http://www.icact.org/journal.asp

ICACT-TACT JOURNAL

Transactions on Advanced Communications Technology

Volume 7 Issue 1, January. 2018, ISSN: 2288-0003

Editor-in-Chief
Prof. Thomas Byeongnam YOON, PhD.

GIRI
Global IT Research Institute

Journal Editorial Board

■ Editor-in-Chief

■ Editors

Dr. Youssef SAID, Tunisie Telecom, Tunisia
Dr. Noor Zaman, King Faisal University, Al Ahsa Hofuf, Saudi Arabia
Dr. Srinivas Mantha, SASTRA University, Thanjavur, India
Dr. Shahriar Mohammadi, KNTU University, Iran
Prof. Beonsku An, Hongik University, korea
Dr. Guanbo Zheng, University of Houston, USA
Prof. Sangho Choe, The Catholic University of Korea, korea
Dr. Gyanendra Prasad Joshi, Yeungnam University, korea
Dr. Tae-Gyu Lee, Korea Institue of Industrial Technology(KITECH), korea
Prof. Ilkyeun Ra, University of Colorado Denver, USA
Dr. Yong Sun, Beijing University of Posts and Telecommunications, China
Dr. Yulei Wu, Chinese Academy of Sciences, China
Mr. Anup Thapa, Chosun University, korea
Dr. Vo Nguyen Quoc Bao, Posts and Telecommunications Institute of Technology, Vietnam
Dr. Harish Kumar, Bhagwant institute of technology, India
Dr. Jin REN, North china university of technology, China
Dr. Joseph Kandath, Electronics & Commn Engg, India
Dr. Mohamed M. A. Moustafa, Arab Information Union (AIU), Egypt
Dr. Mostafa Zaman Chowdhury, Kookmin University, Korea
Prof. Francis C.M. Lau, Hong Kong Polytechnic University, Hong Kong
Prof. Ju Bin Song, Kyung Hee University, korea
Prof. KyungHi Chang, Inha University, Korea
Prof. Sherif Welsen Shaker, Kuang-Chi Institute of Advanced Technology, China
Prof. Seung-Hoon Hwang, Dongguk University, Korea
Prof. Dal-Hwan Yoon, Semyung University, korea
Prof. Chongyang ZHANG, Shanghai Jiao Tong University, China
Dr. H K Lau, The Open University of Hong Kong, Honh Kong
Prof. Ying-Ren Chien, Department of Electrical Engineering, National Ilan University, Taiwan
Prof. Mai Yi-Ting, Hsiuping University of Science and Technology, Taiwan
Dr. Sang-Hwan Ryu, Korea Railroad Research Institute, Korea
Dr. Yung-Chien Shih, MediaTek Inc., Taiwan
Dr. Kuan Hoong Poo, Multimedia University, Malaysia
Dr. Michael Leung, CEng MIET SMIEEE, Hong Kong
Dr. Abu sahman Bin mohd Supa'at, Universiti Teknologi Malaysia, Malaysia
Prof. Amit Kumar Garg, Deenbandhu Chhotu Ram University of Science & Technology, India
Dr. Jens Myrup Pedersen, Aalborg University, Denmark
Dr. Augustine Ikechi Ukaegbu, KAIST, Korea
Dr. Jamshid Sangirov, KAIST, Korea
Prof. Ahmed Dooguy KORA, Ecole Sup. Multinationale des Telecommunications, Senegal
Dr. Se-Jin Oh, Korea Astronomy & Space Science Institute, Korea
Dr. Rajendra Prasad Mahajan, RGPV Bhopal, India
Dr. Woo-Jin Byun, ETRI, Korea
Dr. Mohammed M. Kadhum, School of Computing, Goodwin Hall, Queen's University , Canada
Prof. Seong Gon Choi, Chungbuk National University, Korea
Prof. Yao-Chung Chang, National Taitung University, Taiwan
Dr. Abdallah Handoura, Engineering school of Gabes - Tunisia, Tunisia
Dr. Gopal Chandra Manna, BSNL, India

Dr. Il Kwon Cho, National Information Society Agency, Korea
Prof. Jiann-Liang Chen, National Taiwan University of Science and Technology, Taiwan
Prof. Ruay-Shiung Chang, National Dong Hwa University, Taiwan
Dr. Vasaka Visoottiviseth, Mahidol University, Thailand
Prof. Dae-Ki Kang, Dongseo University, Korea
Dr. Yong-Sik Choi, Research Institute, IDLE co., ltd, Korea
Dr. Xuena Peng, Northeastern University, China
Dr. Ming-Shen Jian, National Formosa University, Taiwan
Dr. Soobin Lee, KAIST Institute for IT Convergence, Korea
Prof. Yongpan Liu, Tsinghua University, China
Prof. Chih-Lin HU, National Central University, Taiwan
Prof. Chen-Shie Ho, Oriental Institute of Technology, Taiwan
Dr. Hyoung-Jun Kim, ETRI, Korea
Prof. Bernard Cousin, IRISA/Universite de Rennes 1, France
Prof. Eun-young Lee, Dongduk Woman s University, Korea
Dr. Porkumaran K, NGP institute of technology India, India
Dr. Feng CHENG, Hasso Plattner Institute at University of Potsdam, Germany
Prof. El-Sayed M. El-Alfy, King Fahd University of Petroleum and Minerals, Saudi Arabia
Prof. Lin You, Hangzhou Dianzi Univ, China
Mr. Nicolai Kuntze, Fraunhofer Institute for Secure Information Technology, Germany
Dr. Min-Hong Yun, ETRI, Korea
Dr. Seong Joon Lee, Korea Electrotechnology Research Institute, korea
Dr. Kwihoon Kim, ETRI, Korea
Dr. Jin Woo HONG, Electronics and Telecommunications Research Inst., Korea
Dr. Heeseok Choi, KISTI(Korea Institute of Science and Technology Information), korea
Dr. Somkiat Kitjongthawonkul, Australian Catholic University, St Patrick's Campus, Australia
Dr. Dae Won Kim, ETRI, Korea
Dr. Ho-Jin CHOI, KAIST(Univ), Korea
Dr. Su-Cheng HAW, Multimedia University, Faculty of Information Technology, Malaysia
Dr. Myoung-Jin Kim, Soongsil University, Korea
Dr. Gyu Myoung Lee, Institut Mines-Telecom, Telecom SudParis, France
Dr. Dongkyun Kim, KISTI(Korea Institute of Science and Technology Information), Korea
Prof. Yoonhee Kim, Sookmyung Women s University, Korea
Prof. Li-Der Chou, National Central University, Taiwan
Prof. Young Woong Ko, Hallym University, Korea
Prof. Dimiter G. Velev, UNWE(University of National and World Economy), Bulgaria
Dr. Tadasuke Minagawa, Meiji University, Japan
Prof. Jun-Kyun Choi, KAIST (Univ.), Korea
Dr. Brownson ObaridoaObele, Hyundai Mobis Multimedia R&D Lab , Korea
Prof. Anisha Lal, VIT university, India
Dr. kyeong kang, University of technology sydney, faculty of engineering and IT , Australia
Prof. Chwen-Yea Lin, Tatung Institute of Commerce and Technology, Taiwan
Dr. Ting Peng, Chang'an University, China
Prof. ChaeSoo Kim, Donga University in Korea, Korea
Prof. kirankumar M. joshi, m.s.uni.of baroda, India
Dr. Chin-Feng Lin, National Taiwan Ocean University, Taiwan
Dr. Chang-shin Chung, TTA(Telecommunications Technology Association), Korea

Editor Guide

■ Introduction for Editor or Reviewer

All the editor group members are to be assigned as a evaluator(editor or reviewer) to submitted journal papers at the discretion of the Editor-in-Chief. It will be informed by eMail with a Member Login ID and Password.

Once logined the Website via the Member Login menu in left as a evaluator, you can find out the paper assigned to you. You can evaluate it there. All the results of the evaluation are supposed to be shown in the Author Homepage in the real time manner. You can also enter the Author Homepage assigned to you by the Paper ID and the author's eMail address shown in your Evaluation Webpage. In the Author Homepage, you can communicate each other efficiently under the peer review policy. Please don't miss it!

All the editor group members are supposed to be candidates of a part of the editorial board, depending on their contribution which comes from history of ICACT TACT as an active evaluator. Because the main contribution comes from sincere paper reviewing role.

■ Role of the Editor

The editor's primary responsibilities are to conduct the peer review process, and check the final camera-ready manuscripts for any technical, grammatical or typographical errors.

As a member of the editorial board of the publication, the editor is responsible for ensuring that the publication maintains the highest quality while adhering to the publication policies and procedures of the ICACT TACT(Transactions on the Advanced Communications Technology).

For each paper that the editor-in-chief gets assigned, the Secretariat of ICACT Journal will send the editor an eMail requesting the review process of the paper.

The editor is responsible to make a decision on an "accept", "reject", or "revision" to the Editor-in-Chief via the Evaluation Webpage that can be shown in the Author Homepage also.

■ Deadlines for Regular Review

Editor-in-Chief will assign a evalaution group(a Editor and 2 reviewers) in a week upon receiving a completed Journal paper submission. Evaluators are given 2 weeks to review the paper. Editors are given a week to submit a recommendation to the Editor-in-Chief via the evaluation Webpage, once all or enough of the reviews have come in. In revision case, authors have a maximum of a month to submit their revised manuscripts. The deadlines for the regular review process are as follows:

Evalution Procedure	Deadline
Selection of Evaluation Group	1 week
Review processing	2 weeks
Editor's recommendation	1 week
Final Decision Noticing	1 week

■ Making Decisions on Manuscript

Editor will make a decision on the disposition of the manuscript, based on remarks of the reviewers. The editor's recommendation must be well justified and explained in detail. In cases where the revision is requested, these should be clearly indicated and explained. The editor must then promptly convey this decision to the author. The author may contact the editor if instructions regarding amendments to the manuscript are unclear. All these actions could be done via the evaluation system in this Website. The guidelines of decisions for publication are as follows:

Decision	Description
Accept	An accept decision means that an editor is accepting the paper with no further modifications. The paper will not be seen again by the editor or by the reviewers.
Reject	The manuscript is not suitable for the ICACT TACT publication.
Revision	The paper is conditionally accepted with some requirements. A revision means that the paper should go back to the original reviewers for a second round of reviews. We strongly discourage editors from making a decision based on their own review of the manuscript if a revision had been previously required.

■ Role of the Reviewer

Reviewer Webpage:

Once logined the Member Login menu in left, you can find out papers assigned to you. You can also login the Author Homepage assigned to you with the paper ID and author's eMail address. In there you can communicate each other via a Communication Channel Box.

Quick Review Required:

You are given 2 weeks for the first round of review and 1 week for the second round of review. You must agree that time is so important for the rapidly changing IT technologies and applications trend. Please respect the deadline. Authors undoubtedly appreciate your quick review.

Anonymity:

Do not identify yourself or your organization within the review text.

Review:

Reviewer will perform the paper review based on the main criteria provided below. Please provide detailed public comments for each criterion, also available to the author.

- How this manuscript advances this field of research and/or contributes something new to the literature?
- Relevance of this manuscript to the readers of TACT?
- Is the manuscript technically sound?
- Is the paper clearly written and well organized?
- Are all figures and tables appropriately provided and are their resolution good quality?
- Does the introduction state the objectives of the manuscript encouraging the reader to read on?
- Are the references relevant and complete?

Supply missing references:

Please supply any information that you think will be useful to the author in revision for enhancing quality of the paperor for convincing him/her of the mistakes.

Review Comments:

If you find any already known results related to the manuscript, please give references to earlier papers which contain these or similar results. If the reasoning is incorrect or ambiguous, please indicate specifically where and why. If you would like to suggest that the paper be rewritten, give specific suggestions regarding which parts of the paper should be deleted, added or modified, and please indicate how.

Journal Procedure

Dear Author,

- **You can see all your paper information & progress.**

- **Step 1. Journal Full Paper Submission**

Using the Submit button, submit your journal paper through ICACT Website, then you will get new paper ID of your journal, and send your journal Paper ID to the Secretariat@icact.org for the review and editorial processing. Once you got your Journal paper ID, never submit again! Journal Paper/CRF Template

- **Step 2. Full Paper Review**

Using the evaluation system in the ICACT Website, the editor, reviewer and author can communicate each other for the good quality publication. It may take about 1 month.

- **Step 3. Acceptance Notification**

It officially informs acceptance, revision, or reject of submitted full paper after the full paper review process.

Status	Action
Acceptance	Go to next Step.
Revision	Re-submit Full Paper within 1 month after Revision Notification.
Reject	Drop everything.

- **Step 4. Payment Registration**

So far it's free of charge in case of the journal promotion paper from the registered ICACT conference paper! But you have to regist it, because you need your Journal Paper Registration ID for submission of the final CRF manuscripts in the next step's process. Once you get your Registration ID, send it to Secretariat@icact.org for further process.

- **Step 5. Camera Ready Form (CRF) Manuscripts Submission**

After you have received the confirmation notice from secretariat of ICACT, and then you are allowed to submit the final CRF manuscripts in PDF file form, the full paper and the Copyright Transfer Agreement. Journal Paper Template, Copyright Form Template, BioAbstract Template,

Journal Submission Guide

All the Out-Standing ICACT conference papers have been invited to this "ICACT Transactions on the Advanced Communications Technology" Journal, and also welcome all the authors whose conference paper has been accepted by the ICACT Technical Program Committee, if you could extend new contents at least 30% more than pure content of your conference paper. Journal paper must be followed to ensure full compliance with the IEEE Journal Template Form attached on this page.

➢ How to submit your Journal paper and check the progress?

Step 1. Submit	Using the Submit button, submit your journal paper through ICACT Website, then you will get new paper ID of your journal, and send your journal Paper ID to the Secretariat@icact.org for the review and editorial processing. Once you got your Journal paper ID, never submit again! Using the Update button, you can change any information of journal paper related or upload new full journal paper.
Step 2. Confirm	Secretariat is supposed to confirm all the necessary conditions of your journal paper to make it ready to review. In case of promotion from the conference paper to Journal paper, send us all the .DOC(or Latex) files of your ICACT conference paper and journal paper to evaluate the difference of the pure contents in between at least 30% more to avoid the self replication violation under scrutiny. The pure content does not include any reference list, acknowledgement, Appendix and author biography information.
Step 3. Review	Upon completing the confirmation, it gets started the review process thru the Editor & Reviewer Guideline. Whenever you visit the Author Homepage, you can check the progress status of your paper there from start to end like this, " Confirm OK! -> Gets started the review process -> ...", in the Review Status column. Please don't miss it!

Volume. 7 Issue. 1

An Improvement of a Checkpoint-based Distributed Testing Technique on a Big Data Environment

Bhuridech Sudsee, Chanwit Kaewkasi

School of Computer Engineering

Suranaree University of Technology, Nakhon Ratchasrima, Thailand, 30000

m5741861@g.sut.ac.th, chanwit@sut.ac.th

***Abstract*— The advancement of storage technologies and the fast-growing number of generated data have made the world moved into the Big Data era. In this past, we had many data mining tools but they are inadequate to process Data-Intensive Scalable Computing workloads. The Apache Spark framework is a popular tool designed for Big Data processing. It leverages in-memory processing techniques that make Spark up to 100 times faster than Hadoop. Testing this kind of Big Data program is time consuming. Unfortunately, developers lack a proper testing framework, which cloud help assure quality of their data-intensive processing programs while saving development time and storage usages.**

We propose *Distributed Test Checkpointing (DTC) for Apache Spark*. DTC applies unit testing to the Big Data software development life cycle and reduce time spent for each testing loop with checkpoint. By using checkpoint technique, DTC keeps quality of Big Data processing software while keeps an inexpensive testing cost by overriding original Spark mechanism so that developers no pain to learn how to use DTC. Moreover, DTC has no addition abstraction layers. Developers can upgrade to a new version of Spark seamlessly. From the experimental results, we found that in the subsequence rounds of unit testing, DTC dramatically speed the testing time up to 450-500% faster. In case of storage, DTC can cut unnecessary data off and make the storage 19.7 times saver than the original checkpoint of Spark. DTC can be used either in case of JVM termination or testing with random values.

***Keyword*— Distributed Checkpointing; Apache Spark; Big Data Testing; Software Testing;**

I. INTRODUCTION

THE increasing and diversity of electronic devices, sensors, IoT devices and the fast-growing numbers of Internet users have been generating tremendous amount of data recently. They are not only the large amount of data but their structures are also complex as well. This complexity makes the traditional data mining tools inadequate to manage today's data [1].

The MapReduce [2] programming model has induced the development of many frameworks such as Apache Hadoop [4], Map-reduce-merge [5] and Apache Spark [6], which aim to process data intensive tasks. Developers only need to rewrite their programming logic in the form of *map* and *reduce* functions in order to process data on a MapReduce framework. These functions will be automatically managed by the framework's default configuration. This mechanism makes the MapReduce framework easy to use. At its simplest form, a MapReduce program usually starts by a *map* function creating key/value pairs from the input. These intermediate key/value pairs are then passed to a *reduce* function to produce the final results. The MapReduce model is parallel by nature. It is designed to allow developers to run MapReduce programs for high performance computing jobs using a commodity cluster, built from low-cost hardwares. With this kind of the cluster architecture, we can handle massive amount of data and process them on numerous cluster nodes without a single point of failure [3].

Although the MapReduce model is easy to use for software development, but it is quite tricky to test software written by the MapReduce model. Software testing is a vital part of the development process. Testing is usually 25-50% of the overall cost [8]. We found that the current mechanism is not enough to assure quality for Big Data processing programs. Unit testing is a software testing technique which properly leads to better levels of quality. However, tools like Scalatest[9] or jUnit[10] have their own limitations to use with a MapReduce framework like Spark. For example, SparkContext and SparkSession objects must be instantiated only once for each running Java Virtual Machine (JVM) to avoid unexpected testing results [12]. Spark-testing-base [11] also does not have a testing mechanism for Spark. Without modification, it cannot work on a Spark cluster because if its inability to distribute class files across worker nodes. There aforementioned techniques are not suitable for Spark simply because they are not designed to test programs that distributelly process large amount of data.

Test-driven development (TDD) is a software development technique that helps developers to focus on

Manuscript received December 27th, 2017. This work was supported by Suranaree University of Technology, and a follow-up of the invited journal to the accepted & presented paper of the 20th International Conference on Advanced Communication Technology (ICACT2018),

Bhuridech Sudsee is with School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasrima, Thailand (corresponding author phone: +66-44-22-4422; e-mail: m5741861@g.sut.ac.th).

Chanwit Kaewkasi is with School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasrima, Thailand (e-mail: chanwit@sut.ac.th).

writing a specific test at a time. It additionally allows code improvement while preserving correctness according to the specification. TDD workflow consists of the following steps, (1) writing a minimum test (2) writing codes to just make the test passed, and (3) refactoring to remove unnecessary codes while still making the current test passed [13]. We call these steps a TDD workflow herein this paper. Applying TDD to data intensive programs is difficult due to the nature of workloads, which need to process on a cluster. So, developers require a special tool to help shorten each loop of the TDD workflow.

Spark has *cache*, *persist* and *checkpoint* methods to help mitigate job failure. These mechanisms however do not help software testing process much. The main reason is that a cluster state cached or persisted by them does not survive across runs of JVMs. A cluster state saved by the *checkpoint* method does survive on disk but unfortunately it cannot be retrieved back by a newly started JVM [14, 15].

In this paper, we present Distributed Test Checkpointing (DTC), a technique that leverages the checkpoint technique to enhance software testing for data intensive jobs. With DTC, developers can increase productivity when testing their software on a distributed cluster repeatedly. DTC applied a hash function on each data partition of a Resilient Distributed Datasets (RDD) [18] to use an identifier. Modification of an RDD or a Dataset can be traced by the hashed number. The testcase that uses the RDD is also hashed at the bytecode level. Combining these techniques, DTC is found to reduce testing time and storage required by checkpointing significantly compared to the original Spark's checkpointing technique.

The remaining of this paper is organized as followed. Section II discusses related works, including Apache Spark. Section III presents the design and internal mechanism of DTC. Section IV presents the system architecture of the cluster used by our experiments, and the experimental results. This paper then ends with conclusion and future works in Section V.

II. Background and Related work

A. Apache Spark

Spark is a data intensive processing framework focusing on in-memory data processing [6], which is implemented in the form of Resilient Distributed Dataset (RDD) [18]. RDD is designed to take care of the data flow and handle the processing mechanism. An RDD could be created using one of the following methods (1) reading data from file (2) parallelizing collection in the driver program (3) transforming from another RDD (4) and by transforming back from a persisted RDD [6]. An RDD comprises with two kinds of command, *transformations* and *actions*. A transformation command transforms an RDD to another RDD. These commands are *map*, *filter* and *groupByKey*, for example. Another set of commands are actions, which are *collect* and *count*, for example. An RDD keeps all previous transformation inside itself. This direct acyclic graph of transformation is known as *lineage*. The beginning of the real computation occurs only when an action is called. This is the lazy evaluation nature of Spark.

A mechanism for failure recovery that helps an RDD to resume the processing without re-computation from scratch are methods such as *cache*, *persist* and *checkpoint*. The *cache* method uses persistency at MEMORY_ONLY, while the *persist* method has several levels of persistency. The checkpoint method, in contrast, uses the technique which save data onto a reliable storage, such as HDFS, Amazon S3 or Ceph. An RDD is usually cached or persisted during its computation to avoid re-computation previous steps [15].

The checkpoint technique is also applicable for Spark Streaming because it truncates the internal lineage, so the RDD does not need to knowledge of its parent. However, this mechanism is not designed for software testing. The re-computation is still required to start from the beginning when the testcase is re-run. The rerunning of the testcase destroys a Block Manager inside an Executor. This Block Manage is responsible for keeping cached and persisted data. The new Driver program and the testcase therefore is not able to access the location of checkpoints.

In addition, Spark has introduced the Dataframe API in 1.3 and Dataset in 1.6. Both abstractions can be used interchangeably because Dataset[Row] is the type safer version of DataFrame. A dataset is also convertible to an RDD. In the case of DTC proposed in this paper, we read and write data directly without triggering any computation of related RDDs.

B. Debugging framework for Spark

A technique used to improve quality of the software is debugging. Developers usually debug to observe certain set of variables they are interested. However, in the Data-intensive Scalable Computing (DISC), the debugging process is difficult as data are computed distributedly on a cluster.

BigDebug [7] is a tool designed to helps Spark's developers deal with debugging a Big Data program. There is a downside that the tool requires user's interaction during the debugging process. Those interactions make the debugging more difficult than those of normal programs because the Big Data programs are distributed by nature. Moreover, a BigDebug program cannot tackle the problem when the RDD being debug requires changes. The whole debugging process needs to start over in that case. In case of the developer changing codes on-the-fly, the RDD will become in-consistent as some partitions of the RDD has been processed by the old version of codes, while other partitions will be processed by the new codes. BigDebug support Spark up to 1.2.1 as the time writing.

C. Checkpoint implementation for Spark

Researchers have been employed the checkpoint of Spark in many ways to improve its efficiency, as follows.

Flint [26] was created atop the original checkpoint technique of Spark. It aims at applying checkpoint and store their data on transient instances to reduce the VM usage cost. A transient instance in a kind of low-cost computing unit, which can be recalled anytime by its cloud provider. Flint solves this problem by writing an RDD's partitions to an HDFS, which is operated on on-demand instances. We found that this implementation lacks a mechanism to prevent re-calculation when JVM is terminated. In addition,

their checkpoint will be saved automatically so developers need to prepare a huge amount of space in order to prevent the full of storage, which can lead to the failure of the whole system.

TR-Spark [27] implements the similar approach as Flint. The difference is that TR-Spark allows fined-granularity checkpoints at task-level. By leveraging this level of checkpoints, the storage usage cloud be reduced in comparison to checkpoint the whole RDD. However, TR-Spark makes it difficult to use as developers need to collect the information of VM failure to let it know the failure probability. TR-Spark does not deal with changes of the Driver program.

Automatic Spark Checkpointing (ASC) [25] was designed to help analyze the trade-off between RDD checkpointing and its restore. ASC performs this computation by estimating them from an RDD lineage. Nevertheless, this technique does not support checkpoint across JVM termination. It also lacks the ability to recognize the similarity or identity of an RDD.

Spark-flow [24] aims to mitigate the effect of JVM termination for checkpoint restoration. It makes use of Distributed Collection (DC), a library similar to the Dataset API. DC is able to analyze an RDD at the bytecode level with ASM. It can identify the location of checkpoint calls, inside an anonymous function. It also uses the MD5 hash function to help detect changes at the bytecode level. However, DC has some downside as the following. First, when calling checkpoint on a DC, the data is re-read again after checkpointing. Second, when restoring from checkpoint, the action *count* will be triggered, so the re-computation kicks in. Finally, computation is mainly done on the Driver machine, so the mechanism is actually not distributed. This often causes Out-of-Memory exception inside the Driver program and it stops working.

```
1  val data = sc.parallelize(Array(1,2,3,4,5))
2  val distData = data.map(x => (x,1))
3  distData.dtCheckpoint()
4  distData.count()
5  distData.collect()
```

Fig. 1. Example of a dtCheckpoint call on an RDD

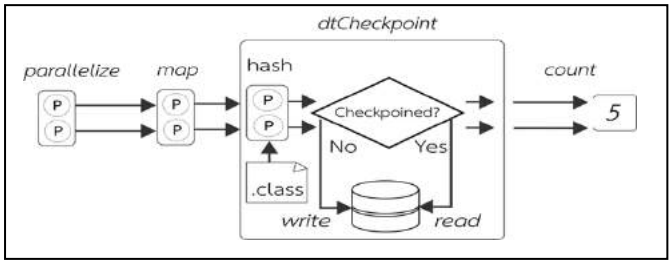


Fig. 2. The dtCheckpointing mechanism inside DTC

III. Design and Implementation

Spark stores the RDD transformations in the form of a lineage graph a.k.a. the logical execution plan. When an action is triggered for a certain RDD, its job will be submitted to the DAG Scheduler to transform the RDD's lineage into a directed acyclic graph, whose a *vertex* is an RDD partition and *edge* is a transformation. After that the staging process will be kicked in. This staging process will be started from the final action going backwards to the beginning of the RDD. However, in the real execution, the process will be performed from the beginning of the RDD forwardly to the final action. After the staging, the system obtains a set of Stages and Tasks.

A checkpoint of an RDD however must be done before the first action is performed. From the source code in the Fig. 1, when a program starts to process an array of integer 1 to 5, the array will be passed as a parameter of method *parallelize* of class SparkContext. This result in a ParallelCollectionRDD stored in variable *data*. At line 2, each element from the *data* RDD is mapped with 1 using the *map* method as a key/value pair. The result is a MapPartitionsRDD stored in variable *distData*. At line 3, method *dtCheckpoint* is invoked. Please note that the original Spark and DTC both use the lazy evaluation mechanism, this means that the checkpoint method only marks at a certain point over the DAG, where checkpoints will happen there. At line 4, command *distData.count()* is the first action. When this first action is triggered, the checkpoint is not yet created. The computation then is started from the beginning of the RDD to the mark point. After that, the checkpoint is stored at the first upper directory level as a hash value generated by the mechanism of DTC. At the line no 5, method *distData.collect()* is invoked as the second action. The system will then check backwards from the action to the beginning of the RDD. This time the system will find a checkpoint already existed because there is a directory whose name matches with the hash. When the DAG Scheduler starts to transform the lineage, it uses the data directly from the checkpoint without re-computation. Please also note that action *count()* and *collect()* belong to the different jobs. The result computed by *count()* will not be included as an input for *collect(),* despite their order of execution.

In Scala, it allows us to implement a new feature for a class by creating an *Implicit Class* then mixes it in to the existing classes, like RDD or Dataset. The DTC mechanisms proposed in this paper are implemented using that technique. With DTC as an Implicit Class, developers could still use all existing properties and behavior of an RDD, while having an additional method from DTC. Developers are also able to upgrade the Spark framework to the newer versions without rewriting this mechanism. DTC is more suitable for testing than Spark-flow, which has many abstraction layers. These abstraction makes it difficult to enhance capability of Spark-flow.

A. DtCheckpointing

This mechanism works when the method *dtCheckpoint* of an RDD or a DataSet is called. This call marks an RDD and also starts the Hashing RDD mechanism to obtain a directory path from hash transformation. If there is no directory matched the hash value, it means that the system never created that checkpoint. After the creation of the directory content of the RDD will be stored inside of it. But if the directory exists, the system will read the content as the data of the RDD. In Fig. 2, when an RDD is created using the *parallelize* method and is transformed with *map* followed by an invocation of *dtCheckpoint*. The sub-system

DtCheckpointing kicks in to mark points in the RDD for later storing when action *count* is called.

We usually perform the test on a Spark Cluster with SBT, which is an interactive build tool to help develop software with Java or Scala. SBT allows us to write a build file using Scala-based Domain Specific Language. It manages a program dependency with Apache Ivy. With DTC, we modify test commands of the SBT namely *test*, *test-only*, and *test-quick* to support not only the local execution but also in the real working cluster. We solve the problem of *ClassNotFoundException* and *NoClassDefFoundError* by making a fat jar via custom SBT task. So, we introduce *testOnCluster* for testing every testcase, *testOnlyOnCluster* to test a specific testcase, and *testQuickOnCluster* to test a certain testcase which may be failed from last time, or never tested or need re-computation. Our modification to SBT allows the new mode of testing on the real cluster.

B. Hashing an RDD

Hash function is a one-way function which can be used to check data modification. Eve one bit of data is changed this function notices that modification. In this paper, we will compare MD5, SHA-1 and SHA-256 because these algorithms have various speed of hash and resource usage.

This technique of the DTC framework is able to track the change of an RDD because the generated transformations. So we can use this mechanism to detect modification of any transformation back to the original RDD. When an action is triggered, the DTC framework detects all RDD dependencies and prepares a clean bytecode available by the CleanF property of the RDD, following by preparing other Java bytecode's files which related to the dependencies. In preparation stage, DTC uses ASM, a tool to manage a Java bytecode [17], which Scala internally uses it for the compilation mechanism. With a ASM, the DTC's hashing an RDD mechanism can access Java class file at runtime and de-serialize them for reverse engineering propose. DTC needs to remove some brittle information such as LINENUMBER or serialVersionUID from a class file. With this information filtered out, we can detect changes of an RDD or DataSet even when the line numbers have been changed.

The result of class file analysis in preparation stage, after unnecessary dependencies was eliminated, these dependencies will compute hash number and input data, which the origin of an RDD will compute hash number also. The computation is distributed computing with Spark's accumulator in the first level hash number computation will compute hash number of input data for every partition, and then collect and reorder result because unpredictable computation time. After that, the DTC will compute hash number of sorted hash number again. Fig. 3, illustrates the steps of hashing mechanism please note that the computation of input data is an option that can specify with *dtCheckpoint(true)*.

```
SET hash_array = empty array of string
IF (HASH_INPUT_DATA = true) THEN
  READ each data partition from (RDD or DataSet)
  COMPUTE hash of each data partition
  APPEND hashes to hash_array
ENDIF
```

Fig. 3. Pseudo codes of the mechanism of Hashing an RDD

IV. Experiments

A. Cluster configuration

The experiments presented in this paper have been conducted on a Spark cluster consisted of 10 nodes. Each node is an Intel Core i5-4570 Quad-core with 4 GB of RAM. The drive node is an Intel Xeon E5-2650V3 Deca-core with 8GB of RAM. We use Apache Spark 2.0 for the experiments along with Ceph as the distributed file system over these 10 nodes. The Ceph storage is 10 TB. The system architecture is illustrated in Fig. 4.

TABLE I
Computation Programs and Input Data of Experimental

Program	Input dataset
Wordcount	31 GB of Wikipedia
Triangle Counting	875,713 vertices and 5,105,039 edges
PageRank	875,713 vertices and 5,105,039 edges
Pi Estimation	10^9 times

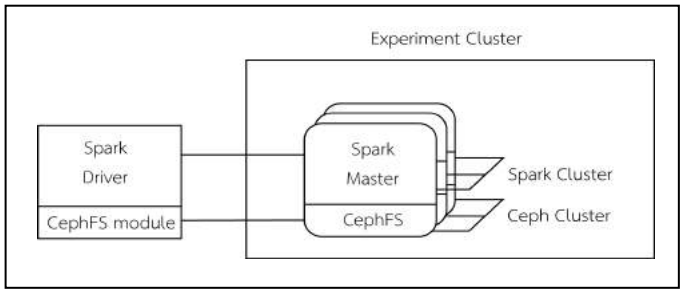


Fig. 4. The cluster architecture used by the experiments

B. Methodology

For the experiments, we use a MapReduce program Wordcount on 31 GB data dump of Wikipedia, Triangle Counting with Google Web Graph [28], PageRank with Google Web Graph and the last one is Pi Estimation with one billion times. Each program with its input dataset is shown in Table I. The Wordcount Program splits sentences into array of words and counts them using both RDD and Dataset (or DC in case of Spark-flow) with different checkpoint mechanisms. We tested each checkpoint mechanism 10 times continuously and measured both in space and time perspectives. Moreover, we tested 5 additional with JVM termination. Then we started the JVM again to test the recovery process of checkpoints.

Table II shows the comparison of checkpoint mechanism properties. If we do not use checkpoint, the system does not have the fault tolerance property. If we use the original Spark, it is not suitable for testing because its checkpoint mechanism does not work well in the test environment. In case of Spark-flow it does not work on the cluster environment out-of-the-box. DTC, on the other hand, is designed to address these problems in the testing

TABLE II
FEATURE COMPARISON BETWEEN CONFIGURATIONS

Method	Failure tolerance	More abstraction layer	Prevent re-calculation from beginning	Suitable for Testing	Cluster
No-Checkpoint	No	No	No	No	Yes
Spark Original	Yes	No	Yes	Not Suitable	Yes
Spark-flow	Yes	Yes	Yes	Yes	No
DTC	Yes	No	Yes	Yes	Yes

TABLE III
THE COMBINATION OF ALL EXPERIMENTAL CONFIGURATIONS

Configuration	Type			Checkpoint Data Format				Hash Algorithm		
	RDD	DataSet	DC	Java	Kryo	Avro	Parquet	MD5	SHA1	SHA256
No-checkpoint	√	√	-	-	-	-	-	-	-	-
Spark Original	√	√	-	√	-	-	-	-	-	-
Spark-flow	-	-	√	-	-	-	√	√	-	-
DTC	√	√	-	√	√	√	√	√	√	√

environment. So, DTC provides the better environment to support unit testing.

Table II shows a brief differentiation of comparison method that we will experiment. That meant, if we have no checkpoint it will lack failure tolerance, the Spark original checkpoint insufficient to testing. The Spark-flow push developer in more abstraction layer by create a higher level of a DataSet and it not work on cluster naturally. In Table III, we show the combination of all experimental configurations. Accordingly, the DTC introduce to rectify that plain.

We compared with MapReduce Wordcount algorithms on Wikipedia 31 GB with separating each word from each other with white space. And then, we filtered only word occurred more than 10 million times, after that asserted with the most word occurred. We consecutively repeated these steps 10 cases and performed testing on 5 cases then stopped the JVM. After that we re-run these 5 cases again on both RDD and DataSet.

Next, we compared with Triangle Counting Program which gathers the number of vertices whose has two adjacent vertices with an edge between them. And then perform PageRank Program to ranks members onto the graph. Input of these programs came from Google Web Graph. with 875,713 vertices and 5,105,039 edges, testing on 5 cases then stop the JVM, after that re-run these 5 cases again on RDD.

Finally, we compared the Pi Estimation program by using Monte Carlo algorithm shows in (1) [29].

$$\begin{aligned}\mathbb{P}\,(drop\ within\ circle) &= \frac{Area\ of\ the\ unit\ circle}{Area\ of\ the\ square}\\ &= \frac{\iint_{\{x^2+y^2\le 1\}} 1\,dx\,dy}{\iint_{\{-1\le x,y\le 1\}} 1\,dx\,dy}\\ &= \frac{\pi}{4}\end{aligned} \quad (1)$$

The algorithm randomly generated two values which represent to coordinate x and y of unit circle (so both x and y are between -1 to 1). After that, trying to addition between square magnitude of x and square magnitude of y and if that result less than or equal to 1 will be count as fall in the unit circle. That number will use to represent $\pi/4$, so that we can multiply by 4 to roughly results Pi number. We tested 5 cases then stop the JVM, after that we re-run these 5 cases again on RDD.

C. Experimental results (consecutively 10 cases)

From the experiments, we start discussing in the case of no hashing input data, denoted *not-hashinput* by running consecutively 10 cases. In this case the input will not be verified by hashing functions before the program starts. We assume that development and during the tests. The experimental results are show in Fig. 5. At the first run, DTC and the `original-checkpoint` mechanism are all slow with insignificant difference. The `DTC-Java-SHA1` is slowest. It uses 636 seconds slightly

TABLE IV
CHECKPOINT'S STORAGE USAGE OF AN RDD

Storage usage	Size	Unit
No-checkpoint	0	MB
Spark original checkpoint	9.870	MB
DTC-Java-with-hash	0.987	MB
DTC-Java-without-hash	0.987	MB
DTC-Kryo-with-hash	*0.501*	*MB*
DTC-Kryo-without-hash	*0.501*	*MB*

TABLE V
CHECKPOINT'S STORAGE USAGE OF DATASET

Storage usage	Size	Unit
No-checkpoint	0	MB
Spark original checkpoint	9.860	MB
DTC-Avro-with-hash	*0.987*	*MB*
DTC-Avro-without-hash	*0.987*	*MB*
DTC-Parquet-with-hash	0.993	MB
DTC-Parquet-without-hash	0.993	MB
Spark-flow	9.930	MB

different from `original-checkpoint`. The `no-checkpoint` configuration does not have this startup overhead, so it run at 136 seconds on average. For the first run, All DTC and the `original-checkpoint` are 4.7 times or slower than the `no-checkpoint` mechanism. However, all DTC configurations are significantly faster in the subsequence runs.

Fig. 6 shows the comparison between cases of applying hash functions over input data to allow the system to detect

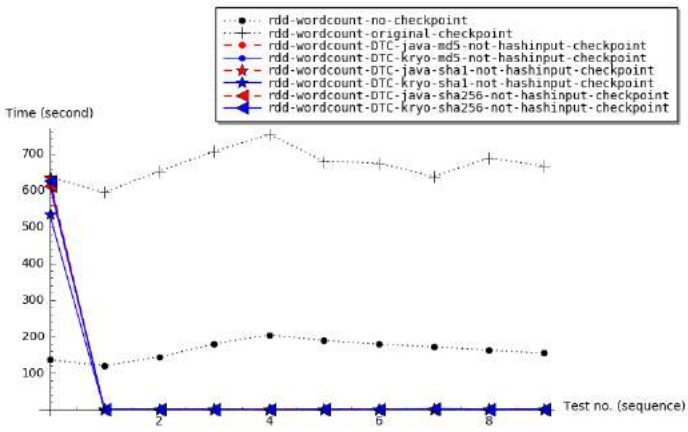


Fig. 5. Comparison of checkpoint time of RDDs without hashing inputs using the Wordcount program. (10 cases consecutively)

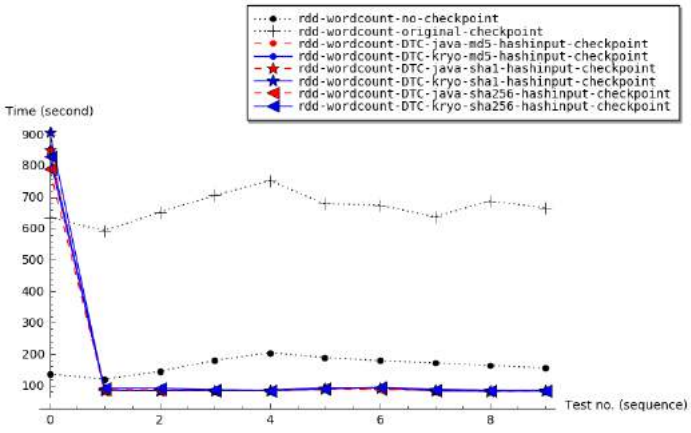


Fig. 6. Comparison of checkpoint time of RDDs with hashing inputs using the Wordcount program. (10 cases consecutively)

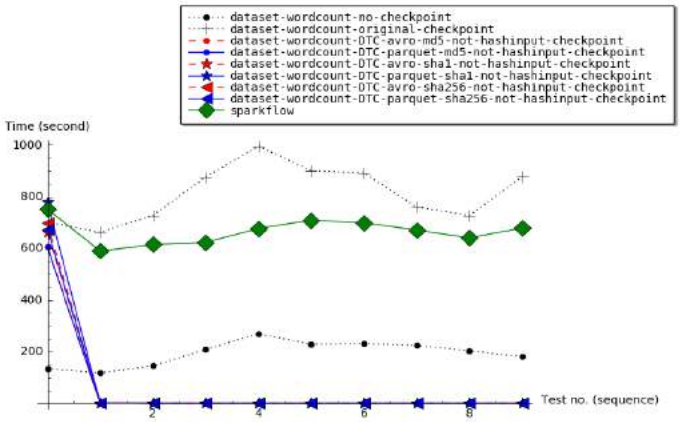


Fig. 7. Comparison of checkpoint time of DataSet,including Spark-flow without hashing inputs using the Wordcount program (10 cases consecutively).

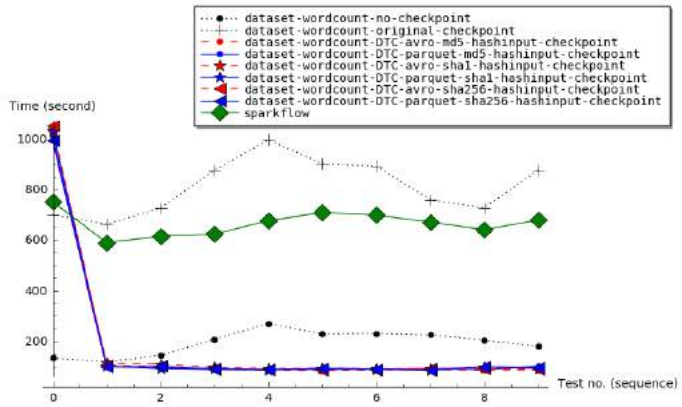


Fig. 8. Comparison of checkpoint time of DataSet,including Spark-flow with hashing inputs using the Wordcount program (10 cases consecutively).

changes of the input. It shows that DTC mechanisms are slower than `no-checkpoint` and `original-checkpoint` only in the first run. In the subsequence runs, DTC mechanisms make the test s faster than those run by `no-checkpoint` and `original-checkpoint`. We found that `DTC-Kryo-SHA1` is slowest in the first run. It uses 908 seconds on average, while `no-checkpoint` uses 136 seconds and `original-checkpoint` use 636 seconds. In the subsequence runs, DTC mechanism uses around 85 seconds on average. It is significantly faster that both `no-checkpoint` and `original-checkpoint`, which is 60%

In the first run with hash input, the fastest DTC mechanism is `DTC-Java-SHA256` it is 480% slower than no-checkpoint and 24% slower than `original-checkpoint`. In the subsequence runs, this mechanism is 40% faster than `no-checkpoint` and 590% faster than `original-checkpoint`. Other cases are in similar trends.

In case of DataSet, we found the similar trends as the case of RDD. During the first run DTC mechanisms are slowest, and significantly faster in subsequence runs. Fig. 7 and Fig. 8 show the comparison between checkpoint mechanisms for the DataSet without hashing input and with hashing input, respectively. We also include `Spark-flow` in these experiments. We found that `Spark-flow` uses 752 seconds at the first run, while `DTC-Parquet-MD5` uses 606 seconds, so DTC is 24% faster than `Spark-flow`. In case of hash input data, DTC is 40% slower than `Spark-flow` for the first run. However, in the subsequence runs, DTC dramatically reduces time spending, according aforementioned trends.

The mechanism of checkpoint usually requires use of storage. The storage usage comparison is then presented in Table IV. According to the table, DTC with Java serializer uses the storage only one-tenth of those used by the original Spark checkpoint. In case of DTC with Kryo, it uses storage only 5% of the `original-checkpoint`.

This storage usages are similar for DataSet. According to Table IV, DTC with Avro format uses only 10% of the original storage. In case of DTC with Parquet format, it uses only 11% of the original storage. Comparison of these results with `Spark-flow`, we are roughly at the same ration.

DTC is designed to allow re-usability of RDDs and DataSets. It can traverse and detect change of the dependency of each RDD or a DataSet. From the experiments, we have found that DTC has a larger overhead than the mechanism of the Original Spark only when a testcases are in first run. When the testcases are in the later runs, DTC makes them 5-6 times faster than running by the Original Spark and `Spark-flow`. Moreover, DTC uses disk space 8-9 times less than both implementations as shown in Table IV and Table V.

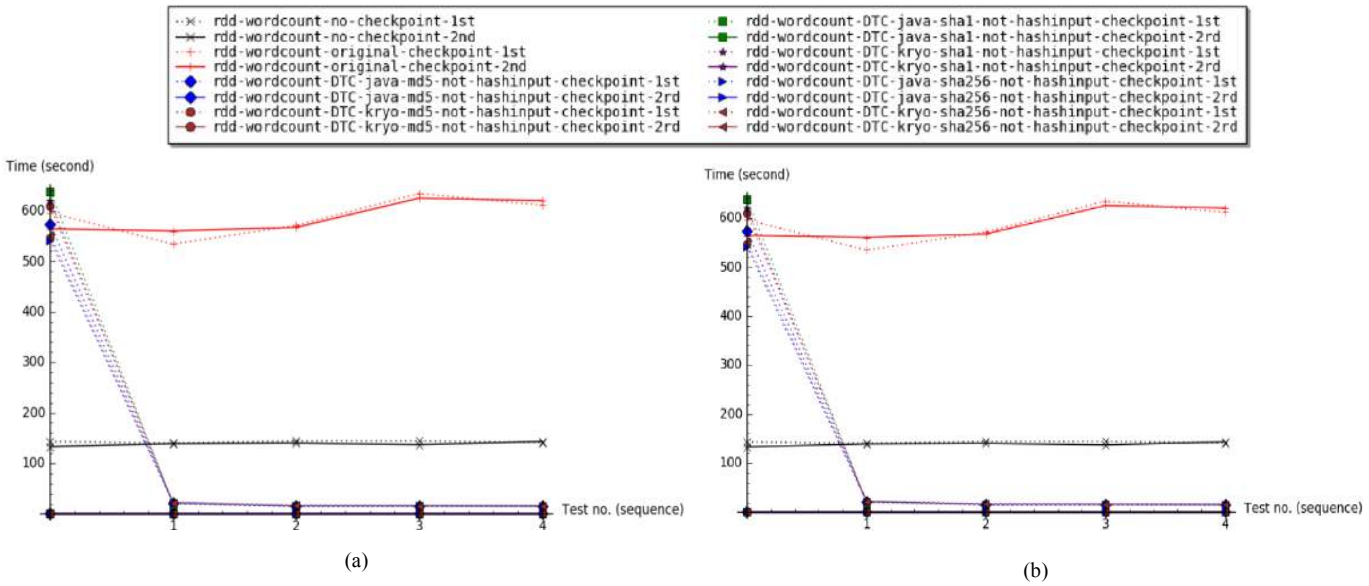


Fig. 9. Comparison of checkpoint time of RDDs using the Wordcount program (5 cases with JVM termination) while (a) without hashing inputs and (b) with hashing inputs.

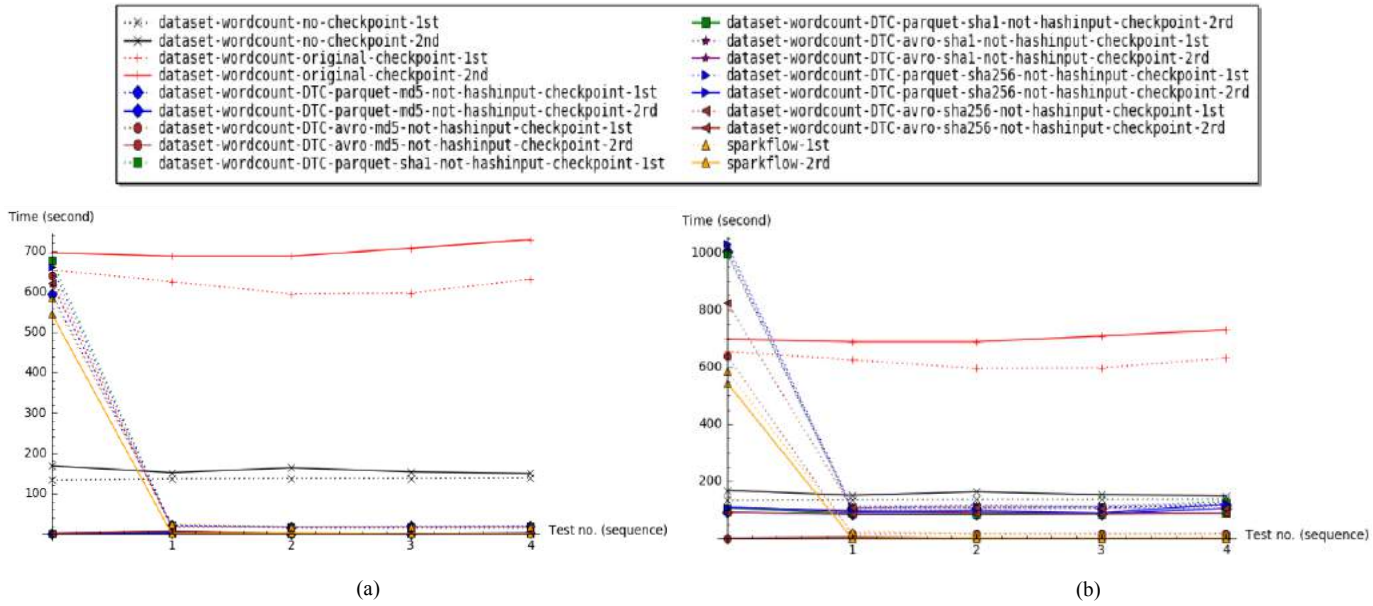


Fig. 10. Comparison of checkpoint time of DataSet using the Wordcount program (5 cases with JVM termination) while (a) without hashing inputs and (b) with hashing inputs.

D. Experimental results (5 cases with JVM termination)

In this section, we discuss the experimental results in case of running 5 cases consecutively, then stopping the JVM, after that the experimental cases were re-run again. Its behavior on different frameworks were observed.

Firstly, we discuss the result of the Wordcount program on RDD. We found that `DTC-Java-SHA256` used 542 seconds at the first run in case of running if before stopping JVM, so DTC is 9% faster than `original-checkpoint` which uses 596 seconds. After stopping JVM or closing the program then re-running the test cases, DTC with all settings used only few seconds to recover checkpoint, while other frameworks used hundreds of second, as showed in Fig 9. In Fig 9, the dashed line is the first running before JVM terminating and the solid line is the second running after restarting the JVM.

In the case of DataSet shown you in Fig 10, the dashed line presents the first run of 5 cases. We found that the `original-checkpoint` used 654 seconds, while `Spark-flow` used 585 seconds. So, `Spark-flow` is 11% faster than the original one. But DTC with the `DTC-Parquet-MD5` configuration, it used 595 seconds, 9% faster than `original-checkpoint`. However, in the second run of 5 cases after restarting the JVM, as the solid line, the results show that the `original-checkpoint` used 697 seconds and `Spark-flow` used 545 seconds, while DTC with any configuration used just few seconds.

Fig. 11 shows the results comparing between frameworks using Triange Counting Program, In the case of not applying hashing to the input data, we showed that in Fig 11 (a), `no-checkpoint`, `original-checkpoint` and DTC used almost the same amount of time for the first runs.

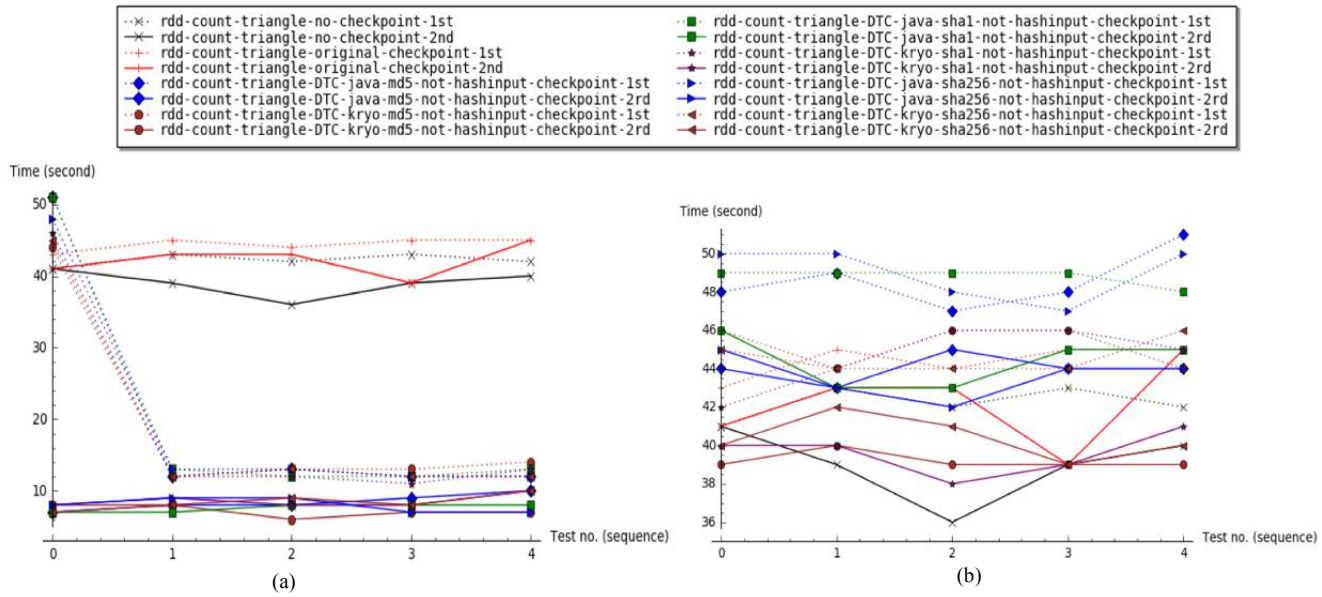


Fig. 11. Comparison of checkpoint time of RDDs using the Triangle Counting program (5 cases with JVM termination) while (a) without hashing inputs and (b) with hashing inputs.

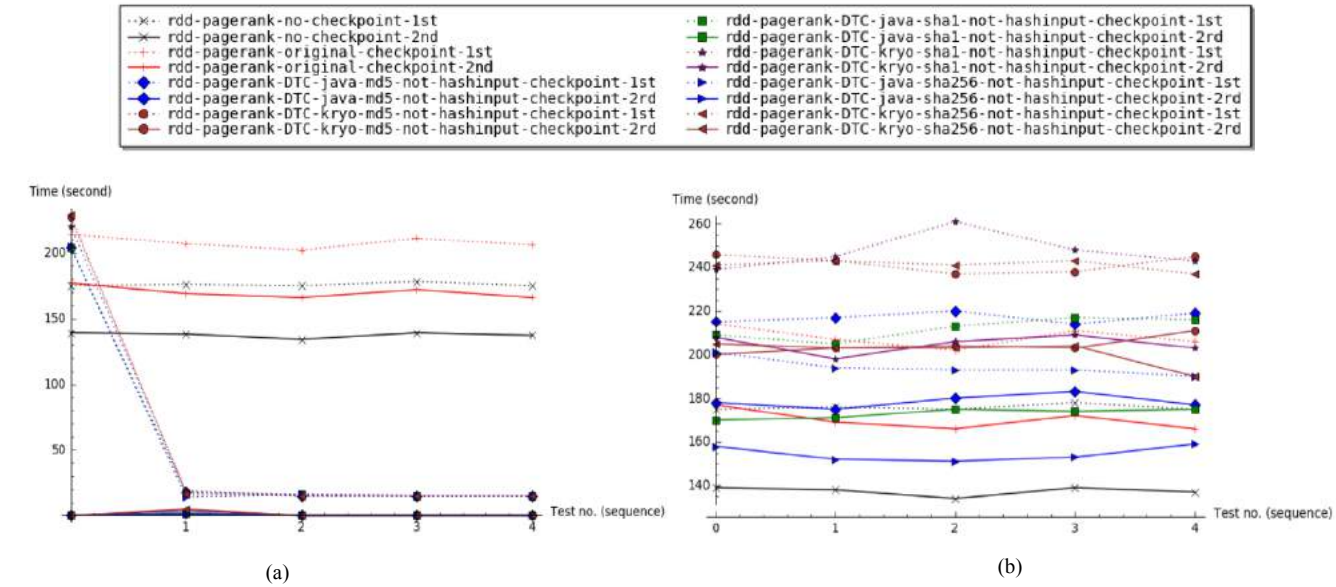
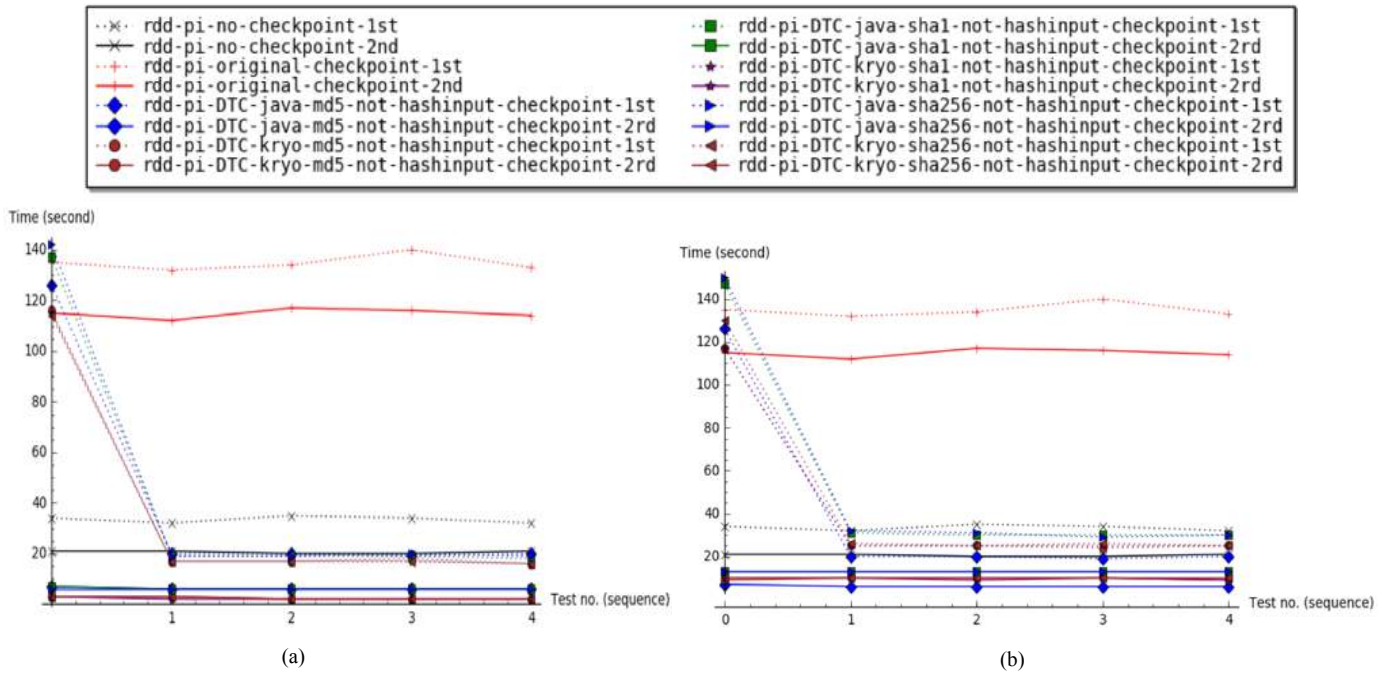


Fig. 12. Comparison of checkpoint time of RDDs using PageRank Program (5 cases with JVM termination) while (a) without hashing inputs and (b) with hashing inputs.

For the second runs after restarting the JVM, we found the same trend as we were discussing earlier. DTC with all configurations could reduce time for testing to just a few seconds. Due to inputs were in the form of graph (vertices and edges) as shown in Fig 11 (b), the underlying mechanism of the Spark Framework tries to perform operations efficiently by casting the partition of the input to class *ShippableVertexPartition*. In the research work reported in this paper, DTC does not import to support to read this kind of data type. Fig 11 (b) shows that DTC with all configurations could not help reduce time much. All frameworks use the same amount of time processing the data.

In Fig 12 shows the experimental results obtained from running the PageRank program. PageRank is a program that processes graphs. It used the same set of inputs as the previous experimental, Triangle Counting. In Fig 12 (a), it shows the results in the case of not applying hashing to the input data. We found that in the first testcase of the first run, the results of DTC with Java serialization, with either MD5 or SHA1 as the hash function, used 204 seconds, while the `original-checkpoint` used 214 seconds. In this comparison, DTC could speed up by 4%. For the rest of testcases, times spent by DTC is cut down to just a few seconds. In Fig 12 (b), we also found the same problem as of the Triangle Counting program. This was the result of hashing input.

Finally, we discuss the results of the Pi Estimation program. In Fig. 13, we showed tenor of comparing frameworks. For the first testcase of the first run, we found



(a) (b)

Fig. 13. Comparison of checkpoint time of RDDs using Pi Estimation Program (5 cases with JVM termination) while (a) without hashing inputs and (b) with hashing inputs.

that without hashing inputs, the `DTC-Kryo-SHA256` used 114 seconds, while the `original-checkpoint` used 135 seconds as shown in Fig 13 (a) DTC was 18% faster in this case. In the consequent testcases, DTC could cut the running time significantly.

In case of hashing inputs, we found the same trend as shown in Fig 13 (b) as the previous results. DTC used processing time almost the same as `original-checkpoint` at the first testcase then dramatically speed up by using only a few seconds for testing each testcase. Moreover, the DTC framework can be detected in case of random values, so that spark developers can reproduce the input which causes software is issues.

V. Conclusions and Future Work

The experimental results have obviously shown that DTC is suitable for improving productivity for unit testing in Big Data applications in terms of time consumption and storage usage. We can perform testing for Big Data either on a local or a cluster. DTC could trace change in testcases with random values. Unfortunately, we found that DTC could work well in case of graph algorithms such as Triangle Counting or PageRank due to spark framework cast partition of an input to *ShippableVertexPartition*. So that one of limitation the DTC is input datatype. We are researching in potential mechanisms which can be used for increasing speed of testing and reducing storage usages such as *cache* and *persist*. The JVM configurations are ones of tuning parameter we are focusing. These subjects are being studied.

References

[1] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," in *ACM SIGKDD Explorations Newsletter*, 2012, vol. 14, pp. 1–5.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 107–113, 2008.

[3] B. Mark and B. Rajkumar, "Cluster Computing: The Commodity Supercomputer," in *Software-Practice and Experience*, 1999, vol. 29(6), pp. 551–576.

[4] "Welcome to Apache™ Hadoop®!" [Online]. Available: https://hadoop.apache.org/. [Accessed: 06-May-2017].

[5] H. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: Simplified Relational Data Processing on Large Clusters," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2007, pp. 1029–1040.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the second USENIX Conference on Hot Topics in Cloud Computing*, 2010, pp. 10–10.

[7] M. A. Gulzar, M. Interlandi, T. Condie, and M. Kim, "BigDebug: Interactive Debugger for Big Data Analytics in Apache Spark," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2016, pp. 1033–1

[8] A. Spillner, T. Linz, and H. Schaefer, Software testing foundations: a study guide for the certified tester exam. Rocky Nook, Inc., 2014.

[9] "ScalaTest." [Online]. Available: http://www.scalatest.org/. [Accessed: 06-May-2017].

[10] "JUnit 5." [Online]. Available: http://junit.org/junit5/. [Accessed: 06-May-2017].

[11] "holdenk/spark-testing-base," *GitHub*. [Online]. Available: https://github.com/holdenk/spark-testing-base. [Accessed: 06-May-2017].

[12] "[SPARK-2243] Support multiple SparkContexts in the same JVM - ASF JIRA." [Online]. Available: https://issues.apache.org/jira/browse/SPARK-2243. [Accessed: 06-May-2017].

[13] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[14] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

[15] "JerryLead/SparkInternals," *GitHub*. [Online]. Available: https://github.com/JerryLead/SparkInternals. [Accessed: 07-May-2017].

[16] H. Karau and R. Warren, High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark. O'Reilly Media, Incorporated, 2017.
[17] E. Kuleshov, "Using the ASM framework to implement common Java bytecode transformation patterns," *Aspect-Oriented Software Development*, 2007.
[18] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 2–2.
[19] "A Universally Unique IDentifier (UUID) URN Namespace," *A Universally Unique IDentifier (UUID) URN Namespace*. [Online]. Available: https://www.ietf.org/rfc/rfc4122.txt. [Accessed: 07-May-2017].
[20] S. Saxena, *Getting Started with SBT for Scala*. Packt Publishing, 2013.
[21] "Home | Apache Ivy™." [Online]. Available: https://ant.apache.org/ivy/. [Accessed: 07-May-2017].
[22] "sbt/sbt-assembly," *GitHub*. [Online]. Available: https://github.com/sbt/sbt-assembly. [Accessed: 07-May-2017].
[23] "The Daily Build - write simple SBT task." [Online]. Available: http://blog.bstpierre.org/writing-simple-sbt-task. [Accessed: 07-May-2017].
[24] "bloomberg/spark-flow," *GitHub*. [Online]. Available: https://github.com/bloomberg/spark-flow. [Accessed: 08-May-2017].
[25] W. Zhu, H. Chen, and F. Hu, "ASC: Improving spark driver performance with automatic spark checkpoint," in *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, 2016, pp. 607–611.
[26] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy, "Flint: batch-interactive data-intensive processing on transient servers," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, p. 6.
[27] Y. Yan, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibroda, "TR-Spark: Transient Computing for Big Data Analytics," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, New York, NY, USA, 2016, pp. 484–496.
[28] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 2009
[29] A. M. Johansen, L. Evers, "Monte Carlo Methods", University of Bristol, Department of Mathematics

Bhuridech Sudsee received B.Eng. in Computer Engineering from Suranaree University of Technology and B.Sc. in Information Technology from Sukhothai Thammathirat Open University, both in Thailand. Currently, he is studying a Master degree in Computer Engineering. His fields of research interests are high-performance computing, distributed computing, data storage, Big Data processing and MapReduce frameworks.

Chanwit Kaewkasi received his PhD in Computer Science from the University of Manchester, United Kingdom in 2010. He is currently an Assistant Professor at School of Computer Engineering, Suranaree University of Technology, Thailand. Dr. Kaewkasi is actively researching in the areas of Low-Power Clusters, Cloud Computing, Big Data and Software Container Technologies.

Improving K Nearest Neighbor into String Vector Version for Text Categorization

Taeho Jo

School of Game, Hongik University, 2639 Sejongro Sejong South Korea 30016
tjo018@hongik.ac.kr

Abstract— This research is concerned with the string vector based version of the KNN which is the approach to the text categorization. Traditionally, texts have been encoded into numerical vectors for using the traditional version of KNN, and encoding so leads to the three main problems: huge dimensionality, sparse distribution, and poor transparency. In order to solve the problems, this research propose that texts should be encoded into string vectors the similarity measure between string vectors is defined, and the KNN is modified into the version where string vector is given its input. The proposed KNN version is validated empirically by comparing it with the traditional KNN version on the three collections: NewsPage.com , Opiniopsis, and 20NewsGroups. The goal of this research is to improve the text categorization performance by solving them.

Keyword— String Vector, K Nearest Neighbor, Text Categorization

I. INTRODUCTION

THE text categorization is defined as the process of classifying a text into its category or categories among the predefined ones. Its preliminary task is to predefine a list of categories and allocate sample texts each of them. As the learning process, using the sample labeled texts, the classification capacity which is given as equations, parameters, or symbolic rules is constructed. As the generalization process, subsequent texts which are given separately from sample labeled ones are classified by the constructed classification capacity. In this research, we assume that the supervised learning algorithms will be used as the approaches, even if other kinds of approaches are available.

Let us consider the facts which provide the motivations for doing this research. Encoding texts into numerical vectors cause problems such as the huge dimensionality and the sparse distribution [1][2][3][5][11]. Previously, we proposed the table based classification algorithm which was called the table matching algorithm, its performance was unstable by impact of noisy examples [2][3]. The computation of the similarity between two tables as the essential task in the approach was very expensive [2][3]. Therefore, in this research, we try to find the solution to the problems by encoding texts into alternatives to both numerical vectors and tables.

What we propose in this research is to encode texts into string vectors and to modify the KNN as solutions to the above problems. Texts are encoded into string vectors as their structured forms, instead of numerical vectors. The semantic similarity measure between two string vectors is defined as the operation which corresponds to the cosine similarity between two numerical vectors. Using the similarity measure, we modify the KNN (K Nearest Neighbor) into the version where a string vector is given as an input vector. The modified version is applied as the approach to the task of classifying news articles and opinions automatically.

In this research, we will validate empirically the proposed approach to the text summarization as the better version than the traditional KNN version. We extract paragraphs from the collections of news articles: NewsPage.com, Opinopsis, and 20NewsGroups. The traditional KNN version and the proposed version are compared with each other. We observe the better results of the proposed KNN version in classifying news articles into their own topics. It potentially possible to require less dimension in encoding texts into string vectors, in addition.

This article is organized into the four sections. In Section II, we survey the relevant previous works. In Section III, we describe in detail what we propose in this research. In Section IV, we validate empirically what is proposed in this research. In Section V, we mention the remaining tasks for doing the further research.

II. PREVIOUS WORKS

Let us survey the previous cases of encoding texts into structured forms for using the machine learning algorithms to text mining tasks. The three main problems, huge dimensionality, sparse distribution, and poor transparency, have existed inherently in encoding them into numerical vectors. In previous works, various schemes of preprocessing texts have been proposed, in order to solve the problems. In this survey, we focus on the process of encoding texts into alternative structured forms to numerical vectors. In other words, this section is intended to explore previous works on solutions to the problems.

Let us mention the popularity of encoding texts into numerical vectors, and the proposal and the application of string kernels as the solution to the above problems. In 2002, Sebastiani presented the numerical vectors are the standard

Manuscript received December 27, 2017. This work is sponsored by 2017 Hongik University Research Fund, and a follow-up of the invited journal to the accepted & presented paper of the 19th International Conference on Advanced Communication Technology (ICACT2017).

Taeho Jo is with School of Game, Hongik University, Sejong, Republic of Korea (phone: +82-44-860-2125; e-mail: tjo018@hongik.ac.kr).

representations of texts in applying the machine learning algorithms to the text classifications [6]. In 2002, Lodhi et al. proposed the string kernel as a kernel function of raw texts in using the SVM (Support Vector Machine) to the text classification [7]. In 2004, Lesile et al. used the version of SVM which proposed by Lodhi et al. to the protein classification [8]. In 2004, Kate and Mooney used also the SVM version for classifying sentences by their meanings [9].

It was proposed that texts are encoded into tables instead of numerical vectors, as the solutions to the above problems. In 2008, Jo and Cho proposed the table matching algorithm as the approach to text classification [2]. In 2008, Jo applied also his proposed approach to the text clustering, as well as the text categorization [13]. In 2011, Jo described as the technique of automatic text classification in his patent document [11]. In 2015, Jo improved the table matching algorithm into its more stable version [12].

Previously, it was proposed that texts should be encoded into string vectors as other structured forms. In 2008, Jo modified the k means algorithm into the version which processes string vectors as the approach to the text clustering [13]. In 2010, Jo modified the two supervised learning algorithms, the KNN and the SVM, into the version as the improved approaches to the text classification [14]. In 2010, Jo proposed the unsupervised neural networks, called Neural Text Self Organizer, which receives the string vector as its input data [15]. In 2010, Jo applied the supervised neural networks, called Neural Text Categorizer, which gets a string vector as its input, as the approach to the text classification [16].

The above previous works proposed the string kernel as the kernel function of raw texts in the SVM, and tables and string vectors as representations of texts, in order to solve the problems. Because the string kernel takes very much computation time for computing their values, it was used for processing short strings or sentences rather than texts. In the previous works on encoding texts into tables, only table matching algorithm was proposed; there is no attempt to modify the machine algorithms into their table based version. In the previous works on encoding texts into string vectors, only frequency was considered for defining features of string vectors. In this research, based on [14], we consider the grammatical and posting relations between words and texts as well as the frequencies for defining the features of string vectors, and encode texts into string vectors in this research.

III. Proposed Approach

This section is concerned with encoding words into string vectors, modifying the KNN (K Nearest Neighbor) into the string vector based version and applying it to the text categorization, and consists of the three sections. In Section III-A, we deal with the process of encoding texts into string vectors. In Section III-B, we describe formally the similarity matrix and the semantic operation on string vectors. In Section III-C, we do the string vector based KNN version as the approach to the text categorization. Therefore, this section is intended to describe the proposed KNN version as the text categorization tool.

A. Text Encoding

This section is concerned with the process of encoding texts into string vectors. As shown in Figure 1, the three steps are involved in encoding texts. A single is given as the input and a string vector which consists of words is generated as the output. The features in each string vector are posting, statistic, and grammatical relationships between a text and a word. Therefore, this section is intended to describe in detail each step involved in encoding texts.

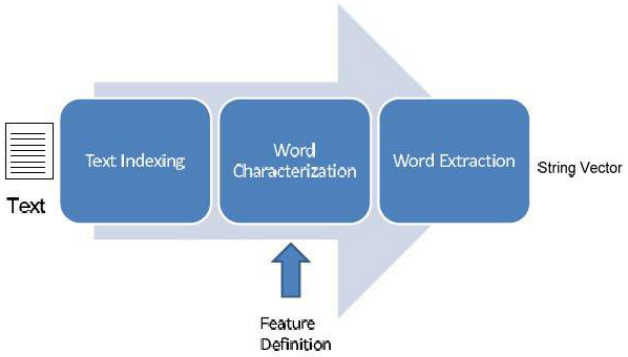


Fig. 1. The Process of Text Encoding.

The first step of encoding texts into string vectors is to index the corpus into a list of words. The texts in the corpus are concatenated into a single long string and it is tokenized into a list of tokens. Each token is transformed into its root form, using stemming rules. Among them, the stop words which are grammatical words such as propositions, conjunctions, and pronouns, irrelevant to text contents are removed for more efficiency. From the step, verbs, nouns, and adjectives are usually generated as the output.

We need to define the relationships between a word and a text as the features of string vectors, and mention the three types of them. The first type is statistical properties of words in a text such as the highest frequent word and the highest TF-IDF (Term Frequency-Inverse Term Frequency) weighted one. The grammatical properties of a word such as subjective noun, objective noun, and verb may be considered as another feature type. Posting properties of a word which indicates its position in the given text, such as the first word in the text, the last of in the text, and the first word in the last paragraph, may be regarded as a feature type. In this research, we define the ten features of string vectors as follows:

- Highest Frequent Word in the given Text
- Second Highest Frequent Word in the given Text
- Third Highest Frequent Word in the given Text
- Highest TF-IDF Weighted Word
- Second Highest TF-IDF Weighted Word
- Third Highest TF-IDF Weighted Word
- The Last Word in the Text
- The First Word in the last Paragraph
- The Last Word in the First Paragraph

Let us explain the process of encoding a text into a string, once the above features are defined. A text is indexed into a list of words, their frequencies, and their TF-IDF weights, and it is partitioned into a list of paragraphs. Corresponding to the above features, words are extracted as elements of the string vector. As the given text representation, the ten dimensional string vector which consists of the above feature values is

constructed. The similarity matrix is required for performing the operation on string vectors, and is described in Section III-B1.

Let us consider the differences between the word encoding and the text encoding. Elements of each string vector which represents a word are text identifiers, whereas those of one which represents a text are word. The process of encoding texts involves the link of each text to a list of words, where as that of doing words does the link of each word to a list of texts. For performing semantic similarity between string vectors, in text processing, the word similarity matrix is used as the basis, while in word processing, the text similarity matrix is used. The relations between words and texts are defined as features of strings in encoding texts and words.

B. String Vector

This section is concerned with the operation on string vectors and the basis for carrying out it. It consists of two subsections and assumes that a corpus is required for performing the operation. In Section III-B1, we describe the process of constructing the similarity matrix from a corpus. In Section III-B2, we define the string vector formally and characterize the operation mathematically. Therefore, this section is intended to describe the similarity matrix and the operation on string vectors.

Similarity Matrix

This subsection is concerned with the similarity matrix as the basis for performing the semantic operation on string vectors. Each row and column of the similarity matrix corresponds to a word in the corpus. The similarities of all possible pairs of words are given as normalized values between zero and one. The similarity matrix which we construct from the corpus is the N X N square matrix with symmetry elements and 1's diagonal elements. In this subsection, we will describe formally the definition and characterization of the similarity matrix.

Each entry of the similarity matrix indicates a similarity between two corresponding words. The two words, t_i, and t_j are viewed into two sets of texts which include them, T_i, and T_j. The similarity between the two words is computed by equation (1),

$$sim(t_i, t_j) = \frac{2|T_i \cap T_j|}{|T_i| + |T_j|} \tag{1}$$

where $|T_i|$ is the cardinality of the set, T_i. The similarity is always given as a normalized value between zero and one; if two words are exactly same to each other, the similarity becomes 1.0 as equation (2):

$$sim(t_i, t_i) = \frac{2|T_i \cap T_i|}{|T_i| + |T_j|} = 1.0 \tag{2}$$

and if two words have no shared texts, $T_i \cap T_j = \varnothing$, the similarity becomes 0.0 as equation (3):

$$sim(t_i, t_i) = \frac{2|T_i \cap T_i|}{|T_i| + |T_j|} = \frac{2 \times 0}{|T_i| + |T_j|} = 0.0 \tag{3}$$

The more advanced schemes of computing the similarity will be considered in next research.

From the text collection, we build N X N square matrix as follows:

$$\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & \cdots & s_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ s_{N1} & s_{N2} & \cdots & s_{NN} \end{bmatrix}$$

N individual words which are contained in the collection correspond to the rows and columns of the matrix. The entry, s_{ij} is computed by equation (1) as equation (4):

$$s_{ij} = sim(t_i, t_j) \tag{4}$$

The overestimation or underestimation by text lengths are prevented by the denominator in equation (1). To the number of words, N, it costs quadratic complexity, $O(N^2)$, to build the above matrix

Let us characterize the above similarity matrix, mathematically. Because each column and row corresponds to its same text in the diagonal positions of the matrix, the diagonal elements are always given 1.0 by equation (2). In the off-diagonal positions of the matrix, the values are always given as normalized ones between zero and one, because of $0 \le 2|T_i \cap T_i| \le |T_i| + |T_j|$ from equation (1). It is proved that the similarity matrix is symmetry, as equation (5):

$$\begin{aligned} s_{ij} &= sim(t_i, t_j) = \frac{2|T_i \cap T_j|}{|T_j| + |T_j|} = \frac{2|T_j \cap T_i|}{|T_j| + |T_i|} \\ &= sim(t_j, t_i) = s_{ji} \end{aligned} \tag{5}$$

Therefore, the matrix is characterized as the symmetry matrix which consists of the normalized values between zero and one.

The similarity matrix may be constructed automatically from a corpus. The N texts which are contained in the corpus are given as the input and each of them is indexed into a list of words. All possible pairs of words are generated and the similarities among them are computed by equation (1). By computing them, we construct the square matrix which consists of the similarities. Once making the similarity matrix, it will be used continually as the basis for performing the operation on string vectors.

String Vector and Semantic Similarity

This section is concerned with the string vectors and the operation on them. A string vector consists of strings as its elements, instead of numerical values. The operation on string vectors which we define in this subsection corresponds to the cosine similarity between numerical vectors. Afterward, we characterize the operation mathematically. Therefore, in this section, we define formally the semantic similarity as the semantic operation on string vectors.

The string vector is defined as a finite ordered set of strings as equation (6):

$$\mathbf{str} = [str_1, str_2, ..., str_d] \tag{6}$$

An element in the vector, str_i indicates a word which corresponds to its attribute. The number of elements of the string vector, **str**, is called its dimension. In order to perform

the operation on string vectors, we need to define the similarity matrix which was described in section 2.1, in advance. Therefore, a string vector consists of strings, while a numerical vector does of numerical values.

We need to define the semantic operation which is called 'semantic similarity' in this research, on string vectors; it corresponds to the cosine similarity on numerical vectors. We note the two string vectors as equation:

$$\mathbf{str}_1 = [t_{11}, t_{12}, ..., t_{1d}] \text{ and } \mathbf{str}_2 = [t_{21}, t_{22}, ..., t_{2d}]$$

where each element, t_{1i} or t_{12}, indicates a word. The operation is defined as equation (7) as follows:

$$sim(\mathbf{str}_1, \mathbf{str}_2) = \frac{1}{d}\sum_{i=1}^{d} sim(t_{1i}, t_{2i}) \quad (7)$$

The similarity matrix was constructed by the scheme which is described in section 2.1, and the $sim(t_{1i}, t_{2i})$ is computed by looking up it in the similarity matrix. Instead of building the similarity matrix, we may compute the similarity, interactively.

The semantic similarity measure between string vectors may be characterized mathematically. The commutative law applies as equation (8):

$$sim(\mathrm{str}_1, \mathrm{str}_2) = \frac{1}{d}\sum_{i=1}^{d} sim(t_{1i}, t_{2i}) = \frac{1}{d}\sum_{i=1}^{d} sim(t_{2i}, t_{1i}) = sim(\mathrm{str}_2, \mathrm{str}_1) \quad (8)$$

If the two string vectors are exactly same, its similarity becomes 1.0 as follows:

If $\mathbf{str}_1 = \mathbf{str}_2$ with $\forall_i, sim(t_{1i}, t_{2i}) = 1.0$,

$$sim(\mathbf{str}_1, \mathbf{str}_2) = \frac{1}{d}\sum_{i=1}^{d} sim(t_{1i}, t_{1i}) = \frac{d}{d} = 1.0$$

However, note that the transitive rule does not apply as follows:

If $sim(\mathbf{str}_1, \mathbf{str}_2) = 0.0$ and $sim(\mathbf{str}_2, \mathbf{str}_3) = 0.0$, not always $sim(\mathbf{str}_1, \mathbf{str}_3) = 0.0$

We need to define the more advanced semantic operations on string vectors for modifying other machine learning algorithms. We define the update rules of weights vectors which are given as string vectors for modifying the neural networks into their string vector based versions. We develop the operations which correspond to computing mean vectors over numerical vectors, for modifying the k means algorithms. We consider the scheme of selecting representative vector among string vectors for modifying the k medoid algorithms so. We will cover the modification of other machine learning algorithms in subsequent researches.

C. The Proposed Version of KNN

This section is concerned with the proposed KNN version as the approach to the text categorization. Raw texts are encoded into string vectors by the process which was described in Section III-A. In this section, we attempt to the traditional KNN into the version where a string vector is given as the input data. The version is intended to improve the classification performance by avoiding problems from encoding texts into numerical vectors. Therefore, in this section, we describe the proposed KNN version in detail, together with the traditional version.

The traditional KNN version is illustrated in Figure 2. The sample words which are labeled with the positive class or the negative class are encoded into numerical vectors. The similarities of the numerical vector which represents a novice word with those representing sample words are computed using the Euclidean distance or the cosine similarity. The k most similar sample words are selected as the k nearest neighbors and the label of the novice entity is decided by voting their labels. However, note that the traditional KNN version is very fragile in computing the similarity between very sparse numerical vectors.

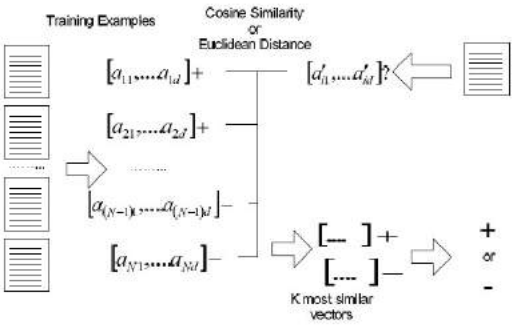


Fig. 2. The Traditional Version of KNN.

Separately from the traditional one, we illustrate the classification process by the proposed version in Figure 3. The sample texts labeled with the positive or negative class are encoded into string vectors by the process described in Section III-A. The similarity between two string vectors is computed by the scheme which was described in Section III-B2. Identically to the traditional version, in the proposed version, the k most similarity samples are selected, and the label of the novice one is decided by voting ones of sample entities. Because the sparse distribution in each string vector is never available inherently, the poor discriminations by sparse distribution are certainly overcome in this research.

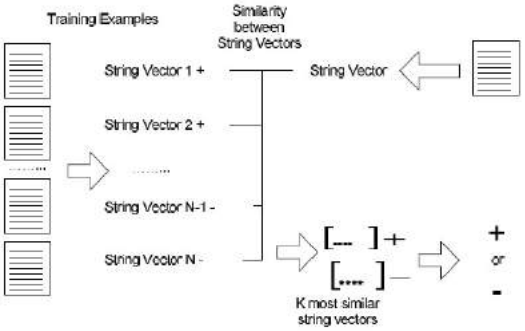


Fig. 3. The Proposed Version of KNN.

We may derive some variants from the proposed KNN version. We may assign different weights to selected neighbors instead of identical ones: the highest weights to the first nearest neighbor and the lowest weight to the last one. Instead of a fixed number of nearest neighbors, we select any number of training examples within a hyper-sphere whose center is the given novice example as neighbors. The categorical scores are computed proportionally to similarities with training examples, instead of selecting nearest neighbors. We may also consider the variants where more than two variants are combined with each other.

Because string vectors are characterized more symbolically

than numerical vectors, it is easy to trace results from classifying items in the proposed version. It is assumed that a novice item is classified by voting the labels of its nearest neighbors. The similarity between string vectors is computed by the scheme which is described in Section III-B2. We may extract the similarities of individual elements of the novice string vector with those of nearest neighbors labeled with the classified category. Therefore, the semantic similarities play role of the evidence for presenting the reasons of classifying the novice one so.

IV. Experimental Results

This section is concerned with the empirical experiments for validating the proposed version of KNN, and consists of the four sections. In Section I, we present the results from applying the proposed version of KNN to the text categorization on the collection, NewsPage.com. In Section II, we show the results from applying it for categorizing texts from the collection, Opinosis. In Section III, we mention the results from comparing the two versions of KNN with each other in categorizing texts from 20NewsGroups. In Section IV, we make the general discussions which is concerned with results from validating the proposed version of KNN, finally.

A. NewsPage.com

This section is concerned with the experiments for validating the better performance of the proposed version on the collection: NewsPage.com. The four categories are predefined in this collection, and texts are gathered from the collection category by category as labeled ones. Each text is classified exclusively into one of the four categories. In this set of experiments, we apply the traditional and proposed version of KNN to the classification task, without decomposing it into the binary classifications, and use the accuracy as the evaluation measure. Therefore, in this section, we observe the performance of the both versions of KNN by changing the input size.

In Table I, we specify the text collection, NewsPage.com, which is used in this set of experiments. This text collection was used for evaluating approaches to text categorization in previous works [1][3][13]. In the collection, the four categories are predefined: Business, Health, Internet, and Sports, and 375 texts are selected at random in each category. In each category, the set of 375 texts is partitioned into the 300 texts as training ones and the 75 texts as test ones. The text collection was built by copying and pasting individual news articles from the web site, newspage.com, in 2005, as plain text files whose extension is 'txt'.

TABLE I
The Number of Texts in NewsPage.com

Category	#Texts	#Training Texts	#Test Texts
Business	500	300	75
Health	500	300	75
Internet	500	300	75
Sports	500	300	75
Total	2000	1200	300

Let us mention the experimental process for validating empirically the proposed approach to the task of text categorization. In this collection, the texts are labeled with one of the four categories which are presented in Table I, and they are encoded into numerical and string vectors. For each test example, the KNN computes its similarities with the 1200 training examples and selects the three most similarity training examples as its nearest neighbors. Each of the 300 test examples is classified into one of the four categories: Business, Sports, Internet, and Health, by voting the labels of its nearest neighbors. We compute the classification accuracy by dividing the number of correctly classified test examples by the number of test examples, for evaluating the both versions of KNN algorithm.

In Figure 4, we illustrate the experimental results from categorizing texts, using the both versions of KNN algorithm. The y-axis indicates the accuracy which is the rate of the correctly classified examples in the test set. In the x-axis, each group indicates the input size which is the dimension of numerical vectors which represent texts. In each group, the gray bar and the black bar indicate the achievements of the traditional version and the proposed version of KNN algorithm, respectively. In the x-axis, the most right group indicates the average over the accuracies of the left groups.

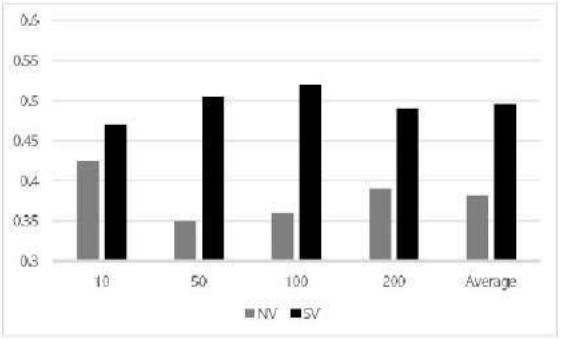


Fig. 4. Results from Classifying Texts in Text Collection: NewsPage.com

Let us make the discussions on the results from doing the text categorization using the both versions of KNN algorithm, as shown in Figure 4. The accuracy which is the performance measure of the classification task is in the range between 0.35 and 0.52. The proposed version of KNN algorithm works strongly better in the all input sizes. The performance difference between the two versions is outstanding in the two input sizes, 50 and 100. From this set of experiments, we conclude that the proposed version works strongly better than the traditional one, in averaging over the four cases.

B. OPINOPSIS

This section is concerned with the set of experiments for validating the better performance of the proposed version on the collection, Opinopsis. The three categories are predefined in the collection, and labeled texts are prepared from it. Each text is classified exclusively into one of the three categories. We do not decompose the given classification into binary classifications and use the accuracy as the evaluation measure. Therefore, in this section, we observe the performances of the both versions of KNN algorithm with the different input sizes.

In Table II, we specify the text collection, Opinosis, which is used in this set of experiments. The collection was used in previous works for evaluating approaches to text categorization. The three categories, 'Car', 'Electronics', and 'Hotel', are predefined, and all texts are used for evaluating the approaches to text categorization, in this set of experiments. We use six texts in each category among all texts as the test set as shown in Table II. We obtained the collection

by downloading it from the web site, http://archive.ics.uci.edu/ml/machine-learningdatabases/ opinion/.

TABLE II
The Number of Texts in Opiniopsis

Category	#Texts	#Training Texts	#Test Texts
Car	23	17	6
Electronic	16	10	6
Hotel	12	6	6
Total	51	33	18

We perform this set of experiments by the process which is described in Section I. We use all of 51 texts which are labeled with one of the three categories and encode them into numerical vectors and string vectors with the input sizes: 10, 50, 100, and 200. For each test example, the both versions of KNN computes its similarities with the 33 training examples and select the three most similar training examples as its nearest neighbors. Each of the 18 test examples is classified into one of the three categories, by voting the labels of its nearest neighbors. The classification accuracy is computed by the number of correctly classified test examples by the number of the test examples for evaluating the both versions of KNN algorithm.

In Figure 5, we illustrate the experimental results from categorizing texts using the both versions of KNN algorithm. Like Figure 4, the y-axis indicates the value of accuracy, and the x-axis indicates the group of both versions by an input size. In each group, the gray bar and the black bar indicate the achievements of the traditional version and the proposed version of KNN algorithm, respectively. In Figure 5, the most right group indicates the averages over results over the left four groups. Therefore, Figure 5 presents the results from classifying each text into one of the three categories by the both versions, on the text collection, Opinosis.

We discuss the results from doing the text categorization using the both versions of KNN algorithm, on Opinosis, shown in Figure 5. The accuracy values of the bother versions range between 0.55 and 1.0. The proposed version works better than the traditional one in the all input sizes. It shows the perfect results in the input size: 200. From this set of experiments, we conclude that the proposed version works outstandingly better than the traditional one, in averaging the four cases.

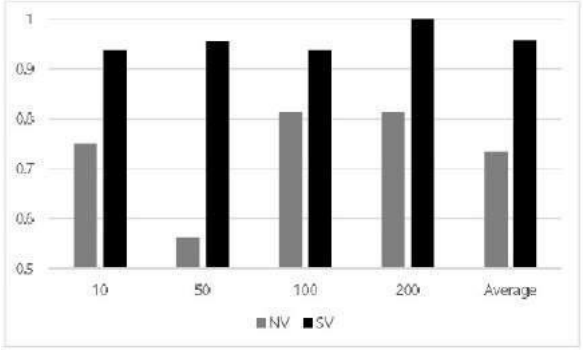


Fig. 5. Results from Classifying Texts in Text Collection: Opiniopsis

C.20NewsGroups

This section is concerned with one more set of experiments where the better performance of the proposed version is validated on 20NewsGroups. In this set of experiments, the four specific categories are predefined in this collection. Each text is exclusively classified into one of the four categories, like the previous sets of experiments. We apply the two versions of KNN algorithm, directly to the classification task, without decomposing it into binary classifications, and use the accuracy as the evaluation metric. Therefore, in this section, we observe the performances of the both versions of KNN algorithm with the different input sizes.

In Table III, we specify the specific version of 20NewsGroups which is used as the test collection, in this set of experiments. Within the general category, sci, we predefine the four categories: 'electro', 'medicine', 'script', and 'space'. In each category, we select 375 texts among approximately 1000 texts, at random. In each category, the set of 375 texts is partitioned into the training set of 300 texts and the test set of 75 texts, like the case in the previous set of experiments.

TABLE III
The Number of Texts in Opiniopsis

Category	#Texts	#Training Texts	#Test Texts
Electro	1000	300	75
Medicine	1000	300	75
Script	1000	300	75
Space	1000	300	75
Total	4000	1200	300

The process of doing this set of experiments is same to that in the previous sets of experiments. We select the balanced number of texts from the collection over categories, and encode them into the representations with the input sizes which are identical to those in the previous set of experiments. We use the two versions of KNN algorithm for their comparisons. Using the two versions of KNN algorithm, we classify each text in the test set into one of the four specific categories within the general category, ' sci': 'electro', 'medicine', 'script', and 'space'. We use the accuracy as the evaluation metric, like the previous set of experiments.

We present the experimental results from classifying the texts using the both versions of KNN algorithm on the specific version of 20NewsGroups. The frame of illustrating the classification results is identical to the previous ones. In each group, the gray bar and the black bar stand for the achievements of the traditional version and the proposed version, respectively. The y-axis in Figure 6, indicates the classification accuracy which is used as the performance metric. The texts are classified directly to one of the four categories like the cases in the previous sets of experiments.

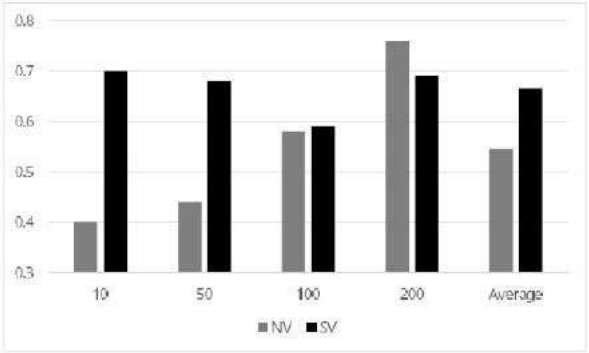


Fig. 6. Results from Classifying Texts in Text Collection: 20NewsGroups

Let us discuss on the results from classifying the texts on the specific version of 20NewsGroups, as shown in Figure 6.

The accuracies of the both versions range between 0.4 and 0.91. The proposed version shows its better performance in the smaller input sizes, but its turning point in the input size, 100. In the traditional version, its performance is proportional to the input size, whereas in the proposed version, its performance is independent of the factor. By the way, from this set of experiments, it is concluded that the proposed version have its outstandingly better performance, by averaging over the accuracies of the four input sizes.

V. Conclusion

Let us discuss the entire results from classifying texts using the two versions of KNN algorithm. The both versions is compared with each other in the task of text categorization, in these sets of experiments. The proposed version show its better results in all of the three collections. The accuracies of the traditional version range between 0.35 and 0.81, while those of the proposed version range between 0.49 and 1.0. From the three sets of experiments, we conclude that the proposed version improves the text categorization performance, as the contribution of this research.

We need to consider the remaining tasks for doing the further research. We will apply and validate the proposed approach in classifying texts in the specific domains such as medicine, engineering, and law, rather than the general domains. In order to improve the performance, we may consider various types of features of string vectors. As another scheme of improving the performance, we define and combine multiple similarity measures between two string vectors with each other. By adopting the proposed approach, we may implement the text categorization system as a module or an independent system.

Acknowledgment

This work was supported by 2017 Hongik University Research Fund.

References

[1] T. Jo, The Implementation of Dynamic Document Organization using Text Categorization and Text Clustering. PhD Dissertation of University of Ottawa, 2006.
[2] T. Jo and D. Cho, “Index Based Approach for Text Categorization”, International Journal of Mathematics and Computers in Simulation, vol. 2, pp. 127-132, 2008.
[3] T. Jo, “Table based Matching Algorithm for Soft Categorization of News Articles in Reuter 21578”, Journal of Korea Multimedia Society, vol. 11, pp. 875-882, 2008.
[4] T. Jo, “`Topic Spotting to News Articles in 20NewsGroups with NTC”, Lecture Notes in Information Technology, pp50-56, vol. 7, 2011.
[5] T. Jo, “Definition of String Vector based Operations for Training NTSO using Inverted Index”, Lecture Notes in Information Technology, pp50-56, vol. 7, 2011.
[6] F. Sebastiani, ``Machine Learning in Automated Text Categorization", ACM Computing Survey, vol. 34, pp. 1-47, 2002.
[7] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, “Text Classification with String Kernels”, Journal of Machine Learning Research, vol. 2, pp. 419-444, 2002.
[8] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble, “Mismatch String Kernels for Discriminative Protein Classification”, Bioinformatics, vol. 20, pp. 467-476, 2004.
[9] R. J. Kate and R. J. Mooney, “Using String Kernels for Learning Semantic Parsers”, Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pp. 913-920, 2006.
[10] S. Chen, B. Mulgrew, and P. M. Grant, “A clustering technique for digital communications channel equalization using radial basis function networks,” IEEE Trans. Neural Networks, vol. 4, pp. 570–578, Jul. 1993.
[11] T. Jo, “Single Pass Algorithm for Text Clustering by Encoding Documents into Tables”, Journal of Korea Multimedia Society, vol. 11, pp. 1749-1757, 2008.
[12] T. Jo, “Device and Method for Categorizing Electronic Document Automatically”, Patent Document, 10-2009-0041272, 10-1071495, 2011.
[13] T. Jo, “Normalized Table Matching Algorithm as Approach to Text Categorization”, Soft Computing, vol. 19, pp. 839-849, 2015.
[14] T. Jo, “Inverted Index based Modified Version of K-Means Algorithm for Text Clustering”, Journal of Information Processing Systems, Vol 4, pp. 67-76, 2008.
[15] T. Jo, “Representation of Texts into String Vectors for Text Categorization”, Journal of Computing Science and Engineering, vol. 4, pp. 110-127, 2010.
[16] T. Jo, “NTSO (Neural Text Self Organizer): A New Neural Network for Text Clustering”, Journal of Network Technology, vol. 1, pp. 31-43, 2010.
[17] T. Jo, “NTC (Neural Text Categorizer): Neural Network for Text Categorization”, International Journal of Information Studies, vol. 2, pp.83-96, 2010.

Taeho Jo works currently as a faculty member in Hongik University, South Korea. He received his Bachelor degree from Korea University in 1994, his Master degree from Pohang University of Science and Technology in 1997, and his PhD degree from University of Ottawa in 2006. His research area spans mainly over text mining, neural networks, machine learning, and information retrieval. He has the four year experience of working for industrial organizations and ten year experience of working for academic ones. Recently, he is awarded in the world wide biography dictionary, Marquis Who’s Who in the World, two times in 2016 and 2018.

Volume 7 Issue 1, January. 2018, ISSN: 2288-0003

ICACT-TACT JOURNAL

GIRI

Global IT Research Institute

1713 Obelisk, 216 Seohyunno, Bundang-gu, Sungnam Kyunggi-do, Republic of Korea 13591
Business Licence Number : 220-82-07506, Contact: tact@icact.org Tel: +82-70-4146-4991