DEVELOPING AND EVALUATING A LOSSLESS COMPRESSION SCHEME FOR SCIENTIFIC DATA FROM A NANOSATELLITE

Spencer Clark, Dwight Makaroff and Kevin Stanley

Department of Computer Science, University of Saskatchewan,
Saskatoon, SK CANADA S7N 5C9
spencer.c@usask.ca, {makaroff, stanley}@cs.usask.ca

ABSTRACT

This paper examines the problem of of developing a lossless compression scheme for data from a nano-satellite being developed by the University of Saskatchewan Space Design team, the USST-Sat. The benefit of compressing scientific data from the satellite will be an increased ability to perform experiments and downlink the results. Goals for the compression scheme are to maximize space savings and result in a net energy savings over storing and transmitting uncompressed data.

Our evaluations show that the custom scheme that we developed, called *USST-Compress*, performs compression as well as or better than the generic compression schemes evaluated, and additionally results in better net energy savings.

Index Terms— Compression algorithms, Data compression, Low earth orbit satellites, Computer science

1. INTRODUCTION

The University of Saskatchewan Space Design Team (USST) designed a nanosatellite for measuring total electron content in the ionosphere as the satellite passed over ground stations, generating significant data. The data downlink rate was estimated to be 100 kilobits per second, and for only a few minutes a day. With the satellite passing over multiple ground stations per day and collecting data at each station, the payload data can accumulate quickly. Higher compression of primary payload data makes room for more downlink time for diagnostic and secondary payload data. The second motivation for compression is to minimize power usage. The cost of transmitting data can be much higher than the cost of collecting and processing [1]. The energy cost of compressing a bit can be much less than the cost of transmitting that bit [2]. By conserving power through compression, the satellite will be able to complete more duty cycles and/or provide more power to other non-essential subsystems.

To properly evaluate a compression scheme for this scientific data, we must obtain a collection of sample data for evaluation. Due to the novel method of data collection, however, there exist no records from previous missions, so a method of simulating record generation was developed. The requirements of the artificial data are as follows: 1) the data generated is easy to place into the format of the USST-Sat's scientific records, 2) the data is generated with random components, 3) erroneous data points are inserted into the data to simulate sensor error, and 4) it is easy to change data distribution parameters.

By exploiting our knowledge of the structure of payload data, we were able to develop a compression scheme (*USST-Compress*) that combined the best aspects of speed and compression ratio of the two generic compression schemes that inspired our experimentation. The compression ratio of *USST-Compress* approaches that of a dictionary coder (*LZ77* [3]), but is 3 orders of magnitude faster in execution time, and therefore consumes less energy for computation.

2. RELATED WORK

Data compression has been studied for years.[1] Generic compression algorithms such as Huffman coding [4] and Lempel-Ziv '77 [3] (*LZ77*) achieve compression without prior knowledge of the data they compress. Huffman coding creates a binary tree of codes, where each code is mapped to a symbol that occurs in the target data, Symbols that appear more frequently are given shorter codes than less frequent symbols. *LZ77* is a dictionary coder, essentially encoding each new symbol as an extension to a symbol that is already in the dictionary.

Tailoring custom compression schemes to specific types of data, such as HTML source or fingerprint images has been shown to achieve improved results. Skibinski exploited the typical structure of HTML documents [5], and was able to achieve compression results 8-15% better than generic compression methods. Zirkind encoded fingerprint images in a tri-color format [6] to achieve a compression ratio of 1:92, compared to the 1:23 achieved by JPEG compression.

[1] History of Lossless Data Compression Algorithms. http://www.ieeeghn.org/wiki/index.php/History_of_Lossless_Data_Compression_Algorithms

3. SYSTEM MODEL AND TESTING ENVIRONMENT

The purpose of our algorithm was to leverage known properties of the data to create a custom data compression scheme which outperformed typical compression, and was capable of being run efficiently on the low power CPUs typical of nanosatellite architectures. Our target architecture was specifically a SAM7S256 microcontroller with 64KB of on-chip RAM and a 32-bit ARM7 processor running at roughly 50MHz executing the eCos operating system.[2]

When the satellite passes over a ground station, it may perform an experiment, where the payload instrument measures a series of time differences (Δt's) between two radio signals (one at UHF and one at VHF) sent from the ground station at 100 Hz for an average of 80 seconds along with GPS fixes for the satellite at both the beginning and end of the experiment.

Based on information from NASA,[3] we devised a method to model the total electron content (TEC) data that the USST-Sat experiment is designed to measure. A set of integers is produced, each representing a time difference between the arrival of VHF and UHF signals from the ground to the satellite, and two artificial GPS fixes are generated for the satellite's position the beginning and end of the experiment. Algorithm 1 describes the process of generating an artificial record.

Algorithm 1: Artificial Record Generation Algorithm

Data: *meanSamples, refpointPct , spread, snr,* λ
Result: An artificial USST-Sat experiment record

```
/* Select a 'seed' integer for
   reference points                          */
```
$x \leftarrow$ RandFromUniform(*range* $= [3995000, 4000000]$)
```
/* Choose a number of samples
   normally distributed                      */
```
numSamples $\leftarrow$ RandFromNormal($\mu =$*meanSamples*, $\sigma =$ *meanSamples* $*0.2$)
```
/* Generate reference points, create
   a cubic spline, representing the
   'pure' signal without noise               */
```
for $i \leftarrow 1$ **to** *(numSamples*$/100)*$ *refpointPct* **do**
    refpoints$[i] \leftarrow$ RandFromNormal($\mu = x$, $\sigma =$ *spread*$*5000$)

pure $\leftarrow$ CubicSpline(*refpoints*)
```
/* Add noise                                 */
```
whiteNoised $\leftarrow$ AddWhiteNoise(*pure, snr*)
final $\leftarrow$ AddDisruptionsPoisson(*whiteNoised,* λ)
return *final*

The algorithm was run 30 times with each combination of 3 different sets of parameters, generating a total of 810 artificial records. The parameter values for the data conditions are shown in table 1. Conditions i, iv, and vii represent a 'clean' condition (Δt's measured have low variance, representing low noise). Conditions ii, v, and viii represent a moderate condition, and conditions iii, vii, and ix represent the 'dirty' condition, with corresponding increases in variance.

Condition	Size	Variability		Noise	
	Size	Ref%	Spread	SNR	λ
i	30000	.02	.06	80	.0001
ii	30000	.04	.12	40	.0002
iii	30000	.08	.24	5	.0004
iv	80000	.02	.06	80	.0001
v	80000	.04	.12	40	.0002
vi	80000	.04	.24	5	.0004
vii	160000	.02	.06	80	.0001
viii	160000	.04	.12	40	.0002
ix	160000	.04	.24	5	.0004

Table 1: Data Conditions.

3.1. Compression Scheme

USST-Compress, exploits knowledge of the experimental record format and consists of three distinct steps: base subtraction, differential modulation, and entropy encoding.

Since the body of a single record consists entirely of a set of integers, they are encoded in 32-bit two's-complement. The payload team expects the maximum range of Δt values will be roughly 5000, so by subtracting the minumum-value Δt in the experiment from each other Δt, we quickly reduce the storage size required for the measurements to 13 bits.

After base-subtraction, integer values remain. The TEC measurements are anticipated to reveal a natural random process that follows a smooth curve between inflection points.We exploit this fact using a technique similar to differential modulation [7]. We define each Δt value in an experiment record as a 'sample.' The compression algorithm begins by marking the first sample as a 'reference sample,' and then iterates over each sample in the record. If the difference between the current sample and the reference sample can be encoded in a predetermined number of bits b_c,[4] the sample is recorded in this way and marked as a 'residual sample.' If the difference between the current sample and the reference sample cannot be encoded in b_c bits, the current sample becomes the new reference sample. A bitmap is maintained which stores the status of each sample as either a reference or residual sample.

The final step is re-encoding the result of the differential encoding step using a general-purpose entropy encoder. The first two stages of *USST-Compress* are tailored to the experiment record data and do not compress based on symbol occur-

[2]SAM7S256 data sheet. http://atmel.com/devices/SAM7S256.aspx
[3]D. Bilitza. International Reference Ionosphere. http://iri.gsfc.nasa.gov/
[4]which should be less than the number of bits used to encode the base-subtracted values

rences; content-independent compression can further reduce the output size.

3.2. Experimental Setup

We employed the compression scheme on data created using the synthetic data creation algorithm to evaluate performance relative to existing generic compression schemes. In keeping with the goals for the compression scheme, we chose to evaluate the candidate compression schemes based on two metrics: Compression Time, the elapsed time from start to finish of compressing a record, and Compression Ratio, the amount of compression achieved. The tests were executed on a 2.53GHz Intel i5 processor and 4GB of RAM running Mac OSX. The compression ratio achieved by the algorithms is independent of hardware. Though the test and target CPUs differ significantly, we are concerned principally with the relative execution times of the compression schemes and less concerned with absolute execution times on target hardware.

Huffman Coding and *LZ77* were the two standard compression methods compared against 4 variants of *USST-Compress*. The *USST-11* and *USST-13* algorithms use *USST-Compress* with the number of bits to encode a residual sample value fixed at 11 and 13, respectively, and no entropy-encoding. *HuffmanUSST-11* and *HuffmanUSST-13* are the same as *USST-11* and *USST-13*, but apply Huffman coding for the entropy-encoding step.

All six algorithms were run with each of the artificial records in the corpus. Tests were conducted in batches based on the clean, moderate, and dirty data conditions.

4. EVALUATION

Figure 1 depicts mean compression ratios for each algorithm in each noise condition. When combined with an entropy-coding pass with Huffman coding, the results of *USST-Compress* are comparable to those of *LZ77*. Figure 1 is representative of the charts for other data conditions.

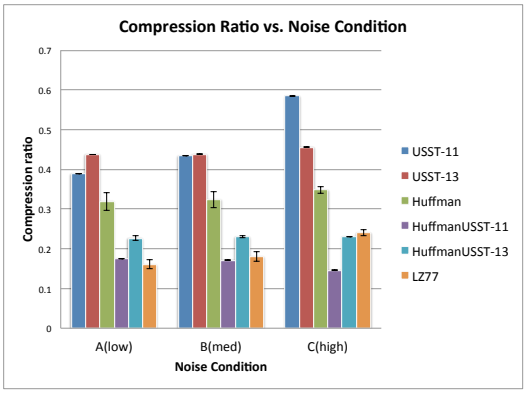

Fig. 1: Compression Ratio vs. Noise Condition. Error bars are at ± one standard deviation.

The mean time elapsed for compressing an experiment record across all data conditions is shown in Figure 2. *LZ77* is omitted because its execution time was 3 orders of magnitude higher than that of the other algorithms. *USST-Compress* without entropy-coding is relatively fast, and adding an entropy-coding pass produced a mean compression time comparable to Huffman coding alone. When compression time is compared between differing noise and variance conditions, the relative performance is very similar. Figure 3 shows compression times increasing directly with record size.

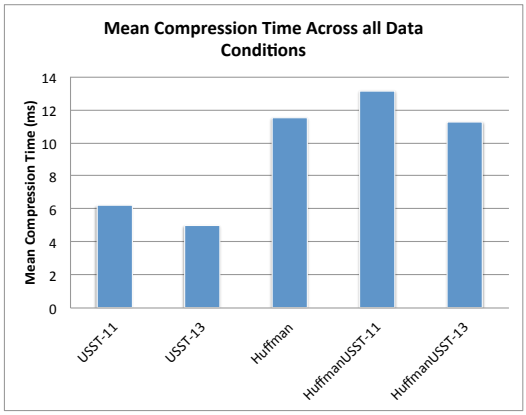

Fig. 2: Overall mean time elapsed

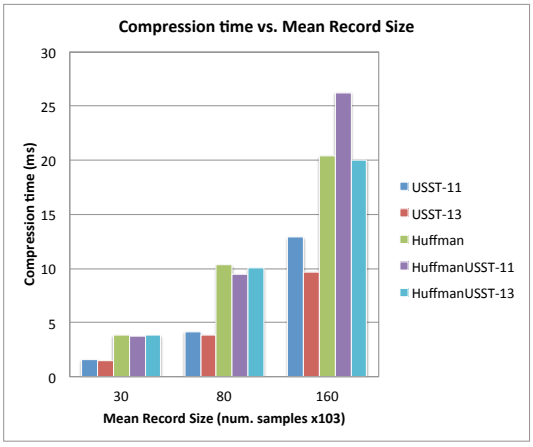

Fig. 3: Mean compression time vs. record size.

The energy, storage, and bandwidth constraints of USST-Sat, combined with knowledge that the bulk of the generated data would be experimental records containing integers of a certain format and range, motivated the design of the content-specific compression scheme *USST-Compress*. In this paper, we described *USST-Compress* and compared it to content-independant compression algorithms on synthetically-generated data.

Compression results for *USST-Compress* with a Huffman coding final pass are comparable to *LZ77*, with overall av-

erages of around 20% of the original record size. Huffman coding alone only managed to compress the test data to about 30% of its original size, while *USST-Compress* variants without a final entropy-coding pass managed to compress to about 45% of the original size. *HuffmanUSST* combinations execution times were comparable to those of Huffman coding, all averaging 10-13ms. *USST-Compress* variants without entropy-coding were the fastest, completing on average in under 6ms. *LZ77* had the longest execution time, taking over 12 seconds.

The results for both evaluation metrics seemed to be generally independent of changes in the data conditions. This was contrary to our expectations, and an interesting result. The conditions chosen were meant to be as realistic as possible for our application, implying that we can trust our compression scheme to perform well under expected operating conditions.

Figure 4 represents the estimated average power savings for each algorithm, for each class of record size. We estimated the cost of transmitting 1 bit at 0.02mJ and the cost per millisecond of cpu time at 0.25W.[5] Compressing saves energy for all record sizes. The low energy cost of running the compression operation on the microcontroller makes even the slowest compression method worthwhile. *USST-Compress* variants without an entropy-coding step run many times faster than *LZ77*, but are worse candidates in terms of energy savings because of their lackluster compression results. The two *HuffmanUSST* variants show the best energy savings.

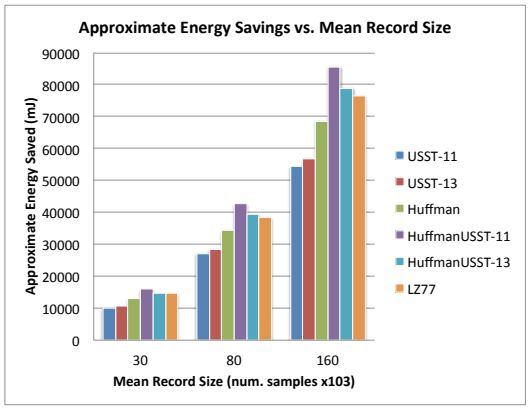

Fig. 4: Approximate Energy Savings

5. CONCLUSIONS AND FUTURE WORK

USST-Compress is evidently no better at compressing the test data than *LZ77*, and no faster than Huffman coding, but it combines the best aspects of each. The *HuffmanUSST* variants consistently proved to compress data 10% better than Huffman coding alone, and are orders of magnitude faster than *LZ77*. This combination of speed and compression performance make *USST-Compress* an excellent method of conserving energy, reducing storage requirements and managing transmission time. It constitutes a novel and interesting application of compression techniques to data management for satellite-based atmospheric scientific measurements.

In the future, we would like to evaluate the performance on other numerical data with and without the differential encoding step to determine how beneficial the step is to the overall performance of the algorithm. We are also curious to see how the number of bits used to store residual values affects the compression ratio. We would like to explore the possibility of first sorting sample values before calculating residuals. Finally, we would like to see the effect of entropy-encoding the bitmap separately from the rest of the record using a different entropy-coding method, such as RLE [8]. We would like to perform more tests with more extreme noise and variance conditions to better determine the sensitivity of *USST-Compress* to changes in the input data. Better measures of power consumption, such as execution counts for specific CPU operations, cache hits and misses or total memory accesses could also prove useful.

[5] SAMS7S256 Data Sheet, Atmel Corporation http://atmel.com/devices/SAM7S256.aspx

6. REFERENCES

[1] N. Kimura and S. Latifi, "A survey on data compression in wireless sensor networks," in *International Symposium on Information Technology: Coding and Computing*, Las Vegas, NV, Apr. 2005, pp. 8–13.

[2] K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression," in *ACM MOBISYS*, San Francisco, CA, June 2003, pp. 231–244.

[3] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[4] D. A. Huffman, "A Method for the Construction of Minimum-redundancy Codes," *IRE*, vol. 40, no. 9, pp. 1098–1101, Sept. 1952.

[5] Przemyslaw Skibinski, "Improving HTML compression," *Informatica*, vol. 33, pp. 363–373, Oct. 2009.

[6] Givon Zirkind, "AFIS data compression: an example of how domain specific compression algorithms can produce very high compression ratios," *SIGGRAPH Comput. Graph.*, vol. 41, no. 4, pp. 3:1–3:36, Nov. 2007.

[7] M. Nelson, *The Data Compression Book*, M&T Books, New York, NY, 1995.

[8] D. Salomon, *Data Compression: The Complete Reference*, Springer-Verlag, New York, NY, 1998.