QaaD(Query-as-a-Data):Scalable Execution of Massive Number of Small Queries in Spark

Abstract

Spark big data processing platform is heavily used in today's IT services for various critical applications such as machine learning tasks for service recommendations or massive volumes of raw sales data analysis. Spark is designed to deliver high performance by enabling a high degree of parallelism while processing various heavy-weight queries that require homogeneous operations on large data. However, it has been observed that workloads made of small and short-running queries coming from various sources are becoming dominant in practice. Unfortunately, the current Spark architecture is unfit to process workloads made of a large number of small queries optimally due to excessive I/Os with small computations. We present a technique, called QaaD, that addresses this problem fundamentally by applying i) transparent conversion of workloads made of small queries into one with large queries and ii) dynamic partition size adjustment for runtime overhead minimization. For this, we introduce a new abstraction, microRDD, to support our design of query merging, the embedding of queries as part of data, and an opportunistic sharing of common input data among queries. Comprehensive evaluation using real-world data shows that QaaD is able to deliver 10.6× to 36.6× speed-up against standard Spark executions for small query workloads.

Keywords: Spark, small query.

1 Introduction

Current heavy reliance on big data processing platforms such as Spark is in part because fulfilling the demands of data analytics applications has become difficult for traditional DBMSs or OLTP systems. Big data analytics platforms were initially designed and optimized for large-size batch jobs. However, it has been noticed that unintended type of workloads categorized as short jobs has emerged and continuously increased their proportion. These queries tend to be light in computation, but are issued in a massive number. Unfortunately, naïve, individual handling of a large number of small queries on Spark results in severe underutilization of its capability. This is because workloads of predominantly small queries do not conform to the expected execution model. Small queries possess an insufficient degree of data parallelism for Spark to take advantage of since the data size may be too small to require enough amount of computation. From the systems viewpoint, the cost of setting up software components, such as loading data to memory through I/O and preparing Spark components for executing just one small query, is far too high compared to the actual computation done for the query execution. The ratio of

setup time to compute time is too large. Consequently, processing a small query is dominated by heavy I/O activities in the overall turn-around time, within which the actual computation is minuscule. A sensible solution should reinstate this setup-to-compute time ratio closer to the ideal value.However, we hypothesize that the most effective fix to the problem is to 'transform the workload' to conform to what Spark was designed. we propose a technique that aims to fundamentally mitigate the problem of excessive I/O overheads coming from repeated setup of Spark sessions in small querydominant workloads which brings it closer toward Spark's ideal execution model.

2 Related works

This paper makes the following contributions in this work. First, it proposes a novel technique that drastically improves the throughput of the massive number of small queries in Spark. It is achieved via new microRDD abstraction that enables users to submit queries as usual, but transparently performs optimal merge and execution plan construction of queries to improve the setup-to-compute time ratio. Second, this paper provides the fully functional prototype implementation of the proposed system, called QaaD, that runs on top of Spark. Third, this paper demonstrates the efficacy of our approach by conducting comprehensive evaluations using real-world workloads.

2.1 Resilient Distributed Datasets (RDD)

It is a main abstraction of data in Spark. Data in an RDD is considered immutable and partitioned across multiple nodes in a Spark cluster. Thus, it enables programmers to perform parallel operations on a large cluster and provides fault-tolerance. Since it is immutable, applying operations, called transformations, on RDDs results in new RDDs. The examples of transformations include map, filter, and join. Programmers define RDDs via transformations on data stored in the distributed storage first. Actual data processing happens when actions are applied on RDDs. Actions are operations returning a result value to an application or a storage system. The examples of actions include collect that returns all records in an RDD and count that returns the number of records in an RDD.

2.2 Partition and Task

A Spark partition is an atomic piece of the dataset stored in one of the nodes in a Spark cluster. The logical RDDs are broken down into a set of partitions to enable parallel processing on nodes. A task is the individual unit of execution corresponding to a partition. When a parallel operation is invoked on an RDD, Spark creates a task to process the corresponding partition of the RDD. The driver node serializes these tasks and sends them to worker nodes. Each worker node deserializes the received tasks. Then, each task starts to process its partition.

2.3 Short Jobs and Small Queries

Although the notion of small queries (or short jobs) has been introduced in several papers, there is still no formally agreed definition to date. Elmeleegy et al. [1] referred to them as 'short jobs' and

discussed them extensively in their work. It was roughly defined in terms of time as a job that finished within 10 minutes. In this work, authors define the 'small query' in terms of data sizes needed by a query rather than the query turnaround time. It is defined as the query whose input data can fit into a single partition of the size specified by spark.sql.files.maxPartitionBytes (currently 128 MB by default) in the Spark configuration during the data loading time. As the proportion of large queries increases in the workload, the execution by QaaD becomes similar to normal Spark runs.

3 Method

3.1 Overview

This system design consists of five components: Query Handler, microDAG Builder, Scheduler, Adaptive Partitioner and Result Handler to process multiple small queries (Figure 1). It is centered around two key techniques operating at (i) planning phase and (ii) the execution phase. For the planning phase, Query Handler, microDAG Builder and Scheduler of QaaD work together to generate a driver program that encloses our QaaD technique. Query Handler first accepts query codes. Combining submitted queries written in our custom APIs (called microRDD APIs) is done at microDAG Builder. Scheduler is responsible for producing a driver program that defines the transformations and actions on RDDs, where each RDD logically contains data for multiple small queries. The driver program starts with an input RDD containing accumulated data referred by a given small query set. For the execution phase, Adaptive Partitioner of QaaD dynamically configures internal partitions in a way that minimizes the overhead of shuffles, the most performance-critical operation in the Spark query processing. The followings are the overview of the techniques employed in these two phases.

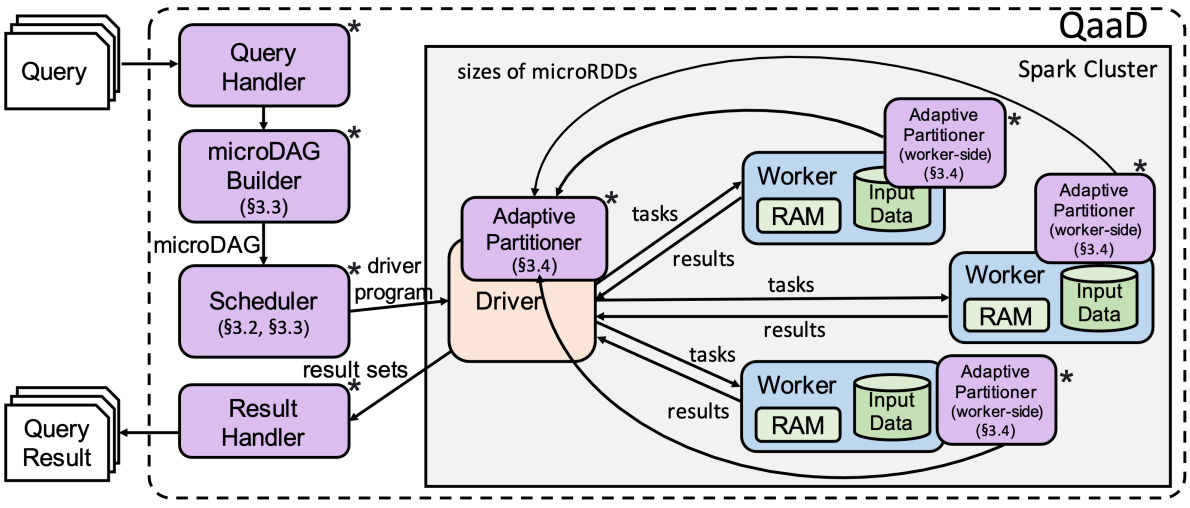

Figure 1. The architecture of QaaD. Our components are marked with asterisks(*).

3.2 MicroRDD APIs and RDD Transformations

All the microRDD APIs are purposely designed to resemble the programming interfaces of the standard RDD. The author follows the same transformation operator names as used in standard RDD

APIs, prefixed with 'q' to the RDD API counterpart. In addition, currently supported action functions are qCollect and qCount. Since it maintain records for multiple microRDDs in an RDD, the paper embed the query information (i.e., microRDD ID) into data (i.e., records), denoted as (,). A set of microRDD IDs serves as a query ID since they are created for each submitted query at the beginning. QaaD can accommodate heterogeneous data types from multiple sources. All records are made to have the same generic type, (Int, Any), where Int is the microRDD ID and Any the record.

3.3 MicroDAG: DAG of microRDDs

As standard Spark query forms a DAG (Directed Acyclic Graph) of RDD dependencies, the paper also maintain a DAG of microRDDs, called microDAG, that represents dependencies amongst microRDDs to keep track of series of transformations on them. This microDAG contains microRDDs as vertices and dependencies as edges between parent and child microRDDs. A microDAG is created by microDAG.

4 Implementation details

4.1 Comparing with the released source codes

Actually, the paper's author provides all the source codes in Github website, Source code for the project is available at https://github.com/postechdblab/qaad. Because the paper use two real-world datasets with 200k records of orders collected and transactions for auction details on websites, except for the huge datasets, the experiment environment is so complicated that it can not be established. Therefore, I build a easy-deploy environment to enhance the deploy situation where we can run the QaaD project by distributed parallel processing on the large-sized datasets. We can see the significant performance improvement of the Spark on workload made of a large number of small queries, also verify of an order of magnitude improved performance on small query workloads through comprehensive evaluations.

4.2 Experimental environment setup

This paper provide information about the experimental settings used in the evaluation. This includes the hardware specification, software environment, dataset properties, queries, and compared techniques.

4.2.1 Hardware and Software Settings

They set up a Spark cluster made of five physical machines – one driver and four worker machines. Each machine has Intel Xeon E5-2450 CPUs with 16 cores and 192 GB of RAM. All machines are equipped with 1TB of HDDs and interconnected by InfiniBand QDR 4x network switch. On the software side, we installed Spark 3.0 and Hadoop 2.7.7 on Ubuntu 20.04.3 LTS. We configured Spark to have four executors. Each executor used 14 cores and 128 GB RAM to run Spark applications. For QaaD, we set minPartitionSize and maxPartitionSize to 100 MB and 200 MB, respectively.

4.2.2 Compared Techniques

QaaD technique is compared against two other techniques SparkS and SparkU.

- SparkS: This represents a standard way of using Spark where all queries are submitted and processed individually and independently. Depending on the resource manager, submitted queries may be assigned to available executor nodes and run in parallel with other outstanding queries.
- SparkU: This technique attempts to remedy Spark's shortcomings with a large number of small query workloads using a simple fix. The key idea is to combine small queries with a UNION operator to transform them into one large Spark query. This unionization is expected to create opportunities for more parallel execution of multiple stages since many queries would have become part of one large query execution plan. It would produce a positive effect of reducing the number of set-ups. However, the time spent waiting for a sufficient number of queries to accumulate may impact their average response time.

4.2.3 Datasets

Using two real-world datasets and 17 synthetic datasets in our evaluations.

- BRA (Brazilian E-commerce public dataset): A dataset with 100K records of orders collected between 2016 and 2018 on a Brazilian online marketplace, known as Olist store [2]. It contains 1.5M records and 50 attributes in the dataset distributed over a total of eight tables.
- eBay (eBay dataset): Transactions for auction details on eBay [3]. It has a single table and contains auction information consisting of 10K records with nine attributes. Both BRA and eBay contain information about orders with sellers, customers, products, and reviews collected from real marketplaces.

4.3 Main contributions

I try to build the experimental environment through one-click operation, which can be used as an open-source platform for operators to manage the entire experimental lifecycle through experimentation, deployment, and testing. It comes in handy when we want to track the performance of the models. It's like a dashboard, for the future, I will develope it to one place where we can:

- monitor Qaad pipelines,
- store model metadata, and
- pick the best-performing model.

This method is our best friend if we want to organize our data pipeline and make QaaD project development much more efficient. We won't have to waste time on code rewrites and will have more opportunities for focusing on robust pipelines. Moreover, helps teams establish collaboration standards to limit delays and build scalable, deployable projects.

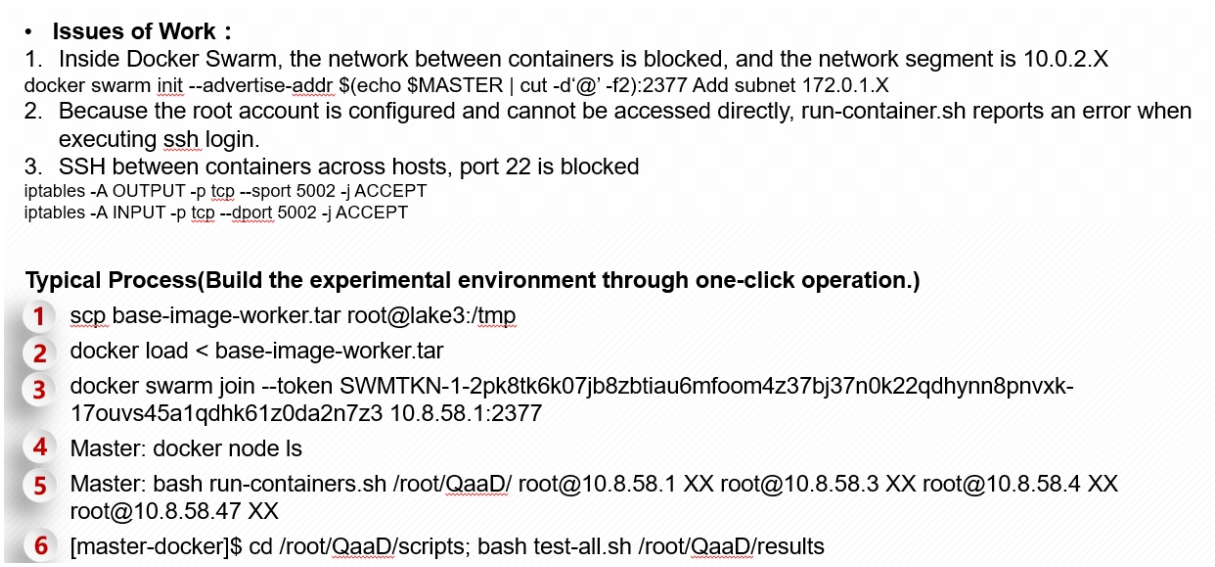

Figure 2. My own work

5 Results and analysis

Figures 3 show the elapsed times of different numbers of queries on BRA and eBay datasets. For both datasets, QaaD consistently outperformed SparkS and SparkU. We see clear trends of widening performance gap between QaaD and two compared techniques as the query sizes scale up. At the highest workload, we obtained 10.6× and 21.6× speed-ups against SparkS for BRA and eBay datasets, respectively.

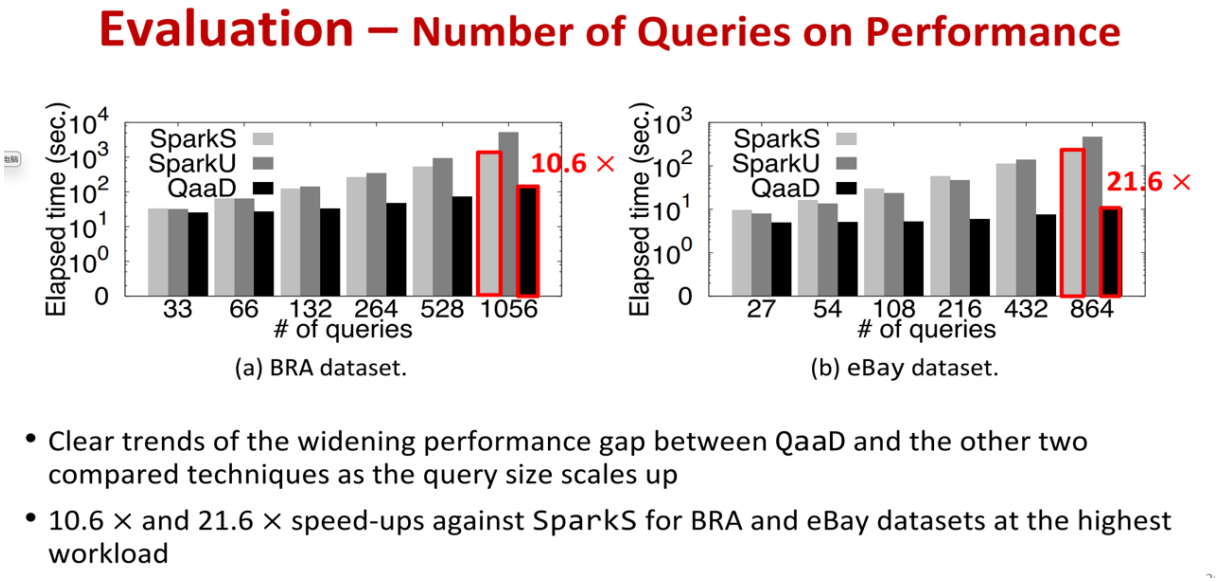

Figure 3. Experimental results

6 Conclusion and future work

This paper presented QaaD, a technique designed to deliver a significant performance improvement to Spark on workloads made of a large number of small queries. The authors conjectured that the current performance level on such workloads was far below the optimal point mainly because of excessive I/O overheads from the set-up of individual queries. The key principle of QaaD is, thus, to transform the small query workloads into large query workloads that match well with the original Spark's execution model. The paper introduced a novel microRDD abstraction for query merging and implemented the APIs. In addition, it be designed and implemented a dynamic partition adjustment scheme to minimize the runtime overheads further. Through comprehensive evaluations, we verified that our approach could deliver an order of magnitude improved performance on small query workloads compared to standard Spark.

References

[1] Khaled Elmeleegy. Piranha: Optimizing short jobs in hadoop. VLDB Endow, 32(11):985–996, 2013.

[2] Online. Brazilian e-commerce public dataset by olist. https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce.

[3] Online. Brazilian e-commerce public dataset by olist. https://www.kaggle.com/datasets/onlineauctions/online-auctions-dataset.