In Proc. IEEE CloudNet 2016

Energy-efficient and Fault-tolerant Mobile Cloud Storage

Chien-An Chen, Radu Stoleru, Geoffrey G. Xie[†]
Department of Computer Science and Engineering, Texas A&M University
[†]Department of Computer Science, Naval Postgraduate School
{jaychen, stoleru}@cse.tamu.edu, xie@nps.edu

***Abstract*—Mobile Cloud Storage (MCS) systems – cloud storage on mobile devices without access to remote data center-type cloud resources, are not only interesting from a theoretical point of view, as they pose the most challenging design settings, but also important in enabling real-world applications such as disaster relief, military operation, and mining in remote areas. Central to MCS design is how to minimize the energy consumption of the battery-powered devices while still maintaining the data reliability and availability. Unfortunately, existing solutions do not model the energy-efficiency and data reliability of MCS in an integrated manner. Their formulations predominantly make use of heuristics, which may over-emphasize energy efficiency and not provide sufficient data reliability for some applications. In this paper, we design an energy-efficient *distributed data storage framework* in MCS under explicit data reliability requirement. The novel formulations produce a reliability-compliant and energy-efficient MCS system. The performance characteristics of our solutions are extensively evaluated through both real-world and synthetic mobility traces.**

I. Introduction

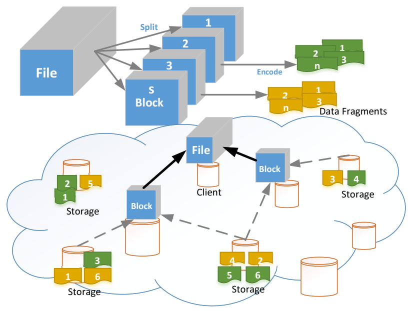
Recent advances in the design and deployment of mobile cloud systems tap into the increasingly abundant storage, processing, and communication resources on smart devices. As a result, mobile cloud computing is becoming a reality [1]. In particular, a class of mobile cloud systems consisting entirely of intermittently connected mobile devices has been conceived and prototyped by researchers [2] [3] [4] [5]. Such mobile cloud storage (MCS) systems are not only interesting from a theoretical point of view, as they pose the most *challenging design settings*, but also important in enabling real-world applications. These applications include both military tactical cloud systems [6] [7] (there is currently an effort by the U.S. military to equip soldiers with smartphones [6]) and civilian use cases such as disaster relief and mining operations in remote areas, where a networking infrastructure does not exist or is congested [1] [8].

MCSs share the same performance objectives with the traditional distributed computing and cloud systems. Prior works identified two major performance concerns unique to MCSs; one is how to deal with the frequent and unpredictable changes in network condition (e.g., network topology and node's availability) and the other is how to minimize the energy consumption of the battery-powered devices while still meeting the performance requirements of applications [9]. However, these studies, particularly those tackling the energy efficiency problem, do not systematically model the reliability of the data. As such, their formulations are mostly based on heuristics derived from specialized network settings, which may over-emphasize the energy efficiency and not provide sufficient protection of the stored data. This is especially problematic in ad-hoc network settings where a drastic and unexpected change in the network topology or nodes' availabilities could greatly impair the performance of applications.

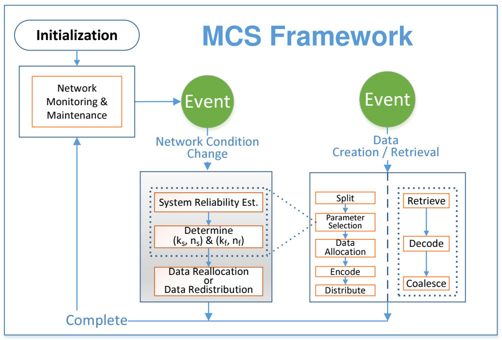
In this paper, we formulate the problem of designing an energy-efficient distributed data storage framework in MCS under explicit reliability requirement. Specifically, the framework realizes a distributed file system in a set of mobile devices. *Reliability* here indicates the probability that a stored data object can be successfully recovered within a predefined time period T_s. We seek fundamental insights towards a systematic methodology for customizing the design to meet a given mix of reliability and energy requirements. We then investigate how each requirement may introduce tradeoffs between application performance and energy efficiency.

We study the k-out-of-n storage system, a more general framework that can describe most of the existing distributed storage systems, e.g., the 1-out-of-n employed by HDFS [11]. Each data object is encoded into n fragments, and the data object can be recovered from any subset of k fragments [3]. While large (k, n) values may be an overkill for static networks, they are necessary for MCSs in order to ensure reliability and more importantly, avoid over-taxing a small number of devices (network hot-spots). We find that it is possible to uphold the data reliability requirement under a dynamic network topology, without re-encoding data most of the time, by using two different (k, n) parameters for data placement (*storage parameter*) and data encoding (*fragment parameter*). When the data reliability drops due to the change of network conditions (e.g. topology change or depleted battery), the framework dynamically reallocates/regenerates data fragments to maintain the system optimality (energy-efficiency and data reliability). Using two (k, n) parameters allows more flexible fragments allocation, and effectively reduces the data maintenance energy by up to 50% in our evaluations.

To the best of our knowledge, this is the first paper to consider *energy efficiency* and *storage reliability* in an integrated manner for MCSs. Focusing on the general problems for designing a distributed data storage framework, we consider only the homogeneous network where all mobile nodes are identical, and leave the heterogeneous issues as future work.



(a)



(b)

Fig. 1. (a). Data creation (top) and data retrieval/processing (bottom). A big file is split into blocks, encoded into data fragments, and distributed to the network. (b). MCS System design.

Our contributions are twofold:

- Novel formulation for a reliability-compliant and energy-efficient MCS
- Extensive evaluation of the proposed design solutions on multiple real-world and synthetic mobility traces

In this paper, Section II reviews the state of art. Sections III and IV present our MCS design with performance constraints and its extensive evaluation. The paper concludes in Section V.

II. MCS Design with Reliability Constraint

Without loss of generality, we assume that the MCS consists of homogeneous mobile devices. Each of these nodes can join or leave the cloud network freely. The mobility of the nodes is primarily due to human movements, e.g., that of soldiers or disaster responders. The mobile cloud provides storage service for applications that need to share or store files reliably. We assume all nodes can be used for storing data, and any node can be a *client node* that retrieves the files stored in MCS.

Each application specifies a minimum reliability requirement. Similar to the traditional distributed computing settings, *reliability* is defined as the probability that a stored data object can be retrieved successfully. Each node can estimate its own reliability based on the *residual energy*, *mobility*, and some *application-specific factors*. The reliability estimation procedure is described in [10]. A node is considered failed if other nodes in the network cannot communicate with it, which may be caused by hardware/software failure, depleted battery, or network partition. Once a node fails, its stored data are no longer accessible. We also assume the existence of a middleware service which detects the change of network condition (e.g., topology or residual energy). The general operational concept of data storage in an MCS is depicted in Figure 1.

Figure 1a shows that a file is first split into blocks and each block is encoded into multiple data fragments. Splitting a file into blocks allows a large file to be encoded/decoded concurrently by multiple nodes. Figure 1b shows the three primary operations of our MCS framework, *data creation*, *data retrieval*, and *data maintenance*. When a file is created, MCS framework estimates the system reliability, determines the encoding parameters, and allocates the fragments to a subset of selected storage nodes. Other nodes in the network can later access the stored files. When the data reliability requirement can no longer hold due to the change of network condition, the scheme relocates or re-generates data fragments to achieve a *graceful degradation* of reliability. In the remainder of this section, we present the data allocation and data encoding scheme that systematically optimizes the MCS system.

A. Reliability-compliant and Energy-aware Data Storage

In a traditional k-out-of-n storage system, files are encoded using a single (k, n) parameter for both data encoding and fragment placement: each file is encoded into n data fragments and each fragment is distributed to a unique storage node. This simple scheme, however, is not suitable for the MCS setting where the network condition may change unpredictably. In other words, each node's reliability and its location can change frequently. Consequently, a single assignment of the (k, n) parameter is unlikely to be sufficient, and adjusting (k, n) on-demand requires a full cycle of data retrieval, data recovery, data encoding, and fragment placement, which not only costs high energy but also incurs long delay. Instead, we propose a proactive approach using two separate (k, n) parameters for data placement and data encoding respectively so that the MCS system can remain energy-efficient and reliable with *minimal maintenance cost*. The maintenance cost here includes the energy for reallocating, regenerating, and redistributing the data fragments.

1) General Concept: We formally define (k_f, n_f) and (k_s, n_s) as the *fragment parameter* and *storage parameter* respectively. n_f is the number of data fragments that a data object is encoded into, and n_s is the total number of storage nodes for placing these fragments. k_f is the minimal number of data fragments required for recovering the data object and k_s is the minimal number of storage nodes that a client needs to contact in order to recover a data object. k_s should meet a *feasibility* requirement in that *any subset of k_s storage nodes must provide k_f or more distinct data fragments*. As an example, in Figure 1a where $(k_s, n_s) = (3, 4)$ and $(k_f, n_f) = (4, 6)$, the yellow block is encoded into $n_f = 6$ data fragments and these fragments are stored on $n_s = 4$ different storage nodes. To recover the yellow block, a client needs to retrieve at least $k_f = 4$ distinct data fragments. Our algorithm ensures that any subset of $k_s = 3$ storage nodes contains at least $k_f = 4$ distinct fragments. Note that for the *security purpose*[1], we assume that data fragments can not be cached on clients and retrieved fragments are deleted once clients finish reading the file.

In this manner, the reliability of a data object depends entirely on the *storage parameter* and the reliability of the storage nodes. (This claim is substantiated in the next subsection.) In other words, we can determine a starting value of (k_s, n_s) from the application data reliability requirement and only need to adjust this parameter (and/or select a different set of storage nodes) to adapt to the changing network condition. As such, we then carefully choose a single fragment parameter (k_f, n_f) for which a range of expected (k_s, n_s) values is feasible, so that when the storage parameter or the set of storage nodes changes, the system simply reallocates the existing fragments and thus avoids going through the full cycle of re-encoding and re-distributing the data object.

2) Estimating Data Reliability from (k_s, n_s)*:* Given the storage parameter (k_s, n_s), a set of k_s storage nodes, and the reliability of each storage node, the reliability of a stored data object can be estimated.

$$R(k_s, n_s) = \sum_{i=k_s}^{n_s} \mathbf{S_i} \quad (1)$$

$$\mathbf{S_i} = \sum_{j=1}^{\binom{n_s}{i}} \prod_{l \in c} R_l \prod_{m \in \bar{c}} Q_m \quad (2)$$

where $\forall c \subset n$ and $|c| = i$

Suppose n is the set of n_s selected storage nodes, c is the subset of functional nodes in n, and $\bar{c} = c \setminus n$ is the subset of failed nodes in n. We consider the probability that k_s or more storage nodes remain functional ($k_s \leq |\mathrm{c}| \leq n_s$).

For each size of $|c|$, there are $\binom{n_s}{|c|}$ combinations that need to be considered. Equation 1 evaluates the reliability of a system with parameter (k_s, n_s). $\mathbf{S_i}$ is the reliability of a system of exactly i functional nodes; R_l is the reliability of the l^{th} node in c; Q_m is the failure probability of the m^{th} node in $\bar{c}$. Although this is a straightforward computation, its time complexity is $O(n!)$ because the number of combinations of subset c in Equation 2 is large. Furthermore, because we are to search for a single storage parameter in Equation 1 that satisfies the reliability requirement, there are $\sum_n^N n$ possible (k_s, n_s) pairs that need to be considered (N is the network size). To account for this computation infeasibility on mobile devices, we propose an approximation algorithm to pre-compute the reliability offline and perform a table-lookup at run time. More specifically, we discretize all the variables in Equation 1 and Equation 2, and build a table of reliability with respect to different (k_s, n_s) and node's reliability.

We first approximate R_l and Q_m by the *mean reliability* r and the *mean failure probability* $(1 - r)$. We then discretize parameters k_s, n_s, and r so that a set of reliabilities can be pre-computed and stored in a table. The reliability calculation in Equation 1 is thus simplified to $R(k_s, n_s) = \sum_{i=k_s}^{n_s} \binom{n_s}{i} r^i (1-r)^{n_s - i}$. This simplified reliability computation can further be written recursively as $R(k_s, n_s) = R(i, j) = (1-r)R(i, j-1) + rR(i-1, j-1)$. The recursive form allows the table to be built efficiently with dynamic programming. Table I is an example look-up table. We will show in the evaluation section (Figure 3a) that this approximation can accurately guide the searching procedure (Tabu search) to derive a good storage parameter.

[1]E.g., military applications require high security and forbid adversaries from recovering intelligence from a single compromised device. It also serves as the performance lower bound.

TABLE I

	n_s=5	n_s=6	n_s=7	n_s=8	n_s=9
k_s=1	0.92	0.95	0.99	0.999	0.999
k_s=2	0.82	0.85	0.87	0.89	0.92
k_s=3	0.72	0.79	0.82	0.84	0.85

Reliability Lookup Table. A 2-D slice of a 3-D lookup table. The reliability r in this 2-D table is 0.8.

3) Determining (k_s, n_s)*:* Storage parameter (k_s, n_s) and the allocation of n_s storage nodes are obtained by solving the k-out-of-n storage allocation problem (Equations 3-7). A set of *Candidate Storage Parameters* is first selected from the table lookup, and a single storage parameter is then found by solving the optimization problem.

$$R_{opt} = \arg\min_{R} \; \mathbf{s}/f_k\Big(\sum_{j=1}^{N} X_j\Big) \times \Big(\sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij}R_{ij} + \sum_{j=1}^{N} D_{jc}X_j\Big) \quad (3)$$

Subject to:

$$\sum_{j=1}^{N} X_j \geq N\mathbf{s}/M \quad (4)$$

$$\sum_{j=1}^{N} R_{ij} \geq f_k\Big(\sum_{j=1}^{N} X_j\Big) \;\forall i \quad (5)$$

$$NX_j \geq \sum_{i=1}^{N} R_{ij} \geq X_j \;\forall j \quad (6)$$

$$X_j \text{ and } R_{ij} \in \{0, 1\} \;\forall i, j \quad (7)$$

Suppose the reliability requirement is 0.8. By checking Table I, we find a set of (k_s, n_s) that meets the reliability requirement (values $\geq$ 0.8). Although multiple k_s in each column may satisfy the reliability requirement, only the one with the highest k_s is selected, as using lower k_s incurs higher data redundancy and maintenance cost. For each possible column where $n_s \leq N$, we pick $k_s = max\{k : R(k, n_s)\} \geq R_{req}$ where R_{req} is the reliability requirement. We call these shaded cells in Table I as *candidate storage parameters* ($(2,5), (2,6), (3,7)...$).

To finally select a single storage parameter, we consider how the storage parameter affects the load on each storage node. Assume a data object of size $\mathbf{s}$ bytes is stored into a system of N nodes and each node requests the data object once during the effective time period $\mathbf{T_s}$. The total number of fragments transmitted during $\mathbf{T_s}$ is Nk_f (each node downloads k_f fragments) and each storage node delivers on average Nk_f/n_s fragments, or equivalently $Nk_f/n_s \times \mathbf{s}/k_f = N\mathbf{s}/n_s$ bytes (each data fragment is approximately $\mathbf{s}/k_f$ bytes). To limit the traffic on each storage node, each storage should not transmit more than $\mathbf{M}$ bytes within $\mathbf{T_s}$. This *bandwidth constraint* is then written as $N\mathbf{s}/n_s \leq M$ or $n_s \geq N\mathbf{s}/M$.

We now describe how the k-out-of-n storage allocation problem is formulated. The objective function, Equation 3, minimizes the data retrieval time. Note that the "transmission time" here implies the "transmission energy" as the RF energy consumption is proportional to the radio transmitting time [10]. Hereafter, we use the terms transmission energy and transmission time interchangeably when referring to the energy. In Equation 3, D_{ij} is the expected transmission time for node j to send a byte to node i, index c is the file creator, R_{ij} is a decision variable indicating whether client node i retrieves data from storage node j, and X_j is a decision variable indicating whether node j is selected as a storage node. The expected transmission time matrix D is estimated based on [15]. $D_{ij}R_{ij}$ indicates total data retrieval cost, $D_{jc}X_j$ indicates the data creation cost, and $\mathbf{s}/f_k(\sum_{j=1}^{N} X_j)$ is the size of each data fragment. Constraint 1 (Equation 4) enforces a lower bound on n_s according to the bandwidth constraint; constraint 2 (Equation 5) ensures that each client is assigned enough storages nodes (k_s or more); function $f_k(n_s)$ returns the corresponding k_s of the given n_s in the candidate storage parameters; constraint 3 (Equation 6) ensures that X_j is 1 if and only if j^{th} column of R has non-zero values. The solution for the optimization problem gives a single storage parameter (k_s, n_s) and the allocation of the storage nodes (X_i) such that the system minimizes the transmission energy while meeting the reliability requirement. This combinatorial optimization problem is solved by our customized Tabu Search heuristic [16].

4) Determining (k_f, n_f)*:* Fragment parameter (k_f, n_f) controls how a data object is encoded and the level of data redundancy. We aim to reduce the data maintenance cost and data redundancy when selecting the fragment parameter. Recall that a storage parameter is considered *feasible* if all the n_f data fragments can be distributed to the n_s selected storage nodes such that any subset of k_s storage nodes contains at least k_f data fragments. When the network topology changes, the number of available fragments n_f may change due to nodes failure or departure from the network. If the remaining fragments can still achieve a feasible allocation (Algorithm 1), the system can be repaired by reallocating data fragments. If there is no feasible allocation, the data object needs to be re-encoded and re-distributed. Note that an infeasible allocation does not mean the data is lost; it simply indicates that the reliability requirement imposed by the storage parameter can no longer be satisfied. The data are still recoverable as long as k_f data fragments are available.

We derive a simple feasibility test to check whether a feasible allocation exists for the given (k_s, n_s) and (k_f, n_f). Algorithm 1 tries to distribute n_f data fragments uniformly to n_s storage nodes. Once all n_f fragments have been allocated, we find the subset of k_s storage nodes with the *least number of fragments*; if this subset contains n_f or more fragments, the current parameter settings have at

Algorithm 1: FeasibilityTest

```
Input: $(k_s, n_s, k_f, n_f)$
Output: $feasible$
$feasible = false$
$uf = n_s - (n_f \mod n_s)$
if $uf \geq k_s$ then
    if $\lfloor \frac{n_f}{n_s} \rfloor k_s \geq k_f$ then
        $feasible = true$
    end
else
    if
    $\lfloor \frac{n_f}{n_s} \rfloor uf + \lceil \frac{n_f}{n_s} \rceil (k_s - uf) \geq k_f$
    then
        $feasible = true$
    end
end
return $feasible$
```

$$uf = n_s - (n_f \mod n_s)$$

$$k_f \leq \begin{cases} \lfloor \frac{n_f}{n_s} \rfloor k_s & \text{if uf} \geq k_s \\ \lfloor \frac{n_f}{n_s} \rfloor uf + (k_s - uf) \lceil \frac{n_f}{n_s} \rceil & \text{otherwise} \end{cases} \tag{8}$$

least one feasible allocation. Otherwise, there is no feasible allocation. uf indicates the number of "under-filled nodes", in which each node has $\lfloor \frac{n_f}{n_s} \rfloor$ fragments (assume $n_f \geq n_s$). uf helps identify the k_s storage nodes that has the fewest data fragments.

Similar to the feasibility test, Equation 8 finds the largest feasible k_f when given the storage parameter and n_f. The rationale is that the subset of k_s storage nodes with the fewest data fragments must have at least k_f fragments in order to have a feasible allocation.

Overall, we prefer a fragment parameter that can support multiple possible storage parameters. The fragment parameter must be carefully selected to adapt to the most possible storage parameters while keeping the data redundancy low. Naturally, the possible storage parameters are those in the *candidate storage parameters*. Suppose $(\hat{k_s}, \hat{n_s})$ is the selected storage parameters and $\hat{r}$ is the mean reliability of the selected storage nodes. We select a (k_f, n_f) that can adapt to the storage parameters in the range specified in Equation 9. α determines the range of storage parameters that (k_f, n_f) can support, and β determines the mean reliability reduction that (k_f, n_f) can tolerate. R_{req} is the minimum reliability requirement.

For example, if we refer to Table I and set $(\hat{k_s}, \hat{n_s}) = (3, 7)$, $\hat{r} = 0.8$, $\alpha = 0.2$, and $\beta = 0.1$. Assume the table contains 21 discrete reliability uniformly spread in $[0, 1]$. The mean reliability that the fragment parameter needs to support is $r = 0.8$ and $r = 0.75$ because these are the only two discrete reliability within range $0.8 \geq r \geq 0.72$. For $r = 0.8$, the storage parameters in the range are $(2, 6)$, $(3, 7)$ and $(3, 8)$ (by table lookup); for $r = 0.75$, another three storage parameters $(2, 6)$, $(2, 7)$ and $(3, 8)$ are in the range. The selected fragment parameter needs to support all these storage parameters.

$$\begin{gathered} \hat{n_s} + \alpha\hat{n_s} \geq n_s \geq \hat{n_s} - \alpha\hat{n_s} \geq 0 \\ \hat{r} \geq r \geq \hat{r} - \beta\hat{r} \geq R_{req} \\ \alpha, \beta \geq 0; \quad n_s \in integer \end{gathered} \tag{9}$$

To determine a single fragment parameter $(\hat{k_f}, \hat{n_f})$, $\hat{n_f}$ is set to the maximum n_s in Equation 9, i.e., $\hat{n_f} = \lceil (1 + \alpha)\hat{n_s} \rceil$. Using this $\hat{n_f}$ and Equation 8, we find the feasible fragment parameters for each candidate storage parameter. E.g, given $(k_s, n_s) = (3, 7)$ and $\hat{n_f} = 8$, the feasible k_f are 1, 2, and 3. In this manner, we obtain an intersection of all the feasible fragment parameters. In the intersection set, the one with the highest k_f is selected. This selected $(\hat{k_f}, \hat{n_f})$ is feasible to all storage parameters in the range specified by Equation 9.

5) Fragment Re-allocation: When the network monitoring component detects a significant change of network topology or nodes' reliabilities, the system re-evaluates the parameters and the allocation of the data fragments. If there are sufficient data fragments, the system simply moves the fragments from the old storage nodes to the new storage nodes. In this section, we present an algorithm based on the *minimum-cost flow problem* to reallocate the fragments with minimal transmission energy.

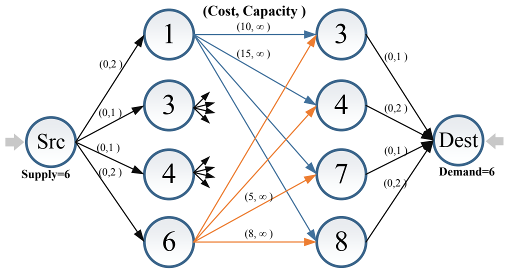


Fig. 2. The minimum cost flow problem formulation

$$x_{opt} = \arg\min_{x} \sum_{i=1}^{n_s}\sum_{j=1}^{n_s} a_{ij}x_{ij} \tag{10}$$

Subject to:

$$\sum_j x_{ji} - \sum_j x_{ij} = \begin{cases} V, & i = src, \\ 0, & i \neq src, dest, \\ -V, & i = dest. \end{cases} \quad \forall i \tag{11}$$

$$0 \le x_{ij} \le c_{ij} \tag{12}$$

The problem is formulated as a directed graph shown in Figure 2 where the left side is the set of current storage nodes and the right side is the set of newly selected storage nodes. All current nodes are connected to a virtual source node and all new nodes are connected to a virtual destination node.

Each arc is associated with a cost a_{ij} and a capacity c_{ij}, represented by (a_{ij}, c_{ij}); the arcs between the current nodes and the new nodes are assigned infinite capacity and their costs are set to D_{ij}; the arcs from the virtual source to the current nodes are assigned zero cost and their capacities are equal to the number of fragments on the current node; the arcs from the new storage nodes to the virtual destination are assigned zero cost and their capacities are equal to the number of required fragments on the new node. The *supply* of the virtual source and the *demand* of the virtual destination are both set to the total number of fragments to be transferred.

The optimization problem is expressed as a Linear Programming problem in Equations 10-12; the objective function (Equation 10) minimizes the cost for sending supply from the source to the destination; x_{ij} is the decision variable indicating the number of fragments node i sends to node j. Equation 11 ensures that the flow conservation property on all nodes except the source and the destination; V is the magnitude of supply/demand; Equation 12 ensures the flow on each arc does not exceed the capacity. This particular linear programming problem is solved efficiently in polynomial time by the *network simplex algorithm* [17].

III. Performance Evaluation

We evaluate our algorithms by extensive trace driven simulations using synthetic traces and Dartmouth Network Trace [18]. Specifically, we are interested in the accuracy of reliability estimation and data maintenance/reallocation energy consumption. These metrics are measured for different network sizes, number of storage nodes, and number of failure nodes. Given an application reliability requirement, we compare the energy consumption for data storage and maintenance using either the traditional single (k, n) parameter and our proposed *split parameters*. When evaluating our framework on the real network trace, we also compare the performance with Hadoop Distributed File System (HDFS).

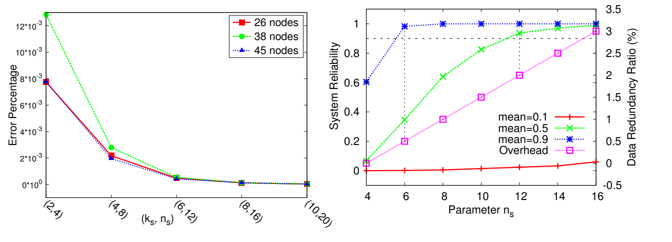


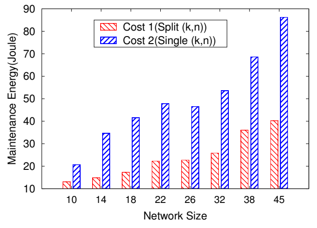
Fig. 3. (a) The error of system reliability estimation in different network sizes. (b) The effect of (k_s, n_s) on the system reliability and data redundancy.

A. Synthetic Network Trace

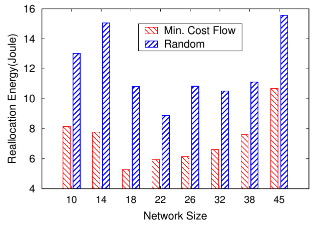
We consider a 400×400m^2 field where 10 to 45 mobile nodes are randomly deployed. The communication range of each node is 130m, as measured on HTC Evo smartphones. Each mobility trace contains 4 hours of data with 1Hz sampling rate, and nodes move with an average speed of 1 m/s. Mobility traces are generated using the Group Mobility (RPGM) model. Figure 3a depicts the accuracy of the table-lookup reliability estimation. Given the storage parameter (k_s, n_s), we compare the true reliability using Equation 1 and the approximated reliability estimated by table lookup. The results show that the error is less than 1% and decreases as n_s increases. When more than half of the nodes in the network are selected, the error drops to below 0.1%. The error decreases with n_s because a larger selected subset is better approximated by the mean network reliability.

Figure 3b shows the effects of storage parameter (k_s, n_s) on the system reliability and data redundancy. In this experiment, the network size is 16, k_s is fixed at 4, (k_f, n_f)=(k_s, n_s), and n_s varies from 4 to 16. Four networks with different mean reliabilities are evaluated. Data redundancy is defined as the increase in size after a data object is encoded by erasure coding. It is obvious that both the system reliability and the data redundancy increase with n_s. Data redundancy is essentially the price of higher system reliability. Assume the reliability requirement is 0.9. The network with 0.9 mean reliability needs only 50% redundancy while the network with 0.5 mean reliability needs 200% redundancy in order to achieve the same reliability requirement.

The purpose of having split storage and fragment parameters is to provide more flexibility and reduce the maintenance cost. If a single fragment parameter can adapt to several storage parameters, the chance of expensive data re-encoding and re-distribution will be lower. The maintenance cost here includes the energy consumption for data reallocation, data retrieval, data encoding, and data redistribution. We estimate the energy consumption based on the energy profile measured from a real smartphone (HTC Evo). Figure 4a shows that split scheme reduces the overall maintenance energy by around 50%. Figure 4b considers only the “data reallocation energy” from the total maintenance energy and compares the performance

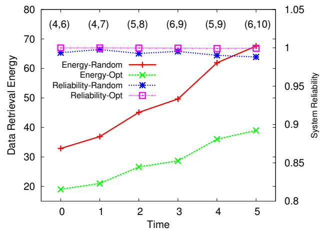


(a)

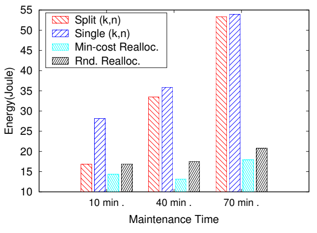


(b)

Fig. 4. (a) Performance comparison of unsplit storage parameter, i.e. $(k_s, n_s) = (k_f, n_f)$, and split parameter. (b) Compare the energy consumption of random reallocation and min-cost flow reallocation.



(a)



(b)

Fig. 5. (a) The effectiveness of the maintenance algorithm. (b) Maintenance energy of Dartmouth dataset at different times.

between *minimum-cost flow reallocation* and *Random* reallocation. *Random* scheme randomly pairs up the new storage node and the old storage node when reallocating fragments. An interesting observation is that network size 10 and 14 have relatively high reallocation energy. The reason is that these two networks have higher data redundancy and more topology changes; when the network size is small, the ratio n_s/k_s is usually higher in order to achieve the system reliability requirement, causing higher redundancy ratio n_f/k_f. More redundant data thus increases the overall reallocation cost.

In our network trace, each node's reliability gradually decreases as time elapsed. Upon a topology change event, the maintenance algorithm may update the storage parameter or data allocation to maintain the system optimality. Figure 5a compares the data retrieval energy of our solution and *Random* solution. In *Random*, n_s storage nodes are randomly selected and clients access data from their closest k_s storage nodes. The x-axis represents the time the maintenance algorithm is triggered and the updated storage parameter is shown at the top of the plot; the left and the right y-axis represent the overall data retrieval energy and system reliability at each time point respectively. The energy efficiency of both schemes degrade as the time elapses because routes between clients and storage nodes become less reliable and the average retrieval time becomes longer. One interesting observation that the system reliability remains almost constant in both schemes. This is because the updated storage parameter keeps the system reliability high regardless of the data allocation.

1) Dartmouth Network Trace: The Dartmouth Outdoor Dataset [18] includes the GPS locations and routing tables of 41 laptops moving in a $255\times365\text{m}^2$ athletic field for 1.5 hours. During the experiment, 7 laptops failed to generate any data, and another 8 laptops became inactive after 30-40 minutes. Less than 26 nodes completed the entire experiment, as most of the laptops reached the end of battery life. This realistic trace serves as a good model for evaluating our solutions.

TABLE II

	5 min.	20 min.	30 min.	40 min.
Avg. N2N dist.	1.84	2.51	3.22	3.27
Avg.Degree	6.97	4.86	4.63	4.11
MCC	20	29	28	26
# of failures	3	3	5	8

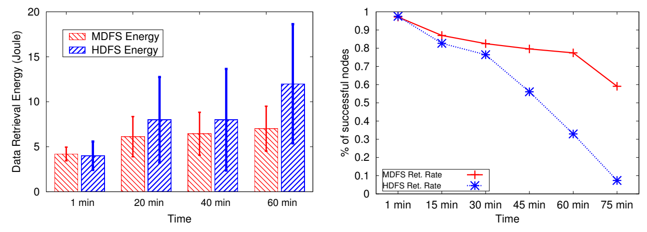
Statistics of Network Trace

A set of files are created at the beginning of the experiment, and each file is split into $4MB$ data blocks. We then evaluate the average maintenance energy (reallocation, retrieval, and redistribution) of each data block. Some important network statistics are summarized in Table II. It shows the average node to node distance in terms of hop-count, the average degree of each node, the size of the maximal connected component (MCC), and the number of failed nodes. Figure 5b shows the maintenance energy at three different times.

Similar to Figure 4, we compare the maintenance cost of single and of split parameter solutions. We then compare the energy consumption of the min-cost flow reallocation and *Random* reallocation. At the beginning of the experiment, all nodes have high reliability and the network has good connectivity. As a result, very low data redundancy is necessary to achieve the reliability requirement and the overall maintenance cost is low. As the time elapses, nodes move towards wider area, some nodes become inactive, and most nodes' reliability drop. These cause higher data redundancy, unstable connectivity, and thus higher maintenance energy. When data re-distribution is inevitable, e.g., at 40 and 70 minutes, the performance of split parameter becomes close to the single storage parameter. Overall, the min-cost flow fragment reallocation achieves 20-50% lower energy than random fragment reallocation.

Figure 6 compares the performance of our storage system and the *Hadoop Distributed File System* (HDFS) in Dartmouth Network Trace. In our storage system (MDFS), a file is encoded, distributed, and maintained based on the framework presented in Sec. II-A. HDFS replicates each data block 3 times to 3 different nodes. Figure 6a evaluates the energy consumption for each node to retrieve the file at different times. The y-axis is the *mean retrieval energy* of all nodes in the network, and the error bar indicates the standard deviation of data retrieval energy among all nodes. It can be seen that our storage framework achieves better energy efficiency, and the variance of data retrieval energy among clients nodes is also much lower than HDFS. This is because our framework allocates data considering the energy efficiency and the data fragments are more evenly distributed in the network. Figure 6b estimates the number of nodes that can successfully retrieve the stored data at different times. The y-axis is the percentage of nodes that can successfully retrieve the stored data. In HDFS, the data becomes unavailable if all 3 nodes fail or if a client node can not access any of the 3 storage nodes. Because our framework continuously maintains the stored data, client nodes in MDFS have much higher probability to successfully recover a file than HDFS.

Node failures are expected in MCS (e.g., data loss, broken



(a) (b)

Fig. 6. Comparison of our storage system (MDFS) with HDFS. (a) Mean data retrieval energy from each node in the network in different times. (b) Data retrieval rate – the percentage of nodes that can recover the data.

links, task re-execution, unstable links, and network partition). Also, when the physical size of the network increases due to mobility, the average node to node distance (hop-count) increases and the number of reachable nodes in the network decreases (some nodes may leave the network). All these changes make the data retrieval and data processing in MCS more difficult, and thus impair the data processing energy as well as job makespan. When all 35 nodes are operational, close to each other, and have good network connectivity at the beginning of the experiment, both processing energy and job makespan achieve the minimal. However, as the network topology changes with time, the performance of all three solutions degrades. One interesting result is that all three solutions have very close data processing energy. A possible explanation is that the network spreads very uniformly and node to node distance has low variance, so the data allocation does not have significant impact on the data retrieval energy. As for the job makespan, minimal-energy solution again has the worst job makespan, and the *Greedy* has slightly longer job makespan than the minimal-makespan solution.

IV. Related Work

Distributed data storage on mobile devices in an infrastructureless network had been studied in many prior works. For example, *STACEE* [12] creates a peer to peer storage system from connected laptops and mobile devices, with an explicit goal to minimize the total energy consumption while maximizing user satisfaction. *WhereStore* [13] is a location-based data storage for smart devices interacting with the cloud. It uses each device's location history to determine what data to replicate locally. *Phoenix* [14] is a distributed communication and storage protocol aiming to make efficient use of storage space and communication bandwidth while maximizing the longevity of stored data. Neither the data reliability or data maintenance are explicitly considered in these works.

Huchton et al. [3] were first to introduce the concept of k-out-n reliability to a mobile cloud setting while targeting primarily military operations. Chen et al. [9] [10] later proposed several generalizations to the concept and a new resource allocation scheme to improve energy efficiency. These constraints have been modeled in ad-hoc routing and other distributed computing literature, however, none of the prior efforts systematically models the reliability and deadline constraints of applications in its problem formulations. By modeling these constraints, we hope to obtain a deeper understanding of the important tradeoff between application performance and energy efficiency in designing MCSs.

V. Conclusions and Future Work

In this paper, we study the issues and challenges for designing an energy-efficient mobile cloud storage (MCS) with explicit reliability constraint. The mobile cloud consists entirely of mobile devices and nodes are connected in ad-hoc manner. We propose algorithms that adaptively reconfigure the system parameters such that the stored data can be accessed and maintained with minimal energy cost while continuously meeting the reliability requirement. The evaluation results show the performance advantages of using our algorithms when compared with the traditional k-out-of$n$ storage system. Our future work will look at the MCS in an delay-tolerant networking environment where connectivity is more sparse. We will also consider the heterogeneous network where each node may have different communication interfaces, computation capabilities, and energy capacities.

References

[1] S. M. George et. al., "Distressnet: a wireless ad hoc and sensor network architecture for situation management in disaster response," *Communications Magazine, IEEE*, 2010.

[2] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. of MCS WKSHPS*, 2010.

[3] S. Huchton, G. Xie, and R. Beverly, "Building and evaluating a k-resilient mobile distributed file system resistant to device compromise," in *Proc. of MILCOM*, 2011.

[4] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," in *Proc. of MobiHoc*, 2012.

[5] A. Mtibaa, A. Fahim, K. A. Harras, and M. H. Ammar, "Towards resource sharing in mobile device clouds: Power balancing across mobile devices," in *Proc. of SIGCOMM WKSHPS*, 2013.

[6] B. McGarry, "Army set to introduce smartphones into combat," http://www.military.com/, March 2013.

[7] J. George, C.-A. Chen, R. Stoleru, G. G. Xie, T. Sookoor, and D. Bruno, "Hadoop mapreduce for tactical clouds," in *Proc. of CloudNet*, 2014.

[8] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, S. Venkataraman, and J. Wang, "A first look at cellular network performance during crowded events," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 41. ACM, 2013, pp. 17–28.

[9] C. Chen, M. Won, R. Stoleru, and G. Xie, "Energy-efficient fault-tolerant data storage and processing in dynamic network," in *Proc. of MobiHoc*, 2013.

[10] ——, "Resource allocation for energy efficient k-out-of-n system in mobile ad hoc networks," in *Proc. of ICCCN*, 2013.

[11] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc of MSST*, 2010.

[12] D. Neumann, C. Bodenstein, O. F. Rana, and R. Krishnaswamy, "STACEE: enhancing storage clouds using edge devices," in *Proc. of WACE*, 2011.

[13] P. Stuedi, I. Mohomed, and D. Terry, "WhereStore: location-based data storage for mobile devices interacting with the cloud," in *Proc. of MCS*, 2010.

[14] R. K. Panta, R. Jana, F. Cheng, Y.-F. R. Chen, and V. A. Vaishampayan, "Phoenix: Storage using an autonomous mobile infrastructure," *TPDS*, 2013.

[15] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wrless Netws.*, 2005.

[16] D. Ghosh, "Neighborhood search heuristics for the uncapacitated facility location problem," *EJOR*, 2003.

[17] J. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," *Mathematical Programming*, 1997.

[18] R. S. Gray et. al., "CRAWDAD data set dartmouth/outdoor," Downloaded from http://crawdad.org/dartmouth/outdoor/, Nov. 2006.