficient data access would be an interesting evolution.

References

[1] M. Albano, S. Chessa, F. Nidito, and S. Pelagatti, "Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks," *Mobile Data Management, 2007 International Conference on*, pp. 166–173, 2007.

[2] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 46–55, 2004.

[3] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[4] C. Ebeling, *An introduction to reliability and maintainability engineering*. McGraw Hill, 1997.

[5] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 1, pp. 77–89, Jan. 2006.

[6] B. Tang, H. Gupta, and S. R. Das, "Benefit-based data caching in ad hoc networks," *Mobile Computing, IEEE Transactions on*, vol. 7, no. 3, pp. 289–304, March 2008.

[7] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table," *Mobile Networks and Applications*, vol. 8, no. 4, pp. 427–442, 2003.

[8] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*, 1990.