Decouple deployment from application logic for web services

Etienne Brodu
etienne.brodu@insa-lyon.fr
Université de Lyon, INRIA,
INSA-Lyon, CITI-INRIA,
F-69621, Villeurbanne, France

Stéphane Frénot
stephane.frenot@insa-lyon.fr
Université de Lyon, INRIA,
INSA-Lyon, CITI-INRIA,
F-69621, Villeurbanne, France

Frédéric Oblé
frederic.oble@worldline.com
Worldline
53 avenue Paul Krüger - CS 60195
69624 Villeurbanne Cedex

Abstract

Web applications evolve constantly, both in features and in performance. At first, the application evolution is dominated by the need for features. The development team logically organizes the application in modules. By keeping modules lights and decoupled, they never get overwhelmed by the growing complexity of the application. But the need for performance grows with the audience of the application. At some point it is necessary to organize the code differently to leverage the parallelism of clusters. A particularly efficient solution is to slice the logic into stages to form a pipeline [3].

Both organizations are carried manually over the source code, and interfere with each other. We believe these interferences freeze the evolution, and are an obstacle for the economic growth. We propose to automate the slicing process. It would allow developers to keep a coherent organization of modules, while beneficing from the efficiency of the pipeline solution. It removes the economic obstacles.

We observe a resemblance between Javascript and pipeline. Javascript popularity recently exploded, in part due to the execution model Node.js. The latter is based on an event-loop and is efficient compared to existing technologies [2]. An event-loop is a pipeline executing on a single-core. We investigate this idea to transform a single event-loop into a fully parallel pipeline.

An event loop triggers handlers to react on events. Event handlers end without return, they asynchronously trigger the next handler execution. The call stacks of two handlers are disjoints. Thus, it is possible to parallelize their execution, as long as the causality is preserved. Every event handlers becomes a stage in the pipeline.

However, the memory is global, and holds both the communications between the handlers, and the state of each handler. In a mono-thread execution model like Node.js, the atomic execution of handlers make synchronization mechanisms useless [1]. To parallelize the handlers, we isolate the state of each handler to allow exclusivity ; and we identify their communications to reproduce a one way flow of messages.

We built a first incomplete compiler as a proof of concept. After successful results with custom applications, we are building a complete compiler to transform real applications.

Références

[1] A Adya, J Howell et M Theimer. “Cooperative Task Management Without Manual Stack Management.” In : *USENIX Annual Technical Conference* (2002).

[2] Kai Lei, Yining Ma et Zhi Tan. “Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js”. In : *2014 IEEE 17th International Conference on Computational Science and Engineering*. IEEE, déc. 2014, p. 661–668. doi : 10.1109/CSE.2014.142.

[3] M Welsh, SD Gribble, EA Brewer et D Culler. *A design framework for highly concurrent systems*. 2000.