A Practical Network Coding Approach for Peer-to-Peer Distributed Storage

Marco Martalò[1], Marco Picone[2], Riccardo Bussandri[3], and Michele Amoretti[2]

[1] WASN Laboratory [2]Distributed Systems Group (DSG)
Department of Information Engineering, University of Parma, Italy
marco.martalo@unipr.it, picone@ce.unipr.it, michele.amoretti@unipr.it

[3] Elfo S.r.l.
Fiorenzuola D'Arda, Piacenza, Italy
riccardo.bussandri@gmail.com

***Abstract*—In this paper, we analyze the performance of a peer-to-peer (P2P) distributed storage network based on the overlay architecture defined by the Wuala project. Although the original system is based on efficient erasure codes, e.g., Reed-Solomon codes, we investigate the use of a "simple" network coding strategy, which leverages on the well-known idea of randomized network coding. In particular, when a resource is published in the network, the fragments are randomly encoded with a predetermined overhead, which is consistent with the erasure coding strategy. Moreover, no regular network maintenance is scheduled to guarantee that a client node is able to successfully complete a resource download. We instead propose that a client node generates new fragments, to be stored in the network, when it is able to successfully download the entire resource. Our simulation results show that this simple coding strategy reduces the amount of data flowing in the network, thus obtaining more free disk space on storage nodes.**

I. Introduction

The last decade has seen an exponential growth of innovative Internet-based applications. In particular, peer-to-peer (P2P) applications have been appealing for the research community due to their issues and challenges, e.g., scalability, fault tolerance, etc. Many practical systems have been developed for content distribution (e.g., BitTorrent) and they have obtained a great level of success all around the world.

Another possible application of the P2P paradigm is in the field of distributed storage, where information (e.g., a file) is stored at different nodes in a network, so that it can be retrieved efficiently and with low probability of error. The main examples of P2P distributed storage systems are OceanStore [1], Total Recall [2], and Wuala [3]. In these systems users store their own data on storage nodes using proper encoding functions, thus generating redundancy to counteract possible faults. They can then retrieve the information by downloading a proper subset of the encoded packets, with the property that this subset is sufficient to reconstruct the original information. When the information available in the overall network is going to fail, a maintenance event is scheduled in order to restore a sufficient amount of information. The most popular encoding function is that of an *erasure code* with the property of maximum distance separable (MDS), which allow the best redundancy-reliability tradeoff with respect to classical replication-based systems. Possible erasure codes for storage applications are Reed-Solomon codes, parity-array codes, and low-density parity-check (LDPC) codes [4].

Since the seminal paper in 2000 [5], network coding has promised to increase the throughput and reliability of a large class of systems such as wireless networks, sensor networks, P2P networks, etc [6]. In [7], the authors, leveraging on the topology management of a classical P2P content distribution system, e.g., BitTorrent, have proposed a novel content propagation system. In this approach, the information is encoded at each node by means of randomized network coding [8], [9]. This scheme is shown to be efficient and fault tolerant. This work has motivated a lot of (theoretical and applied) research in the field of network coding for P2P networks. In [10], the authors analyze a P2P distributed storage system where the data is not encoded by using an erasure code. Random linear combinations, instead, are generated and used, showing that the optimal redundancy-reliability tradeoff can be achieved.

In this paper, we propose a practical network coding approach for P2P distributed storage, based on the architecture of the Wuala project, described in more detail in Section II-A. Although a lot of research has been carried out in the field of distributed storage and network coding for this scenario, our work is the first, to the best of our knowledge, that shows the effective gain brought by the use of network coding in a practical P2P distributed storage application. This gain is obtained without maintaining a complete backup of stored files in some central servers. In particular, our analysis has been carried out by simulating the effective behavior of the considered P2P distributed storage network. We first propose to apply some redundancy, defined by an overhead when the resource is published in the network. The maintenance is handled by the client nodes of the network, which regenerate and store other fragments of a resource when they are able to completely download it. Therefore, it is guaranteed that there exists a sufficient number of resource fragments in the network. Our results show that it is possible to achieve the same average availability of a system based on erasure coding but with limited complexity. The free disk space available at the storage nodes can then be increased with the simple network coding strategy we propose.

This paper is structured as follows. In Section II, we briefly discuss on distributed storage systems, with particular focus on the Wuala project. In Section III, we present the network coding-based architecture used in our scheme and relative simulation results are shown in Section IV. Finally, in Section V concluding remarks are given.

II. Related Work

Distributing data for performance, availability, and durability has been widely studied in the file system and database community. In many cases, *repetition coding* or *erasure coding* (e.g., Reed-Solomon or Tornado codes) are used. In information theory, a repetition code is a coding scheme that repeats the bits across a channel to achieve error free communication. An erasure code is a forward error correction (FEC) code, which transforms a message of k symbols into a longer message (codeword) with n symbols such that the original message can be recovered from a subset of the n symbols.

For example, in OceanStore [1] each version of a data object is stored in a permanent read-only form, which is encoded with an erasure code and spread over hundreds or thousands of servers. A small subset of the encoded fragments are sufficient to reconstruct the archived object; only a global-scale disaster could disable enough machines to destroy the archived object. OceanStore is constructed from interacting resources (such as permanent blocks of storage or processes managing the consistency of data). These resources are virtual in the sense that they are not be permanently tied to a particular piece of hardware and can move at any time. Virtualization is enabled by a structured P2P overlay scheme called Tapestry [11].

The Google File System (GFS) [12] is a distributed file system built for hosting the state of Google's internal applications. GFS uses a simple design with a single master server for hosting the entire metadata and where the data is split into chunks and stored in chunkservers.

Another distributed storage system, Total Recall [2], is composed of the following three layers. The Storage Manager layer (TRSM) handles file requests from clients and maintains file availability for those files for which it is the master. The Block Store layer reads and writes data blocks on storage hosts. The Distributed Hash Table (DHT) layer is used to maintain the ID space and to provide scalable lookup and request routing. The TRSM dynamically adapts its mechanisms and policies for efficiently maintaining data according to the availability of hosts in the system.

Dynamo is a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience [13]. Dynamo is built for latency sensitive applications that require at least 99.9% of read and write operations to be performed within a few hundred milliseconds. For this reason it can be characterized as a zero-hop DHT where each node maintains enough routing information locally to route a request to the appropriate node directly.

In the following, we devote more space to the Wuala P2P storage system [3], since our architecture is based on the same overlay scheme, but with a different strategy for distributing and maintaining data.

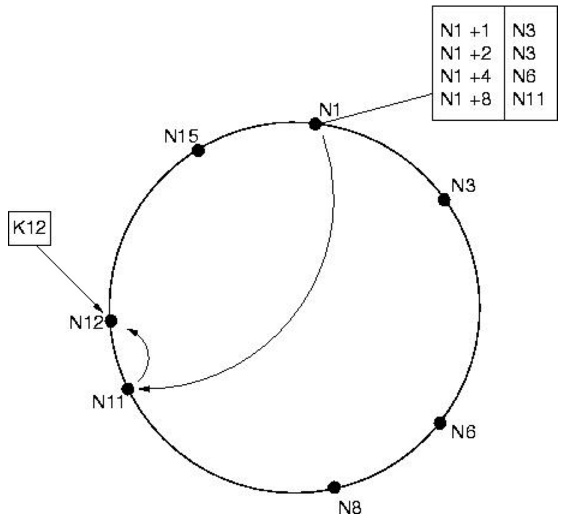


Fig. 1. The finger table entries for node $N1$ and the path taken by a query from $N1$, searching for $K12$ using the scalable lookup algorithm.

A. Wuala

The reason of the success of Wuala [3], which is a commercially distributed storage system with more than 110 million stored files, is its high availability (99.9% guarantee of file preservation). Each new user is immediately allowed to use 1 GB of free storage space. Moreover, users may buy more storage space and possibly sell their own local disk space. The distributed architecture of Wuala is based on Chord [14], probably the most well known P2P overlay scheme adopting the Distributed Structured Model (DSM) [15] i.e. implementing a DHT to store data (in general, information about resources).

In Chord each data block is identified by a unique *key* (a m-bit hash of the block's name) and described by a *value* (typically a pointer to the block's owner). Each node n is assigned a random identifier in the same space of data block keys, and it is responsible for storing key/value pairs for a limited subset of the entire key space. Each node maintains a routing table with up to m entries, called the *finger table*. The i-th entry in the table at node x contains the identity of the first node s that follows x by at least 2^{i-1} on the identifier circle, i.e., $s = successor(x + 2^{i-1})$, where $1 \leq i \leq m$ and all the arithmetic is module 2^m. We call node s the i-th *finger* of node x, and denote it by $x.finger[i]$. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. Fig. 1 illustrates the scalable lookup algorithm based on finger tables. In general, if node x searches for a key whose identifier falls between x and its successor, node x finds the key in its successor; otherwise, x searches its finger table for the node x' whose identifier most immediately precedes the one of the desired key, and then the basic algorithm is executed starting from x'. It is proved that, with high probability, the number of nodes that must be contacted to find a successor in a network composed by N nodes is $O(\log N)$.

With respect to the traditional Chord structure, illustrated above, Wuala defines three node classes: *Super Nodes*, responsible for message routing in the whole network, *Storage Nodes*, whose duty is to provide at least 1 GB disk space for storing data, and *Client Nodes*, which publish and retrieve files. More details are given in Section III.

Data redundancy is achieved by means of erasure coding, i.e., from k pieces in which a file is split, n pieces are generated, so that the file can be reconstructed from any k'-subset of the n-set (in Wuala $k' = k$). After a file has been published by a client node, Super Nodes periodically generate maintenance events with a possible generation of new fragments, in order to guarantee the presence of enough pieces for reconstructing the file.

III. Proposed Network Architecture

The distributed storage system we propose is based on the same P2P approach adopted by Wuala, where peers are both suppliers and consumers of resources, in contrast to the traditional client-server model, where only servers supply and clients consume. Unlike Wuala, we adopt network coding instead of erasure coding, obtaining two main benefits: reduced storage occupancy and reduced file maintenance, with less waste of bandwidth.

As anticipated at the end of Section II-A, the overlay scheme used by Wuala and by our distributed storage architecture is layered, since there are three different groups of nodes with specific behaviors.

Super Nodes build the architecture's kernel, implementing an overlay network based on the Chord overlay scheme. The main purpose of this class of peers is to provide a mechanism for message routing that allows clients to find storage nodes. Super Nodes play a main role and they must be online for 24 hours and 7 days. Therefore, we think that these peers should be managed by the organization that provides the distributed storage service or by any trusted parties. Super Node disconnection or malfunction can propagate serious damage to system functionality, for which Super Node features can't be entrusted to other peers. A Super Node, as mentioned, has a main role in an application based on Chord and has a unique key in the considered keyspace. In order to guarantee storing operations also in unstable situations of the system, each Super Node is also associated to a Storage Node on the same machine. This is also a fundamental operation during the initial phases of the network where there are few nodes that cannot be immediately promoted to the Storage Node class. In this abnormal situation, without "Native Storage Nodes," a client node is not able to find storage space for its requests. These native nodes have the same key of their Super Node and have high storage capacity and also high reliability.

Storage Nodes guarantee a high availability with long sessions and great reliability in the network. They store fragments published by client nodes.

Client Nodes work as data publishers and consumers. They can publish or retrieve resources through the system, as will be discussed later.

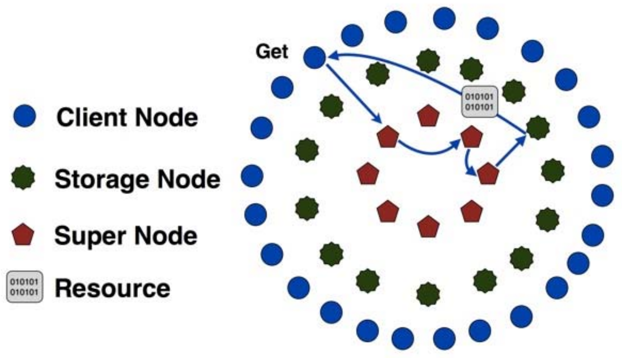


Fig. 2. The considered P2P overlay scheme with an example of resource retrieval process.

Fig. 2 shows an abstract schema of available node classes, placed on separated concentric rings. From the inner to the outer ring, there are Super Nodes, Storage Nodes, and Client Nodes. The "Get" function is also shown and refers to the resource retrieval process, discussed below in the following of this section.

A. Network Join

In order to access the network, a Client Node needs to know at least one Super Node that will manage the creation of a new key for the incoming peer. This key is used to find a Super Node gateway to be used for the communication activities. This kind of operation can be compared with the placement of a resource in the Chord ring, where in this case the resource is the new peer. Key generation is performed like in Chord with a consistent hashing function, by using a SHA-1 algorithm applied to the IP address of the peer. To guarantee the key's uniqueness also with Network Address Translation (NAT) systems typical of modern networks, a suffix is appended to the key.

B. Resource Publishing

In the proposed scheme, when a file must be stored, it is divided into a generation composed by h fragments, which can be collected in a column vector denoted as $\boldsymbol{s} = (s_1, \ldots, s_h)^T$. These fragments are randomly network encoded using a Galois field GF(q) with a pre-determined overhead, and published to storage nodes. Each packet flowing in the network contains both the network coded payload and an header which carries information about the generation to which the fragment belongs and the global coding vector of dimension h. In details, the following procedure is executed to publish a file.

1) A unique key is assigned to the resource, allowing it to route to the responsible Super Node (the one having the nearest key). Moreover, a fixed number of linearly independent randomly network encoded fragments is generated.
2) For each fragment, the Super Node searches for a responsible Super Node. The latter selects the most

suitable Storage Node, among those for which it acts as gateway, applying a load balancing strategy.

3) The Super Node responsible for the resource saves, in a list, the references about the fragments and the corresponding Storage Nodes.
4) At the end of the lookup process, the Client Node can upload generated fragments in a parallel way on selected Storage Nodes. Communications will be created with the lookup features provided by Super Nodes network (Chord based) through specific routing tables.

C. Resource Retrieval

To retrieve a published file, according to the network coding theory, it is necessary to retrieve h linearly independent fragments, using the previously illustrated lookup functionality. After these fragments have been collected, the following system of linear equations is solved using the classical Gauss elimination algorithm:

$$\boldsymbol{\rho} = \boldsymbol{A}\boldsymbol{s}$$

where $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_h)^T$ is a column vector containing the h linearly independent fragments and $\boldsymbol{A}$ is a matrix whose rows correspond to the coding vectors of each retrieved fragment.

Referring to the "GET" function in Fig. 2, the Super Node gateway routes the Client Node's request to the responsible node and then to the Storage Nodes which have the fragments. Once the file has been downloaded, we propose that the Client Node triggers a re-generation process, thus publishing new (linearly independent) fragments.

Fragments re-generation is not always necessary. The Client Node checks, through its gateway Super Node, if the network has been dynamic: if one or more Storage Nodes have been downgraded to Client Nodes within the last 24 hours, a whole new set of linearly independent fragments are re-generated, otherwise each fragment is generated under a probabilistic boolean condition. In other words, before each fragment is generated, a random boolean value is generated; if true, the fragment is generated and published.

D. Evolution from Client Node to Storage Node

The constraint for the evolution of a Client Node to a Storage Node depends on the daily online presence, which must be at least of 4 hours. This presence must be also continuous, being monitored by the gateway for 10 days. After this period, the associated Super Node checks if the client can be moved to the Storage class. If a Storage Node is unavailable for a time superior to 3 days, it is marked as unreliable and downgraded to the Client Node class.

IV. Simulation Results

In this section, we describe the simulation environment and the results obtained with the use of network and erasure coding. The simulation tool used in this paper is DEUS, a discrete event simulator, based on Java and XML, created by the DSG at the University of Parma [16].[1] The simulator is based on the concept of *event* and each new time step is characterized by the generation of an event. The network's time evolution is given in terms of *virtual time* (VT), i.e., each virtual time unit in the network evolution corresponds to the generation of an event. In the following, we will refer, without distinction, to time and VT.

A. Set-up

The following scenario has been considered in order to evaluate the performance of the proposed P2P distributed storage system:[2]

- 10 supernodes with the associated storage nodes;
- 50 storage nodes with a maximum 10 GB disk space each;
- 500 client nodes which can generate resources with a maximum size of 35 MB.

At the beginning of the simulation 10 storage nodes and 80 client nodes are randomly disconnected from the network and the network evolution is analyzed from this point on. The analysis is carried out for 30 days and during each hour 100 events are scheduled (corresponding to generations and searches of resources in the network), so that the overall simulation time is

$$24\,\text{hours} \times 30\,\text{days} \times 100\,\text{events} = 72000 \ \text{VTs}.$$

The resources to be published are divided into generations of h elements, so that the overhead in the packet header containing the coding vector is h bytes (if the operations are in $GF(2^8)$). The simulation results are averaged over 5 independent trials in order to partially reduce statistical fluctuations. In particular, during every run our simulator generates different connections, disconnections, and publishing/download events. As aforementioned, longer simulations are matter of future work.

B. Results

The first analysis carried out in this paper is the resource availability, defined as the success rate of downloading a resource by a client node. In the original system, an average availability of 99.99% is obtained by using erasure coding. Note that in the Wuala project a copy of published resources is stored in remote central servers. This situation has not been simulated in our environment, since we are interested in a complete distributed system where a user is able to download a resource without the help of any central entity. To this end, we have performed simulations by scheduling searches every 10 VTs, whereas connections and disconnections are scheduled every 20 VTs. We also set a maximum percentage of 40% of client nodes which can be promoted to storage nodes, according to the protocol described in Section III-D. The publication

[1]This simulator has been used instead of classical network simulators, e.g., ns-2 or Opnet, since it is optimized to analyze P2P networks at a higher level (up to the overlay network).

[2]Simulations of more complicated networks, e.g., more nodes, were not possible, due to the large time and computational complexity needed. This issue will be solved using the new parallel version of the simulator under development.

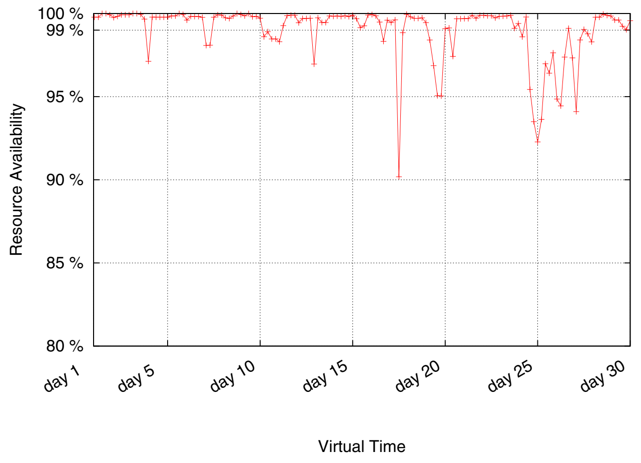


Fig. 3. Resource availability as a function of the virtual time. The considered field size is 2^{16} and the generation size is $h = 100$.

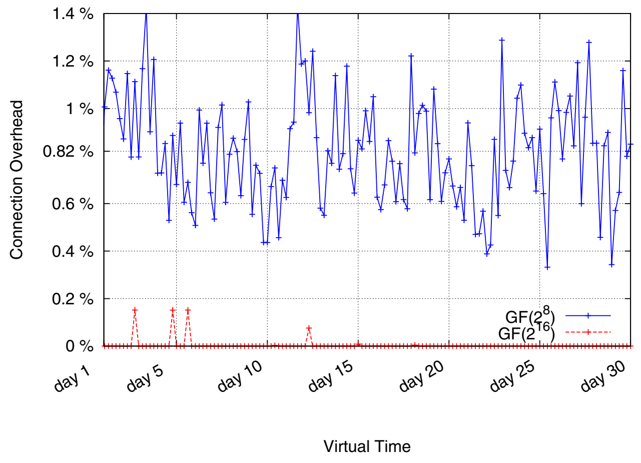


Fig. 4. Connection overhead as a function of the virtual time. Two possible values for the field size are considered: 2^8 and 2^{16}.

overhead is set to 100%, i.e., a number of fragments equal to the double of those created for each generation is published), whereas the maximum overhead of the regeneration after each download is set to 15%. Moreover, if *churn* is detected, i.e., there is a high rate of connections/disconnections of peers from the network, the overhead increases by the 10% of the generation size, in order to compensate for this phenomenon.

In Fig. 3, the resource availability is shown as a function of the virtual time, for a field size equal to 2^{16} and generation size $h = 100$. One can observe that, on average, the resource availability is around 99%. To obtain this result it is necessary that a given number of storage nodes is connected to the network, so that the number of linearly independent fragments in the network is sufficient to retrieve the information. Our results show that this number is always guaranteed during the network functioning. However, at the 17th day a performance loss can be observed and the resource availability drastically reduces to 90%. Our results have shown that this is due to the fact that there is a number of storage nodes in the network which have been recently promoted. Therefore, they do not store a sufficient number of fragments to guarantee a successful resource retrieval.

In Fig. 4, the connection overhead is shown, as a function of the virtual time, for two possible values for the field size: 2^8 and 2^{16}. The connection overhead is defined as the percentage of extra connections required to successfully retrieve a resource. For instance, if $h = 50$ fragments should be downloaded to retrieve the entire resource and 5 extra connections are performed to obtained h linearly independent fragments (due, e.g., to some previous linearly dependent fragments), the connection overhead is $5/50 = 0.1 = 10\%$. One can observe that, by using a sufficiently large Galois field (e.g., GF(2^8)), it is possible to obtain a very low connection overhead, around 0.82%. As expected, if the field size increases to $2^{16} = 65536$, the overhead further reduces and goes to almost zero. One may observe that this field size may be too large. However, the coding vector for generations of 50 elements has a size equal to

$$16\,\text{bits} \times 50 = 100\ \text{Bytes}$$

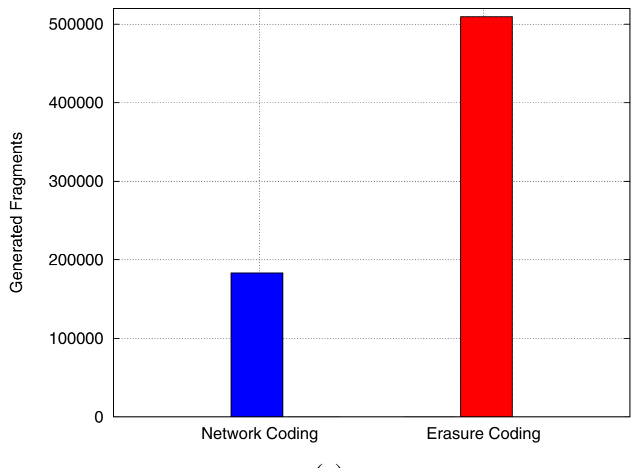
and, therefore, the overhead with respect to a payload of 500 KB is equal to 0.02%. Consequently, it is possible to practically use this field size in the considered P2P distributed storage scenario.

In Fig. 5, a comparison between the performance of erasure and network coding is proposed. Two performance indicators are considered: (a) the number of generated fragments and (b) the average per storage node free available disk space as a function of time. One can directly observe that our "simple" network coding strategy allows us to drastically reduce the number of generated fragments from almost 5×10^5 to almost 2×10^5 (Fig. 5 (a)). This is due to the fact that in the presence of erasure coding a large number of maintenance events are scheduled to have a sufficient number of fragments in the network to retrieve the complete resource. Each maintenance event generates new fragments to be stored in the network and increases the total number of fragments in the network. Let us now define a percentage saving indicator $\epsilon_{\rm f}$ as

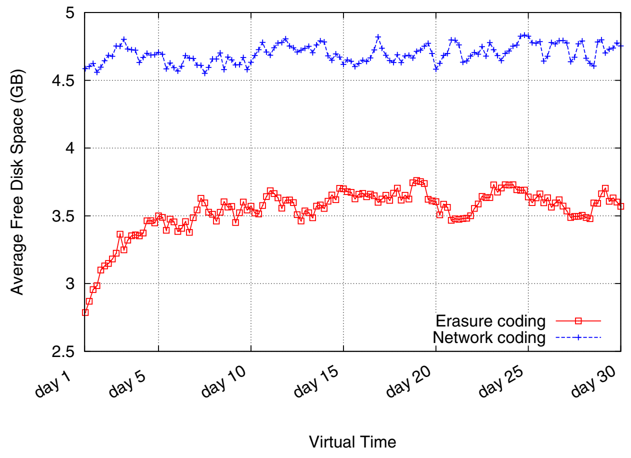
$$\epsilon_{\rm f} = \frac{n_{\rm f}^{\rm EC} - n_{\rm f}^{\rm NC}}{n_{\rm f}^{\rm NC}} \times 100$$

where $n_{\rm f}^{\rm EC}$ and $n_{\rm f}^{\rm NC}$ denote the number of generated fragments with erasure and network coding, respectively. In the case of Fig. 5 (a), a saving $\epsilon_{\rm f} = 178\%$ can be obtained.

This is confirmed by the results in Fig. 5 (b), where the evolution of the average available free disk space per storage node is shown as a function of the time. As one can see, network coding allows us to save, on average, more than 1 GB with respect to the classical erasure coding technique, since it generates less fragments. Since the maximum per node available disk space is 10 GB, it is possible to achieve a saving of 10%. Therefore, although both techniques are applicable to this scenario with good results in terms of resource availability, our approach based on network coding



(a)



(b)

Fig. 5. Comparison between the performance of erasure and network coding. In case (a), the number of generated fragments is considered, whereas in case (b) the average per storage node free available disk is shown as a function of time.

should be preferred. In fact, it is able to dynamically trace the network evolution, guaranteeing, at the same time, lower storage space occupation. Moreover, the resource maintenance is easier for the Client Nodes, since they do not regularly check the resource availability, but re-generate fragments every time the resource is downloaded, as explained in Section III-C. In the near future, our simple approach can be extended and improved, in order to obtain larger gains, especially in terms of available free disk space.

V. Concluding Remarks

In this paper, we have proposed a P2P distributed storage network based on the use of random network coding instead of classical erasure coding. Our architecture is based on that defined by the Wuala project. The proposed practical network coding approach is characterized by two aspects. First, when a resource is published in the network the fragments are randomly network encoded with a pre-determined overhead consistent with the erasure coding strategy. Also, no regular network maintenance is scheduled to guarantee that a Client Node is able to successfully complete the resource download. In our approach, instead, we propose that Client Nodes generate novel fragments to be stored in the network every time they are able to successfully download the resource. Our analysis has been based on simulation results. The main result is that this "simple" coding strategy reduces the amount of data flowing in the network, thus obtaining more free disk space on the storage nodes, at the price of a very small reduction in the resource availability. In the near future, we will consider other (properly designed) network coding strategies, in order to obtain more benefits in terms of both available disk space and resource availability. Moreover, we are going to perform more complex simulations and we will implement a prototype to be tested with PlanetLab.

References

[1] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. D. Kubiatowicz, "Pond: the OceanStore prototype," in *2nd USENIX Conf. on File and Storage Technologies* (FAST), San Francisco, CA, USA, March 2003, pp. 2235–2245.

[2] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total Recall: Systems support for automated availability management," in *Proc. USENIX Symp. on Networked Systems Design and Implementation* (NSDI), San Francisco, CA, USA, March 2004.

[3] "The WUALA Project," URL: `http://www.wuala.com`.

[4] J. S. Plank, "Erasure codes for storage applications," in *4th USENIX Conf. on File and Storage Technologies* (FAST), San Francisco, CA, USA, December 2005.

[5] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[6] C. Fragouli and E. Soljanin, *Network Coding Applications*. Hanover, MA, USA: Now Publisher Foundations and Trends in Networking, 2007.

[7] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. IEEE Conf. on Computer Commun.* (INFOCOM), Miami, FL, USA, March 2005, pp. 2235–2245.

[8] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, and B. J. S. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inform. Theory*, vol. 52, no. 10, pp. 4413–4430, October 2006.

[9] C. Fragouli and E. Soljanin, *Network Coding Fundamentals*. Hanover, MA, USA: Now Publisher Foundations and Trends in Networking, 2007.

[10] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. O. Wainwright, and K. Ramachandran, "Network coding for distributed storage systems," Submitted to *IEEE Trans. Info. Theory*, 2008.

[11] K. Hildruw, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," in *40th ACM Symposium on Parallel Algorithms and Architectures* (SPAA), San Francisco, CA, USA, March 2002, pp. 41–52.

[12] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. ACM Symp. on Symposium on Operating Systems Principles* (SOSP), Landing, NY, USA, October 2003, pp. 29–43.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review Archive*, vol. 41, no. 6, pp. 205–220, December 2007.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, CA, USA, March 2001, pp. 149–160.

[15] M. Amoretti, "A survey of peer-to-peer overlay schemes: Effectiveness, efficiency and security," *BSP Recent Patents on Computer Science*, vol. 2, no. 3, pp. 195–213, September 2009.

[16] M. Amoretti, M. Agosti, and F. Zanichelli, "DEUS: a discrete event universal simulator," in *2nd ICST/ACM International Conference on Simulation Tools and Techniques* (SIMUTools), Rome, Italy, March 2009.