Energy-Efficiency and Performance Trade-offs of Data-reduction techniques

Tim Dobert

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

17.12.2014

Overview

1. Motivation
2. Compression
3. Data Deduplication
4. Recomputation
5. Conclusion

Motivation

Data reduction should be considered for three main reasons:

- Storage/Hard drives constantly use power

Component	Peak power	Count	Total	Percentage
CPU	40 W	2	80 W	37.6 %
Memory	9 W	4	36 W	16.9 %
Disk	12 W	1	12 W	5.6 %
PCI slots	25 W	2	50 W	23.5 %
Motherboard	25 W	1	25 W	11.7 %
Fan	10 W	1	10 W	4.7 %
System total			213 W	

Table I. Component peak power breakdown for a typical server . [1]

- Transmitting data costs energy
- Growing speed gap between CPUs and memory

Motivation

Data reduction should be considered for three main reasons:

- Storage/Hard drives constantly use power

Component	Peak power	Count	Total	Percentage
CPU	40 W	2	80 W	37.6 %
Memory	1 W	4	36 W	16.9 %
Disk	12 W	1	12 W	5.6 %
PCI slots	25 W	2	50 W	23.5 %
Motherboard	25 W	1	25 W	11.7 %
Fan	10 W	1	10 W	4.7 %
System total			213 W	

Table I. Component peak power breakdown for a typical server [1]

- Transmitting data costs energy
- Growing speed gap between CPUs and memory

Reducing the amount of stored data can reduce the energy consumption of a system. There are three main reasons for that. First, running storage devices like hard drives consume energy, so if there is less data to store, not as many hard drives are needed. In a typical server storage makes up about 5% of the overall power consumption[1]. Another factor is data transmission. Especially over wireless, transmitting less data costs less energy. Lastly, while CPU speed has increased rapidly over the years, memory speed has increased much more slowly. This has created a speed gap, that can cause the CPU to wait idly for a significant amount of time, while data is fetched from the memory. Decreasing the amount of data can decrease this time and improve efficiency.

General Approach

Using additional computing power to reduce data.

- This creates a time/energy overhead
- Reduces needed hard drives, could replace with SSDs
- Reduce transmission energy (important on mobile devices)

This can be done using three different techniques.

Using additional computing power to reduce data.

- This creates a time/energy overhead
- Reduces needed hard drives, could replace with SSDs
- Reduce transmission energy (important on mobile devices)

This can be done using three different techniques.

All of the techniques presented in this talk share the same basic approach: Additional computing power is spent to the reduce data. The expectation is, that this will lead to energy savings down the road and have a net positive effect on the overall energy consumption. This obviously introduces an overhead in both energy and time. However, this might be mitigated the use of SSDs, for example, which are faster and more energy efficient, but provide less storage space.

Sections:
Motivation
Compression
Deduplication
Recomp
Conclusion

Compression

General Idea

Encoding output data to reduce redundancy and therefore size

- This involves encoding output, decoding input
- Best case: Integrated into file I/O (i.e. ZFS)
- Compression must be lossless

Data Reduction
Compression
General Idea

Encoding output data to reduce redundancy and therefore size

- This involves encoding output, decoding input
- Best case: Integrated into file I/O (i.e. ZFS)
- Compression must be lossless

Compression is probably the most common and well known technique of data reduction. In this scenario the best option is to directly integrate it into file I/O. ZFS is one file system that already has this option. In most cases, especially in scientific computing, a lossless compression method should be chosen in order to avoid artefacts. Sometimes it might be possible to consider lossy compression though, since it provides better compression ratios.

Trade-off

Whether it's worth it depends on:

- Computational overhead vs. space savings
- Compression algorithm used
- Computing power

Trade-off

Whether it's worth it depends on:

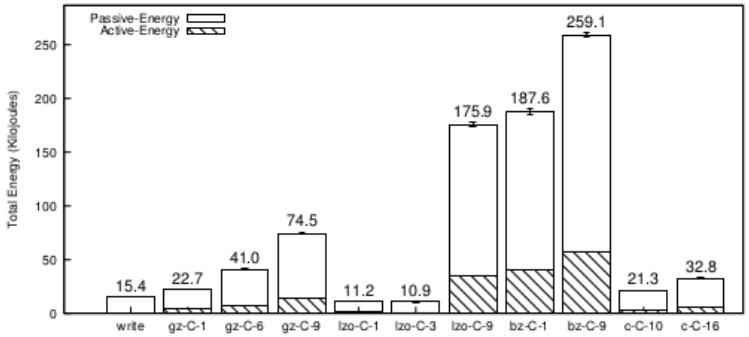
- Computational overhead vs. space savings
- Compression algorithm used
- Computing power

The encoding and decoding process obviously creates an overhead. How big it is, depends on the algorithm, as there is usually a trade-off between speed and compression ratio. The latter is heavily influenced type of data that is being compressed. It is generally more sensible to employ this on faster computers, as this will lessen the time overhead.

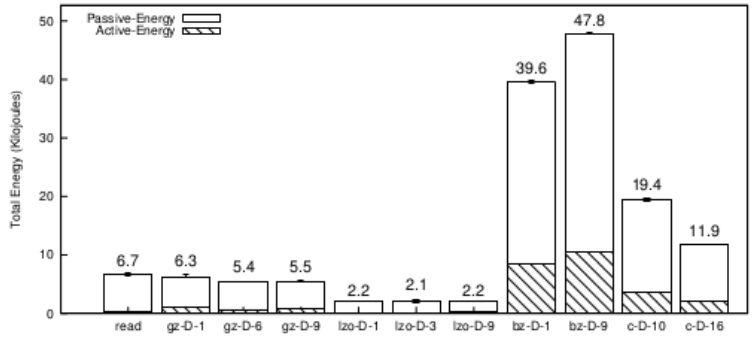
In Practice

Energy Consumption for:

Text file



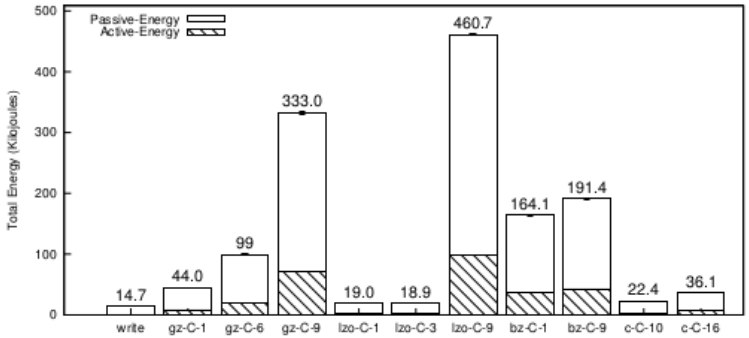
(c) **Energy** consumed for write vs. compression



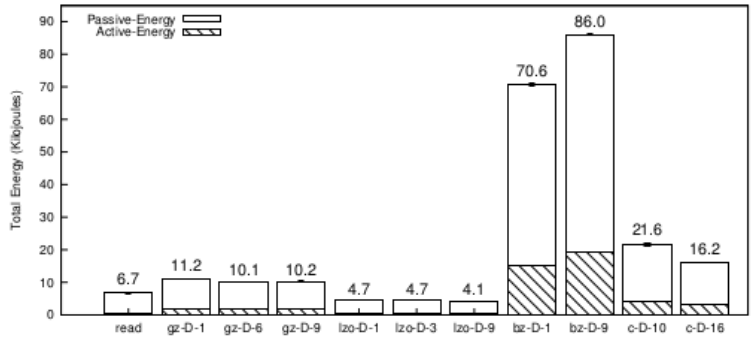
(d) **Energy** consumed for read vs. decompression

[3]

Binary file

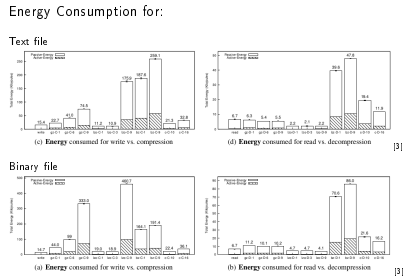


(a) **Energy** consumed for write vs. compression



(b) **Energy** consumed for read vs. decompression

[3]



These graphs are from one particular study on the energy consumption when performing I/O-operations with and without compression. The important insights here are, that in some cases, just on reading or writing alone, energy can be saved when using compression. This depends on the data though, even when using the same algorithm. Also, note that the overhead for reading and writing is not always the same. It is important to keep in mind that in addition, there are also energy savings from having to preserve less data.

Additional Thoughts: Communication

There are different approaches for optimizing energy consumption for a whole system or just parts of it.

- Lots of data transmission $\Rightarrow$ overall savings might outweigh increase on nodes
- Asymmetric compression for single devices

Just choosing the fastest algorithm or the one with the best compression ratio is rarely ideal.

Additional Thoughts: Communication

There are different approaches for optimizing energy consumption for a whole system or just parts of it.

- Lots of data transmission ⇒ overall savings might outweigh increase on nodes
- Asymmetric compression for single devices

Just choosing the fastest algorithm or the one with the best compression ratio is rarely ideal.

When this technique is applied to a system that features a lot of communication between parts, there are more things to consider. Choosing a more expensive algorithm with a better compression ratio could lead to an increase of energy consumption in single nodes, but save energy overall, because there is less data to communicate. Reducing consumption on single nodes can be done with asymmetric compression. This involves encoding data with a fast algorithm on a node, sending it to the server converting it into a format that it is easy to decode on there. This way both encoding and decoding on the nodes can be as efficient as possible. In any case it is rarely optimal to choose the algorithm with the best compression ratio or the fastest speed. One has to consider the ratio thereof and the type of data at hand.

Sections:
Motivation
Compression
Deduplication
Recomp
Conclusion

Data Deduplication

Concept

The Goal is to reduce redundancy across all files.

This is how it's done:

- Data is divided into blocks
- Calculate fingerprints, store in hash table
- Every block is only stored once

This approach is most effective for data centres, HPC and backups.

Concept

The Goal is to reduce redundancy across all files.

This is how it's done:

- Data is divided into blocks
- Calculate fingerprints, store in hash table
- Every block is only stored once

This approach is most effective for data centres, HPC and backups.

Deduplication and compression operate on basically the same principle. Both try to eliminate redundancy in data, but whereas compression works on single files at a time, deduplication works across multiple files. Data is divided into blocks, which are then stored in a hash table. If a block was already saved once, only a reference is stored. This approach is especially effective in systems with backups, since old data is often not much different from newer data. Luckily, errors as a result of hash collisions are extremely unlikely.

Parameters and Trade-offs

A deduplication system can have the following parameters:

- Size of the blocks:
  - static
  - variable
- Block size affects the size of the Table
  - Big, static $\Rightarrow$ small Table
  - Small, variable $\Rightarrow$ Big Table

  Normally a block size of 16kB is used.
- Smaller tables are faster to work with

Like with compression, there is a reduction-overhead trade off.

Data Reduction
Data Deduplication
Parameters and Trade-offs

Parameters and Trade-offs

A deduplication system can have the following parameters:

- Size of the blocks:
  - static
  - variable
- Block size affects the size of the Table
  - Big, static ⇒ small Table
  - Small, variable ⇒ Big Table

  Normally a block size of 16kB is used.
- Smaller tables are faster to work with

Like with compression, there is a reduction-overhead trade off.

How the data is divided into blocks greatly affects performance, both in reduplication and reduction ratio. The size of the blocks can be variable, meaning, that the division is done in a way, that the resulting blocks have a higher probability of occurring again. In practice, a variable, average size of 16kB is used. It is important to note, that, while small block sizes lead to better data reduction, they also lead to big hash tables. This can be a problem, if the table does not fit into the RAM completely, making searches potentially very slow. There are methods to make this event unlikely, but they exceed of the scope of this talk.

In Practice

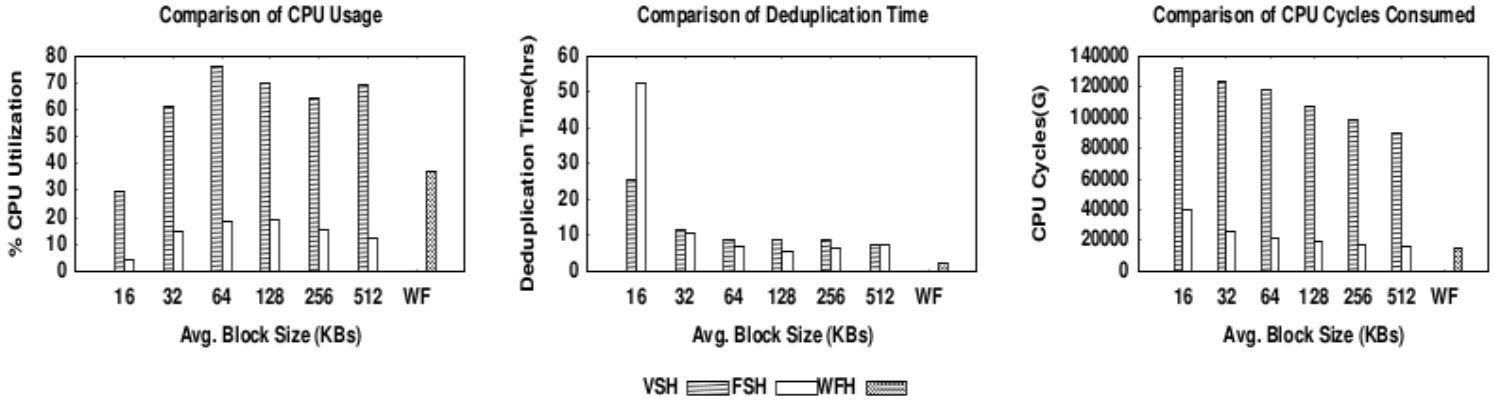
Evaluation:

TABLE II
DEDUPLICATION RATIO USING CONTENT-DEFINED CHUNKING WITH AN AVERAGE CHUNK SIZE OF 8 KB ON DIFFERENT HPC DATA SETS.

Data set	Ratio	Data set	Ratio
BSC-BD	7.0%	DKRZ-B5	29.5%
BSC-MOD	21.3%	DKRZ-B6	22.5%
BSC-PRO	29.3%	DKRZ-B7	14.1%
BSC-SCRA	38.3%	DKRZ-B8	13.9%
DKRZ-A	17.9%	DKRZ-K	49.3%
DKRZ-B1	19.7%	DKRZ-M1	15.0%
DKRZ-B2	27.6%	DKRZ-M2	21.1%
DKRZ-B3	74.4%	RENCI	23.8%
DKRZ-B4	27.1%	RWTH	23.2%

[5]

Efficiency depending on Block Size



(a) CPU Utilization (b) Deduplication Time (c) CPU Cycles

[6]

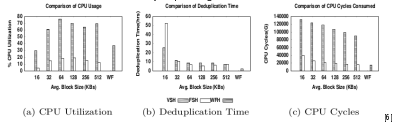
Evaluation:

TABLE II
DEDUPLICATION RATIO USING CONTENT-DEFINED CHUNKING WITH AN AVERAGE CHUNK SIZE OF 8 KB ON DIFFERENT HPC DATA SETS.

Data set	Ratio	Data set	Ratio
BSC-BD	7.0%	DKRZ-B5	29.5%
BSC-MOD	21.3%	DKRZ-B6	22.5%
BSC-PEO	28.3%	DKRZ-B7	14.1%
BSC-SCIBA	58.3%	DKRZ-B8	13.9%
DKRZ-A	17.9%	DKRZ-K	49.3%
DKRZ-B1	19.7%	DKRZ-M1	15.0%
DKRZ-B2	27.6%	DKRZ-M2	21.1%
DKRZ-B3	74.4%	RENCI	23.8%
DKRZ-B4	27.1%	RWTH	23.2%

[5]

Efficiency depending on Block Size



(a) CPU Utilization (b) Deduplication Time (c) CPU Cycles

[5]

The table up top shows that the effectiveness of this technique varies wildly depending on the data. The reduction rate can end up anywhere between 7% and 70%. The three graphs below that are from a different study. Here the takeaway is, that increasing the block size makes deduplication time and CPU cycles go down.

(VFS: Variable Size Hashing, FSH: Fixed Size Hashing, WFH: Whole File Hashing)

Sections:
Motivation
Compression
Deduplication
Recomp
Conclusion

Recomputation

General Idea

The result of a computation is not saved. It will be recomputed when needed.

- Only store input, maybe partial results
- Perform computation every time the data is needed
- Advances in hard- and software may make future computations more efficient

Example: Online video transcoding

General Idea

The result of a computation is not saved. It will be recomputed when needed.

- Only store input, maybe partial results
- Perform computation every time the data is needed
- Advances in hard- and software may make future computations more efficient

Example: Online video transcoding

Recomputation is only an option if the data is accessed very rarely. In some cases the cost of every subsequent recomputation is lower because newer, more efficient hardware is used or a smarter algorithm has been developed. The best example of this technique in practice is online video transcoding. Most web video platforms provide the option of watching content at various resolutions. Instead of having every video in every resolution stored on the server, they usually only store the source file. If a user requests it at a different resolution, the video is only then transcoded. The resulting data remains on the server for a limited amount of time. This approach saves a massive amount of space.

Trade-offs

Of the three techniques, this is the most situational.

- Computing is obviously costly, slow
- Only applicable to rarely used data
- Computation must be deterministic
- Code preservation, emulation needed

It's difficult to say in advance, if the trade-off will be worth it.

Trade-offs

Of the three techniques, this is the most situational.

- Computing is obviously costly, slow
- Only applicable to rarely used data
- Computation must be deterministic
- Code preservation, emulation needed

It's difficult to say in advance, if the trade-off will be worth it.

Recomputation has a number of drawbacks. It is not an option in scenarios where the data is accessed often or has to be gathered quickly. Since this is supposed to replaces normal storage, the computation also has to be deterministic. Even if these are not the case, the code has to be stored somewhere and might not even work the next time the data is needed. Then Emulation would be required, creating a significant overhead. It is also often difficult to predict, how frequently or how soon the data is needed again. All of these factors make recomputation a very situational technique.

Sections:
Motivation
Compression
Deduplication
Recomp
Conclusion

Conclusion

Conclusion

All of the presented techniques come at a cost. In the end, you should keep in mind:

- More reduction is more computationally expensive
- The effectiveness depends on the application and parameters
- Deduplication is not very effective when compression is already being used
- If nothing else, try a light compression method

All of the presented techniques come at a cost. In the end, you should keep in mind:

- More reduction is more computationally expensive
- The effectiveness depends on the application and parameters
- Deduplication is not very effective when compression is already being used
- If nothing else, try a light compression method

In conclusion, all of the three techniques have trade-offs that depend on the parameters and the situation. It is also important to note at this point, that deduplication and compression should not be used at the same time. The reason is, that they both operate on the same principle. If compression has already reduced the redundancy, deduplication can not do it again. In addition, small changes in the source files can lead to big changes in the compressed ones, so two files that would be similar and would benefit from deduplication, do not anymore.

In many cases however it is very easy to implement a light compression algorithm and to see if it has a positive effect on energy consumption.

Experiments

Processing a 4k bitmap image with and without compression.
Compression algorithm: lz4 (https://code.google.com/p/lz4/)
Uncompressed file size: 24.9MB

	normal	compression	compressed size
Picture	read: 0.010s write: 0.027s total: 0.100s	read: 0.030s write: 0.123s total: 0.220s	22.0 MB
Black	- - -	read: 0.017s write: 0.004s total: 0.076s	97.6 kB
Random	- - -	read: 0.020s write: 0.030s total: 0.110s	25.0 MB

One more thought:
Compressing on the GPU. It's generally less efficient, though[9].

Processing a 4k bitmap image with and without compression.
Compression algorithm: lz4 (http://code.google.com/p/lz4/)
Uncompressed file size: 24.9MB

	normal	compression	compressed size
Picture	read: 0.010s write: 0.027s total: 0.100s	read: 0.030s write: 0.123s total: 0.220s	22.0 MB
Black	- - -	read: 0.017s write: 0.004s total: 0.076s	97.6 kB
Random	- - -	read: 0.020s write: 0.030s total: 0.110s	25.0 MB

One more thought:
Compressing on the GPU. It's generally less efficient, though[9].

The table contains the results of a small experiment. Three different 4k image files have been processed (read, invert colors, write) with and without compression. The results are nor surprising for the most part: A normal picture takes more time with compression, a black image is slightly faster. The image with random colors took about the same, probably because there was little redundancy to eliminate. Note that in this experiment the GPU was not utilized. Using the GPU for the image processing and the CPU for compression or vice-versa would probably be worth looking into. Especially if multiple images are precessed in a row, because then the two tasks could be pipelined.

Sources I


[1] A Survey on Techniques for Improving the Energy Efficiency of Large Scale Distributed Systems
https://www.fsl.cs.sunysb.edu/docs/greencomp/green-compress.pdf


[2] Benefits of Data Compression in HPC Storage
http://wr.informatik.uni-hamburg.de/_media/research/publications/2014/epbodcihss14-evaluating_power_performace_benefits_of_data_compression_in_hpc_storage_servers.pd


[3] Energy and Performance Evaluation of Lossless File Data Compression on Server Systems
https://www.fsl.cs.sunysb.edu/docs/greencomp/green-compress.pdf


[4] Energy-aware lossless data compression
http://dl.acm.org/citation.cfm?id=1151692


[5] Deduplication in HPC Storage
http://wr.informatik.uni-hamburg.de/_media/research/publications/2012/asoddihssm12-a_study_on_data_deduplication_in_hpc_storage_systems.pdf


[6] Demystifying Data Deduplication
http://dl.acm.org/citation.cfm?id=1462739

Sources II


[7] Maximizing Efficiency By Trading Storage for Computation
https://www.usenix.org/legacy/event/hotcloud09/tech/full_papers/adams/adams_html/ (09.12.14)


[8] Exascale Storage Systems
http://superfri.org/superfri/article/download/20/6


[9] Parallel Lossless Data Compression on the GPU
http://www.idav.ucdavis.edu/func/return_pdf?pub_id=1087

Zip graphic:

http://wikimediafoundation.org/wiki/File:Simple_Comic_zip.png

Summary

- Less data means less energy consumption
- Three main techniques for data reduction
  - Compression
  - Deduplication
  - Recomputation
- Compression works on single files and can be integrated into the file system
- Deduplication works across all files, can be costly though
- Recomputation is highly situational

	Processor	Memory	Network	Storage
Re-computation of results	-	-	-	+
Deduplication	-	--	0	+
Compression (client side)	-	0	+	++
Compression (server side)	-	0	0	++
User education	+	+	+	+

Table 7. Benefits and penalties of different concepts for data reduction (+ benefits; - penalties)

[8]