Mira: Sharing Resources for Distributed Analytics at Small Timescales

Presentation of work originally published in the Proc. of 2018 IEEE International Conference on Big Data, Seattle, USA

Michael Kaufmann\textsuperscript{1,2}, Kornilios Kourtis\textsuperscript{1}, Adrian Schuepbach\textsuperscript{1}, Martina Zitterbart\textsuperscript{2}

Abstract: Mira is a system for optimized elastic execution of short-running and interactive data-analytics applications with low-latency execution startup, fast resource management and efficient resource utilization on shared clusters. We highlight the key insights and the Mira approach and summarize the most important results.

Keywords: Data Science; Distributed Analytics; Elastic Computing

1 Motivation

Modern distributed analytics stacks consist of application frameworks that enable processing of large amounts of data, and a resource manager that allows applications to share computational resources. These systems were designed to run batch jobs with long lifetimes (e.g., a few hours). New use cases, such as interactive applications, have emerged, which requires operating at smaller timescales (seconds) to share resources efficiently. Small timescales pose a significant challenge for existing systems. To illustrate this, we run a simple Spark application on a YARN-managed cluster. The application spawns 110 tasks, each of which sleeps for 10 seconds. Fig. 1a shows the application’s demands (blue line) and the resources that were allocated to it (red line). There is a significant delay in the application acquiring the needed resources as well as in releasing them after it is done. For interactive applications, those costs dominate – or even exceed – the application runtime. Mira [Ka18] addresses

\textsuperscript{1} IBM Research Zurich, Rueschlikon, Switzerland \{kau,kou,dri\}@zurich.ibm.com
\textsuperscript{2} Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany zitterbart@kit.edu

doi:10.18420/inf2019_36
this issue and enables resource management over small timescales (sub-seconds). Fig. 1b shows that the same application on Mira+Spark executes in significantly less time.

2 Mira

Mira includes two parts: a resource manager (RM) and an application scheduler (AS). The former manages all resources and applications, while the later schedules the tasks of a single application and communicates with the RM. Mira integrates with an existing application framework (AF) (for example Spark) and the AF’s execution environment (EX). Mira achieves efficiency in two ways. First, it treats EX not as ephemeral, but as long-lived, shared resources. This allows it to minimize recurring acquisition costs and benefit from warmed-up executors (e.g., JIT, caches). Second, as consequence of the minimized resource acquisition cost, Mira is able to acquire and release resources almost instantaneously.

To evaluate Mira, we execute two applications concurrently. A background (BG) application, with an infinite loop of stages with 8192 1s-tasks per stage, generates a constant load on the cluster to force the RM to actively balance resource among applications. A foreground (FG) application runs TPC-DS queries. We compare to YARN as baseline. Overall, Mira reduces application runtime by up to $4.2 \times$ and $\approx 2.4 \times$ on average. Fig. 2 compares the resource allocation to the BG (green) and FG (blue) applications executed on YARN vs. Mira. Red areas represent executors not assigned to any application by the RM. Due to Mira’s low executor assignment latency and shorter task runtime (size of blue areas), the FG on Mira is able to execute the same number of queries in 148s instead of 267s. Mira has virtually no unassigned executors (red spikes in Fig. 2b) during the resource-reassignment period. Finally, Spark+YARN suffers from the high cost of resource reacquisition. For query 4, Spark releases executors due to a short dip in task load (at $\approx 165$s) just to reacquire them a few seconds later with a multi-second delay.

![Fig. 2: Resource sharing between BG and FG applications (top) and task load of the FG (bottom). FG application submissions are indicated by labelled vertical black lines.](image)

**Bibliography**