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

Real-time web applications needs to be highly concurrent, and to scale over a large cluster. We identify two types of parallelism to meet such requirements. Data parallelism replicates the whole logic to process each request in a thread [2]. Pipeline parallelism slices logic into stages, each on an independent thread [3]. We believe the two designs are on two ends of a design spectrum; and each application requires a specific combination of the two designs. SEDA [4], Storm and Spark are frameworks to design such combinations.

The two types of parallelism reveals the duality of threads. Data parallelism uses a logical thread for each request. It defines an encapsulation for the logic of processing a request. Pipeline parallelism uses an execution thread for each core. It leverages parallelism. The motives to use logical threads and execution threads are opposite. We argue that it is a mistake to map the former onto the latter.

Pipeline parallelism implies to manually slice an application for its deployment. Because it is a manual process, it implies to reduce logical threads to execution threads. We believe it limits the design and the evolution of a web application. Behren *et. al.* stated that *it is a mistake to attempt high concurrency without help from the compiler* [2]. Following this advice, we propose to automate this slicing process. We expect to improve design flexibility by allowing developers to dissociate logical from execution threads. For this automation, we resolved two main issues. Execution distribution and memory distribution.

After decades of improvements, the event and the thread model are now almost alike for a single core [1]. Still, the event model conserves a particularity. 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 distribute the execution on multiple machines, as long as the causality is preserved.

Each distributed handler needs to access the shared memory. Web applications still heavily rely on single databases to store the whole state. With recent trends around in-memory caches, we believe it is now viable to bring closer the logic and its state. We choose to partition and distribute the memory. We detect the dependencies of each handlers through static analysis. The handlers communicate through a one way flow of messages to exchange the shared states.

We built a first incomplete compiler as a proof of concept. After successful results with custom applications, we are now on the process of 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] JR von Behren, J Condit et EA Brewer. "Why Events Are a Bad Idea (for High-Concurrency Servers)." In : *HotOS* (2003).

[3] J Ousterhout. "Why threads are a bad idea (for most purposes)". In : *Presentation given at the 1996 Usenix Annual Conference* (1996).

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