A Fine-Grained Stateful Data Analytics Method Based on Resilient State Table

Jike Ge, Chongqing University of Science and Technology, Chongqing, China
Wenbo He, McMaster University, Toronto, Canada
Zuqin Chen, Chongqing University of Science and Technology, Chongqing, China
Can Liu, Chongqing University of Science and Technology, Chongqing, China
Jun Peng, Chongqing University of Science and Technology, Chongqing, China
Guorong Chen, Chongqing University of Science and Technology, Chongqing, China

ABSTRACT

This article describes how stateful data analytic frameworks have emerged to provide fresh and low-latency results for big data processing. At present, it is desired to achieve the fine-grained data model in Spark data processing framework. However, Spark adopts coarse-grained data model in order to facilitate parallelization, it is challenging in dealing with the fine-grained data access in stateful data analytics. In this paper, the authors introduce a fine-grained stateful data component, Resilient State Table (RST), to Spark framework. For filling the gap between the coarse-grained data model in Spark and the fine-grained data access requirements in stateful data analytics, they devise the programming model of RST which interacts with Spark’s coarse-grained memory representation seamlessly, and enable users to query/update the state entries in fine granularity with Spark-like programming interfaces. Performance evaluation experiments in various application fields demonstrate that their proposed solution achieves the improvements in latency, fault-tolerance, as well as scalability.

KEYWORDS

Big Data, Resilient Distributed Dataset, Resilient State Table, Spark, Stateful Data Analytics

1. INTRODUCTION

The past few years, from social networks to E-commerce and the Web application, have seen tremendous interests in big data analytics (Wang et al., 2015; Xu et al., 2015; Srinivasa & Bhatnagar, 2012; Raghupathi & Raghupathi, 2014; Wu & Pei, 2015; Fiorini et al., 2016; Oide et al., 2017; Kulkarni et al., 2016), as data volumes in both industry and research continue to outgrow the processing speed of individual machines. With big data processing, the stateful data analytics has drawn significant attentions recently, wherein the results of data analytics are determined not only by the algorithm logic and the inputs, but also the “state” of system. For example, Internet search engines keep the previous URL ranking score as the state, so that they can update the results against the rapidly evolving web pages incrementally rather than re-computing from scratch (Logothetis et al., 2010). The stateful data

DOI: 10.4018/IJSSCI.2018040105
analytics frameworks are expected to provide fine-grained and low-latency state access and scale with the large state size. Recent research has been focusing on developing new programming models and systems for state management to partially or completely fulfill these requirements (Salloum et al., 2016; To, Soto & Markl, 2017; Bhatotia et al., 2011; Castro Fernandez, Migliavacca, Kalyvianaki & Pietzuch, 2013; Fernandez et al., 2014; Gunda, Ravindranath, Thekkath & Zhuang, 2010; Murray et al., 2013). For example, Naiad (Murray et al., 2013) and SEEP (Castro Fernandez, Migliavacca, Kalyvianaki & Pietzuch, 2013) support computation over the state stored the local disk. Nectar (Gunda, Ravindranath, Thekkath & Zhuang, 2010) saves the previous results to derive the new one, but the proposed method does not provide fine-grained state access. SDG (Fernandez et al., 2014) and Piccolo (Power & Li, 2010) offer the full-fledged stateful data analytics functionalities with the imperative programming models.

As above described, there have been some research works to achieve stateful data analytics. These efforts can be classified into two categories, the new frameworks and the refactored stateless frameworks. For the new frameworks, the representative work like Naiad (Murray et al., 2013) and SEEP (Castro Fernandez, Migliavacca, Kalyvianaki & Pietzuch, 2013) adopt graph- and operator-oriented programming model respectively, but they cannot scale to the large state size beyond the capacity of a single host. The other representative studies in this direction include SDG (Fernandez et al., 2014) and Piccolo (Power & Li, 2010). Both of these two works adopt the imperative programming model and support large state in distributed environment.

With regard to the refactored stateless frameworks, which propose to embed the stateful component/abstraction into the stateless frameworks for keeping the capability with the existing frameworks like Hadoop. Some of these proposals have no support to fine-grained access to the state (Bhatotia et al., 2011; Bu, Howe, Balazinska & Ernst, 2010; Gunda, Ravindranath, Thekkath & Zhuang, 2010; Zaharia, Das, Li, Hunter & Stoica, 2013), because they have to cache the results from previous batching jobs and then derive the future results. A few of the frameworks in this category (Zaharia, Das, Li, Hunter & Stoica, 2013; Flink, 2017; Apache samza, 2017) have the limited support of state management. They assume that the data and state entries share the same hash key and an individual data entry can only involve a single state entry.

In order to meet the requirements of stateful data analytics within Spark, we introduce a stateful component, Resilient State Table (RST), with Spark. The programming interface of RST integrates with Spark’s Resilient Distributed Dataset (RDD) seamlessly by bridging the data entries stored in RDD and state entries in RST with a small number of user-defined functions. To associate RDD and RST, users can define StateKeyMapper to map the hash keys of data entries in RDD, and to that of state entries in RST. Users can specify the logic relationship with StateMapping-Func which derives the output from input data entries in RDD and the corresponding state entries RST. Additionally, the runtime of RST co-works with Spark’s data-locality-aware scheduling policy can also avoid/reduce remote state access and accommodate with Spark’s lineage-based fault-tolerance mechanism.

The paper contributions and its structure are as follows: we motivate the need for full-fledged stateful data analytics and propose our resilient state table (RST), and describe the architecture of Spark with RST in section 2; we talk about task scheduling with RST and RDD in section 3; we present the empirical evaluation results in section 4, and followed by related works in section 5, the conclusion and future work in final section 6.

2. ARCHITECTURE OF SPARK WITH RST

2.1. The Motivation

In this paper, we explore the answers to the following questions: (1) How to achieve the full-fledged stateful data analytics within Spark? (2) What benefits we can get from achieving stateful data analytics within Spark?
A Comparison of Human and Computer Information Processing
www.igi-global.com/chapter/comparison-human-computer-information-processing/56126?camid=4v1a

EEG Feature Extraction and Pattern Classification Based on Motor Imagery in Brain-Computer Interface
www.igi-global.com/article/eeg-feature-extraction-pattern-classification/60748?camid=4v1a