Data Compression Cost Optimization

Eyal Zohar*, Yuval Cassuto†

*Yahoo! Labs
eyalz@yahoo-inc.com

†Technion - Israel Institute of Technology
ycassuto@ee.technion.ac.il

Abstract

This paper proposes a general optimization framework to allocate computing resources to the compression of massive and heterogeneous data sets incident upon a communication or storage system. The framework is formulated using abstract parameters, and builds on rigorous tools from optimization theory. The outcome is a set of algorithms that together can reach optimal compression allocation in a realistic scenario involving a multitude of content types and compression tools. This claim is demonstrated by running the optimization algorithms on publicly available data sets, and showing up to 25% size reduction, with equal compute-time budget using standard compression tools.

1 Introduction

Data compression is an essential tool in almost every computing environment, reducing the amount of data sent over networks and stored in various devices. Compression algorithms are now widely deployed in a variety of components and sub-systems within the core infrastructure of large information-technology systems. In particular, lossless data compression now drives two very central information services: *web networking* (via compressed HTML content), and *virtual/cloud storage* (via inline-compressed block storage). In both environments, and more generally so, the system objective is to maximize the utility of compression given the limited available computing resources.

Indeed, deciding how to allocate computing resources to compression tasks is a highly non-trivial problem, with significant ramifications on communication and storage efficiency. Unfortunately, known research results give only a very partial answer to this compression-optimization problem. Prior solutions to this problem considered either specific or limited scenarios, which are hard to extend to the reality of complex systems charged with compressing massive and heterogeneous data sets.

In this work we develop and study a novel framework toward finding the optimal allocation of computing resources for content pending compression in the system. The main strength of this framework lies in its generality. The system's compression tools and its incident data are specified using abstract parameters that are easily computed from the pending compression task. Then the optimal allocation is found using rigorous tools from optimization theory. For the sake of clarity, we present the framework in two steps: in Section 3 the optimization is explained when a single type of document is pending compression; then the fully heterogeneous scenario of multiple document types is addressed in Section 4. An evaluation on massive real data sets in Section 5 shows that the new framework can improve the compression efficiency for every computing budget, and typically by significant amounts.

2 Background and Related Work

We survey some background and prior work on compression in systems with limited computing resources. It is now a standard feature for compression tools to offer an effort-adjustment parameter that allows the user to trade CPU resources for compression ratio. Typically this parameter is specified as an integer level in the range 1 to 9.

There is an extensive research on compression performance in the context of energy-awareness in both wireless [1] and server [2] environments. Inline compression decision [3] presents an energy-aware algorithm for coarse-grain optimization, i.e., answering the "compress or not compress" question per MapReduce job. Similarly, the same "compress or not" question is addressed for real-time storage systems in [4] by a fast sampling of the content compressibility. Finer-grain adaptive compression [5] mixes compressed and uncompressed packets in CPU-bound systems, and [6] extends the mixing idea when a single document is compressed with a novel parallelized compression tool. A web-server system implementing dynamic load-aware elastic compression is detailed in our recent previous work [7].

To the best of our knowledge, this paper presents the first general optimization framework not tailored to a specific application, with ability to handle heterogeneous content and a variety of compression tools simultaneously. In particular, it can also work with relatively new compression techniques [8–10].

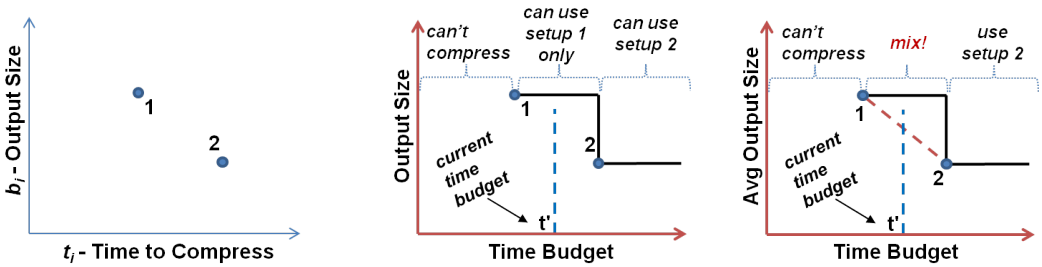
3 Setup Mixing Optimization, Single-Document

In this section we present an algorithm that finds the optimal setups given a single document to compress and a time budget for completion, using a given set of compression tools. The term *document* refers to some data object we wish to compress, with the idea that either the data object is very large or there are many instances of objects similar to it (by some content characteristics). Optimality in this context means that the data-size reduction achieved by the compression is maximized over all setup assignments allowable by the time budget. We further show that mixing as few as two setups is always sufficient for optimality. Building upon this notion of optimal setups and mixing for a single-document, in the next section we present an optimization algorithm for a multiple-documents system.

3.1 Setup mixing

A key feature of our algorithm is a fine-grained effort adjustment. The interpretation we give to the fine-grained operation is a weighted mix between several compression setups, where each setup may be a different compression tool and/or a configuration of such a tool.

To illustrate what setup mixing is, we start with a conventional compression of a single file, and only then turn to present the concept of setup mixing. Suppose we need to run a conventional compression job of a file f that must be completed within a given time budget. We first obtain an estimate of the size reduction and effort



(a) Two sample setups and their performance for f: compression time and output size.

(b) Simple non-mixing algorithm - a budget strictly between the points 1 and 2 cannot be fully utilized.

(c) Mixing setups 1 and 2 achieves full processing utilization and reduces the output size.

Figure 1: The motivation for setup mixing, single-document.

exhibited by each of the compression setups. Figure 1a shows the time it takes to compress f in each of two compression setups 1 and 2, and the resulting compressed size of f. Each setup may be a different compression tool and a specific configuration; for example, setups $1, 2$ may be, for example, two different configurations of the gzip utility. The x coordinate designates the time each setup takes to compress f. The y coordinate designates the resulting compressed size of f achieved by each setup. Setup 2 has a higher computation effort (longer running time) but better (smaller-size) compression outcome compared to setup 1.

In Figure 1b we see how the two setups might be used by a naïve *non-mixing* algorithm to compress f. The x axis in Figure 1b expresses the instantaneous time budget allowed for the compression of f. Suppose that the current time budget is t' time units, as marked with a vertical dashed line on Figure 1b. It is clear from the plot that setup 2 cannot be used, given the time budget t'; on the other hand, setup 1 does not use the entirety of t'.

Figure 1c shows a desirable mix of setups 1 and 2, which fully utilizes the intermediate processing time budget t' between the x locations of the two points. Ultimately, any budget t' satisfying $t_1 < t' < t_2$ can be utilized, yielding an improved average compressed size of b' satisfying $b_2 < b' < b_1$. In graphical view, that means that several points on the line connecting points 1 and 2 would be achieved by mixing setups. In more general terms, mixes allow achieving points on any line connecting two different setups in these plots.

In practice, mixes can be performed in one of several ways: (a) When parallel compression [11] is applicable, each thread may use a different setup. (b) In a web-server, which typically serves several requests at once, mixing is achieved by changing the fraction of requests served by each setup [7]. (c) With large files, it may be possible to switch configurations after every block of input bytes (as can be done with standard gzip and a wrapping script).

3.2 Finding the useful mixes

Given the aforementioned motivation for mixing setups, we now present the algorithm that finds the useful mixes of setups. To this end we explained why mixing two setups is beneficial. Now, we show how mixes can be further enhanced when the content's nature is known in advance. The elements of the algorithm are now defined:

mix A mix is a set of at least two different setups used to serve a compression job of a document. For example, the mix 3,5 means that part of the document is compressed with setup 3 and the rest with setup 5.

useful mix A useful mix is a mix that provides the smallest compressed output size for some time budget. When a specific instantaneous time budget is given, we refer to the corresponding useful mix as *the optimal mix*.

useful setups A useful setup is a setup that participates in at least one useful mix. Hence useful setups form a subset of the setups. The meaning of useful setups and useful mixes is explained in the remainder of this section.

In Proc. 1 below we present an algorithm to obtain the useful setups, which will later be shown to span all the useful mixes, and each of the useful mixes utilizes up to two of those useful setups for any time budget. The input to Proc. 1 are pairs $\{(t_i, b_i)\}$, one for each setup, denoting the time and size resulting from compressing the content with setup i. The algorithm first removes setups that result in a larger size compared to a setup with lower computation time, or consume the same computation time as a setup with no larger size. Then Proc. 1 finds the *convex-hull* of the remaining setups. A convex-hull of a set of points $\{(t_i, b_i)\}$ is defined as the set of (t, b) points that can be obtained by taking any convex combination of points in $\{(t_i, b_i)\}$, i.e. $\sum_i \theta_i(t_i, b_i)$, with $\theta_i \geq 0$ and $\sum_i \theta_i = 1$ (note how the θ_i coefficients precisely capture the mixing of compression setups). Lastly, the setups on the lower polygonal chain of the convex-hull are marked as useful setups. The lower polygonal chain of a convex-hull includes all the points in the hull that have the minimal b value for a given t value among all the hull points. As shown later, setups not on the lower polygonal chain of the convex-hull are never useful, since for any such setup there exists a mix of two setups on the polygonal chain giving smaller average output size for a given average compression time. The proof that the polygonal chain vertices are the only useful setups is given under Claim 1, accompanied by an intuitive graphical illustration.

Claim 1. *The setups found as the vertices of the lower polygonal chain of the convex hull are the only useful setups. Useful mixes will always be of two setups connected by an edge on the lower polygonal chain.*

Proof. We give a proof sketch with the general intuition. A full mathematical proof can be given by formulating the problem as a linear program, and using the KKT optimality conditions (a well-known tool from the theory of convex optimization) [12] to show that mixing two setups is always optimal. We first note that by the definition of the convex-hull, any setup not on the lower polygonal chain has a point on the polygonal chain strictly below it in the vertical direction. Therefore, such a setup cannot be useful as it is dominated by a mix of useful setups giving a smaller output

Proc. 1 Produce a list of useful setups for a document and their convex-hull.

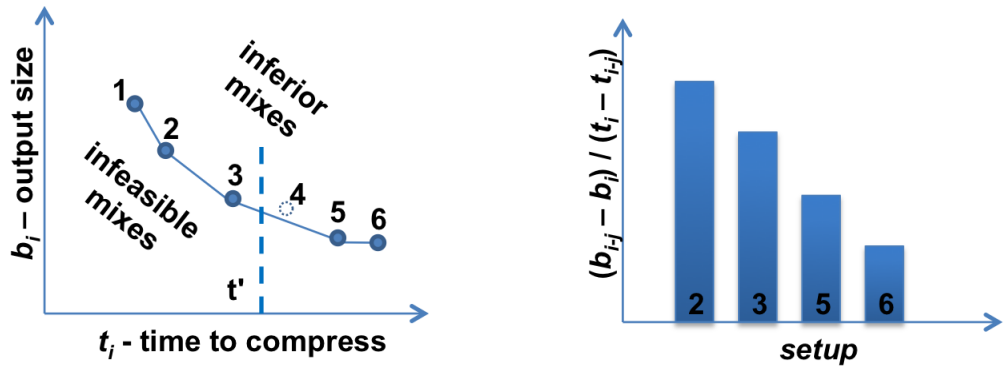
1. Determine t_i, b_i for each setup.
2. Sort the setups in ascending t_i values, with b_i used as a secondary index in descending order.
3. **for all** setups i **do**
4. **if** $b_i \geq b_{i-1}$ **or** $t_i == t_{i+1}$ **then**
5. Remove setup i
6. **end if**
7. **end for**
8. Find the convex-hull of the remaining points $\{(t_i, b_i)\}$.
9. **for all** setups i on edges of the lower polygonal chain **do**
10. Mark setup i as useful setup.
11. **end for**

size for the same processing time (see for example setup 4 in Figure 2a). Now we need to prove that for any time budget t, the useful mix is given by the intersection between the polygonal chain and the vertical line $x = t$. This is easily seen from the fact that, again by the definition of the convex-hull, any mix of *any number* of setups results in a point not below the polygonal chain. □

A graphical illustration of the proof sketch appears in Figure 2a. The solid line intervals show the lower polygonal chain calculated for a set of $m = 6$ setups by a convex-hull algorithm. For a given time budget $t' : t_3 \leq t' \leq t_5$, Claim 1 ensures that the optimal mix will include setups 3 and 5, and no others. Looking at the edges connecting each useful setup to its predecessor, from convexity one can see that the slopes of these edges follow a non-increasing order. These slopes represent the amount of size reduction obtained per additional processing-time unit, and thus capture the benefit of moving to a higher setup on this edge. Formally, we define the *benefit* of a compression setup as the ratio of the size-reduction-delta to the time-delta of the edge that leads to it from left on the convex-hull. The benefit is measured in units of bytes/sec. Figure 2b shows the benefits of all the useful setups in the example. In the next section we use the benefit representation to handle the multiple-document optimization.

3.3 Finding the optimal mix, given a time budget

Given m useful setups ordered in increasing running time, and t' (time budget), we need to determine the optimal mix a, b to use and the fractions of files r_a, r_b that each setup in the mix will serve. The non-trivial case is when $t_1 < t' < t_m$, then the mix is the adjacent useful setups a, b such that $t_b \leq t' \leq t_a$. The respective fractions are: $r_a = \lfloor \frac{t'-t_b}{t_a-t_b} \rfloor$ and $r_b = 1 - r_a$.



(a) The polygonal chain (solid edges) and useful setups (solid vertices). All mixes below the polygonal chain are infeasible given the available setups. All mixes above the chain are inferior to mixes on the chain.

(b) The benefit per useful setup (except for 1) - the setups are always ordered in non-increasing order of their benefits.

Figure 2: Finding the useful setups - an illustrative example with $m = 6$ setups, where setup 4 is found inferior (not useful).

4 Mixing Multiple Documents

In this section we discuss the realistic scenario of a system that compresses multiple different documents using multiple available compression tools. The solution for this scenario will build strongly on the optimization framework developed in the previous section for a single document. Specifically, the core of this section is an algorithm that finds the optimal selection of compression setups for the documents in the system.

4.1 Compression of multiple documents

In the multiple-document setting, each document in general has a different convex-hull, hence the compression optimization has to consider multiple convex-hulls jointly. In particular, in the terminology of Section 3 each document will in general have a different set of useful setups, because the compression efficiency of each tool and setup depends on the information characteristic of the document. For example, a web server typically generates several categories of HTML pages (news, sports, etc.). While different in content, all pages in a category are usually similar in terms of compressibility, so for the purpose of the compression optimization we may regard each category as one distinct document.

Recalling the definition of compression benefit from Section 3, we say that two distinct data objects q, r can be represented by the same document if they have the same set of useful setups, and each useful setup has the same benefit. The above criterion is soft, in the sense that compression performance may not be much affected if it holds only approximately.

4.2 Finding a sequence of useful multi-document setup combinations

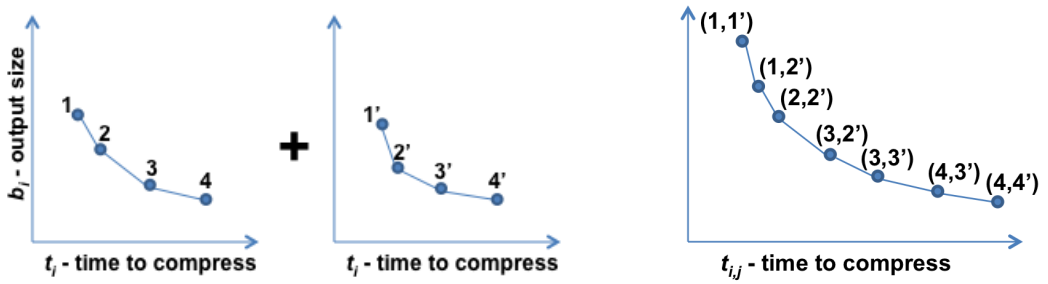
To find the optimal compression strategy in the multiple-documentsetting, we are faced with a complex problem of assigning to each distinct document a compression setup from a multitude of available setups. The different documents may have very different compression behaviors under the same setups. Further, it is possible that system constraints will require each document to choose from a different set of setups.

We show in the sequel that this problem can in fact be solved efficiently. The key step toward the solution is an algorithm that merges the individual convex-hulls of the documents into a single convex-hull, by which the system can easily find the optimal setup combination for a given compute-time budget. This algorithm is presented formally in Proc. 2. The input to the algorithm are n documents, and its output is a sequence of setup combinations for the n documents. Each combination is given as a length n vector whose i-th element is a useful setup of document i. The elements of the vectors in the sequence are non-decreasing in every coordinate. Therefore, the vectors in the sequence represent setup combinations with increasing total compute time.

Proc. 2 Given n documents, produce a sequence of length n vectors representing setup combinations.

1. **for all** documents i **do**
2. run Proc. 1 and obtain the useful setups and the convex-hull
3. set s_i, the i-th coordinate of the setup vector, to the lowest useful setup of document i
4. from the convex-hull calculate D_i – a decreasing ordered list of benefits for the useful setups of i, except for setup s_i
5. **end for**
6. output setup vector $(s_1, ..., s_n)$
7. **repeat**
8. $i^\star \leftarrow i \mid D_i$ has the largest first item among all D's
9. $s_{i^\star} \leftarrow$ next useful setup of document $i^\star$ that is $> s_{i^\star}$
10. remove the first item from $D_{i^\star}$
11. output setup vector $(s_1, ..., s_n)$
12. **until** all D's are empty

Proc. 2 is best understood with an example. Figure 3 gives such an example for $n = 2$ documents. For the 2 input documents, Figure 3a shows the useful setups and the convex-hulls. Recall that the output of Proc. 2 is a sequence of length 2 vectors specifying setup combinations for the 2 documents. This output sequence can be seen on the labels of Figure 3b read from left to right. Setups for document 1 appear as integers s in the first coordinate of the vectors, and for document 2 as integers s' in the second coordinate. Notice that between adjacent vectors in the sequence exactly one of the two elements is increased by at least one setup (in this specific example all changes are +1 because the useful setups happen to be contiguous). In Figure 3b we show, in addition to the setup-combination sequence itself, the slopes in bytes/sec showing the benefits of the setups chosen for increase in every step of the sequence.



(a) Two documents and the convex-hull of each document separately, with slopes showing benefits in bytes/sec.

(b) The merged convex-hull: a sequence of setup combinations and slopes showing the benefits of increased setups.

Figure 3: Building a merged convex-hull to handle multiple documents.

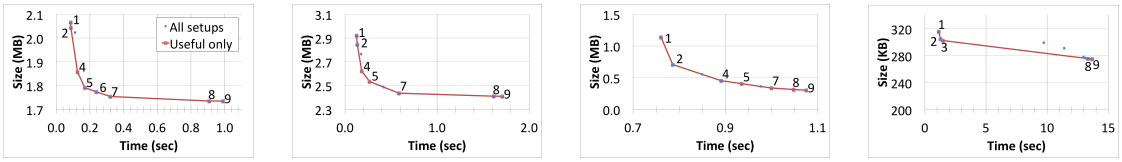
4.3 Finding optimal setup-combinations given compute-time budget

Now that we know how to merge n convex-hulls into a non-decreasing sequence of setup combinations, we turn to specify how the merged sequence can be used to find the optimal setup combination given an arbitrary compute-time budget. The key idea is as follows. We first calculate the required compute time when the lowest useful setups are used for all documents. We can calculate this time from the convex-hulls of the individual documents and the size and/or number of instances of each document pending compression. Then we move to the next setup-combination vector in the output sequence, and repeat the calculation of required compute time. We continue progressing on the sequence of setup combinations until we hit the last setup combination that is within the compute-time budget. Progressing to an adjacent vector in the output sequence amounts to increasing a setup for the single document ($i^\star$ in Proc. 2) that gives the highest incremental benefit at the current compute-time allocation. As we approach the compute-time budget in this progression, at some point we expect the budget to fall in between adjacent setup combinations in the sequence. As we did for the single-document setting in Section 3.3, here too we will mix two adjacent setup combinations. Since exactly one document changes setup between the two adjacent combinations, the final assignment of compression setups to documents will have at most one document mixing two useful setups, while all other documents will use one useful setup for all their instances.

The nice property of the above optimization is that we do not have to know anything about the compression algorithms implemented in the system. We only need to know, in the most abstract way, the ordering of the benefits between the useful setups of the documents pending compression.

5 Evaluation

This section serves to demonstrate the high potential advantage offered by the new optimization framework. To capture the most realistic circumstances for such an optimization, our evaluation will use data from the publicly-available enwiki [13] repository. In addition to working with real massive data-sets, our evaluation will



(a) pagelinks in gzip. (b) langlinks in gzip (c) history in bzip2. (d) history in xz.

Figure 4: Single-document convex-hull of each of the four given documents. The number labels are the useful setups, and the connecting lines are the convex-hull.

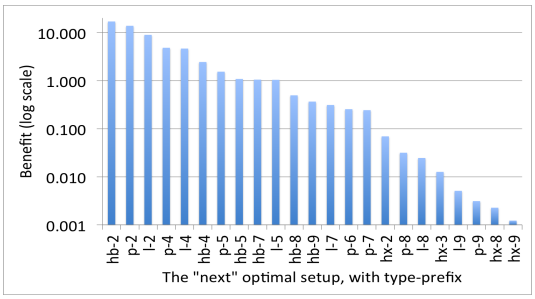
also mimic the way these files are compressed by enwiki snapshot generators.

The data-set: The data-set we use is a complete copy of the English Wikipedia dumps in XML, along with some metadata, and a number of raw database tables in SQL form. These snapshots are provided once or twice a month, after running time-consuming heavy compression for at least 10 days, over more than 12 TB of page histories (as of 2014). For example, the snapshot contains two sets of 169 files of page histories in two compression formats, namely bzip2 and 7z (LZMA). Additional sets also contain gzip forms of various data, making the overall compression task extremely complex and difficult to plan.

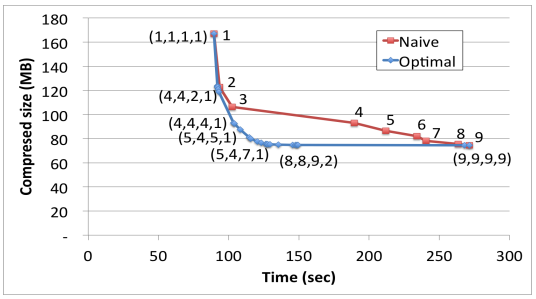
Per-document convex-hull: Figures 4a-4d show the output size and compute time of four of the different documents spanning the enwiki database. The compute time was measured with current standard implementations of bzip2, gzip, and xz (for the 7z/LZMA format) for Mac, using a single CPU and considering all 9 standard setups provided by each of the mentioned tools. The plots also show the resulting useful setups found by Proc. 1. Note that the plots of Figure 4 describe three very different compression algorithms, but this fact is not an issue for our abstract optimization framework.

Merging the documents and assigning setups: Figure 5a shows the benefit calculated for each document and useful setup, except for the lowest setups. The benefits are plotted in non-increasing order. This order is the same as the order by which Proc. 2 changes setups between adjacent setup-combinations in the output sequence. Figure 5b compares between the results of the optimization given in Section 4 to the naïve approach of selecting the same numeric setup to all documents. Each point on the plots represents a compression job for four enwiki documents, where the chosen setups appear on the label[1]. For our results the label is a quadruple of setups, each for one document (in general referring to different compression tools), and for the curve shown for comparison the label designates the setup number chosen for all documents. The vertical coordinate shows the total compressed size, and the horizontal coordinate shows the consumed compute time. It can be observed that the lower curve resulting from this paper's algorithms gives superior compression for all compute budgets except for the extreme ones (none of which is a practical choice: one is poor compression, the other prohibitive compute cost). The advantage is indeed significant: up to 25% stronger compression for the same compute budget. Another advantage lies in the ability to limit the compute budget to half the maximum required, with only 0.7% compromise of compression size.

[1]To avoid clutter, only part of the labels appear on the figure.



(a) The benefits of the useful setups for the enwiki documents, sorted in non-increasing order.



(b) Results for compressing enwiki: 100MB of the first two documents and 1GB of the last two.

Figure 5: Evaluation of compression for four enwiki documents.

6 Conclusion

This paper is the first step toward the goal of fully optimized compression in systems. Future directions include algorithmic enhancement of standard compressors for finer control of their bytes/sec benefits, and joint optimization for additional criteria.

Acknowledgment: This work was partly done under a joint ISF-UGC project. In addition, it was supported by a Marie Curie CIG grant, by the Israel Ministry of Science and Technology, and by an Intel ICRI-CI grant.

References

[1] K. Barr and K. Asanović, "Energy Aware Lossless Data Compression," *Proc. of MobiSys*, 2003.

[2] R. Kothiyal, V. Tarasov, P. Sehgal, and E. Zadok, "Energy and Performance Evaluation of Lossless File Data Compression on Server Systems," in *Proc. of SYSTOR*, 2009.

[3] Y. Chen, A. Ganapathi, and R. H. Katz, "To Compress or Not to Compress - Compute vs. IO Tradeoffs for MapReduce Energy Efficiency," in *Proc. of SIGCOMM Workshop on Green Networking*, 2010.

[4] D. Harnik, R. Kat, D. Sotnikov, A. Traeger, and O. Margalit, "To Zip or Not to Zip: Effective Resource Usage for Real-Time Compression," in *Proc. of FAST*, 2013.

[5] C. Pu and L. Singaravelu, "Fine-Grain Adaptive Compression in Dynamically Variable Networks," in *Proc. of ICDCS*, 2005.

[6] M. Gray, P. Peterson, and P. Reiher, "Scaling Down Off-the-Shelf Data Compression: Backwards-Compatible Fine-Grain Mixing," in *Proc. of ICDCS*, 2012.

[7] E. Zohar and Y. Cassuto, "Automatic and Dynamic Configuration of Data Compression for Web Servers," in *Proc. of LISA*, 2014.

[8] R. Lenhardt and J. Alakuijala, "Gipfeli-High Speed Compression Algorithm," in *Proc. of Data Compression Conference (DCC)*, 2012.

[9] snappy - A Fast Compressor/Decompressor, http://code.google.com/p/snappy/ .

[10] LZO Data Compression Library, http://www.oberhumer.com/opensource/lzo/ .

[11] P. Franaszek, J. Robinson, and J. Thomas, "Parallel Compression with Cooperative Dictionary Construction," in *proc. of DCC*, 1996, pp. 200–209.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[13] Enwiki Dump Progress on 20140903, 2014, http://dumps.wikimedia.org/enwiki/20140903/ .