Basslet: an OS runtime for parallel data processing

Jana Giceva, Gerd Zellweger, Gustavo Alonso, Timothy Roscoe
Systems Group, Department of Computer Science, ETH Zurich
{name.surname}@inf.ethz.ch

1. INTRODUCTION

We revisit the problem of scheduling multiple parallel programs on modern hardware, in the light of new OS and application designs.

Current operating systems are oblivious to the application-level information needed to efficiently schedule and execute parallel data applications (e.g. [2]). Traditionally they decide on the allocation of threads unaware of the internal properties of the application's algorithm, relations between threads or the characteristics of the input data, often leading to unpredictable and sub-optimal performance. As a result, operating systems either open up their interfaces so that the application can take over the management of resources (two-level scheduling [1,4,5]), or make use of parallel runtime systems to perform this task on behalf of the operating system.

This approach works for a single application (albeit at a high cost for the developer) but it does not help when data processing engines must execute many concurrent jobs, as it often happens in server consolidation and multi-tenant scenarios. One option is to use Callisto [3], a shared library that helps parallel runtimes coordinate their execution, but it relies on them to be well behaved and collaborate with each other.

This talk presents the design of Basslet – an OS runtime for parallel data processing. The proposed runtime is integrated with the rest of the OS and uses dynamic, dedicated kernels specialized for task-based scheduling across CPUs.

2. DESIGN AND PROTOTYPE

We are rethinking the separation of responsibilities and push down the decision-making power back to the operating system.

Based on our experience with implementation and scheduling of multiple parallel data-processing systems, we have identified the need for different execution strategies for two types of threads in such applications, and propose the following: (i) an expanded OS interface that allows an application's knowledge of internal *thread* groups to be passed down to the OS, (ii) design principles for an OS-provided *task*-based runtime, (iii) a control/data plane for CPUs that provides resource isolation in the underlying hardware architecture, and (iv) a runtime implementation that is integrated with the Barrelfish operating system [6], and exploits its recent support for dynamically booting dedicated custom kernels [7] to provide low-overhead, system-wide task-based scheduling.

We call our early implementation of this collection of techniques *Basslet*. Figure 1 presents its architecture. At its heart, Basslet bears a resemblance to task-parallel runtimes by allowing applications to enqueue parallel tasks which are then executed concurrently. However, the integration of Basslet as part of the OS allows tasks to profit from stronger guarantees with respect to their scheduling. In addition, Basslet allows related tasks to be further grouped together by introducing the concept of a parallel task

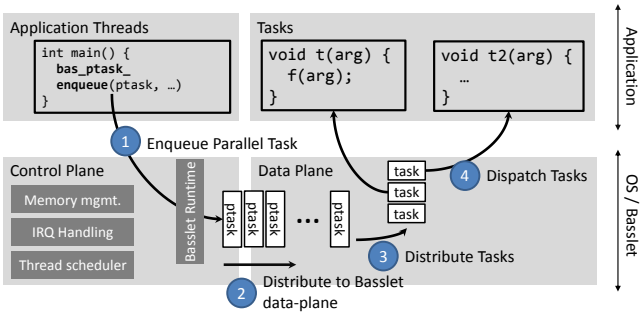

Figure 1: Basslet's architecture

(or ptask). The implementation, uses a novel approach that partitions a multicore machine into a control plane, for the execution of regular application threads, and a data plane for the execution of tasks. Such a separation allows us to specialize the Basslet system-software of the data plane for task-based execution, while also partitioning the hardware resources to minimize interference, without impacting the rest of the OS.

Our goal is to concurrently execute different applications on a multicore or rackscale system with both performance isolation guarantees and optimized resource utilization, without requiring developers to learn complex new programming models or interfaces. We also see Basslet as a means that bridges the problem of scheduling the resources of one machine to the one of a rack, as it increases the unit of management. It also provides a better framework for reasoning about global resource management in a heterogeneous system as Basslet's data plane can be a good match for hardware accelerators (e.g., an Intel Xeon Phi).

3. REFERENCES

[1] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: an operating system architecture for application-level resource management. SOSP '95, pages 251–266, 1995.

[2] J. Giceva, G. Alonso, T. Roscoe, and T. Harris. Deployment of Query Plans on Multicores. *PVLDB*, 8(3):233–244, 2014.

[3] T. Harris, M. Maas, and V. J. Marathe. Callisto: Co-scheduling Parallel Runtime Systems. EuroSys '14, pages 24:1–24:14, 2014.

[4] R. Liu, K. Klues, S. Bird, S. Hofmeyr, K. Asanović, and J. Kubiatowicz. Tessellation: Space-time Partitioning in a Manycore Client OS. HotPar, pages 10–10, 2009.

[5] H. Pan, B. Hindman, and K. Asanović. Composing Parallel Software Efficiently with Lithe. PLDI, pages 376–387, 2010.

[6] The Barrelfish Project. Barrelfish Operating System. `www.barrelfish.org`. accessed 2016-01-27.

[7] G. Zellweger, S. Gerber, K. Kourtis, and T. Roscoe. Decoupling Cores, Kernels, and Operating Systems. In *OSDI 14*, pages 17–31, Oct. 2014.