

Performance Evaluation of Apache Storm With Writing Scripts

Lemi Isaac Yoseke Laku, Alaelddin Fuad Yousif Mohammed, Fawaz AL-Hazemi, Chan-Hyun Youn

Korea Advanced Institute of Science and Technology (KAIST)

291 Daehak-ro, Eoeun-dong, Yuseong-gu, Daejeon, 305-732, South Korea.

Email: {lemi, alaelddin, fawaz, chyoun}@kaist.ac.kr

Abstract—With the exponential growth of stream data emanating from a variety of sources, today, big data presents a new era in data exploration and usage. The understanding of the performance of real-time stream data processing technologies has become a key pre-requisite while considering any deployment. Although many technologies for stream data analytics exist, little is known about the impact of writing scripts on their performance. Using a MapReduce programming model on a pseudo cluster, we conduct a word count, experimental evaluation of Apache Storm using five writing scripts; English, Arabic, Hindi, Chinese and Japanese. We define our word count as the number of time a word is written in a body of text. Static and structured data made up of 300 English sentences is translated into the scripts under study and loaded into Apache Storm. The results show that Apache Storm analyzes, English, Arabic and Hindi script sentences with ease as compared to those written in Chinese and Japanese script. Apache Storm also executes and distinguishes individual words and performs a word count on English, Arabic and Hindi sentences faster than those written in Chinese and Japanese. However, in terms of processing, the reverse is true, sentences written in Chinese and Japanese are processed faster than those written in English, Arabic and Hindi scripts.

Index Terms—Stream data, Real-time, Word count, Writing script.

I. INTRODUCTION

In recent years, daily governmental proceedings have produced and availed more data than there has ever been and it keeps increasing. For instance, the European Parliament releases, new data in 22 languages on a regular basis [1]. The paradigm shift in streaming data drew an extra consideration, predominantly its critical influences on a variety of use cases, including but not limited to real-time government data analysis, log processing, and other metrics analytics. Big data analytics is an approach to gathering, sorting and analyzing bulky sets of data to explore useful insights. To serve the thriving demands of processing stream data or data-in-motion, many computation technologies such as Apache Storm, Flink Spark, and Samza among others have sprung up [2].

Stream processing is the application of continuous queries on a real-time data to get an insight that may necessitate actions. Streaming Analytics links; business systems to outside data sources, allowing the applications to combine certain data into the information flow, or updating a peripheral database with processed intelligence [3].

As far as we know, various studies have been conducted on the performance of the stream data analytic technologies, however, limited devotion has been given to the impact of writing scripts on the performance of these technologies.

In order to help users understand, we seek to answers the questions below:

- 1) What are the effects of a writing script on the performance Apache Storm?
- 2) Does Apache Storm maintain the same level of performance irrespective of the writing script?

We focus on five scripts each representing a writing system. A writing script, also known as a writing system or orthography is any conventional method of visually representing verbal communication. Writing is the use of scripts to combining character set for human communication. It involves the use of visible marks or characters that are related to a particular structural language.

A. Motivation

As mentioned earlier, as of today, many governments produce data in various languages, and the numerous previous works that exist on the stream data analytics only compared the performance of analytic technologies. However, some detected limitations of the previous works presented are as follows:

All of the studies were limited to the key features such as memory consumption, latency, window time, throughput among others.

In addition, the choice of framework for different scenarios is an important factor to consider while planning for any deployment as individual platform is distinctive to the type of data sets, and the message processing semantics, etc.

As far as we know, there has been no earlier attempt to examine the performance of Apache Storm with writing scripts. Therefore, this study is carried out in order to help users in choosing the most appropriate writing system to perform their real-time stream data analytics needs.

B. Contribution

The manifold contributions of this work are as follows:

- 1) The primary contribution is to offer a better understanding of the analytic technologies when deployed for the purpose of processing data written in a particular writing system. Using the MapReduce programming model on

a pseudo cluster, the performance of Apache Storm with different writing scripts or systems is tested. We experimentally test the performance of Apache Storm in terms of the executor's time metrics. We measure:

- a) The number of tuples emitted;
- b) The execution latency and;
- c) The processing latency.

- 2) The lesson can be helpful in stream data analysis scenarios involving content written in native languages.

The organization of this paper is as follows: Section II presents related works; Section III introduces an overview of Apache Storm; Section IV presents the experiment; Section V presents the results and discussion, and we conclude the paper on Section VI and we draw our future work.

II. RELATED WORK

A. Performance Evaluation

Stream data analysis is a new domain in the distributed systems. Various stream processing frameworks and technologies are explored in literature [2], [4]–[9] and many more. Chintapalli et. al., benchmarked the streaming computation framework: including Apache Flink, Spark and Storm. Using various configurations, they compared the performance in terms of throughput and latency [2]. Inoubi et. al. compared stream processing frameworks for big data including Apache Flink, Spark, Storm, and Samza. However, the study was limited to: window time; central processing unit (CPU) consumption; random access memory (RAM) consumption; disk read and write (R/W) consumption; and bandwidth resource usage [4]. Garcia-Gil et. al. compared the scalability of Apache Flink and Spark on batch processing [5]. Lopez, Antoni & Duarte compared the performance of Spark streaming and Apache Flink [6]. Bartolini & Patella compared the performance of Apache Flink, Spark, and Storm on real analysis of multimedia streams (RAMS) framework [7]. Karmov et. al., benchmarked Distributed Stream Processing Engines focusing on major performance indicators such as latency and throughput [8]. Chatterjee & Morin studied and compared the performance and resource utilization of Flink, Storm, and Heron [9]. All the previous studies did not consider the effect of writing systems on the functioning of the stream analytic technologies. This study, therefore, focuses on evaluating the performance of Apache Storm with different writing scripts. Apache Storm is one of the popular and widely-used stream data processing frameworks compatible with many writing systems. It is essential to study and comprehensively match the impact of writing systems (Arabic, Chinese, English, Japanese and Hindi) on the performance of the Apache Storm framework. This will precisely provide an insight for users to wisely choose the correct writing system for stream data that is intended for an Apache Storm platform.

B. Writing Scripts and Word Count

Writing scripts or system consist of various families including alphabets (e.g. English), syllabaries (e.g. Japanese),

logo syllabaries (e.g. Chinese) and others such as Abjad (e.g. Arabic) and those that have the characteristics of both alphabets and syllabaries (e.g. Hindi) [10]. Depending on the encoding and the writing style, words in a sentence can either be difficult or easy to distinguish. For instance, Arabic script is cursive with neighboring letters joined together, has interword space and a word boundary. Arabic, which is an alphabetical language is made up of 28 characters. It has punctuation marks equivalent to those in English. However, the characters have disparities in shapes and forms. An Arabic character has up to four different shapes. The position of a character in the word (initial, medial, final, and alone). Typically, in an Arabic code setting, every letter is denoted by a unique single code which is independent of shapes [11]. This makes it easy to identify the proper shape of a letter basing on its location in a word. For Hindi, a sentence ends with a *purna viram* (!) which identifies the boundary [12]. Hindi has alphabets that are categorized on the basis of similarities of letters. This property makes it free of homonyms, thus obviating the complexity of handling the word recognition [13].

Unlike English and other languages (e.g. Arabic and Hindi etc.), Asian languages like Chinese and Japanese, lack white-space between the words [14]. The Western style of a word boundary is factually unfamiliar to Chinese and Japanese scripts. Chinese and Japanese writings do not use interword spaces or other word delimiters (except for punctuation marks). The absence of word delimiters causes ambiguities in a word segmentation [15]. Since Chinese and Japanese sentences do not separate words by explicit delimiters, segmenting a sentence into individual words becomes a difficult task [16].

Vysokos, reported that the hieroglyphic nature and lack of word separation space make it difficult to count words written in Chinese and Japanese scripts. In case, the word count system is based on the spaces between words, a Chinese sentence will be counted as a single word [17]. Yao et al. explained syntactic and morphological variances concerning the features of alphabetic languages and the Chinese. In terms of morphological organizations, Chinese lacks a clear word boundary which alphabetic writing systems have. Whereas, in alphabetic languages, letters form the basic components of words, by contrast, in Chinese writings, strokes made-up of monosyllabic or disyllabic characters form a word. The words are often multi-syllabic. This morphological and syntactic variances make tools intended for alphabetic textual analysis incompatible and cannot be applied to Chinese texts [18]. Imral, asserted that large amounts of Chinese text can be processed in a shorter time and with relatively low memory usage [19].

III. APACHE STORM

In this Section, we introduce a brief background and architecture of Apache Storm.

Apache Storm is a distributed open-source framework for real-time data processing [20]. It does a real-time processing of stream data (data-in-motion) just as Hadoop does batch processing for data-at-rest [21]. Apache Storm is designed to

accept tons of data coming in extremely fast, possibly from various sources, analyze it, and publish real-time updates to a user interface, without storing any actual data [22].

Apparently, an Apache Storm cluster is comparable to a Hadoop cluster. The notable difference is that Hadoop executes MapReduce tasks, and Storm, uses topologies to execute jobs. Another variation is that a MapReduce job finishes eventually. On the other hand a topology will continuously run and execute messages until the user kills it [23].

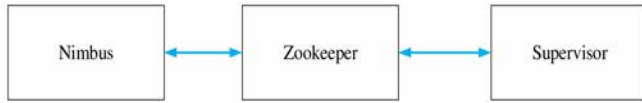


Fig. 1: Apache Storm Architecture.

Apache Storm is made-up of two nodes: the master and the worker nodes. The Nimbus, a daemon in charge of code distribution around the cluster, monitoring failures and assigning tasks to each machine runs in the master node. The Supervisor, a daemon that listens to the Nimbus for a work assignment runs in the worker node. The Supervisor gets instructions from the Nimbus to start and stop worker processes or executors. The Nimbus depends on the services offered by the Apache Zookeeper to monitor messages and process tasks. In addition, the worker nodes communicate with the Nimbus and update their task status through the service of the Apache Zookeeper [22].

Each worker node executes and processes a subset of topology. A running topology may include numerous worker processes spread across many machines. Apache Storm uses topology to do a real-time computation. A topology which is implemented as directed acyclic graph (DAG) data structure, is a graph of computation nodes. Every node in the topology has a processing logic (bolt) and links between nodes indicating the flow and passing of data (streams) around and between them.

A topology is composed of: streams, a set of unbounded data which continuously arrives at the system; spouts, sources of data streams at the edge of the topology and; Bolts, the logical units representing the processing step that ingest streams, applies an operation to them and produces the result as a stream.

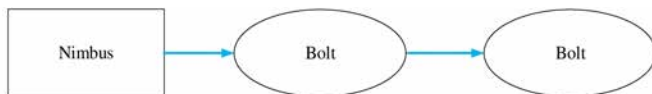


Fig. 2: Apache Storm Topology.

Once a topology is submitted, Nimbus service in a master node inspects the supervisor services on the individual worker nodes and send the topology. Each supervisor has its own separate JVM and executes one or more worker processes [24].

IV. EXPERIMENT

In this section, we first show the environment for our experiment, followed by the system and topology configuration.

Then we discuss the results obtained.

The experiment was performed in a single node. A pseudo cluster composed of: 18.04.1 LTS Ubuntu OS; 16 GB RAM, Core i7-3770 CPU @ 3.40GHz x 8; and HD of 181 GB. Apache Storm-1.1.0 and Apache Zookeeper-3.4.12.

A. System Configuration

```

tickTime=2000
initLimit=5
syncLimit=2
dataDir=/Users/lemilaku/Downloads/exp/zookeeper-3.4.12/data
clientPort=2181
  
```

Fig. 3: Apache Zookeeper configuration.

Shown in, fig. 3 is the Zookeeper, a framework that offers centralized services for maintaining the configurations. It is used to coordinate the exchange of information between the Nimbus and the Supervisor nodes [25].

```

storm.zookeeper.servers:
- "localhost"
- "server2"
storm.local.dir: "/Users/lemilaku/Downloads/exp/apache-storm-1.1.0/data"
nimbus.host: "localhost"
supervisor.slots.port:
- 6700
- 6701
- 6702
- 6703
ui.port: 8080
  
```

Fig. 4: Apache Storm Pseudo Cluster Configuration.

B. Topology Configuration

Shown in, fig. 5 is the topology which a written and programmed in Java. We design and implement a real-time word count system using Apache Storm. We use a topology with a parallelism factor of one (1), consisting of Sentence_Input spout, Sentence_Split bolt, Word_Count bolt, and Count_Summary bolt.

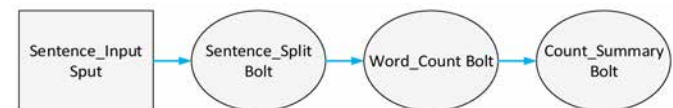


Fig. 5: Word Count Topology Design.

C. Word Count (WC)

Word count is a simple benchmark technique where Apache Storm is used to keep track and frequency of words and their counts from sentences streaming in. In Storm, the ingestion of data is done using spouts. A spout opens and reads a file for real-time event data. However, for simplicity of this experiment, we used a dataset of a finite number of sentences, static data consisting of 300 English sentences. Topology or

DAG, symbolizing the WC, consists of three nodes, as shown in fig. 5, where the Sentence_Input spout releases sentences; the Sentence_Split bolt, the Word_Count bolt, and finally the Count_Summary bolt.

We translate the sentence into four different writing scripts (Arabic, Chinese, Hindi, and Japanese) and sequentially load them into the Sentence_Input spout. The Sentence_Input spout will emit streams of random sentences as tuples to the Sentence_Split bolt.

The Sentence_Split bolt sequentially receives sentences, splits them into words and emits streams of words as tuples, while sending them to the Word_Count bolt where the count is kept. The Word_Count bolt performs fields grouping and determines how many times a particular word is detected or seen. Whenever the Word_Count bolt receives a tuple or a word, it increments the counter and emits a tuple containing the word and the current count to the Count_Summary bolt. The Count_Summary bolt keeps a counter of all the words, including their corresponding counts. When a new tuple arrives, Count_Summary bolt updates the counter and prints the content in a console.

V. RESULTS AND DISCUSSION

The topology (in, fig. 5) is submitted and the metrics of the execution time are taken at the 35th second of the executor's uptime. This is because when a topology is deployed onto Apache Storm, it runs and executes continuously until the user kills it. However, the time for readings of the metrics can vary depending on the investigator's convenience.

To enhance the accuracy of the results, each topology is deployed 25 times and an average of the measured metric is recorded.

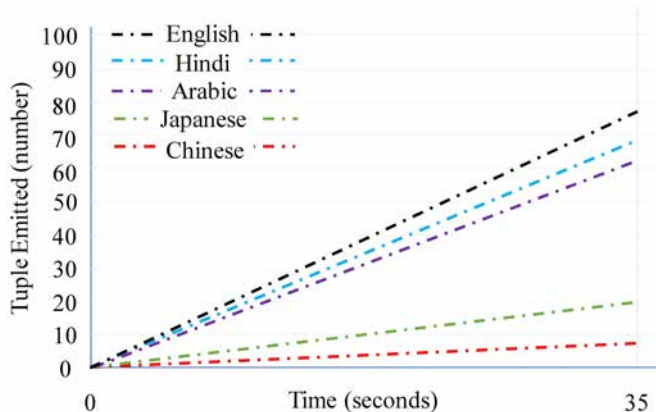


Fig. 6: Average Tuples Emitted.

Shown in, fig. 6 is the results of an average number of tuples emitted or words distinguished which reveals that Apache storm distinguishes words written in English, Arabic, and Hindi writing script easier and faster than those written in Japanese and Chinese. This is because English, Arabic, and Hindi scripts have word spaces between words making it is easy for Apache Storm to detect the words in the sentence [11], [13], [14].

Since Japanese and Chinese scripts have no spaces, it is difficult for Apache Storm to separate the words. In fact, Apache Storm cannot split Chinese and Japanese sentences to distinguish individual words. In most cases, it counts the whole sentence as one word[15]–[19].

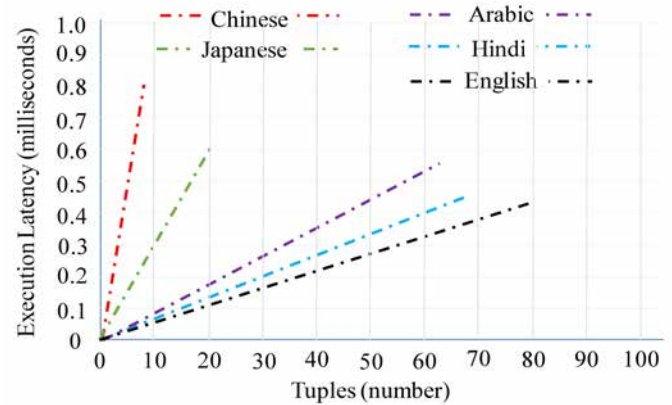


Fig. 7: Average Execution Time.

Shown in, fig. 7, is the results of the average execution latency which shows that Apache Storm is slower in executing or splitting sentences written in Chinese and Japanese scripts. The absence of word delimiters causes obvious ambiguities in sentence segmentation [15]. However, Apache Storm is faster in executing or splitting sentences written in English, Hindi [12], [13], and Arabic scripts [11] as described in fig. 6.

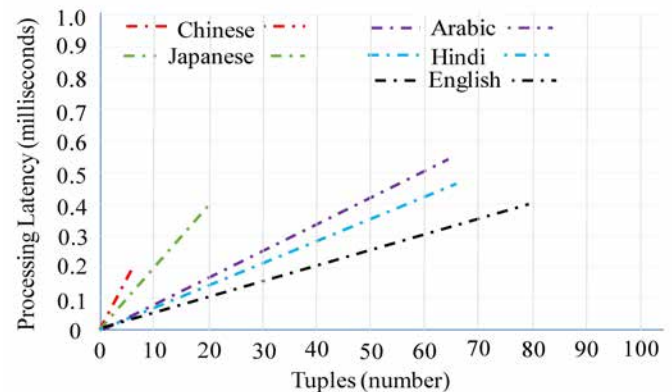


Fig. 8: Average Processing Latency.

Shown in, fig. 8 is the results of an average processing latency which reveals that sentences and words written in Chinese and Japanese scripts are processed pretty faster as compared to those written in English, Arabic and Hindi scripts. Since Apache Storm cannot split and distinguish words in Chinese and Japanese sentences, it ends up processing the whole sentence as a single word [17], as described in fig. 9.

Meanwhile, each Arabic character is denoted by a unique single which is code independent of its shape [11]. Thus it is easy to segment the sentence and count the words using the shape and the position of the word. Hindi sentence has a

	Input	Output
Chinese	牛跳过月亮。 一天一苹果，医生远离我。 四分和七年前。 雪和七个小矮人。 我和大自然在一起。	[牛跳过月亮。]1 [一天一苹果，医生远离我。]1 [四年七年前。]1 [白雪公主和七个小矮人。]1 [我和大自然在一起。]1
Japanese	牛は月を飛び越えた。 リンゴが赤くなると医者が青くなる。 4得点と7年前 白雪姫と7匹の小人。 私は自然と2つです。	[リンゴが赤くなると医者が青くなる。]1 [4つのスコアと7年前。]1 [牛は月を飛び越えた。]1 [私は自然と2つです。]1 [白雪姫と七人のこびと。]1

Fig. 9: Chinese and Japanese Word Count.

boundary marked by *purna viram* (।) [12], and this property makes it easy for the word recognition [13].

VI. CONCLUSION

With the exponential growth of stream data emanating from a variety of sources and which insight is critical to effect actions, understanding the performance of real-time data analytic technologies has become a key pre-requisite while considering a deployment. The main goal of this study has been to experimentally evaluate the implication of writing scripts on the performance of large-scale streaming analytic technologies. Using different writing scripts such as Arabic, English, Hindi, Japanese and Chinese, the performance of Apache Storm was tested in terms of: tuple emission or ease of a sentence and a word distinction; the execution latency or delay in splitting the sentences into distinguishable words and; the processing latency or delay in counting the words and giving the summary. The study found that it is difficult for Apache Storm to split sentences written in Chinese and Japanese script to distinguish the word composition. This is because Chinese and Japanese sentences do not have interword space. And due to the interword spaces, Apache Storm easily splits the sentences written in Arabic, English and Hindi scripts. In addition, to the clear boundaries, Apache Storm executes streams of sentences and distinguishes words written Arabic, English, and Hindi faster than those written in Chinese and Japanese. However, because Apache storm cannot split Chinese and Japanese sentence, they process the whole sentence as a single word, hence less processing time compared to sentences written in Arabic, English, and Hindi.

Though this study has experimentally confirmed that writing scripts can affect the performance of Apache Storm, it has a limitation. A few writing scripts were examined. All the sentences were translated using Google Translate, a tool which the accuracy cannot be trusted. In addition, static data instead of streaming data was used.

As far as we know, this is the first study that points out the impact of writing systems on the performance of Apache Storm. Therefore, we recommend for further studies

using streaming data as opposed to static, expand the study to include more writing scripts such as Hangul, Inuktitut, etc., and also evaluate the impact of sentence structure and character composition on the performance of real-time data streaming analytic technologies.

ACKNOWLEDGMENT

The work was supported by the KAIST GCORE (Global Center for Open Research with Enterprise) grant funded by the Ministry of Science and ICT (Project SDN/NFV and Cloud).

REFERENCES

- [1] A. Levenberg, C. Callison-Burch, and M. Osborne, "Stream-based translation models for statistical machine translation," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 394–402.
- [2] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng *et al.*, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 1789–1792.
- [3] H. Freeman. (2016) Streaming analytics 101: The what, why, and how. [Online]. Available: <http://www.dataversity.net/streaming-analytics-101/>
- [4] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. Nguifo, "A comparative study on streaming frameworks for big data," in *VLDB 2018-44th International Conference on Very Large Data Bases: Workshop LADaS-Latin American Data Science*, 2018, pp. 1–8.
- [5] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "A comparison on scalability for batch big data processing on apache spark and apache flink," *Big Data Analytics*, vol. 2, no. 1, p. 1, 2017.
- [6] M. A. Lopez, A. G. P. Lobato, and O. C. M. Duarte, "A performance comparison of open-source stream processing platforms," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.
- [7] I. Bartolini and M. Patella, "Comparing performances of big data stream processing platforms with ram3s," in *Proc. SEBD*, 2017.
- [8] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, "Benchmarking distributed stream processing engines," *arXiv preprint arXiv:1802.08496*, 2018.
- [9] S. Chatterjee and C. Morin, "Experimental study on the performance and resource utilization of data streaming frameworks," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 143–152.
- [10] P. T. Daniels, "On writing syllables: Three episodes of script transfer," *Studies in the Linguistic Sciences*, vol. 30, no. 1, pp. 73–86, 2000.
- [11] R. I. Elanwar, M. A. Rashwan, and S. A. Mashali, "Simultaneous segmentation and recognition of arabic characters in an unconstrained online cursive handwritten document," in *Proceedings of world academy of science, engineering and technology*, vol. 23, 2007, pp. 288–291.
- [12] C. Thaokar and L. Malik, "Test model for summarizing hindi text using extraction method," in *Information & Communication Technologies (ICT), 2013 IEEE Conference on*. IEEE, 2013, pp. 1138–1143.
- [13] T. Pruthi, S. Saksena, and P. K. Das, "Swaranjali: Isolated word recognition for hindi language using vq and hmm," in *international conference on multimedia processing and systems (ICMPS)*, 2000, pp. 13–15.
- [14] F. Peng, F. Feng, and A. McCallum, "Chinese segmentation and new word detection using conditional random fields," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 562.
- [15] W. J. Teahan, Y. Wen, R. McNab, and I. H. Witten, "A compression-based algorithm for chinese word segmentation," *Computational Linguistics*, vol. 26, no. 3, pp. 375–393, 2000.
- [16] T. Nakagawa, "Chinese and japanese word segmentation using word-level and character-level information," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 466.

- [17] T. Vysokos. (2009) Word count in oriental languages. [Online]. Available: <https://anycount.com/WordCountBlog/word-count-in-oriental-languages/>
- [18] Y.-T. Sung, T.-H. Chang, W.-C. Lin, K.-S. Hsieh, and K.-E. Chang, "Crie: An automated analyzer for chinese texts," *Behavior research methods*, vol. 48, no. 4, pp. 1238–1251, 2016.
- [19] I. S. P. Ltd. (2018) Chinese text analyser. [Online]. Available: <https://www.chinesetextanalyser.com/features>
- [20] V. Yadav, "Big data, hadoop, and hdinsight," in *Processing Big Data with Azure HDInsight*. Springer, 2017, pp. 1–11.
- [21] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.
- [22] X. Liu, N. Iftikhar, and X. Xie, "Survey of real-time processing systems for big data," in *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM, 2014, pp. 356–361.
- [23] A. S. Foundation. (2015) Apache storm tutorial. [Online]. Available: <http://storm.apache.org/releases/1.1.2/Tutorial.html>
- [24] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham et al., "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.
- [25] M. Ficco, R. Pietrantuono, and S. Russo, "Aging-related performance anomalies in the apache storm stream processing system," *Future Generation Computer Systems*, vol. 86, pp. 975–994, 2018.



Lemi Isaac Yoseke Laku received an M.S. degree in Information and Telecommunications Technology (Engineering) from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2018, a B.Sc. degree in Information Technology from Sikkim Manipal University Directorate of Distance Education (SMUDDE), Offshore Campus Kampala Uganda in 2010; and a national diploma in Journalism and Mass Communications from UMCAT School of Journalism and Mass Communications in Kampala Uganda in 2011. Currently, he is working as a Senior Inspector for e-Government and IT Enabled Services (ITES) at the Ministry of ICT and Postal Services, Republic of South Sudan. He had worked as a Research Scientist with the Grid Middleware Research Centre of KAIST, Republic of Korea, where he participated in a national R&D project of developing and demonstrating smart city services using 5G-based technologies. He won a "IJGold Medal" in the 2018 Pohang World University Taekwondo Competition & World University. His research area includes electronic/ mobile government and IT enabled services.



Alaeldin Fuad Yousif Mohammed working as Post-doc Researcher at Media Network Lab at Korea Advanced Institute of Science and Technology (KAIST). Received Ph.D. degree in Information and Telecommunication Technology (Engineering) at the Korea Advanced Institute of Science and Technology, South Korea in 2016, and M.Sc. degree in Computer Networks and M.Sc. degree in the Engineering and Management of Information Systems from the Royal Institute of Technology, Sweden. Received a B.Sc. degree in computer engineering from the University of Gezira, Sudan, have worked as a faculty member at the University of Gezira since 2004 to date. His research interests include energy saving in access networks, modeling and simulation of computer networks, Internet Protocols, Software Defined Networks and Network Function Virtualization.



Fawaz AL-Hazemi received the B.S. degree in Computer Engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in 2003; and the M.S. degree in Information and Communications Engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2010; and the PhD degree in Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2019. He is currently working as a Computer Engineering lecturer in Computer Science Department in College of Information Technology and Computer Sciences with University of Prince Mugrin, Madinah, Saudi Arabia. He has authored or co-authored over 30 technical papers, he is a Technical Program Committee member of several international conferences and serves as a Reviewer for IEEE Transactions/Magazines and International Journals. His research interests are green data center and energy-aware computing. He received "IJBest Paper Award" and "IJBest Student Paper Award" in the FTRA AIM-2014 and the 6th CloudComp-2015, respectively. He is an IEEE Senior Member and ACM Senior Member.



Chan-Hyun Youn Chan-Hyun Youn received the B.Sc and M.Sc degrees in Electronics Engineering from Kyungpook National University, Daegu, Korea, in 1981 and 1985, respectively. He also received a Ph.D. in Electrical and Communications Engineering from Tohoku University, Japan, in 1994. He was with Korean Army as a communications officer, Lieutenant, from 1981 to 1983. Before joining the University, from 1986 to 1997, he was the leader of high-speed networking team at KT Telecommunications Network Research Laboratories, where he had been involved in the research and developments of Centralized Switching Maintenance System (CSMS), Maintenance and Operation system for Various ESS's (MOVE) system, high-speed networking, and HAN/B-ISDN network testbed. Especially, he was a principal investigator of high-speed networking projects including ATM technical trial between KT and KDD, Japan, Asia-Pacific Information Infrastructure (APII) testbed, Korea Research and Education Network (KOREN) and Asia-Pacific Advanced Network (APAN), respectively. Since 2009, he has been a professor at school of Electrical Engineering in KAIST, Daejeon, Korea. He also was an associate vice-president of Office of Planning and Budgets in KAIST, from 2013 to 2017. He was a Visiting Professor at MIT in 2003 and has been engaged in the development of Physio-Grid system with Prof. R.G. Mark's Group in LCP (Laboratory for Computational Physiology) of MIT. He also is a Director of Grid Middleware Research Center at KAIST. Where, he is developing core technologies that are in the areas of cloud computing, deep learning system and applications, distributed computing system, silicon photonics-based interconnection system for supercomputer, and smart car services system. In addition, he served as an Editor-in-Chief in KIPS (Korea Information Processing Society), and Editor of Journal of Healthcare Engineering (U.K.), and served head of Korea branch (computer section) of IEICE, Japan (2009, 2010). He is a member of IEEE, KIPS and KICS, respectively.