Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

A Methodology for Spark Parameter Tuning ☆

Anastasios Gounaris [a,*], Jordi Torres [b]

[a] Department of Informatics, Aristotle University of Thessaloniki, Greece
[b] Department of Computer Architecture, Technical University of Catalonia, Spain

ARTICLE INFO

Article history:
Received 16 January 2017
Received in revised form 24 April 2017
Accepted 4 May 2017
Available online xxxx

Keywords:
Spark configuration
Parameter tuning
Shuffling

ABSTRACT

Spark has been established as an attractive platform for big data analysis, since it manages to hide most of the complexities related to parallelism, fault tolerance and cluster setting from developers. However, this comes at the expense of having over 150 configurable parameters, the impact of which cannot be exhaustively examined due to the exponential amount of their combinations. The default values allow developers to quickly deploy their applications but leave the question as to whether performance can be improved open. In this work, we investigate the impact of the most important tunable Spark parameters with regards to shuffling, compression and serialization on the application performance through extensive experimentation using the Spark-enabled Marenostrum III (MN3) computing infrastructure of the Barcelona Supercomputing Center. The overarching aim is to guide developers on how to proceed to changes to the default values. We build upon our previous work, where we mapped our experience to a trial-and-error iterative improvement methodology for tuning parameters in arbitrary applications based on evidence from a very small number of experimental runs. The main contribution of this work is that we propose an alternative systematic methodology for parameter tuning, which can be easily applied onto any computing infrastructure and is shown to yield comparable if not better results than the initial one when applied to MN3; observed speedups in our validating test case studies start from 20%. In addition, the new methodology can rely on runs using samples instead of runs on the complete datasets, which render it significantly more practical.

1. Introduction

Spark [1,2] has emerged as one of the most widely used frameworks for massively parallel data analytics. In summary, it improves upon Hadoop MapReduce in terms of flexibility in the programming model and performance [3], especially for iterative applications. It can accommodate both batch and streaming applications, while providing interfaces to other established big data technologies, especially regarding storage, such as HDFS and NoSQL databases. Finally, it includes components for SQL-like processing, graph processing, machine learning and data mining. However, its key feature is that it manages to hide the complexities related to parallelism, fault-tolerance and cluster setting from end users and application developers. This feature renders Spark practical for use in real-life data science and big data processing applications.

To support all these, Spark execution engine has been evolved to an efficient albeit complex system with more than 150 configurable parameters. The default values are usually sufficient for a Spark program to run, e.g., not to run out of memory without having the option to spill data on the disk and thus crash. But this gives rise to the following research question: *"Can the default configuration be improved and, if yes, how better configurations can be set efficiently?"*

The aim of this work is firstly, to provide evidence that the answer to the first part of the above question is affirmative, and then, to answer the second part in an efficient manner. Clearly, it is practically impossible to check all the different combinations of parameter values for all tunable parameters. Therefore, tuning arbitrary Spark applications by inexpensively navigating through the vast search space of all possible configurations in a principled manner is a challenging task. Very few research endeavors focus on issues related to understanding the performance of Spark applications and the role of tunable parameters [4–6]. For the latter, Spark's official configuration guides[1] and tuning[2] guides and tutorial book [7] provide a valuable asset in understanding the role of every single parameter.

☆ This article belongs to Big Data & Neural Network.

* Corresponding author.
E-mail addresses: gounaria@csd.auth.gr (A. Gounaris), torres@ac.upc.edu (J. Torres).

[1] http://spark.apache.org/docs/latest/configuration.html.
[2] http://spark.apache.org/docs/latest/tuning.html.

http://dx.doi.org/10.1016/j.bdr.2017.05.001
2214-5796/

Understanding the role of a parameter does not necessarily mean that the impact of each parameter on the performance of arbitrary applications is understood as well. Moreover, such an understanding does not imply that tuning is straightforward. An added complexity stems from the fact that most parameters are correlated and the impact of parameters may vary from application to application and it will also vary from cluster to cluster. In this work, we experiment with the MareNostrum III (MN3) petascale supercomputer at the Barcelona Supercomputing Center. After configuring the cluster in an application-independent way according to the results in [6], we examine the impact of configurable parameters with regards to shuffling, compression and serialization on a range of applications with a view to deriving a simple yet systematic tuning methodology that can be applied to each application separately.

We build upon our previous work in [8], where: (i) We identified the most important parameters with regards to shuffling, compression and serialization in terms of their potential impact on performance and we tested them on MN3. The number of these parameters is 12. (ii) We summarized our experience in a tuning methodology to be applied on an individual application basis. The methodology treats applications as black boxes, follows an efficient trial-and-error approach that involves a low number of experimental runs for just 10 different configurations at most, and leverages the correlation between different parameters.

This article is a heavily extended version of the results in [8] (also repeating all experiments from scratch). The new contributions are summarized as follows:

- We provide new experimental evidence that the default Spark configuration leaves room for performance improvements when tuning the parameters under investigation. We also evaluate how these parameters are correlated.
- We propose a new tuning methodology that can be applied to any computing infrastructure; the new methodology serves as an alternative to the one in [8], which mostly reflected our experience with MN3, did not profile parameter correlations explicitly and is questionable whether it generalizes efficiently.
- We validate and compare the performance impacts of the new methodology and the one in [8]. Our new proposal can yield comparable if not better results than the initial one when applied to MN3. The observed speedups in our validating test case studies start from 20% and reach up to more than 4 times, when compared against the default MN3 configuration.
- With a view to rendering the proposal more practical, we provide evidence that the new methodology can rely on runs using samples instead of runs on the complete datasets.

The remainder of this work is structured as follows. The next section provides an overview of Spark and of the known results to date with regards to Spark tuning. In Section 3, we explain the chosen parameters and we present the methodology in [8]. Our new methodology along with its instantiation on MN3 is presented in Section 4. Section 5 deals with the evaluation of the methodologies. We conclude in Section 6.

2. Overview of existing results for Spark configuration

Apache Spark is an open source massively parallel computing framework. It provides an interface enabling users to develop and deploy applications to run in parallel on clusters of machines, which typically adopt the shared-nothing parallel architecture [9]. For cluster management, the list of supported options includes deploying on an Amazon EC2 cloud cluster instantiated on the fly, employing a third-part manager, such as YARN or MESOS, or launching a standalone cluster. Overall, there are over 150 tunable parameters that define execution details.

2.1. Spark basics

Spark operates on data collections abstracted as Resilient Distributed Datasets (RDDs), which are partitioned across several nodes. A Spark application consists of two types of operations, namely *transformations* and *actions*. The former apply a function on each RDD element and result in a new RDD. Actions trigger the execution of such functions and produce meaningful results. For each action in the application a *job* is performed. For each job, typically several RDD transformations need to be computed. Spark's scheduler creates a physical execution plan for the job based on the directed acyclic graph (DAG) of transformations. The physical plan is divided into *stages*. A *stage* is a sequence of transformations that can be pipelined. The sequence of computations defined by a *stage* instantiated over a single data partition is called a *task*. A *task* is the actual unit of execution of the physical plan. The task scheduler assigns tasks to parallel *workers* via the cluster manager. On each worker node, each applications launches its own *executors*, which are responsible for task execution.

Data may be *repartitioned* across RDDs. This may be done as a result of a specific transformation, such as *groupByKey* and *sortByKey*, which result in a new RDD, where data are partitioned across machines differently. This data re-distribution is commonly referred to as data *shuffling*. Data shuffling is an expensive operation. First it incurs communication cost. Second, it incurs CPU cost, because it involves data serialization. Third, it may incur I/O cost, because it may store temporary data on disk if they cannot fit in main memory; temporary files are kept as long as they are needed for fault tolerance purposes. As such, the cost is multi-dimensional, while memory is stressed as well.

Table 1 provides a categorization of Spark parameters. In this work, we target parameters belonging to the *Shuffle Behavior* and *Compression and Serialization* aspects, which greatly contribute to a Spark application's running time, as supported by our experimental results, the official documentation, and the evidence provided in other works, such as [4,5,3]. Note that there are several other parameters belonging to categories such as *Application Properties, Execution Behavior* and *Networking* that may affect the performance, but these parameters are typically set at the cluster level, i.e., they are common to all applications running on the same cluster of machines, e.g., as shown in [6].

Next, we summarize the known results to date with regards to Spark configuration. These results come from three types of sources: (i) academic works that aimed at Spark configuration and profiling; (ii) official Spark documentation and guides that build on top of this documentation; and (iii) academic works that include evaluation of Spark applications on real platforms and, as a by-product, provide information about the configuration that yielded the highest performance. We also briefly discuss results on Spark profiling, because they directly relate to Spark configuration. The most relevant work to ours is the study of Spark performance on the MN3 in [6], which is complementary to this work and presented separately.

2.2. Optimization of Spark on MN3

The work in [6] sheds lights onto the impact of configurations related to parallelism. In MN3, cluster management is performed according to the standalone mode, i.e., YARN and MESOS are not used. The main results, which are reused in our work, are summarized as follows. First, the number of cores allocated to each Spark executor has a big impact on performance and should be configured in an application-independent manner. In other words,