Bulletin of Electrical Engineering and Informatics
Vol. 12, No. 6, December 2023, pp. 3706~3716
ISSN: 2302-9285, DOI: 10.11591/eei.v12i6.5070

A powerful heuristic method for generating efficient database systems

Haider Hadi Abbas[1], Poh Soon JosephNg[2], Ahmed Lateef Khalaf[3], Jamal Fadhil Tawfeq[4], Ahmed Dheyaa Radhi[5]
[1]Department of Computer Technology Engineering, Al-Mansour University College (MUC), Baghdad, Iraq
[2]Faculty of Data Science and Information Technology, INTI International University, Nilai, Malaysia
[3]Department of Communication Technology Engineering, College of Information Technology, Imam Ja'afar Al-Sadiq University, Baghdad, Iraq
[4]Department of Medical Instrumentation Technical Engineering, Medical Technical College, Al-Farahidi University, Baghdad, Iraq
[5]College of Pharmacy, University of Al-Ameed, Karbala, Iraq

Article Info

Article history:

Received Oct 22, 2022
Revised Dec 14, 2022
Accepted Apr 17, 2023

Keywords:

Database
Heuristic algorithm
Optimized
Security

ABSTRACT

Heuristic functions are an integral part of MapReduce software, both in Apache Hadoop and Spark. If the heuristic function performs badly, the load in the reduce part will not be balanced and access times spike. To investigate this problem closer, we run an optimal database program with numerous different heuristic functions on database. We will leverage the Amazon elastic MapReduce framework. The paper investigates on general purpose, implementation, and evaluation of heuristic algorithm for generating optimal database system, checksum, and special heuristic functions. With the analysis, we present the corresponding runtime results. For the coding part, the records counting part is hasty and can only work for local Hadoop part, it can be debugged and optimized for general purpose implement on Hadoop and Spark and turn into an effective performance monitor tool. As mentioned before, there are strange issue, also the performance of BLAKE2s is unexpectedly slow in that it's widely accepted the performance of BLAKE2s is much better than MD5 and SHA256, we would like to figure out why the common-sense performance of heuristics is deferent from what we got in distributed frameworks.

Corresponding Author:

Poh Soon JosephNg
Faculty of Data Science and Information Technology, INTI International University
Persiaran Perdana BBN Putra Nilai, 71800 Nilai, Negeri Sembilan, Malaysia
Email: joseph.ng@newinti.edu.my

1. INTRODUCTION

MapReduce is a framework to extract information from large datasets ciently. It has to major components: i) mapper which reads, transforms data, and creates key-value pairs and ii) reducer which combines multiple mapper outputs. Each mapper and reducer can run on an individual machine. The most expensive operation in this setup is transferring data between machines. Thus, the necessary data exchange should be kept minimal. Additionally, for the reducer to work correctly, it needs all data which correspond to the same key. To address those two problems heuristic functions are used. They ensure that each key has the same, shorter heuristic value and these heuristic values are as signed to reducer. The reducer then works on their own problem and generate the result for a certain key. Heuristic functions have two core contributions: i) ensure that each key produces the same heuristic value and ii) create a uniformly distributed. The latter is necessary to evenly distribute the workload. Both Apache Hadoop [1] and Apache Spark [2] are two commonly used MapReduce frameworks. If the heuristic functions are slow or distribute data badly, the execution time will potentially increase. A slow heuristic function will add additional computation time at each node and thus increase the overall computation time. If a heuristic function does not distribute the data accordingly, multiple keys might fall into the same heuristic value and thus increase the load on a single reducer. The parallelism is reduced and the execution time is increased. In this paper, we want to give an analysis of common heuristic functions if they are used in Apache Hadoop or Spark. For this purpose, we selected 14 functions which generate a heuristic value. We use each function in an optimal database example in both Apache Hadoop and Spark. Additionally, we use these heuristic functions in the PageRank [3] algorithm on Apache Spark. With the resulting execution times, we show how heuristic functions impact the performance of MapReduce problems and how they can also influence the performance of machine learning algorithms with Apache Spark.

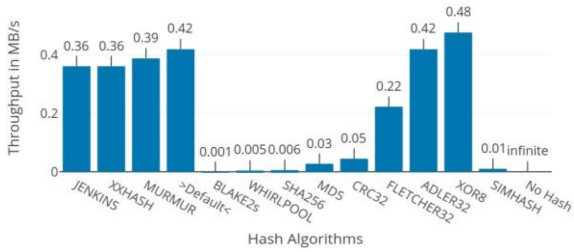
Although heuristic algorithms are crucial to Apache Hadoop and Spark, research about how dedicated heuristic function perform is hard to find. Thus, the found related work is comparably limited. He *et al.* [4] did investigate on using graphics card processing on Apache Hadoop and mentioned the potential impact of a good or bad heuristic function. However, they did not discuss the actual impact of a bad heuristic function. Katsoulis [5] does use heuristic functions extensively, i.e. for joins, but again does not discuss any performance impacts of heuristic functions. However, they further strengthen the reliance on heuristic functions with their research. Bertolucci *et al.* [6] did discuss big data partitioning in Spark in 2015 but again did not focus on heuristic functions. They focused on the difference between dynamic and static partitioning. Kocsis *et al.* [7] proposed a method to repair broken Apache Hadoop heuristic functions but did not discuss the performance impact of bad, broken heuristic functions in detail. Their key contribution is an algorithm to automatically fix or optimize–heuristic functions to perform better. Ramakrishna *et al.* [8] describe the importance of good heuristic functions for high-performance computers but limit themselves to a theoretical approach. To the best of our knowledge, there is no related work which discusses how a heuristic function impacts the overall performance of Apache Hadoop or Spark. While there is research using and relying on heuristic functions, we could not find any which puts numbers on the actual impact. However, it is commonly accepted that heuristic functions do influence the performance of Apache Hadoop and Spark significantly. Nevertheless, a detailed analysis was not found. This paper is structured as follows: section 2 presents the implementation details behind the analysis, section 3 provides the results of the analysis, and section 4 concludes this paper.

2. METHOD

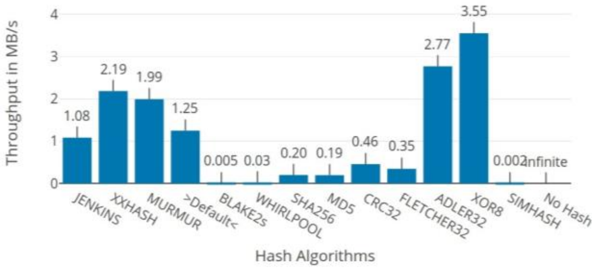
2.1. Heuristic approaches for database

We selected 14 different heuristic function approaches and divided them into four categories. Our defined categories are general purpose heuristic functions, cryptographic heuristic functions, Checksums as heuristic functions, and special heuristic functions. The first category contains heuristic functions which are used to divide data. They should serve the purpose of Apache Hadoop and Spark perfectly as they are supposed to be well balanced between speed and heuristic value distribution. General purpose heuristic functions mostly use a limited amount of operations and are not difficult to implement. The second category contains heuristic functions for cryptographic purposes. They are less efficient but provide the certainty that there is no calculating back from the heuristic value to the original value. One way to ensure this is the festival structure. The input is manipulated in multiple rounds and in each round, it is split up into two segments. Those segments are exchanged. One of them gets XORed with a set of predefined numbers and the other gets again XORed with this result. Through many such iterations both security and distribution are ensured. Although this security not necessary for Apache Hadoop and Spark, they are good markers for bad heuristic functions, as they are usually very slow. In the third category, common checksum algorithms can be found. They are designed to be quick in generating but are created only to detect transmission errors and thus distribution is disregarded. In Figure 1 that such functions tend to have certain values which are more likely the result than others. Their inner workings typically rely on a single iteration in which mathematical operations also modulo operations are applied in combination with predefined numbers. Thus, their heuristic value distribution might be worse than standard heuristic functions, but their throughput is higher. The last category contains special functions which have a heuristic like behavior with certain additional properties.

For each category four representative are chosen. The special heuristic functions category contains two functions. An overview of how the individual functions performed on a standard machine is depicted in Figure 1. On the left side the throughput of an 11-byte input and on the right side the throughput of a 200 MB file. As you can see, some heuristic functions perform better for big inputs and some perform better for small inputs. The results are grouped by the categories and the throughput is calculated in Mbps with a single 3.6 GHz processor on Ubuntu 16.04. Figure 2 shows the distribution of the heuristic functions. It shows how many out of 10,000 numbers in text format fall in a single of 128 buckets. If the bar is thinner and lower, it represents a better distribution. A scattered plot is a bad distribution. The general-purpose heuristic functions have a better distribution than expected. For the cryptographic heuristic functions, SHA256 is a surprising outlier. The others are marginally better than the general-purpose heuristic functions. Both Checksums and special heuristic functions have a bias in the distribution which can end up in reducing parallelism. All evaluated heuristic functions are implementations which can be found on the internet. This approach reduces potential implementation errors. Figures 1(a) and (b) shows the throughput of the different heuristic functions on a single 3.6 GHz core clustered by type 11-byte and 200 MB.



(a)



(b)

Figure 1. Throughput of the different heuristic functions on a single 3.6 GHz core clustered by type (a) 11-byte and (b) 200 MB

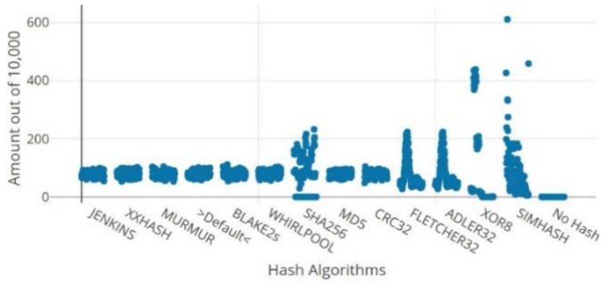


Figure 2. The number of entries of resulting heuristic values modulo 128 of 10,000 numbers in text format

The four leftmost on each side are part of the general-purpose heuristic functions. The next four are in the category cryptographic heuristic functions. The following four are cyclic redundancy checks for error detection. The last two are in the special heuristic function category. Each dot represents one of the 128 modulo buckets. Both general-purpose heuristic functions and cryptographic heuristic functions have a similar distribution. Checksums and special heuristics have an expected worse distribution. The 10,000 dot of no heuristic is not displayed.

2.2. General purpose heuristic functions

The first representative is the default heuristic code () [9] implementation of Java. It is commonly used in Java software and implemented through native code. Thus, the implementation is platform dependent but usually delivers an output well balanced between speed and distribution. The second heuristic function in the category is the Jenkins [10], [11] heuristic. In our test environment, it performs slightly worse than the default implementation of heuristic code (). It should serve as a good reference point for how well the default implementation is on the AWS setup compared to the local one. As a third heuristic MurmurHeuristic [12], [13] is chosen and xxHeuristic [14], [15] as the fourth heuristic. Both are recent developments and focus on the throughput. xxHeuristic is almost capable of twice the throughput than the default Java implementation on Ubuntu 16.04.

2.3. Cryptographic heuristic functions

The first chosen representative of this group is BLAKE2s [16], [17]. It is a novel cryptographic heuristic function which promises superior speed and safety. Strangely, the implementation we use performs worse than the other cryptographic heuristic functions. Nevertheless, we kept it, as it gives the behavior of a very slow heuristic function. The second cryptographic heuristic function is Whirlpool [18], [19]. It is a rather recent development which promises good performance with superior security. In our test setup, it is the second slowest representative in this category. The last two representatives are MD5 [20], [21] and SHA256 [22]. Both have a similar and higher throughput than the others in this category. While MD5 is officially insecure, SHA256 has not suggested for use anymore. However, both are still commonly used in real world applications, which makes them good choices. Since all cryptographic heuristic functions result in more than 32-bit output, only the first 32 bit is used in our analysis.

2.4. Checksums as heuristic functions

Both Fletcher32 [23], [24] and CRC32 [25], [26] have a similar throughput behavior. They are developed specifically for error detection and not for code distribution. However, they do convert an arbitrary input to a 32-bit output and thus can function as heuristic functions. With Adler32 [27], [28] we chose a better performing checksum algorithm for comparison. Similarly, to the previous ones, it generates 32 bits of output. The superior speed is traded for a slightly worse error detection resulting in less distribution. Lastly, we added the XOR checksum for comparison. It is the only function, which generates 8-bit output by simply applying the XOR operation to every byte of the input. Thus, it has the worst distribution of all but outperforms any other function.

2.5. Special heuristic functions

Additionally, we added two special heuristic function to the analysis. With SimHeuristic [29], [30] we evaluate a heuristic function which produces similar heuristic values for similar input values. Although this should not result in an advantage for Apache Hadoop and Spark, we chose it for curiosity purposes. Lastly, we added a no heuristic simulation for comparison purposes. This function always returns the same heuristic values for arbitrary input. Thus, all the data will be gathered at a single reducer and the load is not balanced at all. However, it returns the value instantly and does not need any operations.

2.6. Apache hadoop and spark

The source packages of both Apache Hadoop and Spark provide example implementations for the word count problem. We modified these samples and use them for our analysis. In the Apache Hadoop case, we override the standard org.apache.hadoop.io. text class with our custom implementation. This implementation implements a custom heuristicCode() implementation, which applies a selected heuristicing algorithm for the input. By overriding the heuristicCode() function of the key, it is automatically applied whenever a heuristic code is necessary. For Apache Spark, we create our own class MyString. It acts as a wrapper class for strings and overrides the equals() and heuristicCode() functions. The heuristicCode() functions selects the heuristic algorithm to use. We use this class for the computations with Apache Spark and thus the custom heuristic function is in use wherever possible.

2.7. PageRank

The PageRank algorithm is only used in Apache Spark due to its iterative nature. There exists a standard implementation in the source code of Apache Spark which we modified and used for this paper. Similarly, to the Apache Spark implementation of the optimal database example, we created the class MyString. It acts as a wrapper class for strings and overrides the equals() and heuristicCode() functions. The heuristicCode() functions selects the heuristic algorithm to use. With some modifications of the provided example, we use the MyString class instead of normal strings and thus use the custom heuristic algorithm wherever possible.

3. RESULTS AND DISCUSSION

Amazon provides a bunch of effective frameworks like Amazon elastic compute cloud (EC2) and Amazon elastic MapReduce (EMR). Normally EC2 provides remote servers for a user to customize, in this project, we find that spending a large amount of time setting up a cluster is ineffective since we will run on both Hadoop and Spark and our target is the heuristic algorithms. In this case, we select EMR as our remote environment. Users can easily edit a configuration of framework and the whole cluster can be fully functional within ten minutes. All EMR instances are completely based on Amazon EC2. As commonly known, the file system of Hadoop and Spark is Hadoop distributed file system (HDFS), on EMR, you can choose to use the traditional HDFS or EMRFS with Amazon S3. Considering the limited funds, our project runs on the traditional HDFS. Amazon command line interface (CLI) is the interface for input. For this project, we implement our programs on a cluster consisted of three m4. Large nodes powered by two cores of Intel Xeon E5-2686 v4 or E5-2676 v3 with 8 GB memory. The version of Hadoop is 2.7.3 and for Spark is 2.2.0. The number of practitioners in Hadoop is set to be 5. However, the number of practitioners in Spark is set automatically to be 16 and won't be affected by spark-submit commands, which can better exploit the strongpoint of parallelization. For local running we set up Hadoop on two machines to distribute our work, and the features of one machine are Intel Core i5-7360U with 4 GB memory and another is Intel Core i5-5257U with 8 GB memory. Hadoop is installed in pseudo distributed mode with 5 practitioners and Spark is running in local mode with 5 practitioners as well.

Three datasets are used for word count, first is the air quality index comes from the meteorological stations in China [31], the number size is around 200,000, which contains both integers and decimals. Second is the lorem ipsum generated from an online website [32], lorem ipsum is a text consists of meaningless words generated randomly to minimize the influence of words' meanings in design field, lorem ipsum does have duplicate words and is easy to control the size with accessible generators and is similar to the structure of real articles. The size for local test is 20,777,978 bytes and for EMR is 10,388,990 bytes. On local the size is doubled to make sure that Hadoop can generate enough practitioners. Last is a 49.4 MB duplicate questions file from Quora, which includes over 400,000 question pairs [33] that have the potential to be duplicate, this dataset is closer to real work and can better simulate the situations in real tasks. As to the PageRank part, we wrote a small program that generates a list of websites with their name's represented as numbers, followed by a random PageRank value. We generated a list of 100,000 websites for EMR and 1,300,000 websites one for local test. We took the overall duration time of collective the part on EMR to mimic the practical time elapsed in a cluster, and for local, we took the same part of the time from the terminal output to get the most accurate result for experimental purpose. The test is first launched on Spark EMR, and for every dataset we run it ten times to get the mean value except for some extremely slow heuristic algorithms, only three to five times is launched to make sure the slow performance is not an exception. First comes the result of air quality case, performances of heuristices running on air quality can be seen in Figure 3. Overall, the algorithms except BLAKE2s have the similar time performances, Adler32 and xxHeuristic can even be a little faster than the default heuristicCode in some rounds in such a relatively small dataset. Time consumption from the lorem ipsum test is given in Figure 4. The performance of cryptographic heuristices and special functions start to show a trend of slowing down especially BLAKE2s and its noteworthy that the performance of xxHeuristic even slightly exceeds the one of thus we come to the Quora duplicate questions dataset, the time performance is given in Figure 5. This dataset is big enough and the test on BLAKE2s and no heuristic took so long that we can't even get a result, so we simply represent it as N/A.

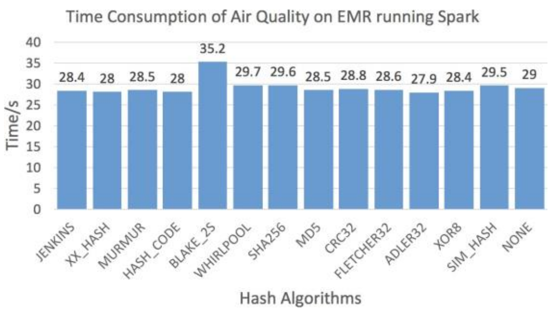


Figure 3. The average time consumption of Spark word count program in collective part. There's no significant performance difference in this case and even ADLER32 can be slightly faster than default heuristic function

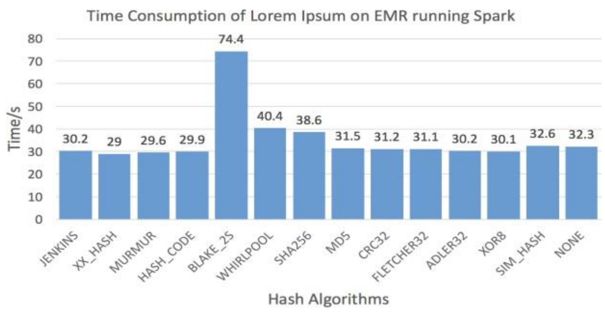


Figure 4. Time consumption of running on Apache Spark shows the relatively worst performance of cryptographic heuristics' and slightly slow performance of special heuristics

We can tell that general heuristics do benefit from their superb throughput and still remain the best, while the cryptographic heuristics and special heuristics have been completely unusable for their more than twice time, which makes sense due to their small throughput. The Checksums except XOR8 also shows the correlation trend but the degree is much lower. Under no heuristic circumstance, all records are delivered to the first reducer, so there is little parallelization in this case and the performance is the worst. Questions can be raised that XOR8 was supposed to show a faster speed as the throughput level is good but here is the opposite.

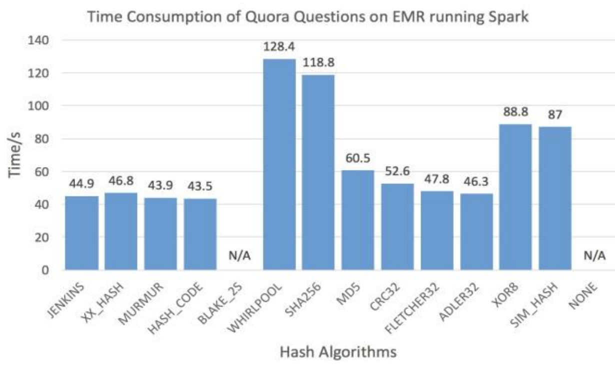


Figure 5. The obvious difference between heuristic types and generally shows a reasonable negative correlation with their throughput level but not completely fitted

The inappropriate correlation level in general purpose heuristics also indicates us that besides throughput, more reasons can be affecting the performance of heuristics. So, we collected the number of records each practitioner receives to see how records are partitioned and distributed to reducers. We took the standard deviation of data to demonstrate the distribution level and as the duty of heuristics is to let out records as evenly as possible, so here higher distribution level represents a worse job. However, when we analyze the Spark distribution result, we are surprised to find Adler32 and Fletcher32 function strangely on Spark with large datasets that the total records generated are more than they are supposed to be, but when we verify this situation with small datasets and on Hadoop, they never happened. Nevertheless, comparing with other results, the result Figure 6 still shows the same pattern and we can still take this as a reference.

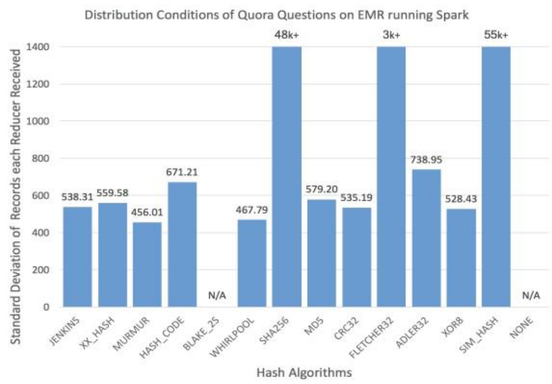


Figure 6. Distribution conditions taken from standard deviation of records each reducer received

General heuristics with higher throughput unluckily have relatively worse distribution and it’s interesting that the heuristics with better throughput right have the worse distribution, this drags down the overall speed and as a result, the difference between general purpose heuristics are neutralized. The cryptographic heuristics are greatly limited by throughput, so the distribution won’t make them better. Checksums performances are hard to predict according to the Figure 2, the records are both likely to be scattered evenly or not. In this case looks like Fletcher32 shows a scattered pattern while Adler32 is acceptable and Adler32’s better throughput made it better than Fletcher. Special heuristics shows such a bad distribution that we should avoid implementing them in practitioner. For the local environment, our data illustrate the similar result Figures 7 and 8, and the significant difference is from Checksums, Adler, and Fletcher seems to be steady this time but XOR8 turns to be extremely bad. For this time, we can get the distribution of records of BLAKE2s, which is quite acceptable, but the little throughput still became its bottleneck and showed an unsatisfactory outcome. Hadoop's results are better able to describe the actual performances of heuristics because the problem only occurred on Spark and not on Hadoop. As the results from EMR are similar and running in pseudo mode can reduce the unpredictable factors on a distributed cluster as much as possible, e.g., for Hadoop, we will primarily discuss the results running on local machines.

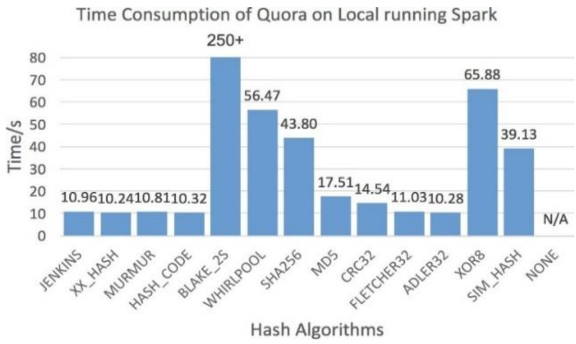


Figure 7. Time consumption on local Spark shows the similar pattern as on a remote cluster on EMR which proves the running mode of Spark won't change the performance of heuristic algorithms

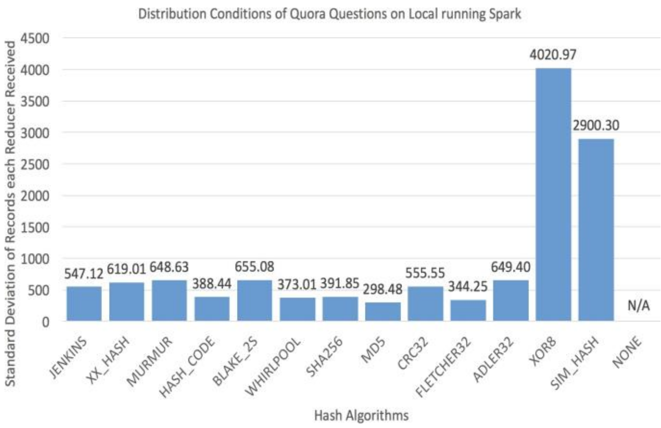


Figure 8. Distribution of local Spark is able to get the data of BLAKE2S, which proves the cryptographic heuristics 'performances are dominated by throughput regardless of the tolerable spread

The lag times in network communication between EMR instances. The performance of XOR8 is the only notable difference between the tests on Spark and Hadoop, which is completely expected given the sporadically well-distributed records. On Hadoop, the previously concluded results are still consistent with the majority of the heuristics. The Hadoop results demonstrate the consistency of the heuristics' operation in both frameworks. Figure 9 shows performance of XOR, which can serve as an excellent example of how distribution affects task speed, demonstrates the main difference between Spark and Hadoop. Figure 10 shows distribution conditions of XOR8 are greatly improved this time, and combining with its outstanding throughput.

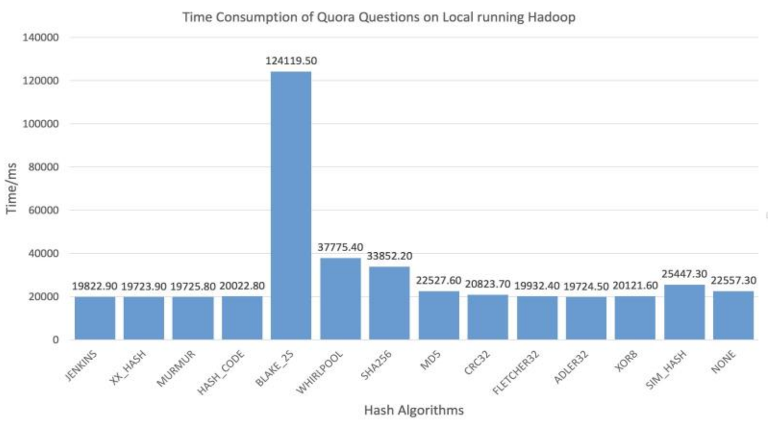


Figure 9. The main discrepancy between Spark and Hadoop is the performance of XOR, which can be an excellent proof of the impact of distribution on the task speed

Figure 10. Distribution conditions of XOR8 are greatly improved this time, and combining with its outstanding throughput

PageRank is an iterative algorithm that only accepts pairs of integers as input and operates entirely differently from word count. Figures 11 and 12 present separate information gathered from EMR and local sources. We received the same time commitments and dispersed records from Hadoop. The independence of heuristic algorithms is demonstrated, and the specifics of the tasks in these two programs have no bearing on the performance of heuristic functions because their responsibility is to map records from appropriate mappers to reducers. They only need to be concerned with the uniformity and speed.

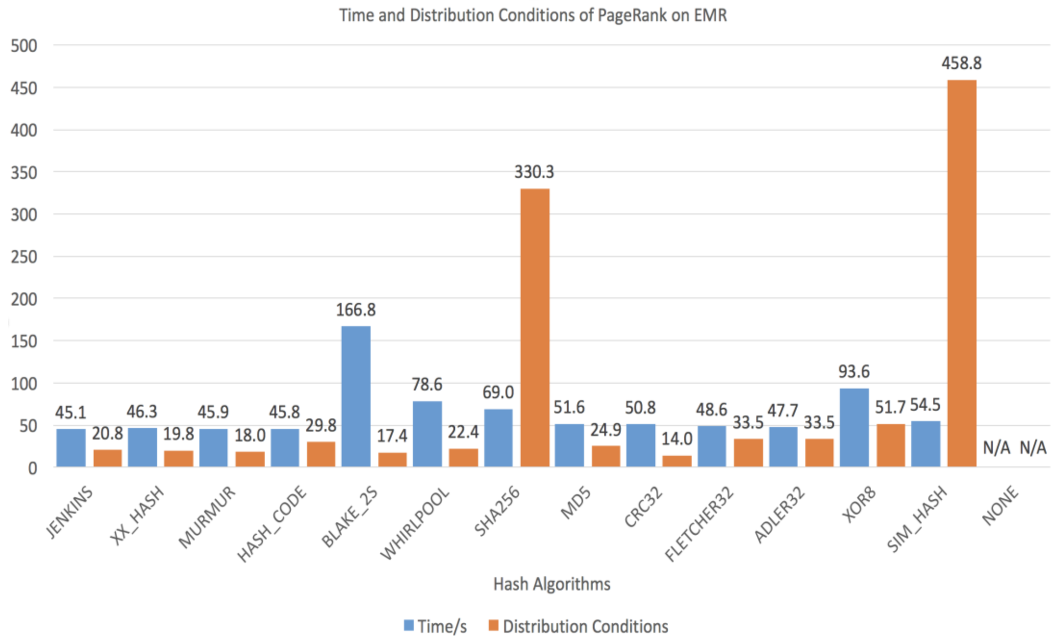


Figure 11. Results of PageRank from EMR have the same features and correlations as the previous results

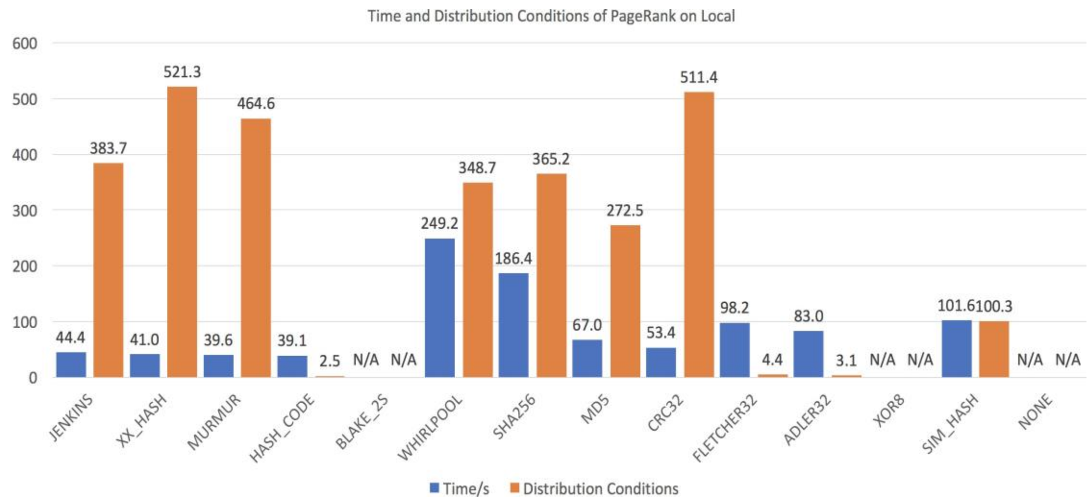


Figure 12. CRC displays poor distribution, slowing speed compared to EMR, but small throughput mitigates speed decrease

4. CONCLUSION

The research has 4 heuristic algorithms divided into 4 groups are tested in this project and unluckily only heuristic has a slight potential of replacing the default algorithms when running with small datasets. We can tell from the data that general purpose heuristics always have the best performance, highly predictable, and similar to each other. Replacing with general purpose heuristics is a safe way if necessary. The cryptographic heuristics only can compare with others under small datasets, their unsatisfactory throughputs are bottlenecks for the speed, thus as the name said, they are better used for encrypting jobs and completely not your choice for a replacement in parallel systems. Checksums generally possess the worst uniformity of distribution. As a result, their performances are unpredictable and can sometimes cost heavily. They are also not recommended for distributing records. Special groups are mainly set for reference and comparison. They are unusable in most case, however, their capability can help us understand how a bad heuristic or even no heuristic would influence the system, which emphasizes the importance of heuristic algorithms in the cloud computing. Basing on the data collected about throughput and distribution.

We can also conclude that the performance of heuristics in the partitioned in greatly dominated by these two factors and usually the impact from throughput is the foundation of the distribution and in a mass decides how it functions inside practitioner. In this program we tested a limited number of heuristic algorithms try to find out the relationship between the performance and the type of heuristics. Only when in small datasets we do find some heuristics such as functions could have the potential of replacing the default heuristic function to provide better performance. But there're still more functions worth our implement and test. We do believe that after relatively mature tests, we can provide a guidebook for when and where to implement which kind of heuristics to avoid blindly trial and error when the default heuristic is not suitable under specific circumstances, but that would require a lot of time and effort to collect and test the commonly used heuristics. We do find the correlations between the performance of tasks and the throughput and uniformity of heuristic algorithms, but these are inferred from small data-sample and we even encounter with some strange behaviors about Adler32 and Fletcher32 on Spark with large datasets. So, we still need more tests with various datasets types and sizes to prove our conclusion. The correlations between performance, throughput, and distribution are qualitative rather than quantitative. It would well if we can find some quantitative relation between the three factors and that can be attempted by implementing machine learning since we do have a bunch of original data.

REFERENCES

[1] M. Vassar *et al.*, "Database selection in systematic reviews: an insight through clinical neurology," *Health Information and Libraries Journal*, vol. 34, no. 2, pp. 156–164, 2017, doi: 10.1111/hir.12176.

[2] J. P. T. Higgins *et al.*, *Cochrane handbook for systematic reviews of interventions*. New Jersey, USA: John Wiley & Sons, 2019, doi: 10.53841/bpsicpr.2020.15.2.123.

[3] M. Bianchini, M. Gori, and F. Scarselli, "Inside pageRank," *ACM Transactions on Internet Technology*, vol. 5, no. 1, pp. 92–128, 2005, doi: 10.1145/1052934.1052938.

[4] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a MapReduce framework on graphics processors," in *Proceedings*

of the 17th international conference on Parallel architectures and compilation techniques, 2008, pp. 260–269, doi: 10.1145/1454115.1454152.
[5] S. Katsoulis, “Implementation of parallel hash join algorithms over Hadoop,” M.S. thesis, School of Informatics, University of Edinburgh, Edinburgh, Scotland, 2011.
[6] M. Bertolucci, E. Carlini, P. Dazzi, A. Lulli, and L. Ricci, “Static and dynamic big data partitioning on apache spark,” *Advances in Parallel Computing*, vol. 27, pp. 489–498, 2016, doi: 10.3233/978-1-61499-621-7-489.
[7] Z. A. Kocsis *et al.*, “Repairing and optimizing hadoop hashCode implementations,” in *Search-Based Software Engineering*, Cham: Springer, 2014, pp. 259–264, doi: 10.1007/978-3-319-09940-8_22.
[8] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, “Efficient hardware hashing functions for high performance computers,” *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997, doi: 10.1109/12.641938.
[9] W. M. Bramer, D. Giustini, and B. M. R. Kramer, “Comparing the coverage, recall, and precision of searches for 120 systematic reviews in Embase, MEDLINE, and Google Scholar: a prospective study,” *Systematic Reviews*, vol. 5, no. 1, pp. 1–7, 2016, doi: 10.1186/s13643-016-0215-7.
[10] J. Rathbone, M. Carter, T. Hoffmann, and P. Glasziou, “A comparison of the performance of seven key bibliographic databases in identifying all relevant systematic reviews of interventions for hypertension,” *Systematic Reviews*, vol. 5, no. 1, pp. 1–6, 2016, doi: 10.1186/s13643-016-0197-5.
[11] M. Arber *et al.*, “Which databases should be used to identify studies for systematic reviews of economic evaluations?,” *International Journal of Technology Assessment in Health Care*, vol. 34, no. 6, pp. 547–554, 2018, doi: 10.1017/S0266462318000636.
[12] D. Chen, Q. Zhi, Y. Zhou, Y. Tao, L. Wu, and H. Lin, “Association between dental caries and BMI in children: a systematic review and meta-analysis,” *Caries Research*, vol. 52, no. 3, pp. 230–245, 2018, doi: 10.1159/000484988.
[13] C. Hayden *et al.*, “Obesity and dental caries in children: a systematic review and meta-analysis,” *Community Dentistry and Oral Epidemiology*, vol. 41, no. 4, pp. 289–308, 2013, doi: 10.1111/cdoe.12014.
[14] M. Hooley, H. Skouteris, C. Boganin, J. Satur, and N. Kilpatrick, “Body mass index and dental caries in children and adolescents: a systematic review of literature published 2004 to 2011,” *Systematic Reviews*, vol. 1, no. 1, pp. 1–26, 2012, doi: 10.1186/2046-4053-1-57.
[15] G. M. Faisal, H. A. A. Alshadoodee, H. H. Abbas, H. M. Gheni, and I. Al-Barazanchi, “Integrating security and privacy in mmWave communications,” *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 5, pp. 2856–2865, 2022, doi: 10.11591/eei.v11i5.4314.
[16] J. P. Aumasson, S. Neves, Z. W.-O. Hearn, and C. Winnerlein, “BLAKE2: Simpler, smaller, fast as MD5,” in *Applied Cryptography and Network Security*, Berlin, Heidelberg: Springer, 2013, pp. 119–135, doi: 10.1007/978-3-642-38980-1_8.
[17] L. W. Li, H. M. Wong, S. M. Peng, and C. P. McGrath, “Anthropometric measurements and dental caries in children: a systematic review of longitudinal studies,” *Advances in Nutrition*, vol. 6, no. 1, pp. 52–63, 2015, doi: 10.3945/an.114.006395.
[18] M. Paisi *et al.*, “Body mass index and dental caries in young people: a systematic review,” *BMC Pediatrics*, vol. 19, no. 1, pp. 1–9, 2019, doi: 10.1186/s12887-019-1511-x.
[19] S. Shivakumar, A. Srivastava, and G. C. Shivakumar, “Body mass index and dental caries: a systematic review,” *International Journal of Clinical Pediatric Dentistry*, vol. 11, no. 3, pp. 228–232, 2018, doi: 10.5005/jp-journals-10005-1516.
[20] N. Manohar, A. Hayen, P. Fahey, and A. Arora, “Obesity and dental caries in early childhood: a systematic review and meta-analyses,” *Obesity Reviews*, vol. 21, no. 3, pp. 1-15, 2020, doi: 10.1111/obr.12960.
[21] M. V. Angelopoulou, M. Beinlich, and A. Crain, “Early childhood caries and weight status: a systematic review and meta-analysis,” *Pediatric dentistry*, vol. 41, no. 4, pp. 261–272, 2019.
[22] D. Eastlake and T. Hansen, “US secure hash algorithms (SHA and SHA-based HMAC and HKDF),” *Internet Engineering Task Force (IETF)*, pp. 1–127, 2011.
[23] Y. K. Salih, O. H. See, S. Yussof, A. Iqbal, and S. Q. M. Salih, “A proactive fuzzy-guided link labeling algorithm based on MIH framework in heterogeneous wireless networks,” *Wireless Personal Communications*, vol. 75, no. 4, pp. 2495–2511, 2014, doi: 10.1007/s11277-013-1479-z.
[24] H. Tao *et al.*, “A Newly Developed Integrative Bio-Inspired Artificial Intelligence Model for Wind Speed Prediction, ” in *IEEE Access*, vol. 8, pp. 83347-83358, 2020, doi: 10.1109/ACCESS.2020.2990439.
[25] P. Koopman, “32-bit cyclic redundancy codes for Internet applications,” in *Proceedings International Conference on Dependable Systems and Networks*, 2002, pp. 459–468, doi: 10.1109/DSN.2002.1028931.
[26] J. Li et *al.*, “Internet of things assisted condition-based support for smart manufacturing industry using learning technique,” *Computational Intelligence*, vol. 36, no. 5, pp. 1-18, 2020, doi: 10.1111/coin.12319.
[27] S. Q. Salih, “A new training method based on black hole algorithm for convolutional neural network,” *Journal of Southwest Jiaotong University*, vol. 54, no. 3, pp. 1–12, 2019, doi: 10.35741/issn.0258-2724.54.3.22.
[28] S. A. Sahy, S. H. Mahdi, H. M. Gheni, and I. Al-Barazanchi, “Detection of the patient with COVID-19 relying on ML technology and FAST algorithms to extract the features,” *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 5, pp. 2886–2894, 2022, doi: 10.11591/eei.v11i5.4355.
[29] Z. A. Jaaz, I. Y. Khudhair, H. S. Mehdy, and I. A. Barazanchi, “Imparting full-duplex wireless cellular communication in 5G network using apache spark engine,” in 2*021 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2021, pp. 123–129, doi: 10.23919/EECSI53397.2021.9624283.
[30] I. A. -Barazanchi, H. R. Abdulshaheed, S. A. Shawkat, and S. R. B. Selamat, “Identification key scheme to enhance network performance in wireless body area network,” *Periodicals of Engineering and Natural Sciences*, vol. 7, no. 2, pp. 895–906, 2019, doi: 10.21533/pen.v7i2.606.
[31] Y. Zheng *et al.*, “Forecasting fine-grained air quality based on big data,” in *Proceedings of the 21th SIGKDD conference on Knowledge Discovery and Data Mining*, 2015, pp. 1–7.
[32] I. A. Barazanchi *et al.*, “Blockchain technology-based solutions for IOT security,” *Iraqi Journal for Computer Science and Mathematics*, vol. 3, no. 1, pp. 53–63, 2022, doi: 10.52866/ijcsm.2022.01.01.006.
[33] S. A. Shawkat, K. S. L. A. -Badri, and I. A. Barazanchi, “Three band absorber design and optimization by neural network algorithm,” *Journal of Physics: Conference Series*, vol. 1530, no. 1, pp. 1–7, 2020, doi: 10.1088/1742-6596/1530/1/012129.

BIOGRAPHIES OF AUTHORS

Haider Hadi Abbas received the B.Sc. degree in Electronics and Communication Engineering from Baghdad University in 1994, Baghdad, Iraq, the M.Sc. degree in Electronics Engineering from University of Technology in 1997, Baghdad, Iraq, and Ph.D. degree in Communications Engineering from Baghdad University, Baghdad, Iraq in 2001. He has been an assistant professor of Data and Information Security since 2014. He has authored more than 25 refereed journal and conference papers. His research interest includes data and information security, machine learning, internet of things, and cloud related issues. He can be contacted at email: haider.hadi@muc.edu.iq.

Poh Soon JosephNg graduated with a Ph.D. (IT), Master in Information Technology (Aus), Master in Business Administration (Aus) and Associate Charted Secretary (UK) with various instructor qualifications, professional certifications and industry memberships. With his blended technocrat mix of both business senses and technical skills, has held many multinational corporation senior management positions, global posting and leads numerous 24×7 global mission-critical systems. A humble young manager nominee twice, five teaching excellence awards recipient, numerous research grants, hundreds of citations and mentored various student competition awards recipient. He has appeared in live television prime time cybersecurity talk show and overseas teaching exposure. His current researches are on strategic IT infrastructure optimization and digital transformation. He can be contacted at email: joseph.ng@newinti.edu.my.

Ahmed Lateef Khalaf received his B.Sc. engineering degree (Control and Systems Engineering) from University of Technology, Iraq (2001) and M.Sc. engineering degree (Computer Engineering) from Middle Technical University, Iraq (2008). He did his Ph.D. research at Universiti Putra Malaysia, Malaysia (2018), in the area of optical sensor based on nanomaterials for chemical sensing applications. Currently, he is a senior lecturer at the Department of Communication Technology Engineering, College of Information Technology, Imam Ja'afar Al-Sadiq University, Baghdad, Iraq. His main research interests are fiber optics sensors, optical chemical sensors, nanomaterials, and computer engineering. He can be contacted at email: ahmedlateef80@gmail.com.

Jamal Fadhil Tawfeq received bachelor in Science of Physics from Mustansiriyah University, Iraq, Baghdad, master of science in Computer Science from University of Technology, Department of Computer Science, Iraq, Baghdad, and Ph.D. in Computer Science in Information Technology “semantic web” form University of Technology, Department of Computer Science, Iraq, Baghdad. He was lecturer in Nahrain University, Computer of science. He was head of Department of Computer Engineering, Madenat Alelem University College. Now, he is associate dean of the College of Engineering Technology, Al-Farahidi University, Baghdad, Iraq. His research interests are software engineering, semantic web developing, metadata, knowledge representation, and database. He can be contacted at email: j.tawfeq@uoalfarahidi.edu.iq.

Ahmed Dheyaa Radhi obtained a bachelor’s degree in Information Technology from the Department of Software at the College of Information Technology at Babylon University, Iraq in 2013. He received a master's degree in Information Technology from the same college in 2018. Currently, he work as a lecturer at the Faculty of Pharmacy, Al-Ameed University in Iraq, as well as in charge of the systems and software division at the university. His current interests are programming, artificial intelligence, website design, databases, and server management. He can be contacted at email: ahmosawi@alameed.edu.iq.