Building a Network Coded Storage Testbed for Data Center Energy Reduction

Ulric J. Ferner, Qian Long, Marco Pedroso, Luis Voloch, and Muriel Médard

***Abstract*—The energy consumption of data centers (DCs) worldwide is approaching that of countries such as Argentina. This paper builds upon the analytical work presented in [1], which showed that reductions in blocking probability, using algorithms such as network coded storage (NCS), can significantly reduce DC energy consumption. We demonstrate that NCS can be deployed in a DC management software with a commercial grade feature set, without interference. We develop a simple method to estimate blocking probability DC networks, and present preliminary blocking results for a single-server single-drive node. The next step of this project is to use the insights gained herein to develop more accurate queueing models for delay and blocking characteristics of DC servers and drives.**

***Index Terms*—Blocking probability; data centers; energy efficiency; network coded storage; storage area networks; testbeds.**

I. Introduction

PROJECTIONS indicate that the worldwide DC industry will require a quadrupling of capacity by the year 2020 [1]. DCs are designed with the goal of serving large numbers of content consumers concurrently, while maintaining an acceptable user experience. One aspect of maintaining a quality user experience is keeping blocking probability small, or the probability that content is unavailable when requested. This paper builds upon the analytical work presented in [1], which showed that reductions in blocking probability, using algorithms such as NCS, can significantly reduce DC energy consumption. We are implementing a testbed to refine the DC network models, and to get early estimates of potential energy savings. This paper outlines the testbed design and select preliminary results. In particular, the contributions of this paper are:

- We demonstrate that NCS can be deployed in DC management software with a commercial grade feature set using exclusively HTTP GETs for communication;
- We develop a simple method to estimate blocking probability in real systems; and
- We provide preliminary blocking probability results for a single-drive single-server system.

The next step of this project will be to develop more accurate queueing models for the delay and blocking characteristics of typical servers and drives, as a function of incoming traffic. Potential follow-on from this project include demonstrating that NCS can improve the quality of service of data centers under normal operating conditions, and translate to significant energy savings.

This material is based upon work supported by the Jonathan Whitney MIT fellowship, Alcatel-Lucent under award #4800484399, the Air Force Office of Scientific Research under awards #FA9550-09-1-0196 and #FA9550-13-1-0023, Georgia Institute of Technology under award #RA306-S1, and France Telecom S.A. under award #018499-00. U. J. Ferner and M. Médard are currently with, and Q. Long, M. Pedroso, and L. Voloch were with the Research Laboratory for Electronics, Massachusetts Institute of Technology, Room 36-512, 77 Massachusetts Avenue, Cambridge, MA 02139 (e-mail: uferner, qlong, marco_p, voloch, medard@mit.edu).

II. Preliminaries

A. An Intuitive Example

This example is adapted from [2]. Consider Fig. 1 which depicts content with two segments f_1 and f_2 and compares two four-drive systems. System (1) is a replication system storing f_1, f_1, f_2, f_2, and System (2) is a coding system storing $f_1, f_2, f_1+f_2, f_1-f_2$. Assume a user requests both f_1 and f_2. In System (1), the user will not be blocked if they access at least one of the top two drives storing f_1, and at least one of the two drives storing f_2. In System (2), the user be will not be blocked if they access and are able to decode *any two* drives.

B. Theoretical Background

We give a brief overview of the theoretical background presented in [1]. Consider a DC storing one or more copies of file f composed of T chunks $f = \{f_1, \ldots, f_T\}$, where f_i is the ith chunk. All incoming user requests for content arrive at an assembler server S in the DC network, where S is connected to a number of drives through an internal DC network. In the uncoded storage (UCS) scheme, each drive stores a subset of the chunks $\{f_i\}_{i=1}^T$. In the NCS scheme, each drive stores coded versions of $\{f_i\}_{i=1}^T$ as follows. Let $c_i^{(k)}$, $i \in \mathcal{B}_l$ be the coded chunk corresponding to the kth replica of f_i, contained in the lth block[1]. Coded chunk $c_i^{(k)}$ is some linear combination of all uncoded chunks in the same block that contains f_i,

$$c_i^{(k)} = \sum_{p \in B_l} \alpha_{p,i}^{(k)} f_p^{(k)} \tag{1}$$

where $\alpha_{p,i}^{(k)}$ is a column vector of coding coefficients drawn from a finite field $\mathbb{F}_q$ of size q [3], and where $f_p^{(k)}$ is

[1]File f is divided into equal-sized blocks, where each block is a group of r chunks.

978-3-901882-56-2

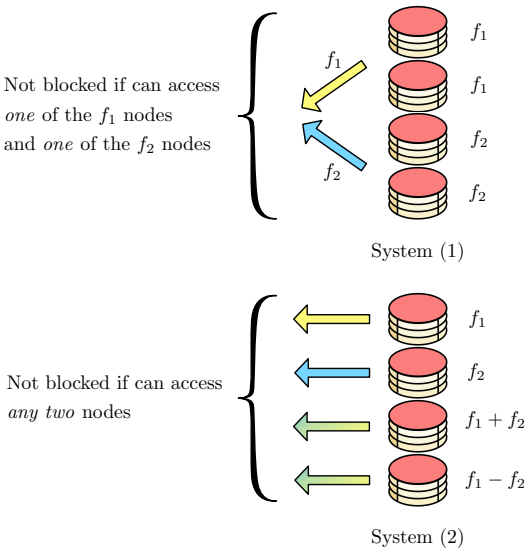


Fig. 1. Adapted from [2]. Intuition behind the system presented in this paper. Assume a user requests both f_1 and f_2. System (2), has a great number of ways to service both the information in f_1 and f_2 than System (1).

treated as a row vector of elements from $\mathbb{F}_q$. We assign coding coefficients that compose each $\alpha_{p,i}^{(k)}$ with a uniform distribution from $\mathbb{F}_q$, mirroring random linear network coding (RLNC) [4]. In this paper we assume that the arrival of user requests does not prompt the update or cycling of coded chunk coefficients. In the NCS scheme, coded chunk c_i provides the user with partial information on all chunks in its block window. When a read request arrives for a coded chunk, the relevant drive transmits both $c_i^{(k)}$ as well as the corresponding coefficients $\{\alpha_{p,i}^{(k)}\}$.

III. Testbed Design & Implementation

This section first discusses the NCS library design and parameters. Second, it details the techniques used to estimate blocking probability.

A. Testbed Structure

(a) Network hardware architecture: There are three types of machines in our testbed, as per Fig. 2. DC network nodes communicate exclusively using HTTP GETs.

- *User node:* External to the DC network. A user node represents one or more users, where each user requests either file f or a file chunk f_i with a single HTTP GET request. A single user node has been built using an Intel Core 2 Quad CPU at 2.83 GHz machine with 2.7GiB RAM. It has a single 100Gb/s ethernet card, generating up to 64 000 simultaneous unique user requests.
- *Assembler:* A member of the DC network. Assembler node S stores no content and is connected to K fetcher nodes. It receives user node GET requests and coordinates loading of content from one or more fetchers. When users request f, it combines chunks from fetchers into a contiguous file stream for each user. Each assembler has been implemented as a Amazon Web Services (AWS) micro-instance, with a single core 64-bit 613 MiB machine.
- *Fetcher:* A member of the DC network. A fetcher stores and services file chunks based on HTTP GET requests from assemblers. The file chunks can be stored as either UCS or NCS chunks. For simplicity, each fetcher has only a single hard drive. Each fetcher has been implemented as a AWS micro-instance, with a single core 64-bit 613 MiB machine.

(b) User node software: The user node uses the open source `curl loader` package written in C to generate an application load of thousands of HTTP clients [5]. We use fresh connections for each user, and each user is assigned a unique IP address or port number. A user requesting content makes an HTTP GET request with filename {*e.*}`filename.b`, where the optional prefix `e.`, if present, denotes NCS and the `.b` postfix denotes the file block number of interest for `filename`.

(c) Assembler & fetcher software: Each server runs commercial-grade DC management software with the NCS library written in C++. The software is a modified version of Vidscale's Mediawarp software under academic license [6]. Mediawarp has a commercial DC management feature set which includes transparent caching, rate throttling and real-time content caching based on hot and cold content. All machines run Linux Fedora 16, and the DC network nodes run a lighttpd webserver. The `json spirit` and `poco` v1.4.1 open source packages are used. The academic version of Mediawarp that runs on each computer is composed of three programs running simultaneously:

- *Conductor:* Receives HTTP GET requests, parses headers, and finds content;
- *Provisioner:* Manages content placement, with the ability to duplicate content onto multiple hard drives if content is under high demand;
- *Relayer:* When redirecting user requests, caches content locally as it is being streamed to the user from a particular node

B. NCS Library

The NCS library is designed to divide large files into chunks and to then encode those chunks using block-based RLNC. This includes both (a) the encoding of files into chunks and storing those coded chunks on the server drive; and (b) decoding a number of coded chunks back into the original file.

The encoding and decoding of content is written in C++ and uses log-lookup tables, with speed improvements based on those presented in [7]. The package can use either 8 bit or 16 bit field sizes for network coding, i.e., a finite field $\mathbb{F}_q$ of size $q = 2^8$ or $q = 2^{16}$. The NCS library was integrated into the Mediawarp software by writing a package to take control of the software after the HTTP

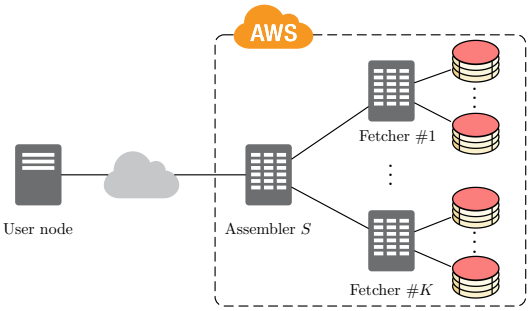


Fig. 2. The testbed is composed of three node types. One or more user nodes that flood the single assembler S with HTTP GET requests for subsets of $\{f_i\}_{i=1}^{T}$. Assembler S is connected to K fetcher nodes. The assembler and fetchers form the DC network and run content management software, as well as the NCS library. To allow for future scalability, they are implemented in AWS.

GET header is parsed for each read request, and return control prior to relaying or caching content. The search for content and the redirecting of content between nodes are done by the NCS library. The relaying of content to caches is done by Mediawarp. Each time a single read request arrives for file$f$ Algorithm 1 is executed.

Algorithm 1 In the NCS scheme, each received user request begins an instance of the following algorithm on the assembler.

Parse the received HTTP GET header
Search system state file for requested content and generate multiple HTTP GET requests to fetchers for coded chunks of file f_i
Receive a degree of freedom (d.o.f.) `filename.b.c` from a fetcher
Check if there are sufficient degrees of freedom to decode
Append `b.c` to the list of received chunks
if Degrees of freedom received is sufficient for user and have not timed out **then**
  Assemble file
  Relay file back to user
else
  Resend HTTP GET requests to all fetchers in cluster, appending list of received chunks
end if

C. Blocking Probability Estimation

Measuring blocking probability directly in data centers can be nontrivial since all measurements are for an instant in time. In this paper we calculate an estimate of blocking probability instead.

Definition *Blocking probability estimate* P_b^e is the probability that a random user who has sent the DC an HTTP GET at time t does not receive any information back from the DC by time $t + x$, where x is measured in seconds.

In this paper we set $x = 5$ seconds. This is consistent with studies that have shown that users begin to abandon a video if it does not start within 2 seconds. Beyond two seconds, each additional second of delay results in a roughly 6% increase in abandonment rate [8]. Hence, our system pins abandonment at approximately 18%, a sizable portion of the user population. such as [9].

We estimate P_b^e as follows. We flood a server with simultaneous user requests from remote machines. All HTTP GETs are transmitted simultaneously to S, and all users request the same coded or uncoded video. A user is blocked if they receive no response from the server within $x = 5$s or if they receive an error message such as an HTTP 404 response.

IV. Preliminary Results

We present both preliminary speed test results for the stand-alone NCS library for comparative purposes, and blocking probability proxy results for a single fetcher node.

A. NCS Library

We performed encoding and decoding speed tests of the NCS library. The speed differences between the 8 bit and 16 bit field sizes were found to be negligible in most cases.

The speed of encoding in most cases is not important because encoding can be done offline prior to distributing content to users. However, if decoding speeds cannot keep up with transmission demands, then the feasibility of NCS for streaming video comes into question.

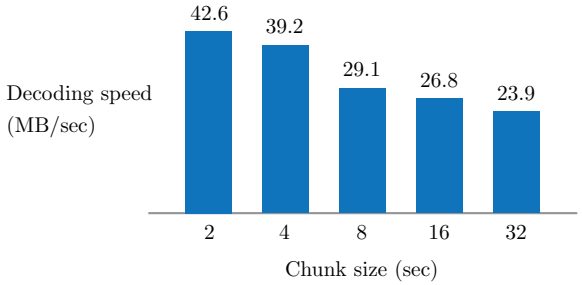
We tested decoding speed on an Fedora 16 Linux machine with Intel Core 2 Quad CPU at 2.83GHz with 2.7GiB memory, a typical 1U server unit in a modern DC rack. It was found that speeds were relatively consistent between differently sized video files. All presented tests are on a 62MB 153 second HD1080p movie trailer. Fig. 3(a) demonstrates that, given a fixed block size, decoding speeds are relatively stable across various chunk sizes. This is comparable to the speeds attained in communication-based network coding testbeds [10].

Fig. 3(b) shows the effect of changing block size, demonstrating that a block size of 1-8 units provides near maximum decoding speed. Decoding speed is significantly degraded if block sizes are too large, e.g., greater than 32 chunks. Although overall decoding speeds are comparable to communication-based network coding testbeds, to maintain such speeds the typical chunk sizes may need to be an order of magnitude larger, and the block sizes an order of magnitude smaller.

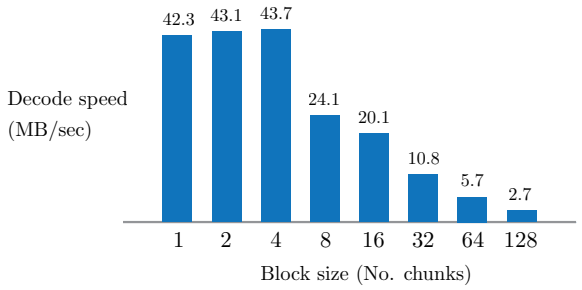
B. Blocking Probability Estimation

Reference [1] showed that blocking probability is a function of the incoming user traffic. Using `curl loader` we measure P_b^e as a function of the number of users present in the system n. For each n, 50 trials were run and confidence intervals are calculated across all data points.

To maintain quality of service, DCs manage both delay and blocking probability. We measure the average delay

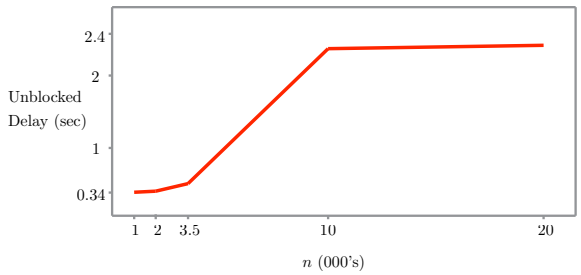


(a) Effect of chunk size on NCS library decoding speed. The block size is set to seven chunks. Chunk size is measured in seconds of an HD1080p video.

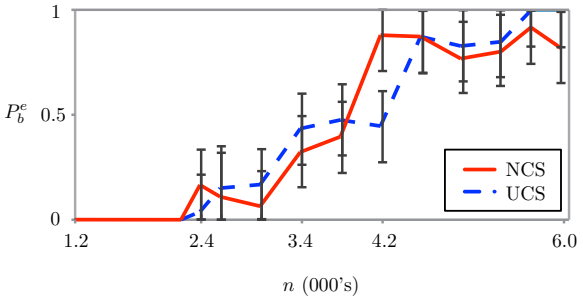


(b) Effect of increasing the number of chunks (block size) on decoding speed, given a fixed 5MB block size. Tests were performed on an 480p video.

Fig. 3. Effect of varying chunk and block size on decoding speed.



(a) Average NCS unblocked delay for one single-drive fetcher, as a function of the number of users n.



(b) NCS and UCS blocking probability estimates as a function of the number of users n. Theory posits that there should be no significant gain from NCS over UCS in the single-drive single-server system [1].

Fig. 4. Fetcher delay and blocking probability estimates.

and P_b^e for a single fetcher being flooded by the user machine with a fetcher ping time of approximately 20ms. The file chunks are small encoded text files of size 200KB. Fig. 4(a) plots the average delay per chunk for all *unblocked* user requests as a function of n. We make two observations. First, the unblocked delay does not increase significantly until n is close to 3 000. Second, the unblocked delay per chunk is approximately bounded at 2.5 seconds by the commercial-grade software. Now consider Fig. 4(b) which plots P_b^e as a function of n. Users begin to be blocked at a smaller n than that which causes substantial unblocked delay. This means that the fetcher is sacrificing users to keep the delay bounded for a small number of users. Blocking probability reaches about 50% prior to seeing substantial unblocked delay. This motivates careful future analysis of the relationship between unblocked delay and blocking probability.

V. Conclusions

We are continuing to build software that helps analyze DC performance. These results will enable us to understand NCS DC energy reduction opportunities.

We have already shown that chunk and block size selection can be markedly different from communication-based network coding systems, and have implemented a technique to estimate blocking probability. The next step of this project involves developing statistical models for the blocking characteristics of servers and drives. We are excited about the promise of demonstrating significant DC energy efficiency gains through NCS.

References

[1] U. J. Ferner, M. Medard, and E. Soljanin, "Toward sustainable networking: Storage area networks with network coding," in *Proc. Allerton Conf. on Commun., Control and Computing*, Champaign, IL, Oct. 2012.

[2] U. J. Ferner, T. Wang, M. Médard, and E. Soljanin, "Resolution-aware network coded storage," *CoRR, http://arxiv.org/abs/1305.6864*, May 2013.

[3] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[4] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[5] R. Iakobashvili and M. Moser, "Curl loader," 2007. [Online]. Available: http://curl-loader.sourceforge.net/

[6] "Vidscale corporation," 2012. [Online]. Available: http://www.vidscale.com/

[7] C. Huang and L. Xu, "Fast software implementation of finite field operations," Washington University in St. Louis, Tech. Rep., 2003.

[8] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proc. Internet Measurement Conf.*, Boston, MA, Nov. 2012.

[9] J. Nielsen, "Top ten mistakes in web design," 1996. [Online]. Available: http://www.useit.com

[10] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Médard, "Network coded TCP (CTCP)," *CoRR, http://arxiv.org/abs/1212.2291*, Apr. 2013.