MemGator - A Portable Concurrent Memento Aggregator

Cross-Platform CLI and Server Binaries in Go

Sawood Alam
Old Dominion University
Norfolk, VA (USA)
salam@cs.odu.edu

Michael L. Nelson
Old Dominion University
Norfolk, VA (USA)
mln@cs.odu.edu

With the growth in the number of public web archives it is becoming important to provide a means to aggregate them for better coverage and completeness. The Memento protocol [2] provides a uniform API to lookup URIs in web archives. Due to the wide support of the Memento protocol in the archiving ecosystem it is now easy to aggregate their holdings for any URI lookup. However, current applications can either use their custom aggregator implementation or rely on centralized services such as LANL's Time Travel portal[1] and ODU Memento Aggregator[2]. While centralized third party services are serving their purpose well, the convenience has the trade-off of lack of customization and control such as the client application cannot choose which archives to be aggregated. Centralized services are usually good for general usages, but not suitable for specialized purposes such as research or heavy traffic applications. For example, certain archives have IP-based traffic throttling policies which might limit the ability of the centralized server in case of heavy traffic. Similarly, the recent surge of OldWeb.today caused increased load on archives, as a result one archive requested its exclusion from being polled. This would have been an issue if they were using a centralized service.

There are a few open source aggregator implementations such as Memento Server[3] and Memento Java Client Library[4], but they are either outdated or require a server setup.

With these issues in mind, we created MemGator that provides a standalone cross-platform binary without any external dependencies. It can be used as a one-off command to retrieve the response on the standard output or run as a web service to replicate necessary features of the centralized Memento aggregator services. We tried to keep the service API as close as possible to the LANL's Time Travel service for greater interoperability. Both the modes (CLI and server) come with a handful of customization options that are documented in the binary itself and can be seen using standard help flag. One such configuration option is to supply a custom list of archives to be aggregated or use the archive profile based archive ranking to query top-K archives only. The tool is currently being used heavily in OldWeb.today, WAIL[5], Mink[6], and our internal archiving research projects. It has proved to be reliable even in the extreme load conditions.

An aggregator is a good example of a concurrent application. It relies on various upstream archives which consumes the maximum amount of the overall time in network I/O while the process sits idle. Performing this operation sequentially will make it useless as the number of upstream services grows. We chose the Go language primarily because it is designed with the concurrency in mind and has features that make development of concurrent web applications easy. Additionally, it provides the ability to create cross-compiled cross-platform static binaries.

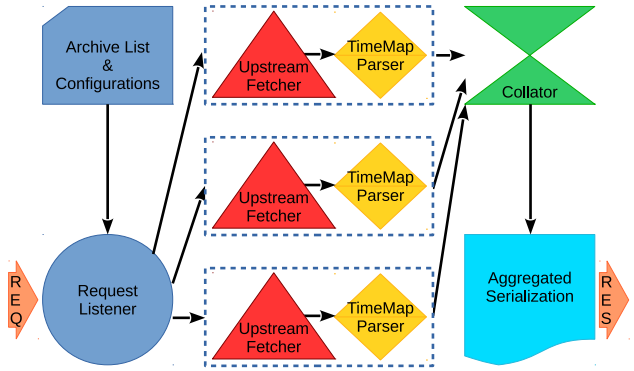

Figure 1: MemGator Workflow Diagram

Figure 1 illustrates the workflow of the MemGator implementation. The main thread (the request listener) loads the list of archives and other configuration options. When a lookup request arrives, it spins off goroutines (lightweight threads of Go) for each individual archives. These individual goroutines fetch the TimeMap from individual archives independently and in case of a successful response they pass the data to a TimeMap parser via a channel (message passing mechanism of Go) which makes a linked list from the response in the chronological order. The parser sends the linked list data to the collator which accumulates responses from each individual goroutine and merges them while maintaining the sorting. Once all goroutines are completed or timeout occurs, the accumulator passes the aggregated linked list to the serializer. Depending on the format requested by the client (such as Link or JSON), the data is serialized and returned as the response.

An expanded version of this work is published at [1]. We made the source code and binaries publicly available[7]. We are also running it as a reference web service[8].

REFERENCES

[1] S. Alam and M. L. Nelson. MemGator - A Portable Concurrent Memento Aggregator: Cross-Platform CLI and Server Binaries in Go. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, JCDL '16, pages 243–244, 2016.

[2] H. Van de Sompel, M. L. Nelson, and R. Sanderson. HTTP Framework for Time-Based Access to Resource States – Memento. RFC 7089, Dec. 2013.

[1] http://timetravel.mementoweb.org/guide/api/
[2] http://mementoproxy.cs.odu.edu/
[3] https://code.google.com/p/memento-server/
[4] https://github.com/ukwa/mementoweb-client-java
[5] http://machawk1.github.io/wail/
[6] https://github.com/machawk1/mink
[7] https://github.com/oduwsdl/memgator
[8] http://memgator.cs.odu.edu/