Towards Fine-Grained Dataflow Parallelism in Big Data Systems

Sebastian Ertel(✉), Justus Adam, and Jeronimo Castrillon

Technische Universität Dresden, Dresden, Germany
{Sebastian.Ertel,Justus.Adam,Jeronimo.Castrillon}@tu-dresden.de

1 Introduction

Over the last decade big data analytics became the major source of new insights in science and industry. Applications include the identification of mutations in cancer genome and the tracking of other vehicles around an autonomously driving car. The big data systems (BDSs) that enable such analyses have to be able to process massive amounts of data as fast as possible. To do so, current BDSs apply coarse-grained data parallelism, i.e., they execute the same code on each core of the nodes in a cluster on a different chunk of the data. Such an application is said to scale with the number of cores in the cluster. However, not every aspect of a big data application exposes data parallelism. For these aspects, current BDSs fail to scale.

2 Scalability Issues of Big Data Systems

A typical big data analysis program assembles a set of predefined operations and applies them to the data in multiple phases. For example, the famous MapReduce programming model defines exactly two phases: a map and a reduce phase [1]. The map phase is data parallel by definition but the data parallelism of the reduce phase depends on the application. For example, in data analytics queries, the join operation for two tables can not be performed in a data parallel way (when the input data is not partitioned). In such a case, a single node receives all results from the map phase and becomes the throughput bottleneck.

BDSs have been traditionally designed to execute applications in a massive coarse-grained data parallel fashion across a cluster of machines. The underlying assumption was that applications would process large amounts of simply structured data, such as text. The effort to serialize and deserialize such data structures, i.e., transforming them to bytes (and its dual operation on the receiver side), is negligible. This setup led to the common belief that network I/O, instead of computation, is the performance bottleneck in these systems.

Only recently, researchers have shown that I/O is not always the limiting factor for performance [3]. Authors in [4] benchmarked the current state-of-the-art BDSs Apache Spark and Apache Flink in a high bandwidth cluster setup. They show that reduce operations do not profit from modern multi-core architectures

L. Rauchwerger (Ed.): LCPC 2017, LNCS 11403, pp. 281–282, 2019.
https://doi.org/10.1007/978-3-030-35225-7

since their cores do not take advantage of fine-grained parallelism. As a result, the data throughput does not increase for faster network devices, i.e., it does not scale with the network.

To better exploit new hardware, the design of BDSs must be revisited [4]. Redesign is non trivial due to the complexity of the code bases of state-of-the-art BDSs, e.g., with over 1.4 million lines of code in Hadoop MapReduce (HMR). Approaching this task with common parallel programming means, like threads, tasks or actors and their respective synchronization via locks, futures or mailboxes, inevitably increases code complexity. As a result, these systems become harder to reason about, maintain and extend. We believe that this redesign can be better achieved with new programming abstractions together with associated compilers and runtimes to help automatically optimize the code depending on the application characteristics. This paper represents first steps in this direction.

3 Implicit Dataflow Programming

In our work, we investigate rewrites for the processing cores of current big data systems to increase data throughput, effectively improving scalability with new hardware. Our rewrites use the implicit parallel programming language, Ohua [2], to provide concise code that is easy to maintain. The corresponding compiler transforms the program into a dataflow graph that the runtime system executes in a pipeline and task parallel fashion across the cores of a single machine. To verify the claim that all BDSs face the above scalability issues, we analyzed the code base of HMR, Spark and Flink. We found that all three systems use the same design patterns to build their data processing pipelines and use them as an indicator for code that can execute in parallel. The rewrite of the data processing cores of HMR with Ohua resulted in concise code that is free of concurrency abstractions and reuses existing code to a large extend. Figure 1 presents first performance results with speed-ups of up to 3.5× for compute-intensive configurations.

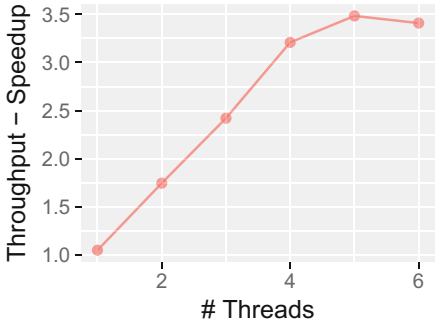

Fig. 1. Map task execution of a black list filter on TPC-H data.

References

1. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004. USENIX Association (2004)
2. Ertel, S., Fetzer, C., Felber, P.: Ohua: implicit dataflow programming for concurrent systems. In: PPPJ 2015. ACM (2015)
3. Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., Chun, B.G.: Making sense of performance in data analytics frameworks. In: NSDI 2015. USENIX Association (2015)
4. Trivedi, A., et al.: On the [ir]relevance of network performance for data processing. In: HotCloud 2016. USENIX Association (2016)