Energy-Efficient Fault-Tolerant Data Storage & Processing in Mobile Cloud

Chien-An Chen, Myounggyu Won, Radu Stoleru, *Member, IEEE,* and Geoffrey G. Xie *Member, IEEE*

Abstract—Despite the advances in hardware for hand-held mobile devices, resource-intensive applications (e.g., video and image storage and processing or map-reduce type) still remain off bounds since they require large computation and storage capabilities. Recent research has attempted to address these issues by employing remote servers, such as clouds and peer mobile devices. For mobile devices deployed in dynamic networks (i.e., with frequent topology changes because of node failure/unavailability and mobility as in a mobile cloud), however, challenges of reliability and energy efficiency remain largely unaddressed. To the best of our knowledge, we are the first to address these challenges in an integrated manner for both data storage and processing in mobile cloud, an approach we call *k-out-of-n computing*. In our solution, mobile devices successfully retrieve or process data, in the most energy-efficient way, as long as k out of n remote servers are accessible. Through a real system implementation we prove the feasibility of our approach. Extensive simulations demonstrate the fault tolerance and energy efficiency performance of our framework in larger scale networks.

Index Terms—Mobile Computing, Cloud Computing, Mobile Cloud, Energy-Efficient Computing, Fault-Tolerant Computing

1 Introduction

PERSONAL mobile devices have gained enormous popularity in recent years. Due to their limited resources (e.g., computation, memory, energy), however, executing sophisticated applications (e.g., video and image storage and processing, or map-reduce type) on mobile devices remains challenging. As a result, many applications rely on offloading all or part of their works to "remote servers" such as clouds and peer mobile devices. For instance, applications such as Google Goggle and Siri process the locally collected data on clouds. Going beyond the traditional cloud-based scheme, recent research has proposed to offload processes on mobile devices by migrating a Virtual Machine (VM) overlay to nearby infrastructures [1], [2], [3]. This strategy essentially allows offloading any process or application, but it requires a complicated VM mechanism and a stable network connection. Some systems (e.g., Serendipity [4]) even leverage peer mobile devices as remote servers to complete computation-intensive job.

In dynamic networks, e.g., mobile cloud for disaster response or military operations [5], when selecting remote servers, energy consumption for accessing them must be minimized while taking into account the dynamically changing topology. Serendipity and other VM-based solutions considered the energy cost for processing a task on mobile devices and offloading a task to the remote servers, but they did not consider the scenario in a multi-hop and dynamic network where the energy cost for relaying/transmitting packets is significant. Furthermore, remote servers are often inaccessible because of node failures, unstable links, or node-mobility, raising a reliability issue. Although Serendipity considers intermittent connections, node failures are not taken into account; the VM-based solution considers only static networks and is difficult to deploy in dynamic environments.

In this article, we propose the first framework to support fault-tolerant and energy-efficient remote storage & processing under a dynamic network topology, i.e., mobile cloud. Our framework aims for applications that require energy-efficient and reliable distributed data storage & processing in dynamic network. E.g., military operation or disaster response. We integrate the k-out-of-n reliability mechanism into distributed computing in mobile cloud formed by only mobile devices. k-out-of-n, a well-studied topic in reliability control [6], ensures that a system of n components operates correctly as long as k or more components work. More specifically, we investigate how to store data as well as process the stored data in mobile cloud with k-out-of-n reliability such that: 1) the energy consumption for retrieving distributed data is minimized; 2) the energy consumption for processing the distributed data is minimized; and 3) data and processing are distributed considering dynamic topology changes. In our proposed framework, a data object is encoded and partitioned into n fragments, and then stored on n different nodes. As long as k or more of the n nodes are available, the data object can be successfully recovered. Similarly, another set of n nodes

- *Radu Stoleru, Myounggyu Won and Chien-An Chen are with Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77840. E-mail:{stoleru,jaychen}@cse.tamu.edu, mgwon1@gmail.com*
- *Geoffrey G. Xie is with Department of Computer Science, Naval Post Graduate School, Monterey, CA 93943 E-mail:xie@nps.edu*

Manuscript received 29 Aug. 2013; revised 24 Jan. 2014; revised 28 Mar. 2014; revised 8 May. 2014

are assigned tasks for processing the stored data and all tasks can be completed as long as k or more of the n processing nodes finish the assigned tasks. The parameters k and n determine the degree of reliability and different (k, n) pairs may be assigned to data storage and data processing. System administrators select these parameters based on their reliability requirements. The contributions of this article are as follows:

- It presents a mathematical model for both optimizing energy consumption and meeting the fault tolerance requirements of data storage and processing under a dynamic network topology.
- It presents an efficient algorithm for estimating the communication cost in a mobile cloud, where nodes fail or move, joining/leaving the network.
- It presents the first process scheduling algorithm that is both fault-tolerant and energy efficient.
- It presents a distributed protocol for continually monitoring the network topology, without requiring additional packet transmissions.
- It presents the evaluation of our proposed framework through a real hardware implementation and large scale simulations.

The article is organized as follows: Section 2 introduces the architecture of the framework and the mathematical formulation of the problem. Section 3 describes the functions and implementation details of each component in the framework. In section 4, an application that uses our framework (i.e., a mobile distributed file system – MDFS) is developed and evaluated. Section 5 presents the performance evaluation of our k-out-of-n framework through extensive simulations. Section 6 reviews the state of art. We conclude in Section 7.

2 Architecture and Formulations

An overview of our proposed framework is depicted in Figure 1. The framework, running on all mobile nodes, provides services to applications that aim to: (1) store data in mobile cloud reliably such that the energy consumption for retrieving the data is minimized (k-out-of-n data allocation problem); and (2) reliably process the stored data such that energy consumption for processing the data is minimized (k-out-of-n data processing problem). As an example, an application running in a mobile ad-hoc network may generate a large amount of media files and these files must be stored reliably such that they are recoverable even if certain nodes fail. At later time, the application may make queries to files for information such as the number of times an object appears in a set of images. Without loss of generality, we assume a data object is stored once, but will be retrieved or accessed for processing multiple times later.

We first define several terms. As shown in Figure 1, applications generate *data* and our framework stores data in the network. For higher data reliability and availability, each data is encoded and partitioned into

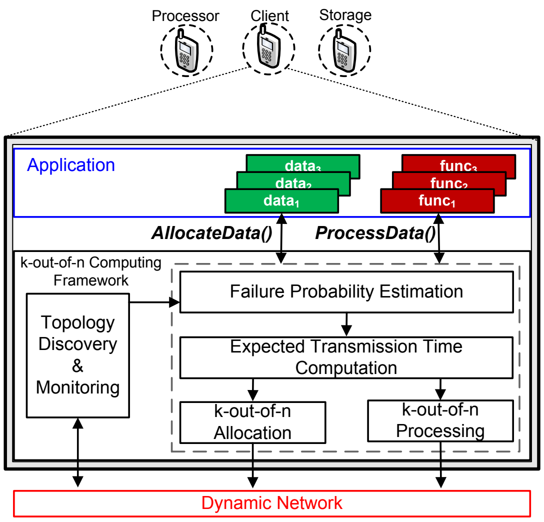

Fig. 1. Architecture for integrating the k-out-of-n computing framework for energy efficiency and fault-tolerance. The framework is running on all nodes and it provides data storage and data processing services to applications, e.g., image processing, Hadoop.

fragments; the fragments are distributed to a set of *storage nodes*. In order to process the data, applications provide *functions* that take the stored data as inputs. Each function is instantiated as multiple *tasks* that process the data simultaneously on different nodes. Nodes executing tasks are *processor nodes*; we call a set of tasks instantiated from one function a *job*. *Client nodes* are the nodes requesting data allocation or processing operations. A node can have any combination of roles from: storage node, processor node, or client node, and any node can retrieve data from storage nodes.

As shown in Figure 1, our framework consists of five components: Topology Discovery and Monitoring, Failure Probability Estimation, Expected Transmission Time (ETT) Computation, k-out-of-n Data Allocation and k-out-of-n Data Processing. When a request for data allocation or processing is received from applications, the Topology Discovery and Monitoring component provides network topology information and *failure probabilities* of nodes. The failure probability is estimated by the Failure Probability component on each node. Based on the retrieved failure probabilities and network topology, the ETT Computation component computes the ETT matrix, which represents the expected energy consumption for communication between any pair of node. Given the ETT matrix, our framework finds the locations for storing fragments or executing tasks. The k-out-of-n Data Storage component partitions data into n fragments by an erasure code algorithm and stores these fragments in the network such that the energy consumption for retrieving k fragments by any node is minimized. k is the minimal number of fragments required to recover a data. If an application needs to process the data, the k-out-of-n Data Processing component creates a job of M tasks and schedules the tasks on n processor nodes such that the energy consumption for retrieving and processing these data is minimized. This component ensures that all tasks complete as long as k or more processor nodes finish their assigned tasks. The Topology Discovery and Monitoring component continuously monitors the network for any significant change of the network topology. It starts the Topology Discovery when necessary.

2.1 Preliminaries

Having explained the overall architecture of our framework, we now present design primitives for the k-out-of-n data allocation and k-out-of-n data processing. We consider a dynamic network with N nodes denoted by a set $V = \{v_1, v_2, ..., v_N\}$. We assume nodes are time synchronized. For convenience, we will use i and v_i interchangeably hereafter. The network is modeled as a graph $G = (V, E)$, where E is a set of edges indicating the communication links among nodes. Each node has an associated *failure probability* $P[f_i]$ where f_i is the event that causes node v_i to fail.

Relationship Matrix R is a $N \times N$ matrix defining the relationship between nodes and storage nodes. More precisely, each element R_{ij} is a binary variable – if R_{ij} is 0, node i will not retrieve data from storage node j; if R_{ij} is 1, node i will retrieve fragment from storage node j. *Storage node list* X is a binary vector containing storage nodes, i.e., $X_i = 1$ indicates that v_i is a storage node.

The *Expected Transmission Time Matrix* D is defined as a $N \times N$ matrix where element D_{ij} corresponds to the ETT for transmitting a fixed size packet from node i to node j considering the failure probabilities of nodes in the network, i.e., multiple possible paths between node i and node j. The ETT metric [7] has been widely used for estimating transmission time between two nodes in one hop. We assign each edge of graph G a positive estimated transmission time. Then, the path with the shortest transmission time between any two nodes can be found. However, the shortest path for any pair of nodes may change over time because of the dynamic topology. ETT, considering multiple paths due to nodes failures, represents the "expected" transmission time, or "expected" transmission energy between two nodes.

Scheduling Matrix S is an $L \times N \times M$ matrix where element $S_{lij} = 1$ indicates that task j is scheduled at time l on node i; otherwise, $S_{lij} = 0$. l is a relative time referenced to the starting time of a job. Since all tasks are instantiated from the same function, we assume they spend approximately the same processing time on any node. Given the terms and notations, we are ready to formally describe the k-out-of-n data allocation and k-out-of-n data processing problems.

2.2 Formulation of k-out-of-n Data Allocation Problem

In this problem, we are interested in finding n storage nodes denoted by $S = \{s_1, s_2, ...s_n\}$, $S \subseteq V$ such that the *total expected transmission cost* from any node to its k closest storage nodes – in terms of ETT – is minimized. We formulate this problem as an ILP in Equations 1 - 5.

$$R_{opt} = \arg\min_{R} \sum_{i=1}^{N} \sum_{j=1}^{N} D_{ij} R_{ij} \tag{1}$$

$$\text{Subject to:} \quad \sum_{j=1}^{N} X_j = n \tag{2}$$

$$\sum_{j=1}^{N} R_{ij} = k \ \forall i \tag{3}$$

$$X_j - R_{ij} \geq 0 \ \forall i \tag{4}$$

$$X_j \text{ and } R_{ij} \in \{0, 1\} \ \forall i, j \tag{5}$$

The first constraint (Eq 2) selects exactly n nodes as storage nodes; the second constraint (Eq 3) indicates that each node has access to k storage nodes; the third constraint (Eq 4) ensures that j^{th} column of R can have a non-zero element if only if X_j is 1; and constraints (Eq 5) are binary requirements for the decision variables.

2.3 Formulation of k-out-of-n Data Processing Problem

The objective of this problem is to find n nodes in V as processor nodes such that energy consumption for processing a job of M tasks is minimized. In addition, it ensures that the job can be completed as long as k or more processors nodes finish the assigned tasks. Before a client node starts processing a data object, assuming the correctness of erasure coding, it first needs to retrieve and decode k data fragments because nodes can only process the decoded plain data object, but not the encoded data fragment.

In general, each node may have different energy cost depending on their energy sources; e.g., nodes attached to a constant energy source may have zero energy cost while nodes powered by battery may have relatively high energy cost. For simplicity, we assume the network is homogeneous and nodes consume the same amount of energy for processing the same task. As a result, only the *transmission energy* affects the energy efficiency of the final solution. We leave the modeling of the general case as future work.

Before formulating the problem, we define some functions: (1) $f_1(i)$ returns 1 if node i in S has at least one task; otherwise, it returns 0; (2) $f_2(j)$ returns the number of instances of task j in S; and (3) $f_3(z, j)$ returns the transmission cost of task j when it is scheduled for the z^{th} time. We now formulate the $k - out - of - n$ data processing problem as shown in Eq 6 - 11.

The objective function (Eq 6) minimizes the total transmission cost for all processor nodes to retrieve their tasks. l represents the time slot of executing a task; i is the index of nodes in the network; j is the index of the task of a job. We note here that T^r, the *Data Retrieval Time Matrix*, is a $N \times M$ matrix, where the element T^r_{ij} corresponds to the estimated time for node i to retrieve task j. T^r is computed by summing the transmission time (in terms of ETT available in D) from node i to its k closest storage nodes of the task.

$$\text{minimize} \quad \sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{M} S_{lij} T^r_{ij} \tag{6}$$

$$\text{Subject to:} \quad \sum_{i}^{N} f_1(i) = n \tag{7}$$

$$f_2(j) = n - k + 1 \;\forall j \tag{8}$$

$$\sum_{l=1}^{L} S_{lij} \le 1 \;\forall i, j \tag{9}$$

$$\sum_{i=1}^{N} S_{lij} \le 1 \;\forall l, j \tag{10}$$

$$\sum_{j=1}^{M} f_3(z_1, j) \le \sum_{j=1}^{M} f_3(z_2, j) \;\forall z_1 \le z_2 \tag{11}$$

The first constraint (Eq 7) ensures that n nodes in the network are selected as processor nodes. The second constraint (Eq 8) indicates that each task is replicated $n-k+1$ times in the schedule such that any subset of k processor nodes must contain at least one instance of each task. The third constraint (Eq 9) states that each task is replicated at most once to each processor node. The fourth constraint (Eq 10) ensures that no duplicate instances of a task execute at the same time on different nodes. The fifth constraint (Eq 11) ensures that a set of all tasks completed at earlier time should consume lower energy than a set of all tasks completed at later time. In other words, if no processor node fails and each task completes at the earliest possible time, these tasks should consume the least energy.

3 Energy Efficient and Fault Tolerant Data Allocation and Processing

This section presents the details of each component in our framework.

3.1 Topology Discovery

Topology Discovery is executed during the network initialization phase or whenever a significant change of the network topology is detected (as detected by the Topology Monitoring component). During Topology Discovery, one delegated node floods a *request* packet throughout the network. Upon receiving the request packet, nodes reply with their neighbor tables and failure probabilities. Consequently, the delegated node obtains global connectivity information and failure probabilities of all nodes. This topology information can later be queried by any node.

3.2 Failure Probability Estimation

We assume a fault model in which faults caused only by node failures and a node is inaccessible and cannot provide any service once it fails. The failure probability of a node estimated at time t is the probability that the node fails by time $t+T$, where T is a time interval during which the estimated failure probability is effective. A node estimates its failure probability based on the following events/causes: energy depletion, temporary disconnection from a network (e.g., due to mobility), and application-specific factors. We assume that these events happen independently. Let f_i be the event that node i fails and let f_i^B, f_i^C, and f_i^A be the events that node i fails due to energy depletion, temporary disconnection from a network, and application-specific factors respectively. The failure probability of a node is as follows: $P\left[f_i\right] = 1 - \left(1 - P\left[f_i^B\right]\right)\left(1 - P\left[f_i^C\right]\right)\left(1 - P\left[f_i^A\right]\right)$. We now present how to estimate $P[f_i^B], P[f_i^C]$, and $P[f_i^A]$.

3.2.1 Failure by Energy Depletion

Estimating the remaining energy of a battery-powered device is a well-researched problem [8]. We adopt the remaining energy estimation algorithm in [8] because of its simplicity and low overhead. The algorithm uses the history of periodic battery voltage readings to predict the battery remaining time. Considering that the error for estimating the battery remaining time follows a normal distribution [9], we find the probability that the battery remaining time is less than T by calculating the cumulative distributed function (CDF) at T. Consequently, the predicted battery remaining time x is a random variable following a normal distribution with mean μ and standard deviation σ, as given by:

$$P\left[f_i^B\right] = P\left[\text{Rem. time} < T \mid \text{Current Energy}\right]$$
$$= \int_{-\infty}^{T} f(x;\mu;\sigma^2)dx, f(x;\mu;\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

3.2.2 Failure by Temporary Disconnection

Nodes can be temporarily disconnected from a network, e.g., because of the mobility of nodes, or simply when users turn off the devices. The probability of temporary disconnection differs from application to application, but this information can be inferred from the history: a node gradually learns its behavior of disconnection and cumulatively creates a probability distribution of its disconnection. Then, given the current time t, we can estimate the probability that a node is disconnected from the network by the time $t+T$ as follows: $P\left[f_i^C\right] = P\left[\text{Node } i \text{ disconnected between } t \text{ and } t+T\right]$.

3.2.3 Failure by Application-dependent Factors

Some applications require nodes to have different roles. In a military application for example, some nodes are equipped with better defense capabilities and some nodes may be placed in high-risk areas, rendering different failure probabilities among nodes. Thus, we define the failure probability $P[f_i^A]$ for application-dependent factors. This type of failure is, however, usually explicitly known prior to the deployment.

3.3 Expected Transmission Time Computation

It is known that a path with minimal hop-count does not necessarily have minimal end-to-end delay because a path with lower hop-count may have noisy links, resulting in higher end-to-end delay. Longer delay implies higher transmission energy. As a result, when distributing data or processing the distributed data, we consider the most energy-efficient paths – paths with *minimal transmission time*. When we say path p is the shortest path from node i to node j, we imply that path p has the lowest transmission time (equivalently, lowest energy consumption) for transmitting a packet from node i to node j. The shortest distance then implies the lowest transmission time.

Given the failure probability of all nodes, we calculate the ETT matrix D. However, if failure probabilities of all nodes are taken into account, the number of possible graphs is extremely large, e.g., a total of 2^N possible graphs, as each node can be either in failure or non-failure state. Thus, it is infeasible to deterministically calculate ETT matrix when the network size is large. To address this issue, we adopt the *Importance Sampling technique*, one of the Monte Carlo methods, to approximate ETT. The Importance Sampling allows us to approximate the value of a function by evaluating multiple samples drawn from a sample space with known probability distribution. In our scenario, the probability distribution is found from the failure probabilities calculated previously and samples used for simulation are snapshots of the network graph with each node either fails or survives. The function to be approximated is the ETT matrix, D.

A sample graph is obtained by considering each node as an independent Bernoulli trial, where the success probability for node i is defined as: $p_{X_i}(x) = (1 - P[f_i])^x P[f_i]^{1-x}$, where $x \in \{0, 1\}$. Then, a set of sample graphs can be defined as a multivariate Bernoulli random variable B with a probability mass function $p_g(b) = P[X_1 = x_1, X_2 = x_2, ..., X_n = x_n] = \prod_{i=1}^{N} p_{X_i}(x)$. $x_1, x_2, ..., x_n$ are the binary outcomes of Bernoulli experiment on each node. b is an $1 \times N$ vector representing one sample graph and $b[i]$ in binary indicating whether node i survives or fails in sample b.

Having defined our sample, we determine the number of required Bernoulli samples by checking the variance of the ETT matrix denoted by $Var(E[D(B)])$, where the ETT matrix $E[D(B)]$ is defined as follows: $E[D(B)] = \left(\sum_{j=1}^{K} b_j p_g(b_j)\right)$ where K is the number of samples and j is the index of each sample graph.

In Monte Carlo Simulation, the true $E[D(B)]$ is usually unknown, so we use the ETT matrix estimator, $\tilde{D}(B)$, to calculate the variance estimator, denoted by $\widehat{Var}\left(\tilde{D}(B)\right)$. The expected value estimator and

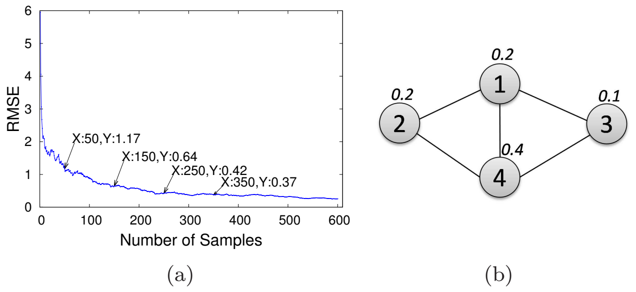

Fig. 2. (a) Root Mean Square Error (RMSE) of each iteration of Monte Carlo Simulation. (b) A simple graph of four nodes. The number above each node indicates the failure probability of the node.

variance estimator below are written in a recursive form and can be computed efficiently at each iteration:

$$\tilde{D}(B_K) = \frac{1}{K}\left((K-1)\tilde{D}(B_{K-1}) + b_K\right)$$

$$\widehat{Var}\left(\tilde{D}(B_K)\right) = \frac{1}{K(K-1)}\sum_{i=1}^{K}\left(b_j - \tilde{D}(B_K)\right)^2$$

$$= \frac{1}{K}\left(\frac{1}{K-1}\sum_{j=1}^{K}(b_i)^2 - \frac{K}{K-1}\left(\tilde{D}(B_K)\right)^2\right)$$

Here, the Monte Carlo estimator $\tilde{D}(B)$ is an unbiased estimator of $E[D(B)]$, and K is the number of samples used in the Monte Carlo Simulation. The simulation continues until $\widehat{Var}\left(\tilde{D}(B)\right)$ is less than $dist\_var_{th}$, a user defined threshold depending on how accurate the approximation has to be. We chose $dist\_var_{th}$ to be 10% of the smallest node-to-node distance in $\tilde{D}(B)$.

Figure 2 compares the ETT found by Importance Sampling with the true ETT found by a brute force method in a network of 16 nodes. The *Root Mean Square Error* (RMSE) is computed between the true ETT matrix and the approximated ETT matrix at each iteration. It is shown that the error quickly drops below 4.5% after the 200th iteration.

3.4 k-out-of-n Data Allocation

After the ETT matrix is computed, the k-out-of-n data allocation is solved by ILP solver. A simple example of how the ILP problem is formulated and solved is shown here. Considering Figure 2(b), distance Matrix D is a 4×4 symmetric matrix with each component D_{ij} indicating the expected distance between node i and node j. Let's assume the expected transmissions time on all edges are equal to 1. As an example, D_{23} is calculated by finding the probability of two possible paths: $2 \rightarrow 1 \rightarrow 3$ or $2 \rightarrow 4 \rightarrow 3$. The probability of $2 \rightarrow 1 \rightarrow 3$ is $0.8 \times 0.8 \times 0.9 \times 0.4 = 0.23$ and the probability of $2 \rightarrow 4 \rightarrow 3$ is $0.8 \times 0.6 \times 0.9 \times 0.2 = 0.08$. Another possible case is when all nodes survive and either path may be taken. This probability is $0.8 \times 0.8 \times 0.6 \times 0.9 = 0.34$. The probability that no path exists between node 2 and node 3 is (1-0.23-0.08-0.34=0.35).

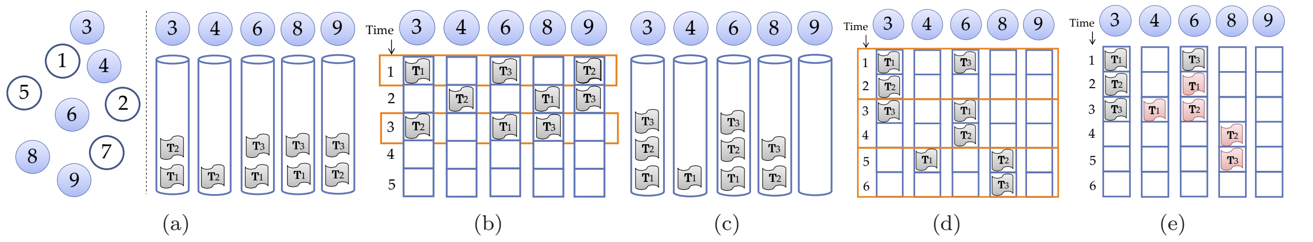

Fig. 3. k-out-of-n data processing example with $N = 9, n = 5, k = 3$. (a) and (c) are two different task allocations and (b) and (d) are their tasks scheduling respectively. In both cases, node $3, 4, 6, 8, 9$ are selected as processor nodes and each task is replicated to 3 different processor nodes. (e) shows that shifting tasks reduce the job completion time from 6 to 5.

We assign the longest possible ETT=3, to the case when two nodes are disconnected. D_{23} is then calculated as $0.23 \times 2 + 0.08 \times 2 + 0.34 \times 2 + 0.35 \times 3 = 2.33$. Once the ILP problem is solved, the binary variables X and R give the allocation of data fragments. In our solution, X shows that nodes $1 - 3$ are selected as storage nodes; each row of R indicates where the client nodes should retrieve the data fragments from. E.g., the first row of R shows that node 1 should retrieve data fragments from node 1 and node 3.

$$D = \begin{pmatrix} 0.6 & 1.72 & 1.56 & 2.04 \\ 1.72 & 0.6 & 2.33 & 2.04 \\ 1.56 & 2.33 & 0.3 & 1.92 \\ 2.04 & 2.04 & 1.92 & 1.2 \end{pmatrix}$$

$$R = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad X = (1\ 1\ 1\ 0)$$

3.5 k-out-of-n Data Processing

The k-out-of-n data processing problem is solved in two stages – *Task Allocation* and *Task Scheduling*. In the Task Allocation stage, n nodes are selected as *processor nodes*; each processor node is assigned one or more tasks; each task is replicated to $n - k + 1$ different processor nodes. An example is shown in Figure 3(a). However, not all instances of a task will be executed – once an instance of the task completes, all other instances will be canceled. The task allocation can be formulated as an ILP as shown in Eqs 12 - 16. In the formulation, $\overline{R_{ij}}$ is a $N \times M$ matrix which predefines the relationship between processor nodes and tasks; each element $\overline{R_{ij}}$ is a binary variable indicating whether task j is assigned to processor node i. $\overline{X}$ is a binary vector containing processor nodes, i.e., $X_i = 1$ indicates that v_i is a processor node. The objective function minimizes the transmission time for n processor nodes to retrieve all their tasks. The first constraint (Eq 13) indicates that n of the N nodes will be selected as processor nodes. The second constraint (Eq 14) replicates each task to $(n - k + 1)$ different processor nodes. The third constraint (Eq 15) ensures that the j^{th} column of R can have a non-zero element if only if X_j is 1; and

$$\overline{R_{opt}} = \arg\min_{\overline{R}} \sum_{i=1}^{N} \sum_{j=1}^{M} T_{ij}^{r} \overline{R_{ij}} \tag{12}$$

$$\text{Subject to:} \quad \sum_{i=1}^{N} \overline{X_i} = n \tag{13}$$

$$\sum_{i=1}^{N} \overline{R_{ij}} = n - k + 1 \ \forall j \tag{14}$$

$$\overline{X_i} - \overline{R_{ij}} \geq 0 \ \forall i \tag{15}$$

$$\overline{X_j} \text{ and } \overline{R_{ij}} \in \{0, 1\} \ \ \forall i, j \tag{16}$$

the constraints (Eq 16) are binary requirements for the decision variables.

Once processor nodes are determined, we proceed to the Task Scheduling stage. In this stage, the tasks assigned to each processor node are scheduled such that the energy and time for finishing at least M distinct tasks is minimized, meaning that we try to shorten the job completion time while minimizing the overall energy consumption. The problem is solved in three steps. First, we find the minimal energy for executing M distinct tasks in $\overline{R_{ij}}$. Second, we find a schedule with the minimal energy that has the shortest completion time. As shown in Figure 3(b), tasks 1 to 3 are scheduled on different nodes at time slot 1; however, it is also possible that tasks 1 through 3 are allocated on the same node, but are scheduled in different time slots, as shown in Figure 3(c) and 3(d). These two steps are repeated n-k+1 times and M distinct tasks are scheduled upon each iteration. The third step is to shift tasks to earlier time slots. A task can be moved to an earlier time slot as long as no duplicate task is running at the same time, e.g., in Figure 3(d), task 1 on node 6 can be safely moved to time slot 2 because there is no task 1 scheduled at time slot 2.

The ILP problem shown in Equations 17 - 20 finds M unique tasks from $\overline{R_{ij}}$ that have the minimal transmission cost. The decision variable W is an $N \times M$ matrix where $\overline{R_{ij}} = 1$ indicates that task j is selected to be executed on processor node i. The first constraint (Eq 18) ensures that each task is scheduled exactly one time. The second constraint (Eq 19) indicates that $\overline{W_{ij}}$ can be set only if task j is allocated to node i in $\overline{R_{ij}}$.

$$W_E = \arg\min_W \sum_{i=1}^{N}\sum_{j=1}^{M} T_{ij}\overline{R_{ij}}W_{ij} \tag{17}$$

$$\text{Subject to: } \sum_{i=1}^{N} W_{ij} = 1\ \forall j \tag{18}$$

$$\overline{R_{ij}} - W_{ij} \geq 0\ \forall i,j \tag{19}$$

$$W_{ij} \in \{0,1\}\ \forall i,j \tag{20}$$

$$\underset{Y}{\text{minimize}}\ \mathrm{Y} \tag{21}$$

$$\text{Subject to:} \sum_{i=1}^{N}\sum_{j=1}^{M} T_{ij} \times \overline{R_{ij}} \times \overline{W_{ij}} \leq E_{min} \tag{22}$$

$$\sum_{i=1}^{N} \overline{W_{ij}} = 1 \quad \forall j \tag{23}$$

$$\overline{R_{ij}} - \overline{W_{ij}} \geq 0\ \forall i,j \tag{24}$$

$$Y - \sum_{j=1}^{M} \overline{W_{ij}} \geq 0 \quad \forall i \tag{25}$$

$$\overline{W_{ij}} \in \{0,1\} \quad \forall i,j \tag{26}$$

The last constraint (Eq 20) is a binary requirement for decision matrix W.

Once the minimal energy for executing M tasks is found, among all possible schedules satisfying the minimal energy budget, we are interested in the one that has the minimal completion time. Therefore, the minimal energy found previously, $E_{min} = \sum_{i=1}^{N}\sum_{j=1}^{M} T_{ij}\overline{R_{ij}}W_E$, is used as the "upper bound" for searching a task schedule.

If we define $L_i = \sum_{j=1}^{M} \overline{W_{ij}}$ as the number of tasks assigned to node i, L_i indicates the completion time of node i. Then, our objective becomes to *minimize the largest number of tasks in one node*, written as $\min\left\{\max_{i\in[1,N]}\{L_i\}\right\}$. To solve this *min-max* problem, we formulate the problem as shown in Equations 21 - 26.

The objective function minimizes integer variable Y, which is the largest number of tasks on one node. $\overline{W_{ij}}$ is a decision variable similar to W_{ij} defined previously. The first constraint (Eq 22) ensures that the schedule cannot consume more energy that the E_{min} calculated previously. The second constraint (Eq 23) schedules each task exactly once. The third constraint (Eq 25) forces Y to be the largest number of tasks on one node. The last constraint (Eq 26) is a binary requirement for decision matrix $\overline{W}$. Once tasks are scheduled, we then rearrange tasks – tasks are moved to earlier time slots as long as there is free time slot and no same task is executed on other node simultaneously. Algorithm 1 depicts the procedure. Note that k-out-of-n data processing ensures that k or more functional processing nodes complete all tasks of a job with probability 1. In general, *it may be possible* that a subset of processing nodes, of size less than k, complete all tasks.

Algorithm 1 Schedule Re-arrangement

```
 1: L=last time slot in the schedule
 2: for time $t = 2 \rightarrow L$ do
 3:     for each scheduled task $J$ in time $t$ do
 4:         $n \leftarrow$ processor node of task $J$
 5:         while $n$ is idle at $t-1$ AND
 6: $J$ is NOT scheduled on any node at $t-1$ do
 7:             Move $J$ from $t$ to $t-1$
 8:             $t = t-1$
 9:         end while
10:     end for
11: end for
```

Algorithm 2 Distributed Topology Monitoring

```
 1: At each beacon interval:
 2: if $p > \tau_1$ and $s \neq U$ then
 3:     $s \leftarrow U$
 4:     Put +ID to a beacon message.
 5: end if
 6: if $p \leq \tau_1$ and $s = U$ then
 7:     $s \leftarrow NU$
 8:     Put −ID to a beacon message.
 9: end if
10:
11: Upon receiving a beacon message on $V_i$:
12: for each ID in the received beacon message do
13:     if ID > 0 then
14:         $\mathcal{ID} \leftarrow \mathcal{ID} \bigcup \{\mathrm{ID}\}$.
15:     else
16:         $\mathcal{ID} \leftarrow \mathcal{ID} \setminus \{\mathrm{ID}\}$.
17:     end if
18: end for
19: if $|\{\mathcal{ID}\}| > \tau_2$ then
20:     Notify $V_{del}$ and $V_{del}$ initiate topology discovery
21: end if
22: Add the ID in $V_i'$s beacon message.
```

3.6 Topology Monitoring

The Topology Monitoring component monitors the network topology continuously and runs in distributed manner on all nodes. Whenever a client node needs to create a file, the Topology Monitoring component provides the client with the most recent topology information immediately. When there is a significant topology change, it notifies the framework to update the current solution. We first give several notations. A term s refers to a state of a node, which can be either U and NU. The state becomes U when a node finds that its neighbor table has drastically changed; otherwise, a node keeps the state as NU. We let p be the number of entries in the neighbor table that has changed. A set $\mathcal{ID}$ contains the node IDs with p greater than τ_1, a threshold parameter for a "significant" local topology

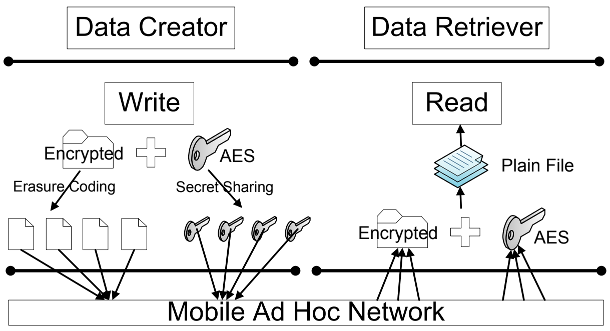

Fig. 4. An overview of improved MDFS.

change.

The Topology Monitoring component is simple yet energy-efficient as it does not incur significant communication overhead – it simply piggybacks node ID on a beacon message. The protocol is depicted in Algorithm 2. We predefine one node as a *topology_delegate* V_{del} who is responsible for maintaining the global topology information. If p of a node is greater than the threshold τ_1, the node changes its state to U and piggybacks its ID on a beacon message. Whenever a node with state U finds that its p becomes smaller than τ_1, it changes its state back to NU and puts $-$ID in a beacon message. Upon receiving a beacon message, nodes check the IDs in it. For each ID, nodes add the ID to set $\mathcal{ID}$ if the ID is positive; otherwise, remove the ID. If a client node finds that the size of set $\mathcal{ID}$ becomes greater than τ_2, a threshold for "significant" global topology change, the node notifies V_{del}; and V_{del} executes the Topology Discovery protocol. To reduce the amount of traffic, client nodes request the global topology from V_{del}, instead of running the topology discovery by themselves. After V_{del} completes the topology update, all nodes reset their status variables back to NU and set $p = 0$.

4 System Evaluation

This section investigates the feasibility of running our framework on real hardware. We compare the performance of our framework with a random data allocation and processing scheme (Random), which randomly selects storage/processor nodes. Specifically, to evaluate the k-out-of-n data allocation on real hardware, we implemented a Mobile Distributed File System (MDFS) on top of our k-out-of-n computing framework. We also test our k-out-of-n data processing by implementing a face recognition application that uses our MDFS.

Figure 4 shows an overview of our MDFS. Each file is encrypted and encoded by erasure coding into n_1 data fragments, and the secret key for the file is decomposed into n_2 key fragments by key sharing algorithm. Any *maximum distance separable code* can may be used to encoded the data and the key; in our experiment, we adopt the well-developed Reed-Solomon code and Shamir's Secret Sharing algorithm. The n_1 data fragments and n_2 key fragments are then distributed to

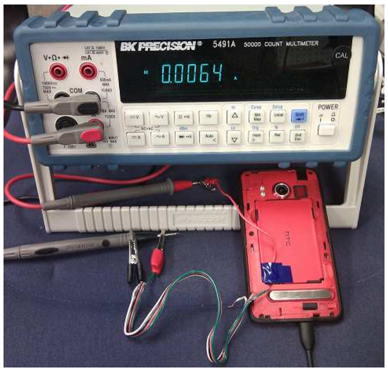

Fig. 5. Energy measurement setting.

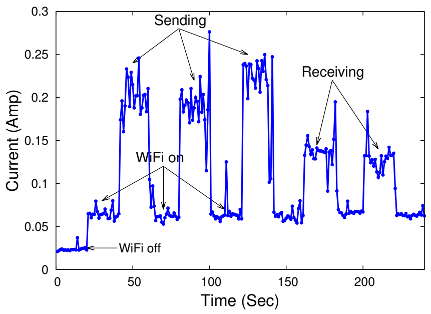

Fig. 6. Current consumption on Smartphone in different states.

nodes in the network. When a node needs to access a file, it must retrieve at least k_1 file fragments and k_2 key fragments. Our k-out-of-n data allocation allocates file and key fragments optimally when compared with the state-of-art [10] that distributes fragments uniformly to the network. Consequently, our MDFS achieves higher reliability (since our framework considers the possible failures of nodes when determining storage nodes) and higher energy efficiency (since storage nodes are selected such that the energy consumption for retrieving data by any node is minimized). Any

We implemented our system on HTC Evo 4G Smartphone, which runs Android 2.3 operating system using 1G Scorpion CPU, 512MB RAM, and a Wi-Fi 802.11 b/g interface. To enable the Wi-Fi AdHoc mode, we rooted the device and modified a config file – wpa_supplicant.conf. The Wi-Fi communication range on HTC Evo 4G is 80-100m. Our data allocation was programmed with 6,000 lines of Java and C++ code.

The experiment was conducted by 8 students who carry smartphones and move randomly in an open space. These smartphones formed an Ad-Hoc network and the longest node to node distance was 3 hops. Students took pictures and stored in our MDFS. To evaluate the k-out-of-n data processing, we designed an application that searches for human faces appearing in all stored images. One client node initiates the processing request and all selected processor nodes retrieve, decode, decrypt, and analyze a set of images. In average, it took about $3-4$ seconds to process an image of size 2MB. Processing a sequence of images, e.g., a video

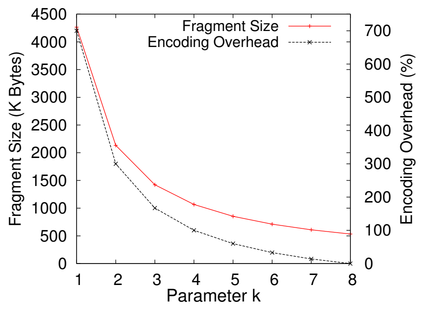

Fig. 7. A file of 4.1 MB is encoded with n fixed to 8 and k swept from 1 to 8.

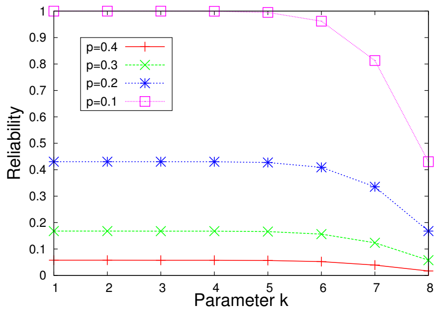

Fig. 8. Reliability with respect to different k/n ratio and failure probability.

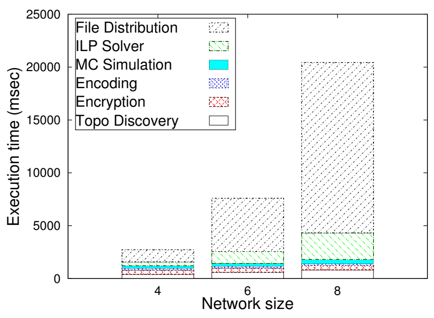

Fig. 9. Execution time of different components with respect to various network size.

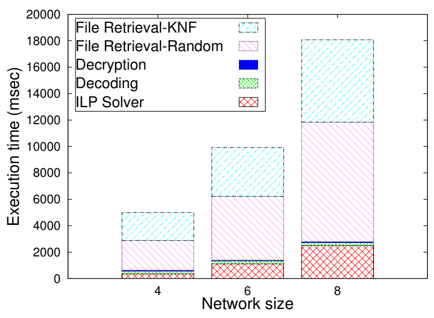

Fig. 10. Execution time of different components with respect to various network size. File retrieval time of Random and our algorithm (KNF) is also compare here

stream, the time may increase in an order of magnitude. The peak memory usage of our application was around 3MB. In addition, for a realistic energy consumption model in simulations, we profiled the energy consumption of our application (e.g., WiFi-idle, transmission, reception, and 100%-cpu-utilization). Figure 5 shows our experimental setting and Figure 6 shows the energy profile of our smartphone in different operating states. It shows that Wi-Fi component draws significant current during the communication(sending/receiving packets) and the consumed current stays constantly high during the transmission regardless the link quality.

Figure 7 shows the overhead induced by encoding data. Given a file of 4.1MB, we encoded it with different k values while keeping parameter $n = 8$. The left y-axis is the size of each encoded fragment and the right y-axis is the percentage of the overhead. Figure 8 shows the system reliability with respect to different k while n is constant. As expected, smaller k/n ratio achieves higher reliability while incurring more storage overhead. An interesting observation is that the change of system reliability slows down at $k = 5$ and reducing k further does not improve the reliability much. Hence, $k = 5$ is a reasonable choice where overhead is low ($\approx$ 60% of overhead) and the reliability is high($\approx$ 99% of the highest possible reliability).

To validate the feasibility of running our framework on a commercial smartphone, we measured the execution time of our MDFS application in Figure 9. For this experiment we varied network size N and set $n = \lceil 0.6N \rceil$, $k = \lceil 0.6n \rceil$, $k_1 = k_2 = k$, and $n_1 = n_2 = n$. As shown, nodes spent much longer time in distributing/retrieving fragments than other components such as data encoding/decoding. We also observe that the time for distributing/retrieving fragments increased with the network size. This is because fragments are more sparsely distributed, resulting in longer paths to distribute/retrieve fragments. We then compared the data retrieval time of our algorithm with the data retrieval time of random placement. Figure 10 shows that our framework achieved 15% to 25% lower data retrieval time than Random. To validate the performance of our k-out-of-n data processing, we measured the completion rate of our face-recognition job by varying the number of failure node. The face recognition job had an average completion rate of 95% in our experimental setting.

5 Simulation Results

We conducted simulations to evaluate the performance of our k-out-of-n framework (denoted by KNF) in larger scale networks. We consider a network of 400×400m^2 where up to 45 mobile nodes are randomly deployed. The communication range of a node is 130m, which is measured on our smartphones. Two different mobility models are tested – Markovian Waypoint Model and Reference Point Group Mobility (RPGM). Markovian Waypoint is similar to Random Waypoint Model, which randomly selects the waypoint of a node, but it accounts for the current waypoint when it determines the next waypoint. RPGM is a group mobility model where a

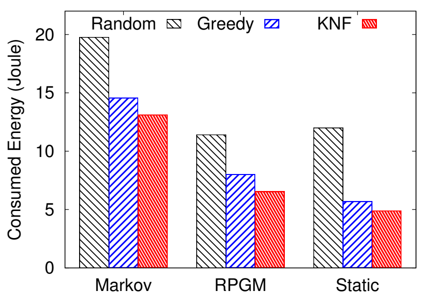

Fig. 11. Effect of mobility on energy consumption. We compare three different allocation algorithms under different mobility models.

subset of leaders are selected; each leader moves based on Markovian Waypoint model and other non-leader nodes follow the closest leader. Each mobility trace contains 4 hours of data with 1Hz sampling rate. Nodes beacon every 30 seconds.

We compare our KNF with two other schemes – a greedy algorithm (Greedy) and a random placement algorithm (Random). Greedy selects nodes with the largest number of neighbors as storage/processor nodes because nodes with more neighbors are better candidates for cluster heads and thus serve good facility nodes. Random selects storage or processor nodes randomly. The goal is to evaluate how the selected storage nodes impact the performance. We measure the following metrics: consumed energy for retrieving data, consumed energy for processing a job, data retrieval rate, completion time of a job, and completion rate of a job. We are interested in the effects of the following parameters – *mobility model*, *node speed*, *k/n ratio*, τ_2, and *number of failed nodes*, and *scheduling*. The default values for the parameters are: $N = 26$, $n = 7$, $k = 4$, $\tau_1 = 3$, $\tau_2 = 20$; our default mobility model is RPGM with node-speed 1m/s. A node may fail due to two independent factors: depleted energy or an application-dependent failure probability; specifically, the energy associated with a node decreases as the time elapses, and thus increases the failure probability. Each node is assigned a constant application-dependent failure probability.

We first perform simulations for k-out-of-n data allocation by varying the first four parameters and then simulate the k-out-of-n data processing with different number of failed nodes. We evaluate the performance of data processing only with the number of node failures because data processing relies on data retrieval and the performance of data allocation directly impacts the performance of data processing. If the performance of data allocation is already bad, we can expect the performance of data processing will not be any better.

The simulation is performed in Matlab. The energy profile is taken from our real measurements on smartphones; the mobility trace is generated according to RPGM mobility model; and the linear programming problem is solved by the Matlab optimization toolbox.

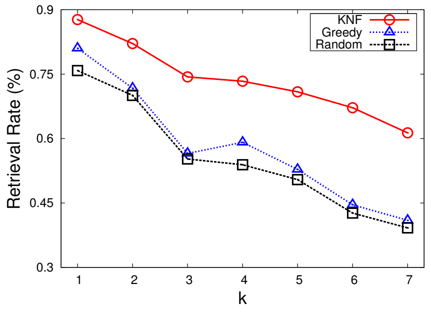

Fig. 12. Effect of k/n ratio on data retrieval rate when $n = 7$.

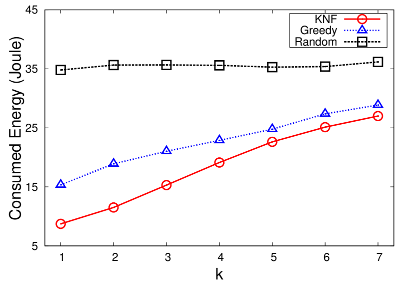

Fig. 13. Effect of k/n ratio on energy efficiency when $n = 7$.

5.1 Effect of Mobility

In this section, we investigate how mobility models affect different data allocation schemes. Figure 11 depicts the results. An immediate observation is that mobility causes nodes to spend higher energy in retrieving data compared with the static network. It also shows that the energy consumption for RPGM is smaller than that for Markov. The reason is that a storage node usually serves the nodes in its proximity; thus when nodes move in a group, the impact of mobility is less severe than when all nodes move randomly. In all scenarios, KNF consumes lower energy than others.

5.2 Effect of k/n Ratio

Parameters k and n, set by applications, determine the degree of reliability. Although lower k/n ratio provides higher reliability, it also incurs higher data redundancy. In this section, we investigate how the k/n ratio (by varying k) influences different resource allocation schemes. Figure 12 depicts the results. The data retrieval rate decreases for all three schemes when k is increased. It is because, with larger k, nodes have to access more storage nodes, increasing the chances of failing to retrieve data fragments from all storage nodes. However, since our solution copes with dynamic topology changes, it still yields 15% to 25% better retrieval rate than the other two schemes.

Figure 13 shows that when we increase k, all three schemes consume more energy. One observation is that the consumed energy for Random does not increase much compared with the other two schemes. Unlike

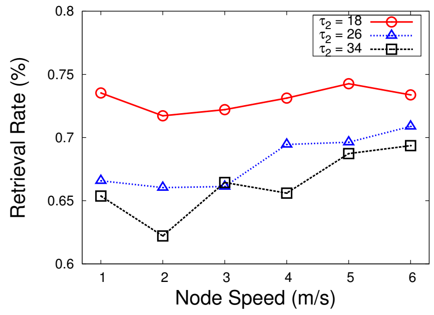

Fig. 14. Effect of τ_2 and node speed on data retrieval rate

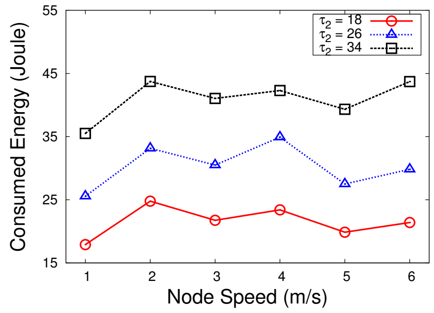

Fig. 15. Effect of τ_2 and node speed on energy efficiency.

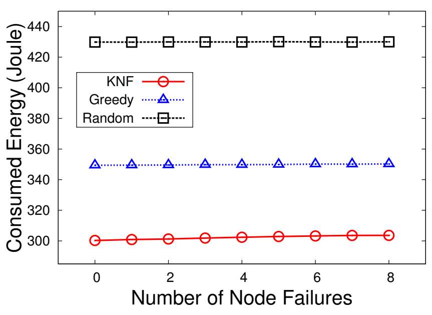

Fig. 16. Effect of node failure on energy efficiency with fail-slow.

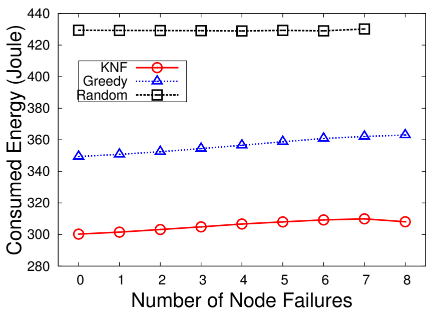

Fig. 17. Effect of node failure on energy efficiency with fail-fast.

KNF and Greedy, for Random, storage nodes are randomly selected and nodes choose storage nodes randomly to retrieve data; therefore, when we run the experiments multiple times with different random selections of storage nodes, we eventually obtain a similar average energy consumption. In contrast, KNF and Greedy select storage nodes based on their specific rules; thus, when k becomes larger, client nodes have to communicate with some storage nodes farther away, leading to higher energy consumption. Although lower k/n is beneficial for both retrieval rate and energy efficiency, it requires more storage and longer data distribution time. A 1MB file with $k/n = 0.6$ in a network of 8 nodes may take 10 seconds or longer to be distributed (as shown in Figure10).

5.3 Effect of τ_2 and Node Speed

Figure 14 shows the average retrieval rates of KNF for different τ_2. We can see that smaller τ_2 allows for higher retrieval rates. The main reason is that smaller τ_2 causes KNF to update the placement more frequently. We are aware that smaller τ_2 incurs overhead for relocating data fragments, but as shown in Figure 15, energy consumption for smaller τ_2 is still lower than that for larger τ_2. The reasons are, first, energy consumed for relocating data fragments is much smaller than energy consumed for inefficient data retrieval; second, not all data fragments need to be relocated. Another interesting observation is that, despite higher node speed, both retrieval rates and consumed energy do not increase much. The results confirm that our topology monitoring component works correctly: although nodes move with different speeds, our component reallocates the storage nodes such that the performance does not degrade much.

5.4 Effect of node failures in k-out-of-n data processing

This section investigates how the failures of processor nodes affect the energy efficiency, job completion time, and job completion rate. We first define how Greedy and Random work for data processing. In Greedy, each task is replicated to n-k+1 processor nodes that have the lowest energy consumption for retrieving the task, and given a task, nodes that require lower energy for retrieving the task are scheduled earlier. In Random, the processor nodes are selected randomly and each task is also replicated to n-k+1 processor nodes randomly. We consider two failure models: fail-fast and fail-slow. In the fail-fast model, a node fails at the first time slot and cannot complete any task, while in the fail-slow model, a node may fail at any time slot, thus being able to complete some of its assigned tasks before the failure.

Figure 16 and Figure 17 show that KNF consumes 10% to 30% lower energy than Greedy and Random. We observe that the energy consumption is not sensitive to the number of node failures. When there is a node failure, a task may be executed on a less optimal processor node and causes higher energy consumption. However, this difference is small due to the following reasons. First, given a task, because it is replicated to n-k+1 processor nodes, failing an arbitrary processor may

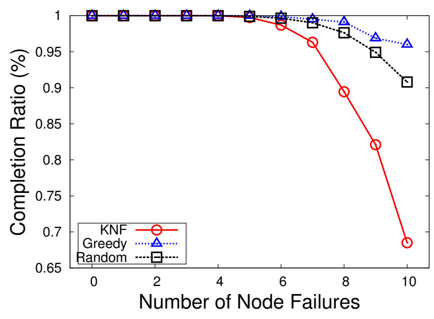

Fig. 18. Effect of node failure on completion ratio with fail-slow.

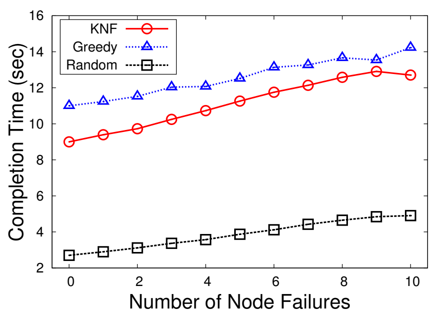

Fig. 20. Effect of node failure on completion time with fail-slow.

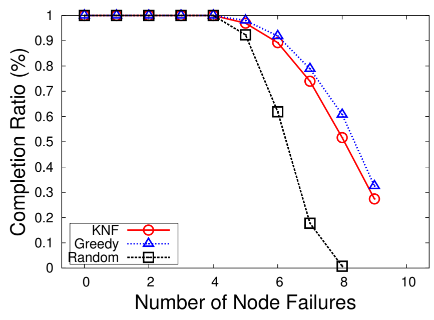

Fig. 19. Effect of node failure on completion ratio with fail-fast.

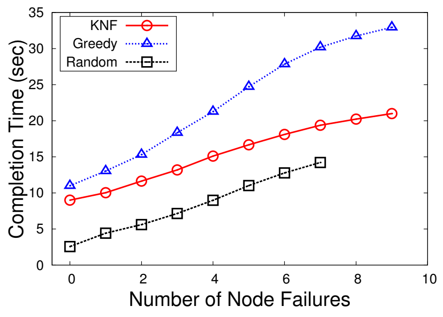

Fig. 21. Effect of node failure on completion time with fail-fast.

have no effect on the execution time of this task at all. Second, even if a processor node with the task fails, this task might have completed before the time of failure. As a result, the energy difference caused by failing an additional node is very small. In the fail-fast model, a failure always affects all the tasks on a processor node, so its energy consumption increases faster than the fail-slow model.

In Figure 18 and Figure 19, we see that the completion ratio is 1 when no more than $n-k$ nodes fail. Even when more than $n-k$ nodes fail, due to the same reasons explained previously, there is still chance that all M tasks complete (tasks may have completed before the time the node fails). In general, for any scheme, the completion ratio of the fail-slow model is higher than the completion ratio of the fail-fast model. An interesting observation is that Greedy has the highest completion ratio. In Greedy, the load on each node is highly uneven, i.e., some processor nodes may have many tasks but some may not have any task. This allocation strategy achieves high completion ratio because all tasks can complete as long as one such high load processor nodes can finish all its assigned tasks. In our simulation, about 30% of processor nodes in Greedy are assigned all M tasks. Analytically, if three of the ten processor nodes contain all M tasks, the probability of completion when 9 processor nodes fail is $1-\binom{7}{6}/\binom{10}{9} = 0.3$. We note that load-balancing is not an objective in this article. As our objectives are energy-efficiency and fault-tolerance, we leave the more complicated load-balancing problem formulation for future work.

In Figure 20 and Figure 21, we observe that completion time of Random is lower than both Greedy and KNF. The reason is that both Greedy and KNF try to minimize the energy at the cost of longer completion time. Some processor nodes may need to execute much more tasks because they consume lower energy for retrieving those tasks compared to others. On the other hand, Random spreads tasks to all processor nodes evenly and thus results in lowest completion time.

5.5 Effect of scheduling

Figure 22 and Figure 23 evaluate the performance of KNF before and after applying the scheduling algorithms to k-out-of-n data processing. When the tasks are not scheduled, all processing nodes try to execute the assigned tasks immediately. Since each task is replicated to $n-k+1$ times, multiple instances of a same task may execute simultaneously on different nodes. Although concurrent execution of a same task wastes energy, it achieves lower job completion time. This is because when there is node failure, the failed task still has a chance to be completed on other processing node in the same time slot, without affecting the job completion time. On the other hand, because our scheduling algorithm avoids executing same instances of a task concurrently, the completion time will always be delayed whenever there is a task failure. Therefore, scheduled tasks always achieve minimal energy consumption while unscheduled tasks complete the job in shorter time. The system reliability, or the completion ratio, however, is not affected by the scheduling algorithm.

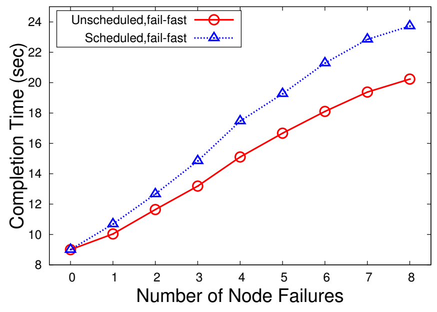

Fig. 22. Comparison of performance before and after scheduling algorithm on job completion time.

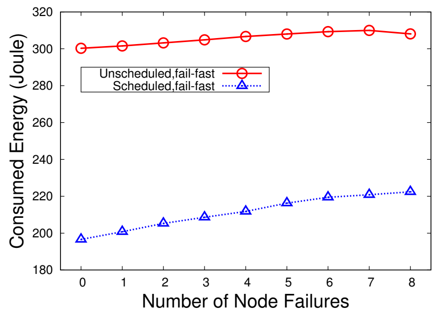

Fig. 23. Comparison of performance before and after scheduling algorithm on job consumed energy.

6 Related Work

Some researchers proposed solutions for achieving higher reliability in dynamic networks. Dimakis et al. proposed several erasure coding algorithms for maintaining a distributed storage system in a dynamic network [11]. Leong et al. proposed an algorithm for optimal data allocation that maximizes the recovery probability [12]. Aguilera et al. proposed a protocol to efficiently adopt erasure code for better reliability [13]. These solutions, however, focused only on system reliability and do not consider energy efficiency.

Several works considered latency and communication costs. Alicherry and Lakshman proposed a 2-approx algorithm for selecting optimal data centers [14]. Beloglazov et al. solved the similar problem by applying their Modified Best Fit Decreasing algorithm [15]. Liu et al. proposed an Energy-Efficient Scheduling (DEES) algorithm that saves energy by integrating the process of scheduling tasks and data placement [16]. [17] proposed cloudlet seeding, a strategic placement of high performance computing assets in wireless ad-hoc network such that computational load is balanced. Most of these solutions, however, are designed for powerful servers in a static network. Our solution focuses on resource-constrained mobile devices in a dynamic network.

Storage systems in ad-hoc networks consisting of mobile devices have also been studied. *STACEE* uses edge devices such as laptops and network storage to create a P2P storage system. They designed a scheme that minimizes energy from a system perspective and simultaneously maximizes user satisfaction [18]. *MobiCloud* treats mobile devices as service nodes in an ad-hoc network and enhances communication by addressing trust management, secure routing, and risk management issues in the network [19]. *WhereStore* is a location-based data store for Smartphones interacting with the cloud. It uses the phone's location history to determine what data to replicate locally [20]. *Segank* considers a mobile storage system designed to work in a network of non-uniform quality [21].

[22], [23] distribute data and process the distributed data in a dynamic network. Both the distributed data and processing tasks are allocated in an energy-efficient and reliable manner, but how to optimally schedule the task to further reduce energy and job makespan is not considered. Compared with the previous two works, this paper propose an efficient k-out-of-n task scheduling algorithm that reduces the job completion time and minimizes the energy wasted in executing duplicated tasks on multiple processor nodes. Furthermore, the tradeoff between the system reliability and the overhead, in terms of more storage space and redundant tasks, is analyzed.

Cloud computing in a small-scale network with battery-powered devices has also gained attention recently. *Cloudlet* is a resource-rich cluster that is well-connected to the Internet and is available for use by nearby mobile devices [1]. A mobile device delivers a small Virtual Machine (VM) overlay to a cloudlet infrastructure and lets it take over the computation. Similar works that use VM migration are also done in *CloneCloud* [2] and *ThinkAir* [3]. *MAUI* uses code portability provided by Common Language Runtime to create two versions of an application: one runs locally on mobile devices and the other runs remotely [24]. MAUI determines which processes to be offloaded to remote servers based on their CPU usages. Serendipity considers using remote computational resource from other mobile devices [4]. Most of these works focus on minimizing the energy, but do not address system reliability.

7 Conclusions

We presented the first k-out-of-n framework that jointly addresses the energy-efficiency and fault-tolerance challenges. It assigns data fragments to nodes such that other nodes retrieve data reliably with minimal energy consumption. It also allows nodes to process distributed data such that the energy consumption for processing the data is minimized. Through system implementation, the feasibility of our solution on real hardware was validated. Extensive simulations in larger scale networks proved the effectiveness of our solution.

Acknowledgment

This work was supported by Naval Postgraduate School under Grant No. N00244-12-1-0035 and NSF under Grants #1127449, #1145858 and #0923203.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14–23, 2009.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in *Proc. of EuroSys*, 2011.

[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of INFOCOM*, 2012.

[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] S. M. George, W. Zhou, H. Chenji, M. Won, Y. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, "DistressNet: a wireless AdHoc and sensor network architecture for situation manaigement in disaster response," *IEEE Communications Magazine*, vol. 48, no. 3, 2010.

[6] D. W. Coit and J. Liu, "System reliability optimization with k-out-of-n subsystems," *International Journal of Reliability, Quality and Safety Engineering*, vol. 7, no. 2, pp. 129–142, 2000.

[7] D. S. J. D. Couto, "High-throughput routing for multi-hop wireless networks," PhD dissertation, MIT, 2004.

[8] Y. Wen, R. Wolski, and C. Krintz, "Online prediction of battery lifetime for embedded and mobile devices," in *Power-Aware Computer Systems*. Springer Berlin Heidelberg, 2005.

[9] A. Leon-Garcia, *Probability, Statistics, and Random Processes for Electrical Engineering*. Prentice Hall, 2008.

[10] 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.

[11] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Su, "A survey on network codes for distributed storage," *Proc. of the IEEE*, vol. 99, no. 3, pp. 476–489, 2010.

[12] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocation for high reliability," in *Proc. of ICC*, 2010.

[13] M. Aguilera, R. Janakiraman, and L. Xu, "Using erasure codes efficiently for storage in a distributed system," in *Proc. of DSN*, 2005.

[14] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of INFOCOM*, 2012.

[15] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012.

[16] C. Liu, X. Qin, S. Kulkarni, C. Wang, S. Li, A. Manzanares, and S. Baskiyar, "Distributed energy-efficient scheduling for data-intensive applications with deadline constraints on data grids," in *Proc. of IPCCC*, 2008.

[17] D. Shires, B. Henz, S. Park, and J. Clarke, "Cloudlet seeding: Spatial deployment for high performance tactical clouds," in *WorldComp*, 2012.

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

[19] D. Huang, X. Zhang, M. Kang, and J. Luo, "MobiCloud: Building secure cloud framework for mobile computing and communication," in *SOSE*, 2010.

[20] P. Stuedi, I. Mohomed, and D. Terry, "WhereStore: location-based data storage for mobile devices interacting with the cloud," in *Proc. of the Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010.

[21] S. Sobti, N. Garg, F. Zheng, J. Lai, Y. Shao, R. Y. Wang, and et al., "Segank: a distributed mobile storage system," in *in Proc. of FAST*, 2004.

[22] C. Chen, M. Won, R. Stoleru, and G. Xie, "Resource allocation for energy efficient k-out-of-n system in mobile ad hoc networks," in *Proc. of ICCCN*, 2013.

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

[24] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proc. of MobiSys*, 2010.

Chien-An Chen received the BS and MS degree in Electrical Engineering from University of California, Los Angeles in 2009 and 2010 respectively. He is currently working toward the PhD degree in Computer Science and Engineering at Texas A&M University. His research interests are mobile computing, energy-efficient wireless network, and cloud computing on mobile devices.

Myounggyu Won is a postdoctoral researcher in the Department of Information & Communication Engineering at Daegu Gyeongbuk Institute of Science and Technology (DGIST), South Korea. His research interests are wireless mobile systems, vehicular ad hoc networks, intelligent transportation systems, and wireless sensor networks. He won the Graduate Research Excellence Award from the Department of Computer Science and Engineering at Texas A&M University in 2012. He received a Ph.D. in Computer Science from Texas A&M University at College Station in 2013.

Radu Stoleru is currently an associate professor in the Department of Computer Science and Engineering at Texas A&M University, and heading the Laboratory for Embedded & Networked Sensor Systems (LENSS). Dr. Stoleru's research interests are in deeply embedded wireless sensor systems, distributed systems, embedded computing, and computer networking. He is the recipient of the NSF CAREER Award in 2013. Dr. Stoleru received his Ph.D. in computer science from the University of Virginia in 2007. While at the University of Virginia, Dr. Stoleru received from the Department of Computer Science the Outstanding Graduate Student Research Award for 2007. He has authored or co-authored over 60 conference and journal papers with over 2,600 citations (Google Scholar). He is currently serving as an editorial board member for 3 international journals and has served as technical program committee member on numerous international conferences.

Geoffery G. Xie received the BS degree in computer science from Fudan University, China, and the PhD degree in computer sciences from the University of Texas, Austin. He is a professor in the Computer Science Department at the US Naval Postgraduate School. He was a visiting scientist in the School of Computer Science at Carnegie Mellon University from 2003 to 2004 and recently visited the Computer Laboratory of the University of Cambridge, United Kingdom. He has published more than 60 articles in various areas of networking. He was an editor of the Computer Networks journal from 2001 to 2004. He co-chaired the ACM SIGCOMM Internet Network Management Workshop in 2007 and is currently a member of the workshop's steering committee. His current research interests include network analysis, routing design and theories, underwater acoustic networks, and abstraction driven design and analysis of enterprise networks.