Adaptive Log Compression for Massive Log Data

Robert Christensen Feifei Li
robertc@eng.utah.edu, lifeifei@cs.utah.edu, School of Computing, University of Utah

ABSTRACT

We present a novel adaptive log compression scheme. Results show 30% improvement on compression ratios over existing approaches.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management – *Systems*

Keywords

Log compression, adaptive log compression, log data management

1. PROBLEM STATEMENT

Log data is ubiquitous and humongous. The standard log compression method is to compress the entire log data together.

But in practice log entries are often heterogeneous, with varying patterns over time. They also have strong temporal locality. Thus, a better approach is to adaptively distribute entries to different buckets, and compress buckets separately in parallel. Formally,

Definition 1 (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 sorted log entries. Given any compression method Z, $Z(D)$ is the compressed output with size (in bytes) $|Z(D)|$. The objective is to maximize $|Z(D)| - \sum_{j=1}^{g} |Z(B_j)|$.

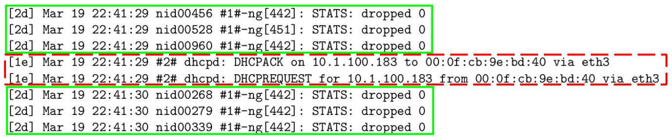
2. TECHNICAL APPROACH

Consider a snippet from a real log data in Figure 1. To minimize heterogeneity in a log bucket, ideally, log entries in green solid box should be partitioned into one log bucket, and entries in red dashed box will be in another bucket. The challenge is to achieve adaptive distribution online to get homogeneous buckets.

We impose a sliding window w_i of size m on bucket B_i, that keeps track of most recent m entries in B_i. The jth entry in w_i is $e_{i,j}$ ($e_{i,1}$ being the most recent). For each entry, we construct a *signature* with a mapping function σ, i.e., $\sigma_{i,j} = \sigma(e_{i,j})$. Assume a similarity function $\mathsf{sim} : (\sigma_1, \sigma_2) \to [0, 1]$ which gives a similarity score for any two signatures. For an incoming log entry e_{next}, we define the *score* of e_{next} on the ith bucket B_i as follows:

$$s(e_{\text{next}}, i) = \text{avg}\{\mathsf{sim}(\sigma(e_{\text{next}}), \sigma_{i,j}), \text{ for } j \in [1, m]\}, \quad (1)$$

i.e., the average similarity between e_{next}'s signature and any signature of the m entries in w_i. Our *adaptive scheme* sends e_{next} to bucket B_j with the maximum score, i.e, $j = \text{argmax}_{i \in [1,g]}\, s(e_{\text{next}}, i)$.



```
[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
```

Figure 1: Snippet from the Red Storm HPC server log.

When all entries have been processed, we compress $B_1, \ldots, B_g$ independently in parallel. We base our construction of σ by viewing each log entry as a set of elements (e.g., q-grams). We then develop a log signature σ based on the k-minimum value synopses [1]. Different similarity functions sim can be used; we report only the one that is a variant of the *Jaccard similarity*.

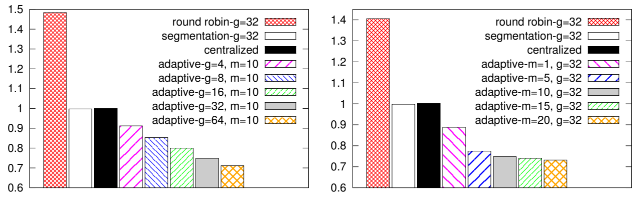
3. MAIN RESULTS

Compressing the entire log together is dubbed *centralized*. Two naive bucketization methods, *round robin* and *segmentation*, were also tested. The former distributes entries to different buckets in a round robin fashion; the latter divides D into g *disjoint but contiguous* segments with same number of entries.

We show the results using gzip on an Apache web server log. It has 26 million entries, totaled 8GB. Other compression methods and datasets were also tested, giving similar results.

We use *centralized* as a reference point which has a compressed size of 533MB using gzip. The output size of any method is shown as a *ratio to the output size* of *centralized* in Figure 2. For *adaptive*, *round robin*, and *segmentation* methods, the default bucket budget g is 32; for *adaptive*, the default sliding window size m is 10.

Figure 2 shows up to 30% improvement in compression ratios achieved by *adaptive* over *centralized*, using small g and m values. It also shows the ineffectiveness of naive bucketizations. Lastly, because *adaptive* is a streaming algorithm, and can distribute entries and compress buckets independently in parallel, its overall running time is comparable to that in *centralized* (omitted for space).



(a) Vary bucket budget g. (b) Vary sliding window size m.

Figure 2: Effect of bucket budget and history depth.

4. CONCLUDING REMARKS

An interesting challenge is to improve *adaptive* to learn the best values for g and m online and adjust them dynamically over time.

5. REFERENCES

[1] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, 2007.