Advanced Operating Systems
EbbRT: A Framework for Building Per-Application Library Operating Systems

Presented by
BARALLON Lucas and SID-LAKHDAR Riyane
(M2 MoSIG: ENSIMAG / UGA)

January 3, 2017

Abstract

In this document, we refer to the paper of D. Schatzberg et al [3] in order to present the library OS framework "EbbRT". We first present the main design specifications of this framework. We link each of this design to a specific performance or portability objective.
In a final section, we show how, in our vies, the performance elasticity of the EbbRT is clearly reached while its portability may be not be assessed regarding the provided evaluations.

Contents

1 The EbbRT Framework

In this paper the authors exposed this problem : there is a trade-off between performance and generality with actual operating systems. With this in mind they want to bring high performance to the broader number of software possible.

There idea is to used a framework for constructing per-application library operating systems : the Elastic Building Block Runtime (EbbRT). With this framework, there are 3 objectives: The high performance of specialized application, a wide applicability to reach the largest number of software, and also simplify the development of system specific software.

To achieve these goals the framework used a particular structure. A typical deployment of an EbbRT application is composed of several part (called representative). These representative are running at user-level library (hosted runtime) or directly as a part of the OS (native runtime). Also these elements are link to Elastic Building Block(Ebb) which encapsulate the data and the details of the object manipulated. Ebb are highly customizable and are the element facilitating the work of the developers.

An EbbRT execution is also "non preemptive and event-driven". In their case the overhead of the abstraction provided is very low in such a way that the software is directly connect to the hardware interrupt. The cost of scheduling are thus avoided.

To make EbbRT lightweight and not enforcing the compatibility to all existing interfaces, the software doesn't rely on a complete Linux or Posix compatibility. Leverage by the offload of functionalities with the heterogeneous platform.

An other point allowing software to used blocking systems calls is the cooperative threading model. By explicitly saving and restoring state(stack/registers) it allow to pause and resume an event during the event loop.

A more technical aspect are take into the implementation of the memory management, the control flow of the application and then the network stack. A lot of details are provide from the multi-core partition of the memory thanks to the heterogeneous structure to fine grain detail TCP into the network stack.

The authors end by a part about performance where they show first different evaluation about the memory allocation and the network performance. They used benchmark with Memcached, a Javascript bench with Node.js and a webserver evaluation also with Node.js. The result gather show that EbbRT have better result in each of the test performed, having a slight amelioration with test like the webserver to outstanding result in Memcached.

2 EbbRT: efficient application-fitting framework

The considered Schatzberg et al paper presents a framework to build library OS. But throughout this implementation, there is an interesting method that is described to build a bridge between general-purpose OS (that hardly adapt to the performance specifications of an application) and the application specific OS (that require a significant engineering effort and can not host most "general applications").

The efficiency fitness of the EbbRT is reached through a set of simple designs such as an event-oriented environment, object-oriented programming mechanism. Thus this implementation may reach an important characteristic for an OS: be affordable for the highest number of programmers (with few system knowledge).

Meanwhile, the EbbRT has covered an interesting set of application performance specification. Different general-purpose OS bottleneck have been considered such us the dynamic memory allocation or the network card packet transfer. Different implementations are available depending on the usage pattern of the hosted application.

Thus, the performance drawback of general-purpose OS are properly identified and tackled using a coherent set of measures. Thus, the EbbRT may well fit the need of a specific application. But what about the "generality degree" reached by this implementation.

3 Limitations of the EbbRT evaluation

The main limitations highlighted are linked to the "broad applicability" stated about the EbbRT-based OS. The considered paper argues that despite the avoidance of some mechanisms from the general-purpose OS, the EbbRT may port a significant set of general purpose applications. In this section, we show the limits and the potential incoherence of this statement and its assessment.

- The event oriented environment implemented by the EbbRT framework obviously reduces the execution time of a single application (as it does not stop it at each quantum). But it is also a drawback to port thread-oriented applications on the EbbRT OS. The paper handles this issue by proposing a concurrent thread-oriented running environment on the top of the native non-preemptive one. But is this adaptation cost-less in term of performance? What is the overhead due the dynamic preemption of running events. How important is this overhead compared to the EbbRT performance gain.
- One of the main arguments highlighted to promote the EbbRT is that the framework allows to embed heterogeneous execution environment with **a**

relatively reduced engineering effort[1]. And this was one of the angular-stones that make the EbbRT built OS reach significant "broad applicability". Two issue are raised by such a statement: First the notion of "reduced engineering effort" is very weakly defined. Indeed, we know that when developing such a general purpose environment, the main development time is dedicated to the debugging and the testing (due to the very large matrix of potential usages). The paper provides no evaluation of this environment within "real constraints".

Second, the paper provides a performance evaluation of the developed *node.js* execution environment. But once again, this test is meaningless regarding "broad applicability" of the EbbRT framework: it does not assess the different expectation that a general purpose application may expect of the node (such as implementation of multi-threading, system calls).

References

[1] Wikipedia: Unikernel, os library. `https://en.wikipedia.org/wiki/Unikernel#Library_operating_systems`.

[2] D. Schatzberg, J. Appavoo, J. Cadden, and O. Krieger. Multilibos: An os architecture for cloud computing. Technical report, Computer Science Department, Boston University, 2012.

[3] D. Schatzberg, J. Cadden, H. Dong, O. Krieger, and J. Appavoo. Ebbrt: A framework for building per-application library operating systems. 2016.

[1] "... node.js, a managed runtime environment was ported in two weeks by a single developer" [3], The EbbRT Framework