A Fast Data Plane for Data-Intensive Applications

Shoumik Palkar and Matei Zaharia.

Motivation. Data scientists compose libraries from a diverse range of domains (relational processing, graph analytics, machine learning, etc.) to build complex workflows. Even though functions in these libraries are often highly optimized, programmers who *compose* these libraries can still be orders of magnitude off bare-metal performance because of a lack of *end-to-end optimization* in their workflow.

We believe there are two key challenges for providing bare-metal performance in data-intensive workflows. First, fast implementations often need to peek "under the hood" and throw out abstractions which impede optimization. The humble function call is one such abstraction. Programmers compose libraries by composing function calls but do not achieve "machine level" performance because of data movement costs between these functions and lack of optimizations such as vectorization or tiling across them.

Second, libraries must leverage ever-improving parallel hardware to achieve the best performance. With Moore's Law ending speedups in applications will come from specialized parallel architectures rather than steady improvements in commodity CPUs. Supporting hardware such as FPGAs, GPUs, and coprocessors such as Intel's Xeon Phi will become increasingly necessary and an interface which makes targeting these platforms easier must emerge. Finally, determining *when* to leverage these platforms is also a challenge; systems will need to consider the entire application when making these scheduling decisions.

Our Existing Efforts. At CIDR 2017, we proposed a small position paper on a system called Weld, which is one step toward creating an interface which in part addresses these two challenges. Weld contains an interface which allows library developers to express their core computations in a small parallel IR. This interface is invoked through an API which can compose fragments of this IR across function calls even between different libraries. When an application needs a result, Weld *generates* parallel code from the IR for the entire workflow by and eliminates unnecessary data movement between functions while performing optimizations such as vectorization and tiling; it also targets different parallel platforms. We showed that Weld accelerated workflows in Spark SQL by 7x, TensorFlow by up to 30x, and workloads which combined libraries by up to 30x on a single thread and 270x on multiple threads by automatically parallelizing tasks.

Proposal. We propose taking the idea of generating code for a workflow one step further. Data intensive applications can run at scale across many nodes and involve many different components such as the network, I/O, and the operating system. A system like Weld can generate code for each query across a cluster to optimize each of these components.

As an example, parsing data loaded from disk can take millions of CPU cycles. Given a specification of the data format, Weld can use knowledge of the query it will run to generate faster parsing code for the input data (and perhaps even offload it to other kinds of hardware). As another example, a runtime like Weld can be extended to generate code which handles tasks generally attributed to the *operating system*, such as sending and reading network messages directly to the network card (using zero-copy IO). Apart from avoiding overheads such as context switching and buffer copies between the kernel and user space, Weld can use information about the *full query* when generating code to further optimize these tasks. For example, in a distributed setting Weld can use UDP-based datagrams instead of ordered TCP messages when performing a reduce operation since reordered traffic does not affect correctness. it can also *generate* optimized code for the networking layer since Weld will know which protocols will need to be implemented.

In all, achieving bare-metal performance across analytics queries from a diverse range of domains requires a system which can generate a fast, parallel, end-to-end "data plane" for these queries which handles *every* aspect of executing the query – from loading data, to parsing, to processing to marshaling and sending data over the network. Simultaneously, the "control plane" for such as system must provide the typical data scientist the same set of abstractions in place today – disjoint libraries written in expressive, high level programming languages which perform domain specific optimizations. For bare-metal performance across a workflow, these libraries should then invoke a runtime like Weld and do little more.

Impact. The impact of such a data plane is twofold. First, library developers can focus on implementing smarter algorithms without having to spend time optimizing their code for a given hardware platform; if a backend for a platform exists, all libraries using Weld will benefit. Second, programmers will continue to leverage libraries from many domains to build their applications; to achieve the best performance on the available hardware, the interface for building data intensive libraries must change to offset data movements costs, eliminate overheads imposed by existing abstractions, and to leverage parallelism effectively.