Energy-Efficient Fault-Tolerant Data Storage & Processing in Dynamic Networks

Chien-An Chen, Myounggyu Won, Radu Stoleru
Department of Computer Science and Engineering, Texas A&M University
{jaychen,mgwon,stoleru}@cse.tamu.edu

Geoffrey G. Xie
Department of Computer Science, Naval Postgraduate School
xie@nps.edu

ABSTRACT

With the advance of mobile devices, cloud computing has enabled people to access data and computing resources without spatiotemporal constraints. A common assumption is that mobile devices are well connected to remote data centers and the data centers securely store and process data. However, for systems like mobile cloud deployed in infrastructureless dynamic networks (i.e., with frequent topology changes because of node failure/unavailability and mobility), reliability and energy efficiency remain largely unaddressed challenges. To address these issues, we develop the first "k-out-of-n computing" framework that ensures nodes retrieve or process data stored in mobile cloud with minimum energy consumption as long as k out of n storage/processing nodes are accessible. We demonstrate the feasibility and performance of our framework through both hardware implementation and extensive simulations.

Categories and Subject Descriptors

C.2 [**Networks**]: Mobile Networks, Mobile ad hoc networks, Dynamic Network; H.3 [**Information Systems**]: Distributed Storage

Keywords

Mobile Cloud, Distributed Storage, Distributed Data Processing

1. INTRODUCTION

With the explosion of personal mobile devices, cloud computing has gained popularity and become the norm in the past decade. However, due to limited resources of mobile devices (e.g., energy, processing, and memory), executing sophisticated applications (e.g., video and image storage and processing, or map-reduce type) on mobile devices remains challenging. To enable such applications on mobile devices, many commercial applications like Google Goggle and Siri rely on offloading data and computation-intensive tasks to high-performance computers. Nevertheless, in situations such as disaster response or military operations, high-performance computers are not immediately available; instead, a quick and flexible ad hoc network is required.

In this paper, we propose the first framework to support fault-tolerant and energy-efficient remote storage & processing under a dynamic network topology. We integrate the k-out-of-n reliability mechanism into distributed computing in dynamic networks. The k-out-of-n system, a well-studied topic in reliability control, ensures that a system of n components operates correctly as long as k or more components work [5]. More specifically, we investigate how to store data as well as process the stored data 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 partitioned into n fragments and stored on n different nodes. As long as k or more of the n nodes are available, the data object can be recovered. Similarly, another set of n nodes are assigned tasks for processing the stored data and all tasks can be completed as long as k or more of the n nodes finish the assigned tasks. The parameters k and n determine the degree of reliability; system administrators select these parameters based on their reliability requirements. The contributions of this paper are summarized 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 dynamic network, where nodes fail or move, joining/leaving the network.
- it presents the evaluation of our proposed framework through real hardware implementation as well as extensive simulations particularly for large-scale networks.

2. RELATED WORK

Replicating data to multiple locations is a common technique for improving performance and reliability in distributed computing. Erasure coding has also been a widely used technique for enhancing data reliability [12]. Replicating data incurs storage overhead while achieving higher reliability; coding techniques reduce storage overhead while

MobiHoc'13, July 29–August 1, 2013, Bangalore, India.

not performing well when nodes are not reliable, e.g., in dynamic networks [13].

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 [6]. Leong et al. proposed an algorithm for optimal data allocation that maximizes the recovery probability [9]. Aguilera et al. proposed a protocol to efficiently adopt erasure code for better reliability [1]. These solutions, however, focused only on system reliability but did not consider energy efficiency.

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 [10]. 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 [8]. 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 [4]. 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 [11]. Most of these works focus on minimizing the energy and do not address system reliability.

3. ARCHITECTURE

An overview of our proposed framework is depicted in Figure 1. Applications generate *data*; our framework stores the data in the network. For higher data reliability, each data is encoded and partitioned into *fragments*. These fragments then 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 into multiple *tasks* that will process the data simultaneously. Nodes that execute these tasks are *processor nodes*. We call a set of tasks instantiated from one function as a *job*. *Client nodes* are the nodes that request 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. Without loss of generality, we consider a single dataset – which is stored once and may be retrieved multiple times by any node. 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 is received from applications, the *Topology Discovery and Monitoring component* provides network topology information and failure probabilities of nodes. The failure probability, which depends on the battery lifetime, network connectivity, and application-specific factors, 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 two nodes in the network. The "expected" communication cost indicates that the estimated ETT between nodes considers the dynamically changing topology. Given the ETT matrix, our framework finds the locations for storing fragments. The *k-out-of-n Data Allocation component* partitions a data object into n fragments and stores these fragments in the network such that the energy consumption for retrieving k fragments by any node is minimized. 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 processing these data is minimized. This component ensures that all tasks complete as long as k or more processor nodes finish their assigned tasks.

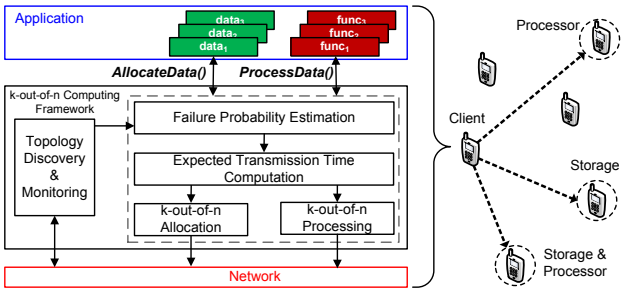

Figure 1: Architecture for integrating the k-out-of-n data allocation framework.

4. PROBLEM FORMULATION

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. 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 which defines 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 may retrieve data 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 [3] 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 two 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.

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 a battery may have relatively high energy cost. For simplicity, we assume the network is homogeneous and nodes consume the same amount of energy for proces-

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

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

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

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

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

sing the same task. As a result, only the *transmission energy* affects the energy efficiency of the final solution.

4.1 k-out-of-n Data Allocation

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 as shown in Equations 1 - 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 constraint (Eq 5) is binary requirements for the decision variables.

4.2 k-out-of-n Data Processing

The objective of this problem is to find n nodes in V as processor nodes such that energy consumption for processing M tasks of a single job is minimized. In addition, all tasks must complete as long as k or more processor nodes can finish their assigned tasks.

In this problem, n nodes are selected as *processor nodes*; each processor node is assigned one or more tasks. Each task is replicated to $n-k+1$ processor nodes. However, not all instances are processed – 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 Equations 6 - 10. 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., $\overline{X_i} = 1$ indicates that v_i is a processor node. The objective function minimizes the transmission time for n processor nodes to

$$\overline{R_{opt}} = \arg\min_{\overline{R}} \sum_{i=1}^{N} \sum_{j=1}^{M} T_{ij}^{r} \overline{R_{ij}} \quad (6)$$

$$\text{Subject to:} \quad \sum_{i=1}^{N} \overline{X_i} = n \quad (7)$$

$$\sum_{i=1}^{N} \overline{R_{ij}} = n - k + 1 \ \forall j \quad (8)$$

$$\overline{X_i} - \overline{R_{ij}} \geq 0 \ \forall i \quad (9)$$

$$\overline{X_j} \text{ and } \overline{R_{ij}} \in \{0,1\} \ \forall i,j \quad (10)$$

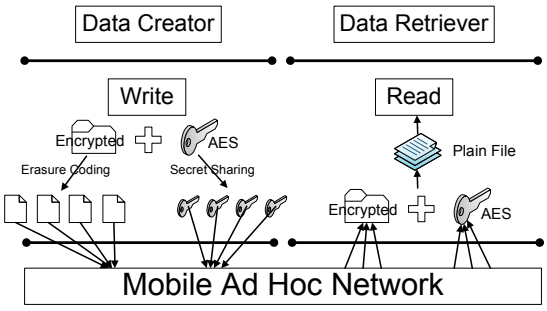

Figure 2: An overview of improved MDFS.

retrieve all tasks and their instances. The first constraint (Eq 7) indicates that n of the N nodes will be selected as processor nodes. The second constraint (Eq 8) replicates each task to $(n-k+1)$ different processor nodes. The third constraint (Eq 9) ensures that the j^{th} column of $\overline{R}$ can have a non-zero element if only if $\overline{X_j}$ is 1; and constraint (Eq 10) is binary requirements for the decision variables.

5. SYSTEM IMPLEMENTATION

This section investigates the feasibility of running our framework on real hardware. We implemented a mobile distributed file system (MDFS) on top of the k-out-of-n framework. Figure 2 shows an overview of our MDFS. As shown, each file is encrypted by a secret key and partitioned into n_1 file fragments. The secret key is also decomposed into n_2 key fragments by using Shamir's key sharing algorithm. The file and key fragments are distributed independently to 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 allows nodes to optimally distribute the file and key fragments, when compared with the state-of-art MDFS that distributes file and key fragments uniformly in the network [7]. The ratio k_1/n_1 determines the reliability of a file and the ratio k_2/n_2 indicates the security of the file. Decreasing k_1/n_1 ratio improves the file reliability, while increasing k_2/n_2 ratio strengthens the security. Consequently, our k-out-of-n enabled MDFS to achieve higher reliability ,energy efficiency, and security. On top of our MDFS, we also test our k-out-of-n data processing component by implementing a face recognition application that counts the number of faces appearing in a set of files stored in the network. Our k-out-of-n framework selects n processor nodes to retrieve and analyze the files in an energy-efficient and reliable way.

We implemented our system on HTC Evo 4G Smartphones, which run Android 2.3 OS 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. For this experiment we varied network size N and set $n_1 = \lceil 0.6N \rceil$, $k_1 = \lceil 0.3n_1 \rceil$ and $n_2 = \lceil 0.6N \rceil$, $k_2 = \lceil 0.6n_2 \rceil$. Figure 3 shows measured time of each component. As shown, distributing/retrieving fragments takes much longer time than computation, which affirms that our goal of minimizing communication energy is necessary. We also observe that larger network sizes incur longer distributing/retrieving time. This is because fragments are more sparsely distributed, resulting in more hops to distribute/retrieve fragments.

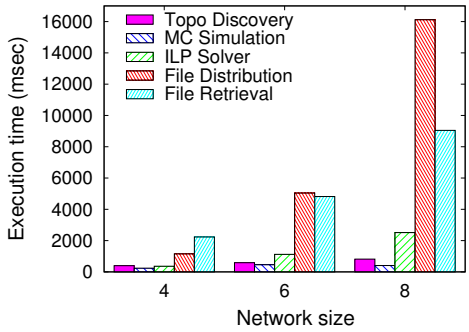

Figure 3: Execution time of different components.

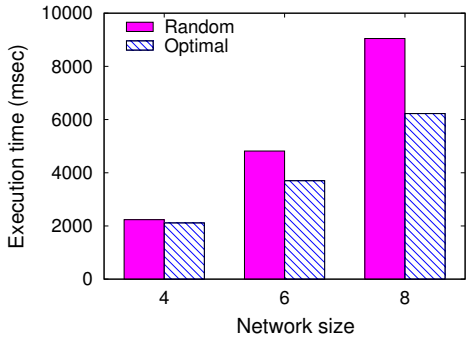

Figure 4: Data retrieval time.

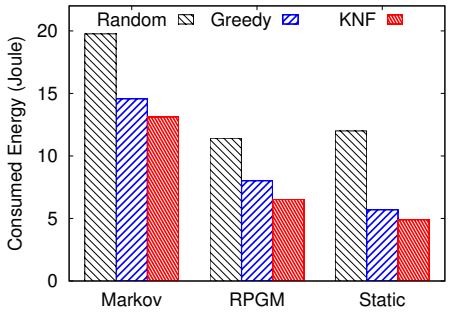

Figure 5: Effect of mobility.

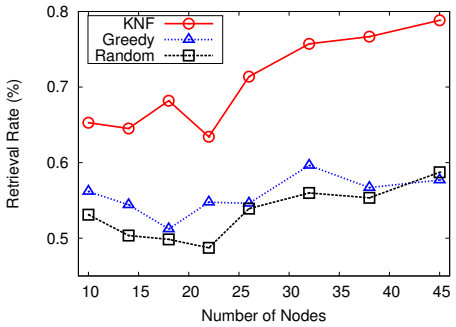

Figure 6: Effect of node density.

We then compared the file retrieval time between our allocation and random allocation. As shown in Figure 4, our framework achieves 15% to 25% smaller 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 algorithm by varying the number of node failures. The face recognition algorithm had an average completion rate of 95% in our experimental setting.

6. SIMULATION RESULTS

We performed simulations to evaluate the performance of our k-out-of-n framework (denoted by KNF) in larger scale networks. We consider a network of $400\times400\text{m}^2$ where up to 45 mobile nodes are randomly deployed. The communication range of a node is 130m. Two different mobility models are tested – Markovian Waypoint Model and Reference Point Group Mobility (RPGM). Markovian Waypoint is similar to Random Waypoint Model in which it 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 set of leaders are selected; leaders move based on Markovian Waypoint; and other nodes follow their closest leaders.

We compare 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 nodes. Random selects storage 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, and completion rate of a job. A node may fail due to two independent factors: depleted energy and an application-dependent failure probability; specifically, the energy associated with a node decreases as the time elapses. The lower the energy is, the higher the failure probability. Each node is assigned a constant application-dependent failure probability. In particular, for the simulations of our k-out-of-n data processing, we also artificially force various number of nodes to fail.

6.1 k-out-of-n data allocation

Figure 5 shows that mobility causes nodes to spend higher energy in retrieving data compared with the static network and the energy consumption for RPGM is smaller than that for Markov. 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.

Figure 6 shows that KNF achieves 15% to 25% higher retrieval rate than others because KNF takes the dynamic nature of the network into account when selecting storage nodes. We also observe that the retrieval rates increase with higher network density. The explanation is that, with higher network density, client nodes can find more reliable paths to storage nodes and the network is less likely to be partitioned due to node failures. Since the simulations run for 4 hours and several nodes fail in the last hour due to depleted energy, the highest retrieval rate of KNF is only 80%.

6.2 k-out-of-n data processing

This section investigates how the failures of processor nodes affect the energy efficiency and job completion rate. 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. In Random, the processor nodes are selected randomly and each task is also replicated to n-k+1

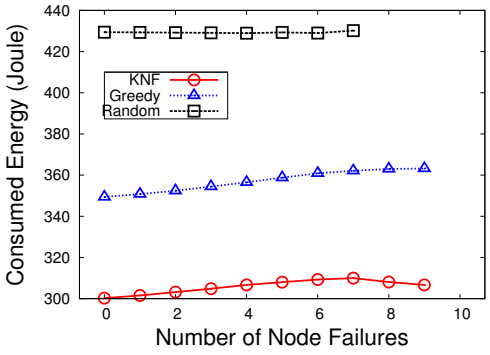

Figure 7: Effect of node failure on energy efficiency.

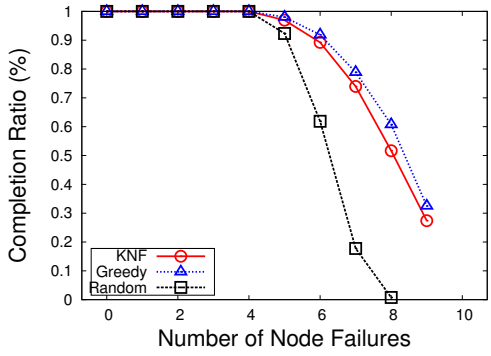

Figure 8: Effect of node failure on completion ratio.

processor nodes randomly. When a node fails, we assume none of the tasks assigned to it can be completed.

Figure 7 shows 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 because given a task, it is replicated to n-k+1 processor nodes and failing an arbitrary processor may have no effect on the execution time of the job at all.

In Figure 8, we see that the completion ratio is 1 when no more than $n-k$ nodes fail. An interesting observation is that Greedy has the highest completion ratio. The main reason is because, in Greedy, the load on each node is highly uneven, meaning that some processor nodes may have many tasks but some may not have a task. This allocation strategy achieves high completion ratio because all tasks can complete as long as one of 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 $1 - \binom{7}{6}/\binom{10}{9} = 0.3$.

7. CONCLUSIONS

We presented the first k-out-of-n data allocation/processing framework that jointly addresses the energy-efficiency and fault-tolerance challenges in dynamic networks. It allocates data fragments to storage nodes such that other nodes retrieve data reliably with minimal energy consumption; when applications need to process data stored in the network, the framework selects a set of processor nodes to execute the job such that the job can be completed with minimal energy consumption and high completion rate. We demonstrated the effectiveness of our framework through both system implementation and simulations in large-scale networks.

Acknowledgment

This work was supported by Naval Postgraduate School under Grant No. N00244-12-1-0035.

8. REFERENCES

[1] M.K. Aguilera, R. Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of Dependable Systems and Networks*, 2005.

[2] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of EuroSys*, 2011.

[3] Douglas S. J. De Couto. *High-Throughput Routing for Multi-Hop Wireless Networks*. PhD dissertation, MIT, 2004.

[4] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of MobiSys*, 2010.

[5] Jiachen Liu David W. Coit. System reliability optimization with k-out-of-n subsystems. *Int. Journal of Reliability, Quality and Safety Engineering*, 7(2):129–142, 2000.

[6] A.G. Dimakis, K. Ramchandran, Y. Wu, and Changho Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99:476–489, 2011.

[7] S. Huchton, G. Xie, and R Beverly. Building and evaluating a k-resilient mobile distributed file system resistant to device compromise. In *Proceedings of Military Communications Conference*, 2011.

[8] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of INFOCOM*, 2012.

[9] D. Leong, A.G. Dimakis, and Tracey Ho. Distributed storage allocation for high reliability. In *Proceedings of ICC*, 2010.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8:14–23, 2009.

[11] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of MobiHoc*, 2012.

[12] A Shokrollahi. Raptor codes. *Information Theory*, 52:2551–2567, 2006.

[13] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of Peer-to-Peer Systems*, 2002.