Adaptive Log Compression for Massive Log Data

Robert Christensen and Feifei Li
robertc@eng.utah.edu
lifeifei@cs.utah.edu
Computer Science Department, University of Utah, Salt Lake City, Utah

THE UNIVERSITY OF UTAH®

Motivation

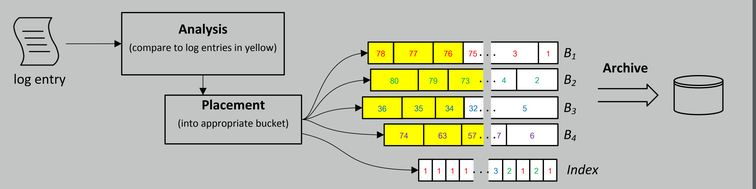
```
[2d] Mar 19 22:41:29 nid00456 #1#-ng[442]: STATS: dropped 0
[2d] Mar 19 22:41:29 nid00528 #1#-ng[451]: STATS: dropped 0
[2d] Mar 19 22:41:29 nid00960 #1#-ng[442]: STATS: dropped 0
[1e] Mar 19 22:41:29 #2# dhcpd: DHCPACK on 10.1.100.183 to 00:0f:cb:9e:bd:40 via eth3
[1e] Mar 19 22:41:29 #2# dhcpd: DHCPREQUEST for 10.1.100.183 from 00:0f:cb:9e:bd:40 via eth3
[2d] Mar 19 22:41:30 nid00268 #1#-ng[442]: STATS: dropped 0
[2d] Mar 19 22:41:30 nid00279 #1#-ng[442]: STATS: dropped 0
[2d] Mar 19 22:41:30 nid00339 #1#-ng[442]: STATS: dropped 0
```

- Log data is **humongous** and often stored for an extended period of time.
- Cost to store log data is proportional to the number of bytes the data occupies.
- Log entries are often **heterogeneous**, with varying patterns over time
- Compressors such as *bzip2* and *gzip* are most **effective** when compressing **predictable data**.

Definition (Adaptive Log Compression)

For a log data D with n log entries $e_1, \ldots, e_n$ (sorted on arrival timestamps), a budget g, produce g *disjoint* log buckets $B_1, \ldots, B_g$, such that $\forall i \in [1, n]$, $\exists j \in [1, g], e_i \in B_j$, and $\forall x, y \in [1, g]$, $B_x \cap B_y = \emptyset$. Each bucket also stores log entries in sorted order. Given any compression method Z, $Z(D)$ is the compressed output with size (in bytes) $|Z(D)|$. The objective is to maximize $|Z(D)| - \sum_{j=1}^{g} |Z(B_j)|$.

System Overview



- Analysis: A new log entry is compared to the m most recent entries in $B_1, \ldots, B_g$ using a similarity function.
- Placement: The log entry is placed into the bucket B_k which resulted in the **best similarity score** during the **analysis** phase. The index k is saved to an index log, so the log can be reassembled when unarchived.
- Archive: Each bucket $B_1, \ldots, B_g$ and the **Index** is compressed separately and in parallel using a compressor such as *bzip2* or *gzip* before being saved on a hard disk.

Test Data

- Red Storm is a system log from the supercomputer **Red Storm**.
- Apache log is an Apache web server log for the website **MesoWest**.

Table: Real world data

Source	Uncompressed (MB)	*bzip2* (MB)	*gzip* (MB)	log entries	med entry length
Red Storm	32,596	664	1080	219,096,168	99
Apache log	7501	296	533	26,568,851	277

References

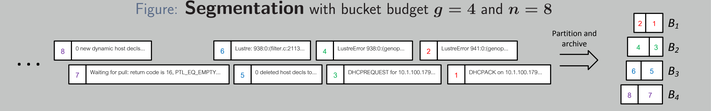
[1] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla.
On synopses for distinct-value estimation under multiset operations.
In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 199–210. ACM, 2007.

Similarity Functions

Figure: **Centralized**

Centralized places each entry in a single archive file in chronological order. This is the method applied currently in industry when compressing logs.

Figure: **Segmentation** with bucket budget $g = 4$ and $n = 8$



Segmentation only considers the total number of log entries n, and the bucket budget g. Each bucket will be given an continues chunk of the log with an equal number of log entries. B_1 will contain $e_1, \ldots, e_{\frac{n}{g}}$, B_2 will contain $e_{\frac{n}{g}+1}, \ldots, e_{2\frac{n}{g}}$, etc.

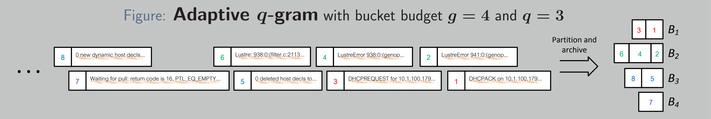
Figure: **Round Robin** with bucket budget $g = 4$

Round Robin inserts the ith log entry into the bucket B_k, where $k = i \bmod g$.

Figure: **Adaptive Word** with bucket budget $g = 4$

Adaptive Word inserts each word of an incoming log entry into a set W. The incoming log entry will be inserted into the bucket B_k which has the largest **average** Jaccard index when compared with the log entries in B_k within the sliding window.

Figure: **Adaptive q-gram** with bucket budget $g = 4$ and $q = 3$



Adaptive q-gram splits an incoming log entry into q-grams, which are inserted into a set W. The incoming log entry will be inserted into bucket B_k which has the largest **average** Jaccard index when compared with the log entries in B_k within the sliding window.
Computing the Jaccard index between sets which contain a large number of q-grams can be very expensive. In order to make **Adaptive q-gram** practical, the Jaccard index was **estimated** using a KMV synopsis[1]. Estimating the Jaccard this way did not degrade the final archive compression ratio.

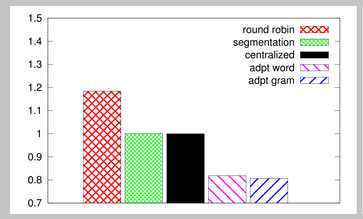
Experimental Setup

We use **centralized** as a reference for comparing the effectiveness of the other methods. The output size of any method is shown as a **ratio to the output size of centralized**.

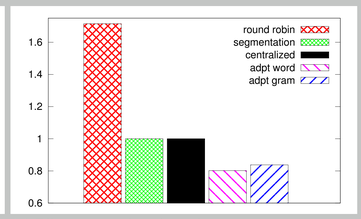
Table: default settings for the similarity functions. The value is ignored if not applicable.

variable	value
bucket budget	$g = 32$
sliding window size	$m = 10$
q-gram length	$q = 6$
synopsis size	$k = 60$

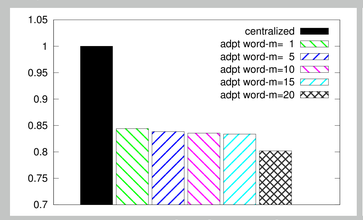
Results



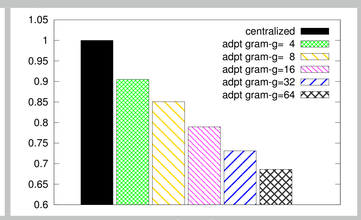
(a) Effectiveness of similarity functions on **Apache log** using the *bzip2* compressor when compared to **centralized**.



(b) Effectiveness of similarity functions on **Red Storm** using the *gzip* compressor when compared to **centralized**.



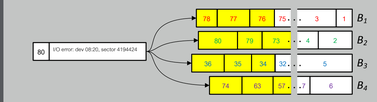
(c) Effectiveness of the **adaptive word** similarity function on **Red Storm** using the *gzip* compressor while varying the sliding window depth m when compared to **centralized**.



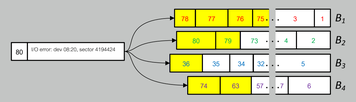
(d) Effectiveness of the **adaptive q-gram** similarity function on **Apache log** using the *bzip2* compressor while varying the number of buckets g when compared to **centralized**.

- Naive log distribution can significantly **decrease** the compression effectiveness, as shown by **round robin** in figure (a) and (b).
- **adaptive word** improves the compression ratio on **Red Storm** by up to **20%**. see figure (c).
- **adaptive q-gram** improves the compression ratio on **Apache log** by up to **30%**. see figure (d).
- The number of buckets **positively** correlates with the compression ratio. see figure (d).

Future Work

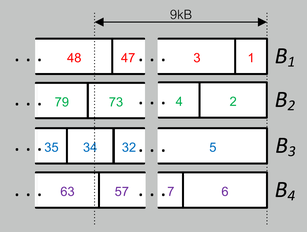


(a) current method, **static** sliding window depth

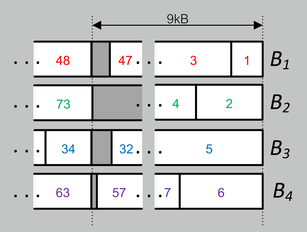


(b) proposed method, **dynamic** sliding window depth

- Sliding window depth m should be dynamic.
  - ▷ Smaller when bucket contents is nearly **identical**.
  - ▷ Larger when bucket contents are more **varied**.



(a) current method, compression blocks are **not** aligned



(b) proposed method, compression blocks are aligned

- Compression utilities compress data is small compression blocks.
  - ▷ Maximum block size for *bzip2* and *gzip* is **9KB**.
  - ▷ Compression tables are **not shared** between blocks.
- Straddling log entries between two compression blocks can **decrease** compression **effectiveness**.
- Avoiding straddling log entries **isolates** archive **corruption**.