

# Processing in Mobile Cloud with Energy-Efficient Fault-Tolerant Data Storage

**Syed Sajid Hussain**

*Assistant Professor CSE,  
Islamic university of science and technology, India.*

## Abstract

We are living in a world of ubiquitous computing and everything is available to the human race. 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 enormous 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.

**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 and Siri process the locally collected data on clouds. Going beyond the traditional cloudbased scheme, recent research has

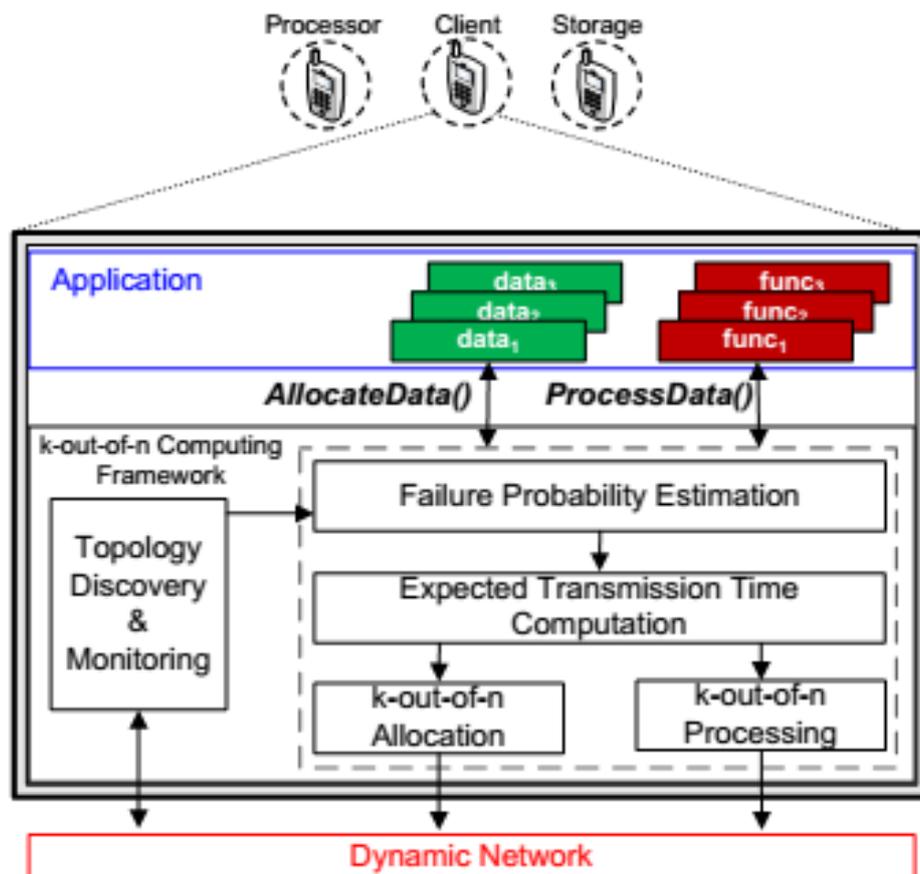
proposed to offload processes on mobile devices by migrating a Virtual Machine (VM) overlay to nearby infrastructures [1], [2], [3]. This strategy essentially allows offloading any process or application, but it requires a complicated VM mechanism and a stable network connection. Some systems (e.g., 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.

In this paper, we propose the 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 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. Section 4 presents the performance evaluation of our k-out-of-n framework through extensive simulations. Section 5 reviews the state of art.

## 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.



**Figure 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.

For higher data reliability and availability, each data is encoded and partitioned into 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 Formulation of k-out-of-n Data Allocation Problem

In this problem, we are interested in finding n storage nodes denoted by  $S = \{s_1, s_2, \dots, s_n\}$ ,  $S \subseteq V$  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.

$$R_{opt} = \arg \min_R \sum_{i=1}^N \sum_{j=1}^N D_{ij} R_{ij} \quad (1)$$

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

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

$$X_j - R_{ij} \geq 0 \quad \forall i \quad (4)$$

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

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.

## 2.2 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)  $f1(i)$  returns 1 if node  $i$  in  $S$  has at least one task; otherwise, it returns 0; (2)  $f2(j)$  returns the number of instances of task  $j$  in  $S$ ; and (3)  $f3(z, j)$  returns the transmission cost of task  $j$  when it is scheduled for the  $z^{\text{th}}$  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.

$$\text{minimize } \sum_{l=1}^L \sum_{i=1}^N \sum_{j=1}^M S_{lij} T_{ij}^r \quad (6)$$

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

$$f_2(j) = n - k + 1 \quad \forall j \quad (8)$$

$$\sum_{l=1}^L S_{lij} \leq 1 \quad \forall i, j \quad (9)$$

$$\sum_{i=1}^N S_{lij} \leq 1 \quad \forall l, j \quad (10)$$

$$\sum_{j=1}^M f_3(z_1, j) \leq \sum_{j=1}^M f_3(z_2, j) \quad \forall z_1 \leq z_2 \quad (11)$$

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  $f^{B_i}$ ,  $f^{C_i}$ , and  $f^{A_i}$  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:

$$P[f_i] = 1 - (1 - P[f_i^B]) (1 - P[f_i^C]) (1 - P[f_i^A]).$$

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  $\mu$  and standard deviation  $\sigma$ , as given by:

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

#### 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:  $P[f_i^C] = P[\text{Node } i \text{ disconnected between } t \text{ and } t+T]$ .

### 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 nonfailure 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

$$p_g(b) = p [X_1 = x_1, X_2 = x_2, \dots, X_n = x_n] = \prod_{i=1}^N p_{X_i}(x_i).$$

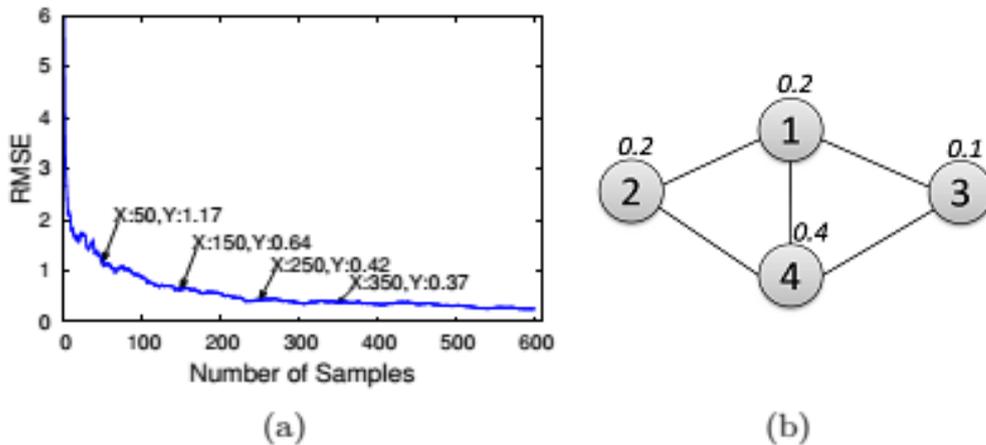
$x_1, x_2, \dots, x_n$  are the binary outcomes of Bernoulli experiment on each node.  $b$  is an  $1 \times N$  vector representing one sample graph and  $b[i]$  in binary indicating whether node  $i$  survives or fails in sample  $b$ . Having defined our sample, we determine the number of required 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  $\widehat{Var} (D (B))$ . The expected value estimator and variance estimator below are written in a recursive form and can be computed efficiently at each iteration:

$$\begin{aligned} \tilde{D} (B_K) &= \frac{1}{K} ((K - 1) \tilde{D} (B_{K-1}) + b_K) \\ \widehat{Var} (\tilde{D} (B_K)) &= \frac{1}{K(K - 1)} \sum_{i=1}^K (b_i - \tilde{D} (B_K))^2 \\ &= \frac{1}{K} \left( \frac{1}{K - 1} \sum_{j=1}^K (b_j)^2 - \frac{K}{K - 1} (\tilde{D} (B_K))^2 \right) \end{aligned}$$

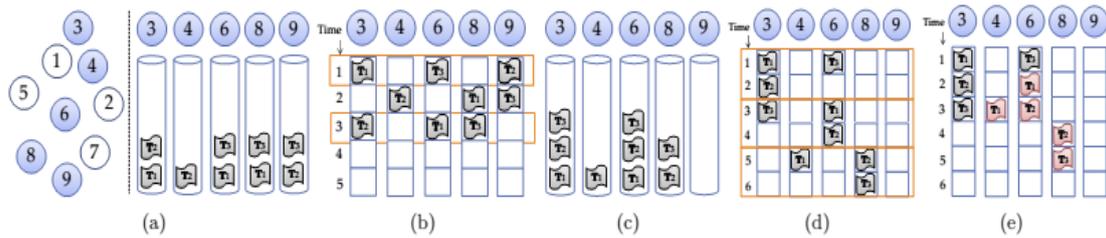
Here, the Monte Carlo estimator  $\tilde{D} (B)$  is an unbiased estimator of  $E [D (B)]$ , and  $K$  is the number of samples used in the Monte Carlo Simulation. The simulation continues until  $\widehat{Var} \tilde{D} (B)$  is less than  $dist\_var_{th}$ , a user defined threshold depending on how 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.



**Figure 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.

### 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  $(1 - 0.23 - 0.08 - 0.34) = 0.35$



**Figure 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,  $R_{ij}$  is a  $N \times M$  matrix which predefines the relationship between processor nodes and tasks; each element  $R_{ij}$  is a binary variable indicating whether task j is assigned to processor node i.  $X_i$  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 jth column of R can have a non-zero element if only if  $X_j$  is 1; and

$$\overline{R}_{opt} = \arg \min_{\overline{R}} \sum_{i=1}^N \sum_{j=1}^M T_{ij}^r \overline{R}_{ij} \quad (12)$$

$$\text{Subject to: } \sum_{i=1}^N \overline{X}_i = n \quad (13)$$

$$\sum_{i=1}^N \overline{R}_{ij} = n - k + 1 \quad \forall j \quad (14)$$

$$\overline{X}_i - \overline{R}_{ij} \geq 0 \quad \forall i \quad (15)$$

$$\overline{X}_j \text{ and } \overline{R}_{ij} \in \{0, 1\} \quad \forall i, j \quad (16)$$

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  $R_{ij}$ . 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 nk+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  $R_{ij}$  that have the minimal transmission cost. The decision variable  $W$  is an  $N \times M$  matrix where  $R_{ij} = 1$  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  $W_{ij}$  can be set only if task  $j$  is allocated to node  $i$  in  $R_{ij}$ .

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

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

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

$$W_{ij} \in \{0, 1\} \quad \forall i, j \quad (20)$$

$$\text{minimize } Y \quad (21)$$

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

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

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

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

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

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:  $\min\{\max_{i \in [1, N]} \{L_i\}\}$ . 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.  $W_{ij}$  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  $W$ . 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  $p > \tau I$  and  $s \neq U$  then
3:  $s \leftarrow U$ 
4: Put  $+ID$  to a beacon message.
5: end if
6: if  $p \leq \tau I$  and  $s = U$  then
7:  $s \leftarrow NU$ 
8: Put  $-ID$  to a beacon message.
9: end if
10: Upon receiving a beacon message on  $V_i$ 
11: for each  $ID$  in the received beacon message do
```

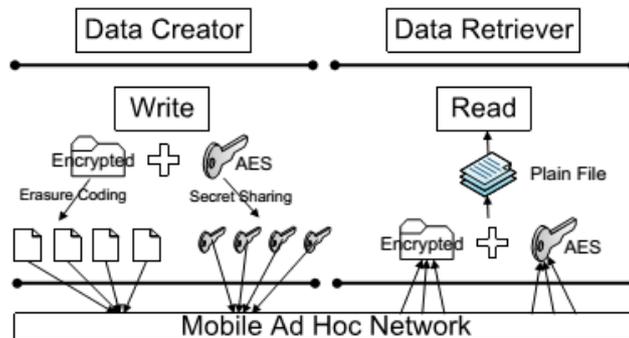
```

12: if  $ID > 0$  then
13:  $ID \leftarrow ID \cup \{ID\}$ .
14: else
15:  $ID \leftarrow ID \setminus \{ID\}$ .
16: end if
17: end for
18: if  $|\{ID\}| > \tau_2$  then
19: Notify  $V_{del}$  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  $ID$  contains the node IDs with  $p$  greater than  $\tau_1$ , a threshold parameter for a “significant” local topology change.



**Figure 4.** An overview of improved MDFS

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 Vdel who is responsible for maintaining the global topology information. If  $p$  of a node is greater than the threshold  $\tau_1$ , 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  $\tau_1$ , 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 ID if the ID is positive; otherwise, remove the ID. If a client node finds that the size of set ID becomes greater than  $\tau_2$ , a threshold for “significant” global topology change, the node notifies Vdel ; and Vdel executes the Topology Discovery protocol. To reduce the amount of traffic, client nodes request the global topology from Vdel , instead of running the topology discovery by themselves. After Vdel completes the topology update, all nodes reset their status variables back to NU and set  $p = 0$

#### 4. 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  $400 \times 400 m^2$  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

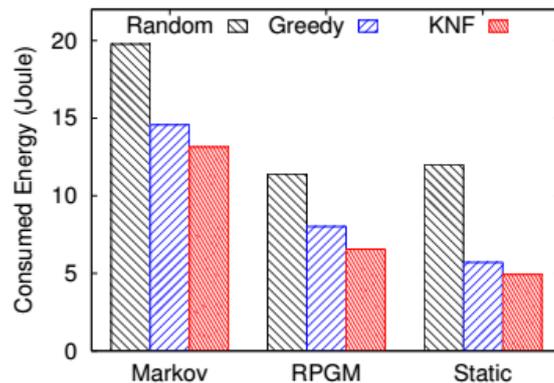


Figure 11. Effect of mobility on energy consumption.

Where a 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 Hz 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,  $\tau_2$ , 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.

### 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 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).

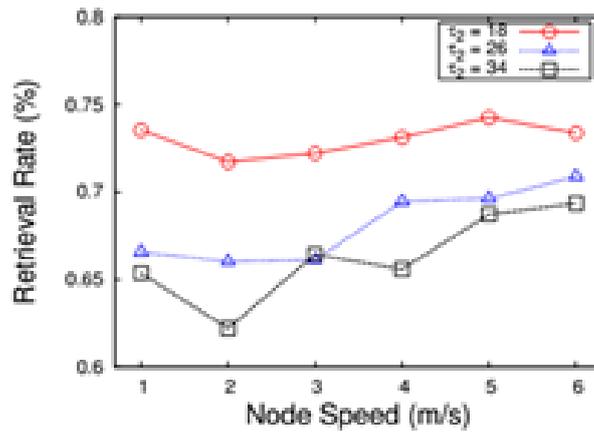


Figure 14. Effect of  $T_2$  and node speed on data retrieval rate

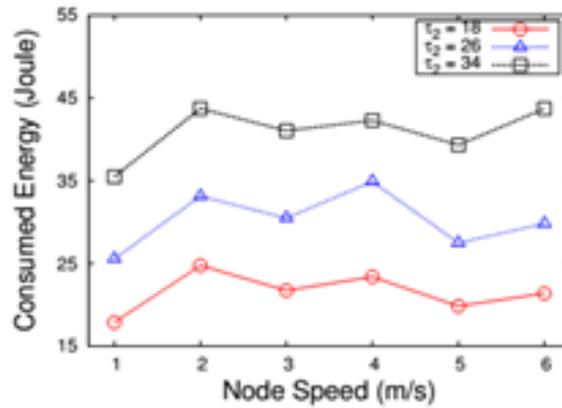
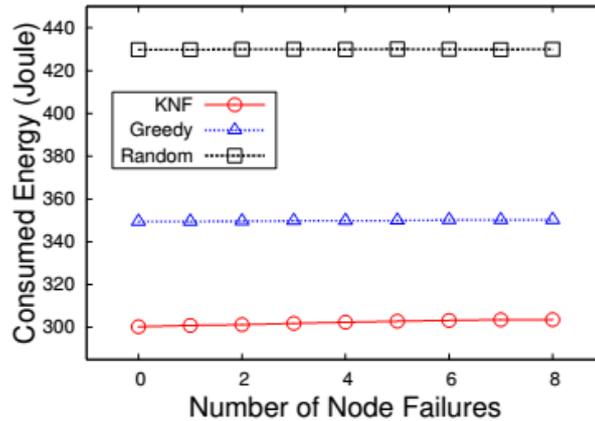


Figure 15. Effect of  $T_2$  and node speed on energy efficiency

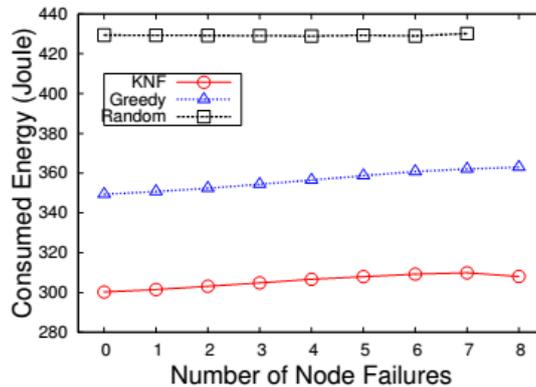
### 5.3 Effect of $\tau_2$ and Node Speed

Figure 14 shows the average retrieval rates of KNF for different  $\tau_2$ . We can see that smaller  $\tau_2$  allows for higher retrieval rates. The main reason is that smaller  $\tau_2$  causes KNF to update the placement more frequently. We are aware that smaller  $\tau_2$  incurs overhead for relocating data fragments, but as shown in Figure 15, energy consumption for smaller  $\tau_2$  is still lower than that for larger  $\tau_2$ . 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.



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



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

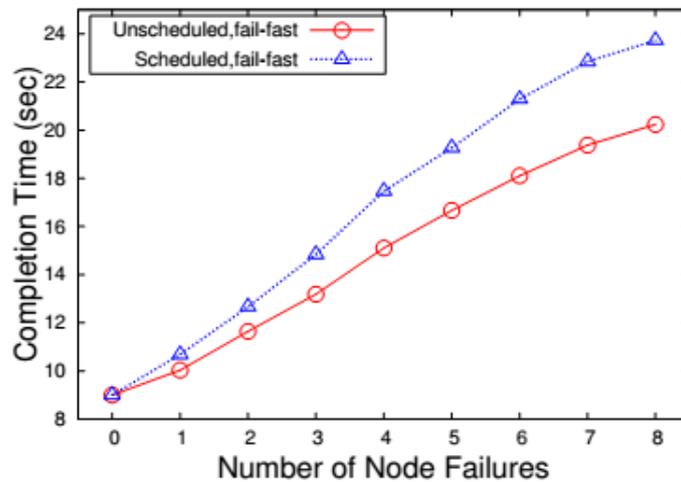
#### 5.4 Effect of node failures in k-out-of-n data processing

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-

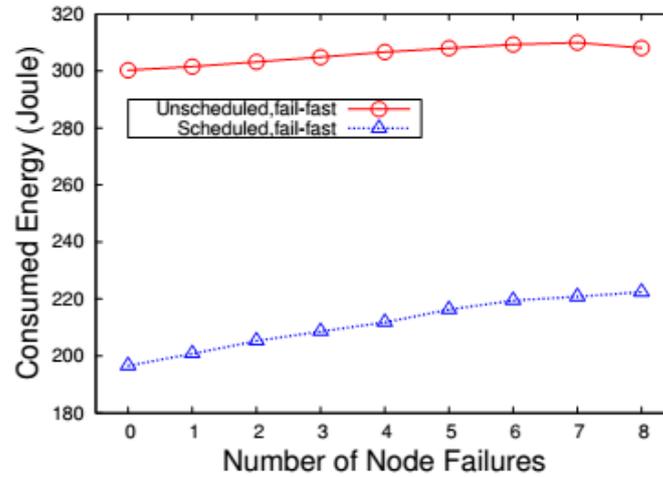
fast and fail-slow. In the fail-fast model, a node fails at the first time slot and cannot complete any task, while in the fail-slow model, a node may fail at any time slot, thus being able to complete some of its assigned tasks before the failure. Figure 16 and Figure 17 show that KNF consumes 10% to 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  $nk+1$  processor nodes, failing an arbitrary processor may

### 5.5 Effect of scheduling

Figure 18 and Figure 19 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.



**Figure 18.** Comparison of performance before and after scheduling algorithm on job completion time



**Figure 19.** Comparison of performance before and after scheduled algorithm on job consumed energy

## 6. 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.

## 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. Baroah, "DistressNet: a wireless AdHoc and sensor network

- architecture for situation management 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 *PowerAware 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. Bodenstern, 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, "Energyefficient 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.