Data Model Optimization for Reducing Computational Cost at Apache Spark

Rohyoung Myung, Han-Yee Kim, Sukyong Choi, Taeweon Suh, Heonchang Yu
Department of Computer Science and Engineering, Korea University

1. Abstract

- As the performance of distributed parallel processing on big data is considered as a main concern, Apache Spark the most prevalent open source based distributed processing engine has triggering much interest for performance optimization.
- According to the user's purpose, Apache Spark provides diverse scope of system customization via system parameters.
- In this paper, we propose a data model that reduces the execution cost of parallel processing model on Apache Spark.
- we verify our scheme by adapting the proposed model on general big data application and prove the validity of it by scaling up its input data size and the number of executors.

3. Background

- Spark has a specific architecture for distributed parallel processing which is shown at Fig. 1.

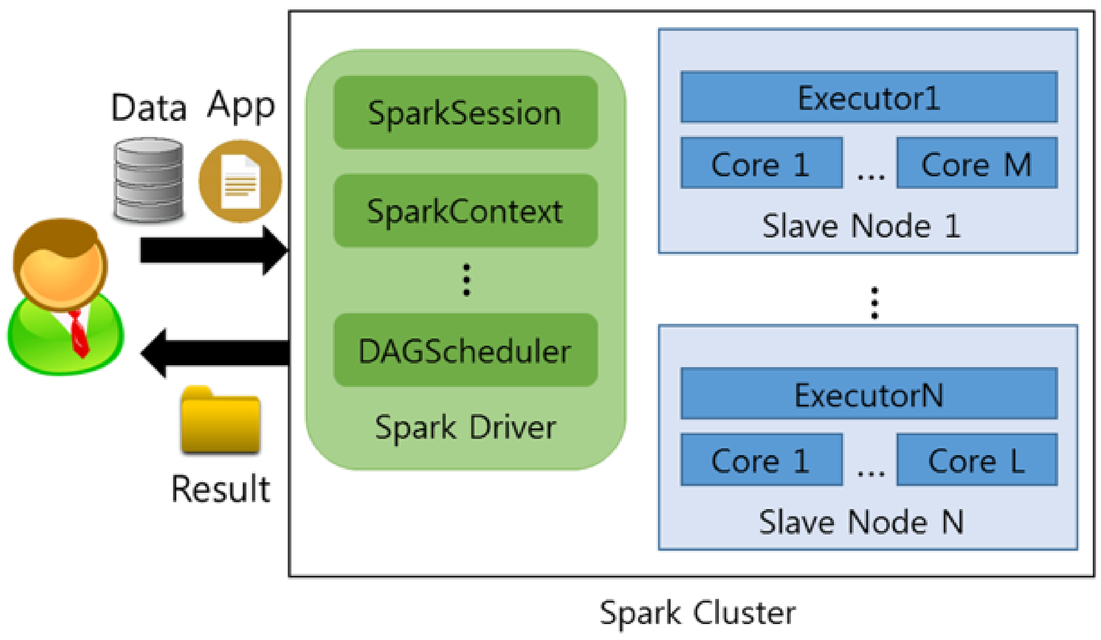

Figure 1: Spark cluster system architecture and interaction between user and Spark cluster

- For executing the application, Spark Driver transform submitted data to RDD and express executing procedure of the RDD into Directed Acyclic Graph(DAG).
- A RDD consists of at least one partition and a partition consists of at least one element which is depicted at Fig. 2.
- There are two relationships among RDDs: wide dependency and narrow dependency according to relationship between parent and child partitions. Partitions which have narrow dependency can be pipelined as each of them doesn't have any dependency with one another.

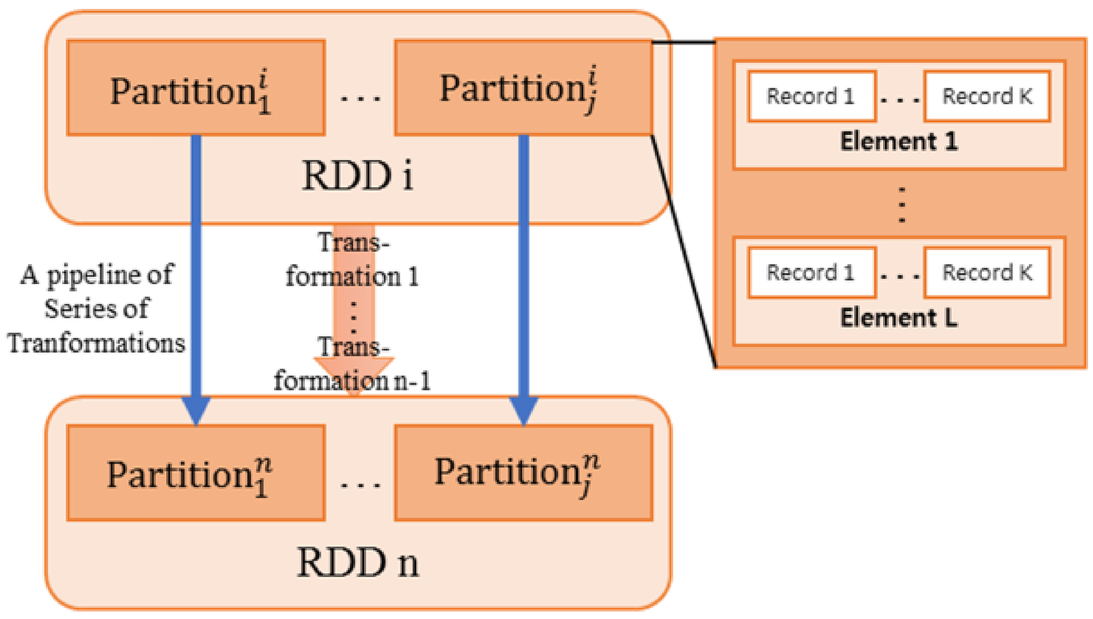

Figure 2: Pipelining procedure and inner structure of RDD

4. Data Submission Model

- When executors processing allocated tasks, each task is defined as transformation or action. The transformations or actions which are own API of Spark[7] are divided into two types.
- The first type such as 'groupByKey' and 'HashPartitioning' which has predefined functions execute task manually
- The second type such as 'map' and 'flatMap' has inner functions which should be defined by Spark user. For example, when 'map' whose inner function is 'split' is processed, whole sequence of executing single RDD which is processed at each core is described at Fig. 3.
- When an executor process single partition which is known as 'task' at Spark, total execution time is divided into two parts.

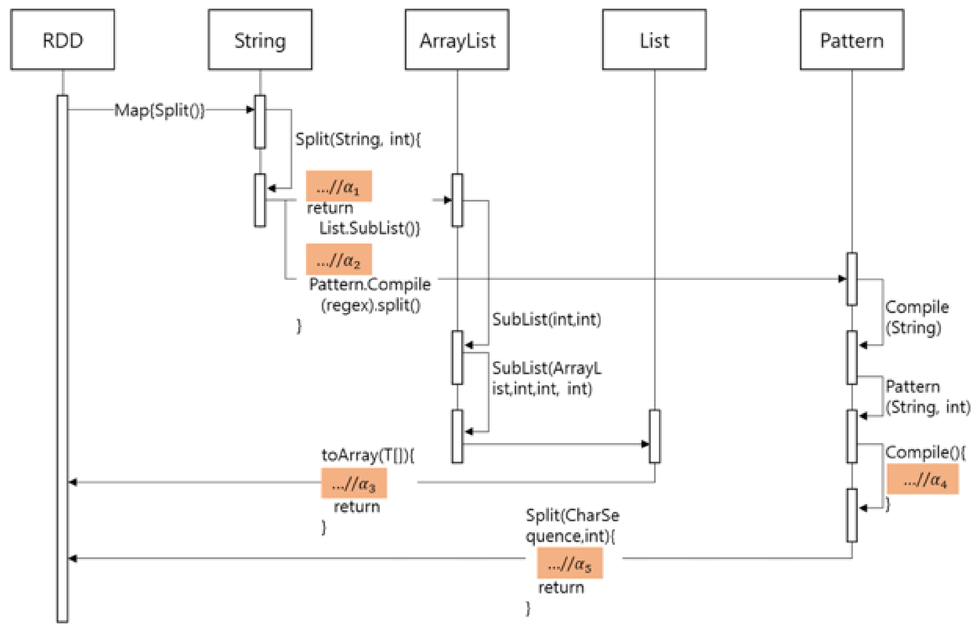

Figure 3: Sequence diagram when Spark API 'map' and its inner function 'split' is processed. The shaded parts(α_1-α_5) are the additional cost for each element.

- The first part includes the cost that when an executor traverse whole records of single partition and apply its inner functions into each records iteratively
- The second part is additional cost that not correlated to the number of records but correlated to the number of elements.
- The definitions of all notations required for calculating the two costs are described at Table 1.

Table 1: Definition of the notations for computational cost model

Notation	*Meaning*
e	Total number of elements in a partition
r	The number of records in an element
l	The iterative overhead for executing single record
α	The additional overhead for executing single element
K	The number of records per element
C	Original total cost for executing single task
C'	Optimized total cost for executing single task

- Equation 1. models the summation of the first and the second part. When the number of records in an element is multiplied by K total cost can be described as Equation 2.

$$C = r * l + e * \alpha \quad (1) \qquad C' = r * l + \frac{1}{K} * e * \alpha \quad (2)$$

- Notice that when K is increased, task size for executing an element is also increased which causes additional 'garbage collection overhead' which is proportional to K.

5. Experimental result

- Fig. 4. describes that when applying our model with proper value 'K', in our system: 100, the execution time of the application is reduced at most 3.5x more than the vanilla data model.
- Fig. 5. describes that when the number of executors are increased linearly, the execution time of the application is decreased linearly. It also describes that our model reduces the execution time at least 2x.

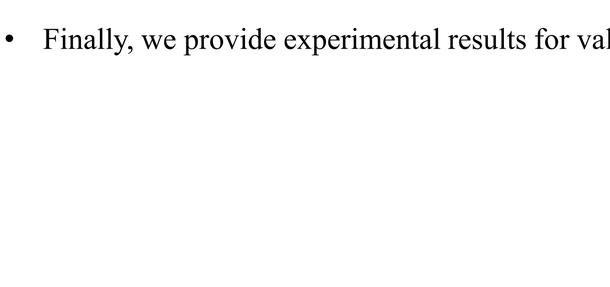

Figure 4: Execution time(s) of 'WordCount' when data size is linearly increased(1-5GB) with single executor and 'K': 1(vanilla model), 100(our scheme applied).

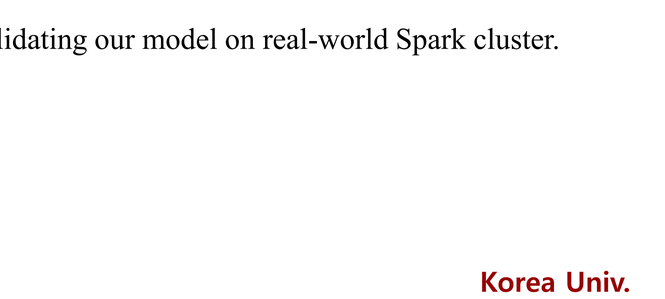

Figure 5: Execution time(s) of 'WordCount' when the number of executors is linearly increased(1-3) with data size:5GB and 'K': 1(vanilla model), 100(our scheme applied).

6. Conclusion

- Our research analyzes structure details of Spark, its distributed data structure, and its details of distributed processing procedure.
- We demonstrate how our data model reduces execution cost by deterministically modeling the cost of processing multiple series of data in parallel.
- Finally, we provide experimental results for validating our model on real-world Spark cluster.