Decentralized Coding Algorithm in Data Centric Storage for Wireless Sensor Networks

Khandakar Ahmed
RMIT University
Melbourne, Australia
khandakar.ahmed@rmit.edu.au

Mark A. Gregory
RMIT University
Melbourne, Australia
mark.gregory@rmit.edu.au

Abstract—For robust data recovery in catastrophic scenarios in wireless sensor networks, obtaining reliable storage over unreliable nodes is desirable. A distributed Data Centric Storage (DCS) system provides reliable access to data through redundancy spread over individually unreliable nodes. However, instead of using simple replication, it is possible to achieve a similar level of reliability with less redundancy by utilizing erasure encoding. In this paper, a Decentralized Distributed Erasure Coding (DDEC) algorithm is implemented in a Disk Based Data Centric Storage (DBDSC) architecture dividing the rectangular field into a matrix of storage cells (referred to as sectors) where row and column represent track (T) and sector (S) respectively. The acquired information is encoded into n fragments and disseminated across n nodes inside a sector so that the original source packets can be recovered from any k surviving nodes. The performance of DDEC was analyzed and found to provide an improvement over comparable approaches.

Keywords—Wireless Sensor Network; Data Centric Storage; regenerating codes; sector based distance routing; repair bandwidth

I. INTRODUCTION

There are three orthodox methods of data storage and retrieval in Wireless Sensor Networks (WSN) — Local Storage (LS), External Storage (ES) and Data Centric Storage (DCS) [1]. In DCS, data is reliably stored over long periods using a distributed collection of storage nodes, which may be individually unreliable. However, erasure encoding can achieve improved or similar reliability utilizing partial redundancy rather than replication of the data.

According to classical encoding theory, a data packet consisting of k pieces can be encoded into n (n = k+m) coded fragments using an (n, k) maximum distance separable (MDS) code and stored in n nodes. The original packet, then, can be recovered from any set of k coded fragments stored in k nodes. The complexity of encoding, updating and decoding of classical erasure encoding is \(O(mn), O(m)\) and \(O(mn)\) respectively. However, classical MDS erasure codes are optimal in terms of the redundancy-reliability tradeoff i.e. they maximize the error tolerance for minimum redundancy. The most well known family of MDS erasure codes includes Reed Solomon Codes (RS), which work for any value of k and n. However, n-way dot product for each coding device and Galois Field (GF) multiplication operations slow down MDS RS performance. One of the most popular variations of RS is Cauchy Reed Solomon Codes [2]. Another line of research for MDS erasure coding in storage applications is parity array codes. This type of encoding solely depends on XOR operations with the objective of achieving lower encoding, decoding and updating complexity. Popular variants of parity array codes includes EVENODD [3], X-code [4], STAR [5], WEAVER [6], HOVER [7], BLAUM-Roth [8] and others. In contrast to MDS erasure encoding, the distinctly non-MDS Low Density Parity Check (LDPC) [9] codes perform far better than optimal MDS. LDPC depends solely on parity checking. Despite having increased complexity, LDPC is easy to implement in practice. A widely used variation of LDPC is the Luby-Transform (LT) code. Table I and Fig. 1 show the performance comparison among different classical erasure encoding approaches.

In contrast to the ordinary erasure codes, another family of codes named Fountain codes are rateless codes and have been of increasing interest to the research community. In Fountain codes [10], encoder samples \(\rho(k)\) for a value \(1 \leq \rho(k) \leq k\), at the \(t^{th}\) step, where \(B = \{b_1, b_2, \ldots, b_n\}\) and \(\rho(k)\) denote a set of k native data blocks and a probability distribution respectively. \(d_i\) random data blocks are uniformly selected by the encoder from the set B. The blocks are then XORed in encoder using \(F_2\) generating an encoded set \(e_i\). The encoder then appends a \(k\)-dimensional binary encoding vector \(G_i = \{g_1, g_2, \ldots, g_k\}\) to \(e_i\). It then sets every entry \(g_j\), which is referred to as encoding coefficient, of \(G_i\) to 1 if \(b_j\) was used to construct \(e_i\) and 0 otherwise. Given, a set of encoded blocks and encoding vectors denoted by \(E = \{e_1, e_2, \ldots, e_k\}\) and \(G_i\) respectively, the decoder on the receiving side keeps on receiving encoded blocks until the system of linear equations \(E_{G_i} = B_{G_i}\) \(G_{i}\) is solved for \(B_{i}\). In general, \(I + \epsilon\) k encoded blocks are required by the decoder to recover all native data blocks, where \(\epsilon > 0\).

Coding theories generally focus on the tradeoff between redundancy and error tolerance. Redundancy must be continually invigorated as nodes fail or leave the region of interest. This rejuvenation involves large data transfers across the network. Being motivated by the approach taken in [11], this paper proposes DDEC, also referred to as minimum-bandwidth regenerating (MBR) code, which draws
a fundamental tradeoff between storage and repair bandwidth. In contrast to MDS, also referred to Minimum-Storage Regenerating (MSR) codes, MBR minimizes bandwidth by having fragments slightly bigger in size. In this paper, DDEC, designed based on a product-matrix MBR code construction technique [12], is implemented in a Disk Based Data Centric Storage (DBDCS) architecture [13-14], where a rectangular network is divided into $m$ tracks (rows) and $n$ sectors (columns). The smallest storage unit of the deployed network is the sector, which is considered to be a system unit. Hence, the nodes in the network are divided into $S$ ($mn$) sectors. Each sector comprises a Sector Head (SH) and sector members. Member nodes communicate via one hop to the SH. An event (file of size $O$), which is mapped to a sector to store, is divided into $k$ pieces (of size $O/k$), which are then encoded into $n$ fragments. The encoded fragments are stored in $n$ nodes within the corresponding network sector. Then, the original event can be recovered by the SH from any set of $k$ coded fragments retrieved from $k$ nodes.

In this paper, an MBR code called DDEC is implemented in a DBDCS Data Centric Storage Architecture. The performance of DDEC is evaluated using traces of failures and different ($n, k$) configurations for different measurement matrices.

### Table I. Performance Comparison of Erasure Encodings

<table>
<thead>
<tr>
<th>Performance</th>
<th>Read Solomon</th>
<th>Hover</th>
<th>Weaver</th>
<th>LDPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Capacity (% of memory occupied by data)</td>
<td>80%</td>
<td>76%</td>
<td>50%</td>
<td>80%</td>
</tr>
<tr>
<td>Number of Failures Tolerated</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3.81</td>
</tr>
<tr>
<td>Coding Operations (XORs)</td>
<td>44.6/CW</td>
<td>12.2/CW</td>
<td>3/CW</td>
<td>30/4/CW</td>
</tr>
<tr>
<td>Encoding Time (sec)</td>
<td>59.5</td>
<td>18.73</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Decoding (sec) per failed device</td>
<td>14.9</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Update Operations (XORs) per Updated Node</td>
<td>12.4</td>
<td>5</td>
<td>5</td>
<td>3.53</td>
</tr>
</tbody>
</table>

*20 Storage Devices Resilient to 4 Failures, CW: Coding Word*

The remainder of this paper is organized as follows: Section II provides an overview of the related work in the literature. Section III describes the system model that includes network architecture and coding model. This is followed by the simulation results and DDEC performance evaluation presented in Section IV. The paper is concluded in Section V.

### II. RELATED WORK

In [15], the Redundant Residue Number System (RRNS) is used as a distributed erasure code and employed in the two Data Centric Storage approaches: DCS-GHT [16] and Q-NIGHT [17]. RRNS also defines a probabilistic approach to evaluate the probability of correct data encoding and decoding on the parameters of the erasure code, WSN configuration, number of expected faults and the way in which the erasure code keys are distributed to the sensors. However, the proposed model evaluation was carried using a probabilistic model and Monte Carlo based simulation where there was insufficient analysis of how the model might apply to implementable systems.

A decentralized Fountain code based algorithm is presented by Kong, et al. [18] for distributed storage in WSN where the $k$ source packets are encoded and stored in $n$ storage nodes so that the original source packets ($k$) can be recovered from any set of $k(1+\epsilon)$ nodes where $\epsilon > 0$. In contrast to all previously developed schemes, the authors rationalized their algorithm as truly distributed since nodes do not know $n, k$ or network connectivity except in their neighborhood. In addition, the nodes do not maintain a routing table. However, the network model turns out to be expensive due to the inflated ratio between storage node and sensor node (9:1). Moreover, during the decoding process the requisite number of packets that need to be collected to recover the original source packets is not limited to any explicit figure which results in variable bandwidth during recovery operations.

The code efficiency and low complexity of Fountain codes inspired Dimakis et al. [19] to propose a decentralized realization of Fountain codes using fast random walks to disseminate source data packets to the storage nodes and geographic routing over a grid with the assumption that every node knows its own location. Another solution employing random walks with stops and the Metropolis Algorithm specifying transition probabilities to the random walks is proposed by Lin et al. [20].

However, despite having low complexity encoding and decoding, the Robust Soliton Distribution [21] that has been considered in [18-20] is constrained to the initial distribution of data. Nevertheless, the random walk that has been used requires excessive energy to implement in WSNs where limiting energy consumption is crucial. Moreover, random walk protocols require a large number of nodes to converge to the essential distribution.
III. SYSTEM MODEL

A. Network Architecture

The structural design of a platter of a magnetic disk storage provides an interesting approach that has been applied to a rectangular sensing area. Based on this assumption an architecture referred to Disk Based Data Centric Storage (DBDCS) is proposed. DBDCS divided the rectangular field into a matrix of storage cells, referred to as a Sector, each comprising a SH and member nodes (Fig. 2(a)). Here, rows and columns are denoted by tracks ($T_j$) and sectors ($S_j$) respectively. Communication from member nodes to SH, referred to as the intra-sector communication, is restricted to one hop limiting the transmission range within the sector. On the contrary, the inter-sector communication via SHs is multi-hop. In Figure 2(b), the sensor nodes inside each sector are not shown explicitly. In lieu, the total traffic from member nodes to head node is represented by an aggregated link (see Fig. 2(c)). It is also assumed that at the time of deployment, each node is configured to be acquainted of the deployment layout by knowing: (1) All member nodes know their own node id and (2) All SHs are assigned with the sector number as a virtual address and node id.

It is assumed that a unique hashing algorithm is employed based on an event attribute value and weight matrix that is assigned to different attributes according to the importance in the event description. A one-dimensional domain of derived hash keys is then mapped to a two-dimensional metric space of the DBDCS network using a generalized and adapted variant of $iDistance$ and hence the event generating the common hash key is stored in the same sector. Given a $j^{th}$ sector containing $\eta_j$ member nodes, the member nodes are sorted in ascending order by $SH_j$ based on Received Signal Strength Indicator (RSSI). The member nodes are then divided into $r$ segments. Each segment forms a ball, denoted by $B_{(X,Y)}(r_i)$, with the ball centred at $(X,Y)$ and radius $r_i$. The number of balls or segments depends on the WSN application, the size of a sector and the number of member nodes in each sector.

B. Coding Models

1) Information Flow Graph

Given a $j^{th}$ sector, a directed acyclic graph is considered to show the information flow between SH and a set of storage nodes (denoted by $\eta_j$, where $|\eta_j| = n$), which is a subset of member nodes (i.e. $\eta_j \subseteq M_j$ and $M_j = \{m_1, m_2, m_3, \ldots, m_n\}$).

Hence, the graph can be denoted by $G$ consisting of three kinds of nodes: a Data Source ($SH_j$), storage nodes ($\eta_j$), and a data collector ($SH_i$). The $SH_i$ divides an object $O$ into $k$ initial fragments of size $O/k$. Each $k$ fragment is then encoded into $n$ encoded fragments of size $\theta$, where $\theta$ consists of $\alpha$ symbols and is slightly greater than $O/k$. The fragments are then forwarded to $\eta_j$. This creates a virtual link between $SH_i$ and active storage nodes denoted by $SH_i \rightarrow \eta_j$. During this step, the storage nodes are active and they represent a distributed erasure code, corresponding to the desired state of the system. If a node $m_{i\alpha} (N = \psi_j + l, \text{where } l > 0)$ joins this sector, it can only be connected to the active storage nodes creating a directed edge from active storage nodes to new node denoted by $\eta_j \rightarrow m_{i\alpha}$. If a node leaves the sector, the surviving nodes communicate a packet of size $\mu$ with the new node producing a new erasure code to be stored in $m_{i\mu}$. During the repair process, it might be necessary for nodes to communicate more data than they store (i.e. $d_{\mu} > a$, where $d$ is the number of surviving nodes). Finally, a data collector ($SH_i$), which corresponds to a request to reconstruct the data, is connected to the subset of active nodes through directed edges denoted by $\beta \rightarrow SH_i$, where $\beta \subset \eta_j$.

2) Code Construction

In this paper, the product-matrix MBR code construction technique [12] has been used to generate a codeword. Under this framework, $C$ is an $(n \times \alpha)$ code matrix which is the product of an $(n \times d)$ encoding matrix and an $(d \times \alpha)$ message matrix denoted by $P$ and $O$ respectively.

$$C = PO$$

(1)
The encoded message that will be stored in $i^{th}$ node can be given by:

$$c_i = \phi_i O$$  \hspace{1cm} (2)

In (2), the $i^{th}$ row of $P$, denoted by $\phi_i$, refers to the encoding vector for node $i$.

In (2), the $i^{th}$ row of $P$, denoted by $\phi_i$, refers to the encoding vector for node $i$.

With the assumption that it is possible to construct an $[n, k, d]$ MBR code for $\mu = 1$ then the possibility of construction of an MBR code for larger value of $\mu$ is accepted. Hence, in order to have an MBR code with $\mu = 1$, the MBR code should hold data-reconstruction and exact-regeneration properties. The code should satisfy (3).

$$\alpha = d, B = kd - \left(\frac{k}{2}\right)$$  \hspace{1cm} (3)

Hence, the message matrix $O$ can be defined as the $(d \times d)$ symmetric matrix, given by:

$$O = \begin{bmatrix} S & T \\ T^T & 0 \end{bmatrix}$$  \hspace{1cm} (4)

In (4), $S$ is a $(k \times k)$ matrix. The entries of the upper-triangular region of $S$ are $\left\{ \alpha \right\}_{j=1}^{k}$ distinct messages from the $\left\{ \alpha \right\}_{k}$ set. The $(k × (d-k))$ of (4) is filled up with the remaining $(d-k)$ message symbols. The $P$ in (1) can be either a Vandermonde or a Cauchy matrix.

The data collector (SH) can recover all the $B$ message symbols connecting to any of the $k$ nodes from $P_{DC}O$. $P_{DC}$ is the sub-matrix of $P$ consisting of the $k$ rows taken from $k$ nodes to which the SH connects. $P_{DC}O$ can be expanded as:

$$P_{DC} = [\Pi_{DC} + \Delta_{DC}]$$  \hspace{1cm} (5.1)

$$P_{DC}O = [\Pi_{DC}S + \Delta_{DC}T^T]/\Pi_{DC}T]$$  \hspace{1cm} (5.2)

Hence, first $T$ and then subsequently $S$ can be recovered from $P_{DC}O$ matrix by multiplying it with $\Pi_{DC}^{-1}$.

In the repair process, the code of a failed node can be restored exactly by:

$$[\phi_i O][1 \ldots 1]^T]\Pi_{DC}^{-1} = P_{DC}^{-1}$$  \hspace{1cm} (6)

In (6), $P_{DC}^{-1}$ is the sub-matrix of $P$ consisting of $d$ rows for $d$ storage nodes referred to as helper or surviving nodes. Here, $i$ denotes the index of a node from an arbitrary set of $d$ helper nodes \{h1, h2, ..., h4\} to which SH1 connects.

### IV. Evaluation

Several simulations have been performed using Castalia v3.2 [22] in order to verify DDEC and to get a better understanding on how well the proposed solution works. The system parameters and their settings that have been used in the experiments are presented in Table II. With varying-channel affecting seeds, simulations were run 30~40 times to provide results that included average and 95% confidence interval. The model network (presented in Section III-A) was tested in a 150x150 m² rectangular field consisting 64 sectors (8x8 grid). Each SH is considered as both source and data collector while member nodes inside a sector are considered as the super set of storage nodes. A sector can have a maximum of 16 member nodes and thus the value of $n$ is varied up to 16 i.e. a $k$-fragmented packet is encoded into a maximum of 16 encoded fragments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Size</td>
<td>150x150 m²</td>
</tr>
<tr>
<td>Number of Nodes (member nodes, SH)</td>
<td>1088 (1024, 64)</td>
</tr>
<tr>
<td>Member Node Density ($\rho_m$)</td>
<td>1 node / 21.97 m²</td>
</tr>
<tr>
<td>Sector Head Node (SH) Density ($\rho_{SH}$)</td>
<td>1 node / 351.56 m²</td>
</tr>
<tr>
<td>Radio Range (member node), Radio Range (SH)</td>
<td>~8 m, ~20 m</td>
</tr>
<tr>
<td>Transmission Power</td>
<td>0 dBm (SH), -5 dBm (member node)</td>
</tr>
<tr>
<td>Power Consumption in Sending and Receiving Messages</td>
<td>57.42 mW (SH), 46.2 mW (member node)</td>
</tr>
<tr>
<td>Power Consumption Per Sensing</td>
<td>0.02 mJ/s</td>
</tr>
<tr>
<td>Modulation Type, Data Rate, Bits Per Symbol, Bandwidth, Noise</td>
<td>PSK, 250 Kbps, 4, 20 MHz, 194 MHz, -95 dBm, -100 dBm</td>
</tr>
<tr>
<td>Reference Distance ($d_0$), pathLossExponent, Gaussian Zero-Mean Random Variable ($\chi_0$)</td>
<td>55, 1.0 m, 2.4, 4.0</td>
</tr>
<tr>
<td>MAC Protocol, Routing Protocol</td>
<td>SMAC [23], SBD [13]</td>
</tr>
<tr>
<td>SMAC Acknowledgment, Synchronization, RTS, CTS Packet Size</td>
<td>11, 11, 13, 13 bytes</td>
</tr>
</tbody>
</table>

### A. Storage Performance

In this section, two experiments were conducted to evaluate the storage efficiency of DDEC. In both experiments, DDEC is compared against 3-Way Replication as it is one of the most popularly used flat replication approaches. In the first...
experiment (Figure 4), the simulation was run for 20 rounds where data production rate was varied between 1–25 packets/sector/round. SBD operation [13] was divided into rounds that included a learning phase followed by a relaying phase. Figure 4 shows that DDEC consumed almost 50% less resources to maintain a similar or even higher magnitude of redundancy. In the second experiment, the simulation was run for five different \((n, k)\) configurations to observe how storage overhead varies. For each configuration, the simulation was run for 20 rounds and the data production rate was 10 Packet/Sector/Round. For this experiment, the cost of storing one byte is defined as the metric of how many bytes are stored for each useful byte. The 3-Way Replication is used as the baseline for comparison in this experiment. The results are presented in Figure 5. It shows that the normalized cost of DDEC decreases as \(n\) grows when \(n-k\) is fixed. From both Figures 4 and 5, it can be concluded that DDEC needs approximately half the storage of 3-Way Replication.

B. Reliability

In this Section, three experiments were conducted to analyze DDEC data reliability. The experiments were conducted for 20 rounds with a \((16, 12)\) configuration. An event namely \(\text{fail}_\text{event}\) was implemented in each node. The \(\text{fail}_\text{event}\) was scheduled to trigger at different nodes at different times. The \(\text{fail}_\text{event}\) was carefully scheduled in the nodes to ensure a uniform distribution of failed nodes among sectors over time. In each round, the same 100 queries were executed where the result set of the queries was distributed throughout the network. Figure 6 shows the percentage of message decoded in function of number of nodes failed. From Figure 6, it is seen that the percentage of messages decoded was steady until the average number of surviving nodes for each sector was greater than \(k\).
Replication as a base-line. Note that the bandwidth when using 3-way replication is constant across different \((n, k)\) since there is no dependency on these parameters. The bandwidth performance of DDEC was lower than 3-Way Replication. It is not surprising due to the amount of data required to be communicated to recover data from the lost node.

![Normalized Repair Bandwidth](image)

Fig. 9. Normalized Repair Bandwidth

V. CONCLUSION AND FUTURE WORK

This paper presents an MBR code referred to as DDEC in a distributed DCS WSN. The paper shows the use of DDEC in a DCS layout denoted by DBDCS. Simulation was done in Castalia. Modifications to the Castalia models allowed the new decentralized coding algorithm to be modeled. Further, DDEC was evaluated against 3-Way Replication using different performance metrics. The results highlight that the use of erasure encoding in network storage can provide the desired level of data availability at a smaller memory overhead when compared to replication. Research into a more robust implementation of DDEC continues. Using the Texas Instruments’ (TI) CC2530 Evolution Module (CC2530EM) [24], which is ZigBee/IEEE 802.15.4 compliant System-on-Chip with an optimized 8051 MCU core and radio for the 2.4 GHz unlicensed ISM/SRD band, a prototype of DDEC will be implemented in future.

REFERENCES