Tibor: Event-Driven Static File Cache and Use in an Erlang File Server

Daniel Lovasko[1], Jacopo Notarstefano[2], Laura Rueda[2] and Jan Stypka[2]
[1] *lovasko@freebsd.org (The FreeBSD Project)*
[2] {*jacopo.notarstefano, laura.rueda, jan.stypka*}*@cern.ch (CERN)*

***Abstract*—In this paper we introduce Tibor, an innovative implementation of an event-driven server-side cache for static files. Tibor was developed during the EPFL Facebook Hackathon, in the wee hours of the morning. The authors take full responsibility for the horrors contained in its codebase.**

1. Introduction

Drawing on their experience as contributors to Invenio, the library powering several of the biggest digital libraries of Europe[1], the authors noticed the opportunity for creating a software specialized in the server-side caching of static files.

In fact, the serving of static files of sizes ranging from modest (for example, the papers of the High Energy Physics community stored in INSPIRE) to significant (for example, the historical movies stored in the CERN Document Server) is at odds with the aging architecture of the Invenio library, a problem that can only be alleviated, but not solved, by falling back to the Apache Web Server.

The authors propose Tibor, a novel solution written in C as a stand-alone library with minimal dependencies. To demonstrate its versatility, the authors provide a proof-of-concept integration with Elli, an Erlang web server.

Even if this won't provide a solution to the problem the authors are facing, we confide that this integration constitutes sufficient proof of the wide applicability of this solution to disparate domains[2].

DL, JN, LR, and JS
April 19, 2015

2. High Level Architecture

As most caches, Tibor in structured in two levels, conventionally called "L1" and "L2". It is expected that accesses to the L1 cache are significantly faster than those to the second level, at the price of a smaller cache size. In the next section we will describe the data structures and the search algorithms that Tibor uses to achieve this goal.

Tibor is configured, as nowadays is common, by editing a file in the JSON format. In the third section we will describe its format, the currently supported features and the plans for extending them.

In the fourth section we will describe Jiří, the mechanism that Tibor uses to communicate with the Elli web server. As it will be clear from the discussion, communicating between BEAM, the Erlang Virtual Machine, and `tibord`, the daemonized version of the Tibor library, requires a little ingenuity and some knowledge of the inner working of these two pieces of software.

Finally, in the last section, we will describe the Simko prototype to visualize and provide guidance in improving Tibor's performance.

3. Data Structures

The L2 cache is arranged as a trie whose leaves contain the contents of the files in the cache. Every internal node represents a prefix of some number of paths to those files, with a child for every valid next character in the path to some file.

In the L1 cache the data is arranged into a small table of the following format:

hash	length	key	data
...	...	...	...

The first field is the hash value of the file path, calculated using the FNV1a algorithm. Then we have the length of the key, the key itself and a pointer to a leaf in the L2 trie.

The lookup begins from the L1 cache. First we compute the hash of the requested filename. Then we compare it in turn with every element of the first level; in case of a match the lengths and the keys itself are compared. If all of those tests succeded then we are sure we have an L1 cache hit, and we return the pointer to the data. Otherwise we begin to lookup in the L2 cache using the standard trie visit algorithm.

4. Configuration

We first present a complete example of a Tibor's configuration file, and then we will comment on its features.

```
{
  "path" : "/root/tibor/images",
  "max_file_size" : "1M",
  "l1_size": 10,
```

1. Among which the CERN Document Server, INSPIRE and the Infoscience project at EPFL

2. Possibly including the problem of visiting California, or at least getting a few t-shirts.

```
  "mushrooms" : {
    "1.png" : {
      "on_delete" : "drop",
      "on_modify" : "update",
      "on_attrib" : "keep",
      "on_rename" : "follow",
      "on_l1": true
    },
    "2.png" : {
      "on_delete" : "drop",
      "on_modify" : "update",
      "on_attrib" : "keep",
      "on_rename" : "follow",
      "on_l1": false
     }
}
```

Tibor supports some expected settings, such as the root path of the files served, the size of the L1 cache in MB and in the number of files. In the `mushrooms` section the user can specify which files are stored in the cache and with which policies. For example, the policy "`"on_delete": "keep"`" specifies that, on file cancellation, the file should stay in the cache, while "`"on_attrib": "drop"`" says that, following a change in permissions, the file should be evicted from the cache.

We remark a feature afforded by Tibor's unique architecture: it is possible to invalidate a single cached element just by changing its permissions, provided that the appropriate policy was selected during configuration.

The authors recognize that configuring a large cache could prove to be tedious. Therefore, they are planning to include the possibility of defining *memory pools* described by regular expressions, as in the following example:

```
{

  [...]

  "pools" : {
    "all_images" : {
      "filename_pattern" : ".*\\.jpg$",
      "size" : "10M",
      "on_delete" : "drop",
      "on_modify" : "update",
      "on_attrib" : "keep",
      "on_rename" : "follow"
    },
    "text_files" : {
      "filename_pattern" : ".*\\.txt$",
      "size" : "10M",
      "on_delete" : "drop",
      "on_modify" : "update",
      "on_attrib" : "keep",
      "on_rename" : "follow"
    }
  }

  [...]

}
```

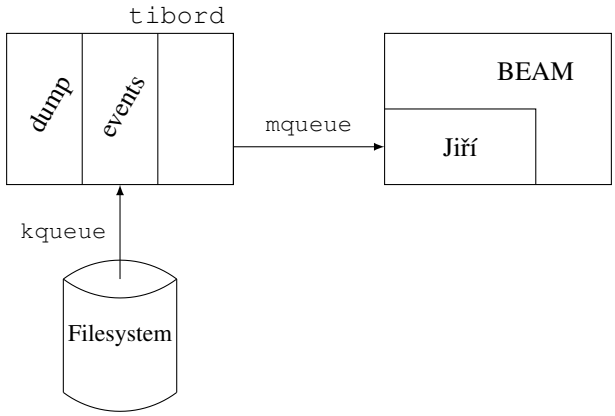
5. Internals

Since Tibor is a small C library, we expect to be able to integrate it in any web server written in a language that is either C or that allows FFI calls to C code. This unfortunately is not true with Erlang, which requires a more convoluted approach.

We first daemonize the Tibor library into the Tibor daemon `tibord`. This demon contains at least three threads: one to periodically dump diagnostic data, another to listen to file system events and one that communicates with Elli, the Erlang web server.

We would like to share memory between `tibord` and Elli, but programs written purely in Erlang do not allow us to do that. Instead, we run some C code (a component we called Jiří) inside of the Erlang Virtual Machine, which then allows other external C code to communicate with it using SystemV message queues and shared memory.

The following picture illustrates the relations between the various components:



6. Simko: a visualization layer for Tibor

While developing Tibor, the authors noticed that its users could benefit from inspecting the status of the cache and diagnosing its inefficiencies, either caused by misconfiguration or by bugs in the software. Therefore the authors decided to implement a graphical tool that would allow the users in easily identifying missed opportunities for optimization and wasted resources.

Since Tibor's support for logging is still in its infancy, the communication between Simko and Tibor is achieved by reading and writing shared files in the CSV format. We auspicate that in the future a more robust medium of communication is estabilished between these two softwares.

Despite this limitation, Simko can show the current status of the cache, automatically refreshing every 5 seconds. The colors and the lenghts of the bars encode the number of times an item was accessed and found in L1, L2 or not found at all.

The user can immediately see which files where accessed often despite not being picked for the cache. On the other hand, she can also see which files she selected weren't accessed as much as she expected.

Simko goes even further by automatically suggesting which files should be evicted by the user from the cache and which files should be included instead.

7. Conclusion

In this paper we presented Tibor, an event-driven server-side cache for static files. We demonstrated in which sense it is event-driven, and we argued why this matters. We also introduced Simko, the first tool of the Tibor ecosystem.

Acknowledgments

The authors would like to thank their bosses Tibor Šimko, Jiří Kunčar and Lars Holm Nielsen for being constant sources of inspiration. The authors also thank Marios Kogias for the productive conversations that shaped the design of the Tibor library, and for having the courage to step back at a crucial moment in Tibor's history.