ISSN 2278-3091

Volume 9, No.1.3, 2020

International Journal of Advanced Trends in Computer Science and Engineering

Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse0791.32020.pdf

https://doi.org/10.30534/ijatcse/2020/0791.32020

An Analysis of Standardized Data for Fog Computing Storage Capacity using Non-Relational Database

Mohd Hariz Naim @ Mohayat[1], Jasni Mohamad Zain[2], Kamarularifin Abd Jalil[3], Lee Mei Ph'ng[4], Lizawati Salahuddin[5], Mohd Hakim Abdul Hamid[6], Halizah Basiron[7]

[1,5,6,7]Centre for Advanced Computing Technology C-ACT, Fakulti Teknologi Maklumat dan Komunikasi,
[4]Centre for Technopreneurship Development C-TED, Center for Language and Human Development,
Universiti Teknikal Malaysia Melaka Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia.
[2,3]Department of Computer Technology and Network, Fakulti Sains Komputer dan Matematik,
Universiti Teknologi Mara, Shah Alam, Malaysia.
mohdhariz@utem.edu.my, jasni@tmsk.uitm.edu.my, kamarul@tmsk.uitm.edu.my, lizawati@utem.edu.my, mohdhakim@utem.edu.my, halizah@utem.edu.my

ABSTRACT

Computer applications nowadays relies on physical storage either to store the computation information or for the application itself. As the application become more complex and being used day after day, the data will be growing causing issue with lack of free space especially in fog computing where the storage resource is limited. Despite increasing disk storage or even migrating the data into the cloud would resolve this issue but this would also increase the overall cost as well. Thus, the objective of this paper is to analyse the difference between non-relational database with relational database in term of storage capacity. First the data from relational database will be taken and converted into a standard format and later turned into non-relational database. The result from the analysis will provide motivation for proposing database storage to be implemented in fog computing environment.

Key words: Fog Computing, Non-Relational Database, Standardized Data.

1. INTRODUCTION

With the mass usage of software applications, the amount of data that is received and send could become countless of terabyte[1]. According to research done by [2], the number of devices on the internet become more than 50 billion devices by the year 2020 and the number will keep on increasing day by day together with the popularity of Internet of Things(IoT) [3] that is connected though via network . This is supported by [4] where the author emphasize that there are 16 million wireless network devices that only make small fraction of network devices around the world. A study performed by [5] also highlight the issue of storage technology in Big Data for storing data in vast amount is becoming serious issue. These massive of devices scattered around the edge of internet are capable to generate zettabytes of data which raise the issue of how will the gigantic data is stored? The most widely use database to store information in the web is via relational database [6] such as Oracle and MySQL which sometime known as relational database management system (RDBMS).

The RDBMS will store the information in the form of tables, rows and columns or also known as schema [7] which refers to the design or conceptual how the data will be stored in the physical storage. Regardless of different types of database, all of them should contain physical files residing on server hard disk storage and those files will keep on growing and taking spaces as it keeps on receiving data from applications from time to time [8]. The schema will then be saved in physical storage in term of files with certain extension depending on type of database being used for example a MySQL database will have files with extension of .ibd and .frm which refer to a certain table that is declared in the MySQL. This means if there are multiple tables that are declared in the database, then it should have multiple .ibd and .frm files and each of the file will allocate certain size in the hard disk or physical storage. Theoretically, the bigger the data, the larger the physical files that will allocate in hard disk storage.

As data becoming complex and growing, there is a need for software applications to save and retrieve data on the edge of network without having to depend on the type of database [1]. In addition, the vulnerabilities face by RDMBS such as SQL injection [9] would make the data residing on both cloud and fog devices exposed to data unavailability and interruption. Instead, the data would rather being stored in plain text file with standard key-value format [10] which can be read by any devices from different platform and hardware [11] that is known as non-relational database. The data in the file can be stored or transmitted from one device to other devices or it can also send to the central cloud server. This characteristic of database is suitable to be used by fog computing devices such as Raspberry Pi or Mobile devices that has similar concept with research done by [12].

This paper is structured into five sections as follows; Section 2 will explain the related works that involve in this research while Section 3 will describe the methodology for the conversion of data from relational database to non-relational database as Section 4 will discuss on the result as well as the analysis from the result. Lastly the Section 5 will conclude the research as well as highlight the significant of the result and the future recommendation.

2. RELATED WORKS

In this section, we will explain on the type of storage that are available to be implemented in fog computing. In this paper we will categorized two types of distributed database storage as follow; relational database [7] and non-relational database [13].

2.1. Relational Database

The relational database is a type of database that stores data in a table and information should be normalized [14] which mean the information must be separated in different table to ensure data is not redundant and also to ensure data integrity. Each table could be linked together with a foreign key [7]. Through the foreign key, the data from two or more table could be joined when performing query to fetch back the information.

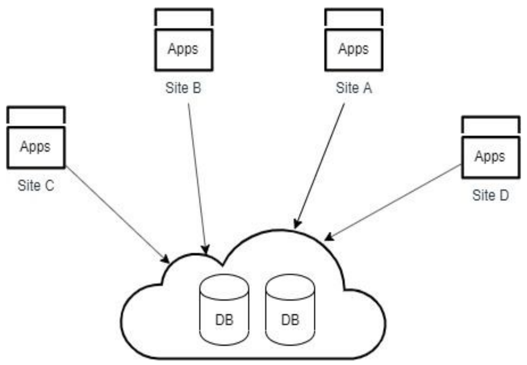


Figure 1: The Centralized Cloud Database

Information should be correctly filtered in order to store the data in the correct table thus, a database design is crucial [7] to identify which data belong to which table. The design could be complex depending on functionality and modules of the application and even two or more application could be sharing the same database. The database could be parked on cloud hosting as shown in Figure 1 or it can be installed on certain machine which will become the server. The server would become distributed database as they have multiple copies of database as depicts in Figure 2. Here, the database servers are deployed at the same site with application server. If the server is not park on the cloud, the server should be equipped with high CPU processing and larger storage to store all the information.

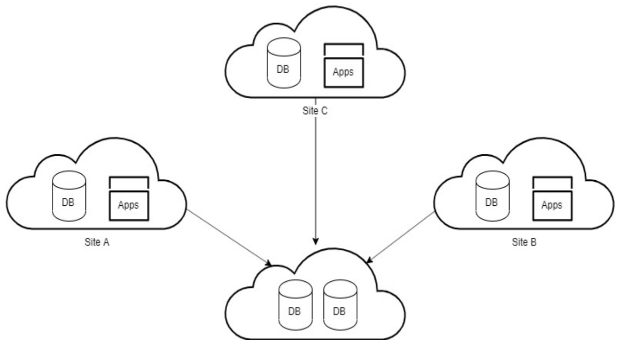


Figure 2: The Distributed Database

Most of relational database use Structured Query Language (SQL) in order to perform Create, Read, Update and Delete information as exemplified in Figure 3. However, if the database containing humungous of data, there could be issue with the performance [8] as the server could take longer time to run required query especially through complex and nested query. Relatively, a bigger data would also cause the database file size increase. For MySQL database, the file is the .ibd file [15] which is file type associated with InnoDB storage engine by Oracle Corporation and this file cannot simply be opened by other software or tool except MySQL InnoDB engine compiler. Another example of database from Microsoft SQL Server (MSSQL) would also contain physical file with extension .mdf and .ndf [16] where these file also taking hard disk spaces as more data is being stored into the database.

```
SELECT * FROM TBL_USER
INSERT INTO TBL_USER VALUES (VALUE1,VALUE2)
UPDATE TBL_USER SET COLUMN COL1 = VAL1
```

Figure 3 The example of SQL statement

2.2 JavaScript Object Notation and Non-Relational Database

Non-relational database is another type of data storage technique that does not have relationship with other data [17] and the data is not store in a view of row and column. Instead each of data value is paired with a key, for example, if there are 10 data value, then it should be paired with 10 different keys. The data is usually in presented in standard arrangement of either in format extensible mark-up language (XML) or in format of JavaScript Object Notation (JSON) as exampled in Figure 4. Here, there are three values and each will be paired with key.

Both of the format can be stored as a plain text file without having to rely on specific engine or compiler which is suitable to be placed in devices of different hardware and operating system[18]. A research done by [14] who also focus on storing data in plain text having format of comma separated value(CSV), however the author discover that CSV file type even though have smaller size but it is much complex to perform information retrieval and eventually take more processing resources such as memory and processor. A research done by [19] where the authors mentioned about using Microsoft Excel with enhanced user interface to perform query for users that do not familiar with SQL statement. This however, would need to depend on specific driver called ActiveX Data Object (ADO) which not every devices are able to be installed with.

In term of file size, the standardized data format would depend on number of characters that present in the file [14]. This means the more characters it contains; the number of file size will also increase. From here, it is clear that XML format generates more characters as compared to JSON format due to open and close tag as depicts in Figure 4. Since the data is paired with a key, the process of data retrieval could be simplified and applications may access the data by referencing the key as opposed by CSV where applications need to loop through many lines of data. In additions, the technique proposed by [14] also prone to logical error due to misinformation to map the information to which data.

```
<user>                           {
<u_fname>John</u_fname>           "u_fname":"John",
<u_lname>Doe</u_lname>           "u_lname":"Doe",
<u_gender>Male</male>            "u_gender":"Male"
</user>                          }
```

Figure 4: Example of XML(left) and JSON(right)

2.3 Fog Computing Technology – Shifting the Paradigm

According to research done by [20], the term fog computing is referred as the moving the computation process from the cloud to the edge of network where IoT devices have the capabilities to perform process such as logical process for detecting and sending notification direct to user without having to wait for cloud processing. Another researcher mention that fog computing is another new computing paradigm that extends the usual cloud computing service run at the edge of network [2] that capable to provide communication, computation and storage services without relying on the cloud. In other term, the fog devices have their own resources such as CPU, memory, storage and even operating system to perform similar task with cloud servers.

By running the computational process on the edge of network, it could benefit from several aspects especially in term of bandwidth and network latency [21] as there is less communication required towards cloud servers which proved to be effective [22] whenever the client location are distance away from central cloud servers. In addition, the processing done at fog devices could reduce the workload and save cloud resources as only consumption ready information [23] may be transmitted to the cloud for further processing and information dissemination.

The challenge with fog computing however would relate with the hardware capabilities in term of storage and processing power [20] that may not be same level with the cloud for handling humongous volumes of data. In addition, the issue with power supply and depending on batteries as source of power also limiting the capabilities of fog and IoT devices [1], [23] as fog devices are distributed and scattered around the edge of network restricting the duration and number of tasks that able to perform. The limitation of data storage [23] has proven that the fog devices are restricted to certain amount of data and then would need to perform data cleanup [24] by deleting old data that is no longer required in order to continue the computational task. This has eventually contributed to another issue with important old data that still need to be stored for future refences as these data could be massive and keep on consuming the storage resources.

3. METHODOLOGY

3.1 Database file Selection

In this section, the step to run and test the file conversion will be explained in detail. First, we will take a sample relational database from MySQL where it consists of several tables. For the sake of experimentation, we will going to take three tables from a database and each of the table is labelled with Table_A, Table_B and Table_C. Due to restriction in personal data protection act, the content of the data will not be revealed only the general information of database will be explained. The database in summary, is a database for a sport application that is developed in web-based application. The reason the database is selected as it will simulate the database that will be deployed on edge device network which later be used for middleware in future work.

The database is a MySQL database running on Windows platform. Here, we open through the folder named ***mysql*** installation folder, inside the folder it should have a folder named ***data*** and then there are other folders that resemble the database entity. We select three identical databases having similar schema and then the .ibd file size will be taken note. By right the .ibd files should tally with the database conceptual design.

3.2 Database Table Information

In this experimentation, the number or rows will be varied while the number of columns will be constant at 8 columns. Table_A consist of 4098 rows of data with .ibd file size of 352 kilobyte(kb) whereas Table_B consist of 3276 rows of data taking 288 kilobyte(kb) of file size. The Table_C on the other hand consist of 3217 rows of data consuming 320 kilobyte(kb) of storage size. Each of the table file will be converted to comma-separated (csv) first before turn into non-relational JSON format.

The data type for each of the column are presented as shown in Table 1. The table in general is the bridge table that store user information and the score obtained by the user thus it consists of a foreign key to relate the data with other table. Since the table is a bridge table, the table are mean to have many-to-many relationship with other tables. In this experiment we do not take other physical files of the interconnected tables yet.

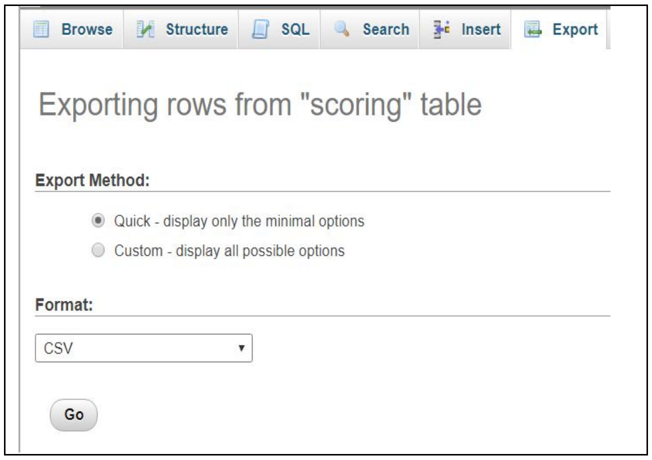
Table 1: The Design of the Table

Column Name	Data Type	Data Length
user_id	varchar	4
range_id	varchar	255
round_no	Int	11
score_mark	Int	11
count_X	Int	1
count_10	Int	1
count_9	Int	1
count_M	Int	1

3.3 Conversion Procedure

The reason why conversion to csv is needed is because the .ibd file is not readable by any kind of compiler other than MySQL itself thus it needed to be exported first as a plain readable format which is the csv by using the feature in MySQL as shown in Figure 5. For the conversion from the csv, we have developed a tool that read each of the csv file according to the separated delimiter which will then arrange the data according to key-value of JavaScript Object Notation.

Although there are available file conversion tools to convert the CSV files but for some reason the available existing tools generate unnecessary data and characters such as date of generated data and some tools even has restricted number of data that is generated. As for the reason, we prefer to develop on our own tools where later we can enhance the tools for the future works. The tool that has been developed will generate a JSON array consisting of thousands of JSON objects. All the JSON formatted data will then be arranged a single line file which is the minimized JSON file that has smaller size.



Exporting rows from "scoring" table

Export Method:

● Quick - display only the minimal options
○ Custom - display all possible options

Format:

CSV

Go

Figure 5: MySQL CSV Conversion Tools

The process flow of the experimentation is visualized in Figure 6 starting by selecting the table to be exported into csv which is generated by tool provided by MySQL. Then, we took the csv file to the developed tool to turn the csv into json file. The process is repeated until all the 3 files from the database has been converted into json. All the three files are compared, and the file size is recorded in the Table 2. The result and the analysis of the file will be discussed in the next section.

4. RESULT AND DISCUSSIONS

From the conversion that had been done in previous section, we have compared all the three files size and we presented it on Table 2. For Table_A, the original .ibd file has size of 352 kilobytes and then having size of 84 kilobytes when converted to CSV while the JSON file has 523 kilobytes. For Table_B, it has been discovered that the .ibd file has 288 kilobytes and 67 kilobytes for its CSV file whereas the JSON file has 456 kilobytes. Likewise, for Table_C which has 320 kilobytes for its .ibd file and has 62 kilobytes and 344 kilobytes for its JSON file. All the JSON files has been truncated for any empty spaces to minimize the size as smallest as possible.

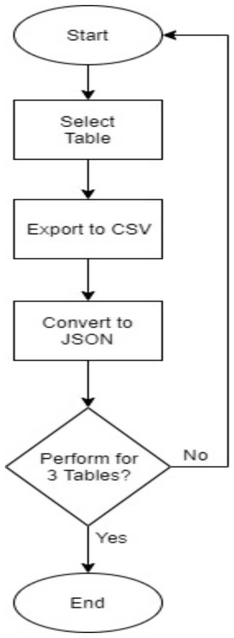


Figure 6: The process flow of the experimentation steps

Table 2: Result of the File Size

File Size(kb) / Table	IBD	CSV	JSON	No of Rows
Table_A	352	84	523	4380
Table_B	320	67	456	3540
Table_C	288	62	394	3300

4.1 Analysis of Result

It has been discovered from the experimentation result that the JSON file is larger due to additional characters that are generated which is the key for each of the value. This means that each of the value should be paired with a key and since there are thousands of rows and each of row has 8 columns, making the key that is generated is multiplied by eight as resulted in Table 3.

Surprisingly, even though there are thousands more characters that are generated, the percentage of file size increment is not significant. The increasing of size is only 171 kilobytes for Table_A, 168 kilobytes for Table_B and 24 kilobytes of file for Table_C, making the percentage of increase are 48.58%, 58.33% and only 36.80%. For Table_C, it has been observed that the data for each row is much smaller in length as compared to other table which explain why the additional size increment is significantly lower.

From the result, the CSV file showed less file size as there are less character as compared to JSON as CSV file does store the key for each value and each value is only separated by comma and this proves the study made by [14]. Although in the result showing the CSV file has much lower file size, it still has flaws in term of complex information retrieval where much resource in term of memory and CPU utilization are required.

The size of JSON file is indeed larger than the MySQL tablespace .ibd file. This mainly due to MySQL has formatted the table database file using its own compiler and only certain software is able to read the file format which is not suitable to be deployed in multiple platform of different hardware specifications that do not support the needed software.

As for the CSV file, the size is substantially small but as for the processing of information could take more resources. For devices that have limited resource of memory and CPU, this could contribute to the bottleneck of performance. Plus, the absence of key to map the value will make computational process much complex and could lead to inaccuracy of information retrieval.

Table 3: The Key Generated in the JSON file

Table	No of Keys	Additional Size(kb)	Percentage File Increase(%)
Table_A	35040	171	48.58
Table_B	28320	168	58.33
Table_C	26400	106	36.80

JSON file format on the other hand is just a standard text file that could be placed in any devices with storage capability. In conjunction to that, fog computing devices could vary from different hardware and platform, also the hardware could be lower specifications of memory and processing power. Therefore, a standard file format such as JSON seems suitable to be the storage file for fog computing

The Figure 7 depicts the analysis of the three tables involve in the experimentation where the percentage of increase show insignificant amount of increment even though the number of key generated has multiplied by eight times. This eventually will indicate that with the amount of data increase, the number of data storage size consumed will not be affected much and only taking about 50% less than the total number of data in a certain table.

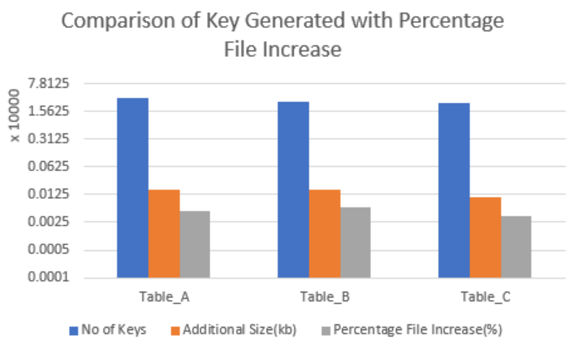


Figure 7: Graph Analysis of Three Different Data Table

5. CONCLUSION AND FUTURE WORKS

In this paper, the comparative analysis of the file size has been presented where we have compared three different file formats as an option to be deployed in fog devices as data storage. It can be concluded that the smallest file size is the CSV file whereas the biggest file is the JSON file format. The CSV file however would suffer from complexity when performing computation task such as information retrieval as the data is just separated by a delimiter. It also may lead to inaccurate information retrieval as the data is not mapped to other relational data.

As more row of data is being stored into the database, the size of JSON file format is not affected much by the increasing number of keys for each value. In fact, the CPU and memory resource may also reduce during information processing and retrieval as the tasks are simplified through the usage of key value in the JSON file. The file size reduction would be advantageous to be used in fog computing as it will provide more data storage to contrast with fog device storage limitations. Here, we can conclude that fog computing has the trade off between CPU and memory resources versus storage resource in optimizing the file size reduction.

In the future work, we will try to reduce the JSON file size by implementing certain file compression technique such as lossless compression or using distributed file system technique to share file storage among other fog computer nodes. Another issues and challenge with compression file technique is to reduce the overhead of CPU and memory resource during information retrieval. This is due to the limitation with fog devices that have capabilities not as par with cloud devices and having a heavy computational task would mean slower latency for application running on fog devices.

ACKNOWLEDGEMENT

This research has been supported and funded by UTeM through short term grant with references PJP/2019/PBPI(9D)/S01698. The author would like to express gratitude to Center for Technopreneurship (C-TED) Center for Languages and Human Development, Center of Advanced Computing Technology (C-ACT) Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka and Center of Computer Technology and Network (CTN) Faculty of Computer Science and Mathematics Universiti Teknologi Mara, Malaysia for offering facilities in term of research accommodations.

REFERENCES

1. I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara. **A review of edge computing reference architectures and a new global edge proposal**, *Future Generation Computer Systems*, vol. 99, no. 2019, pp. 278–294, 2019, doi: 10.1016/j.future.2019.04.016.
2. P. Hu, S. Dhelim, H. Ning, and T. Qiu. **Survey on fog computing: architecture, key technologies, applications and open issues**, *Journal of Network and Computer Applications*, vol. 98, no. September, pp. 27–42, 2017, doi: 10.1016/j.jnca.2017.09.002.
3. N. Muhammad and J. M. Zain. **Conceptual Framework for Lightweight Ciphertext Policy-Attribute Based Encryption**, *Journal of Computers*, vol. 4, no. 1, pp. 237–245, 2019.
4. N. Baharudin, F. H. M. Ali, M. Y. Darus, and N. Awang. **Wireless intruder detection system (WIDS) in detecting de-authentication and disassociation attacks in IEEE 802.11**, *2015 5th International Conference on IT Convergence and Security, ICITCS 2015 - Proceedings*, pp. 1–5, 2015, doi: 10.1109/ICITCS.2015.7293037.
5. M. H. A. Hamid, S. N. M. Mohamad, A. Idris, and Z. Zahriladha. **Satisfiable Integer Programming Algorithm (SIP-DIPC)**, vol. 11, no. 08, pp. 315–322, 2019.
6. K. Kaur and H. Singh. **Distributed database system on web server : A Review**, *International Journal of Computer Techniques*, vol. 3, no. 6, pp. 12–16, 2016.
7. K. K. Ezéchiel, S. Kant, and R. Agarwal. **A systematic review on distributed databases systems and their techniques**, *Journal of Theoretical and Applied Information Technology*, vol. 97, no. 1, pp. 236–266, 2019.
8. J. Klein and I. Gorton. **Runtime performance challenges in big data systems**, *WOSP-C 2015 - Proceedings of the 2015 ACM/SPEC Workshop on Challenges in Performance Methods for Software Development, in Conjunction with ICPE 2015*, pp. 17–22, 2015, doi: 10.1145/2693561.2693563.
9. T. F. Abdul Rahman, A. G. Buja, K. A. Jalil, and M. A. Fakariah. **SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm**, *Journal of Computers*, vol. 12, no. 2, pp. 183–189, 2017, doi: 10.17706/jcp.12.2.183-189.
10. B. Mouad and B. Abdessamad. **RDFMongo: A MongoDB Distributed and Scalable RDF management system based on Meta-model**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 3, pp. 734–741, 2019, doi: 10.30534/ijatcse/2019/62832019.
11. M. H. Naim, M. K. A. Ghani, A. S. H. Basari, B. Aboobaider, L. Salahuddin, and W. N. A. Rashid. **Synchronization technique via raspbery Pi as middleware for hospital information system**, *Advances in Intelligent Systems and Computing*, vol. 734, pp. 262–271, 2018, doi: 10.1007/978-3-319-76351-4_27.
12. K. A. Jalil and N. S. Kamal. **A New Technique for Protecting Server Against MAC Spoofing via Software Attestation**, *Advanced Science Letters*, vol. 21, no. 10, pp. 3019–3023, 2015, doi: https://doi.org/10.1166/asl.2015.6482.
13. A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena. **NoSQL databases: Critical analysis and comparison**, *International Conference on Computing and Communication Technologies for Smart Nation, 2017*, pp. 293–299, 2018, doi: 10.1109/IC3TSN.2017.8284494.
14. S. Adaszewski. **Mynodbcsv: Lightweight zero-config database solution for handling very large CSV files**, *PLoS ONE*, vol. 9, no. 7, pp. 1–8, 2014, doi: 10.1371/journal.pone.0103319.
15. **InnoDB Tablespace SDI Extraction Utility**. https://dev.mysql.com/doc/refman/8.0/en/ibd2sdi.html (accessed Nov. 20, 2019).
16. **MSSQL Database Files and Filegroups**. https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups?view=sql-server-ver15 (accessed Nov. 23, 2019).
17. E. Botoeva, D. Calvanese, B. Cogrel, M. Rezk, and G. Xiao. **OBDA over non-relational databases**, *CEUR Workshop Proceedings*, vol. 1644, 2016.
18. K. Maeda. **Performance evaluation of object serialization libraries in XML, JSON and binary formats**, *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012*, pp. 177–182, 2012, doi: 10.1109/DICTAP.2012.6215346.
19. A. E. Karrar, M. Fadl, and I. Fadl. **SpreadDB: Spreadsheet-Based User Interface for Querying and Updating Data of External Databases**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 7, no. 1, pp. 1–5, 2018.
20. K. Bilal, O. Khalid, A. Erbad, and S. U. Khan. **Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers**, *Computer Networks*, vol. 130, no. 2018, pp. 94–120, 2018, doi: 10.1016/j.comnet.2017.10.002.

21. C. Li, J. Zhang, Y. Chen, and Y. Luo. **Data prefetching and file synchronizing for performance optimization in Hadoop-based hybrid cloud**, *Journal of Systems and Software*, vol. 151, pp. 133–149, 2019, doi: 10.1016/j.jss.2019.02.007.
22. Y. Aldwyan and R. O. Sinnott. **Latency-aware failover strategies for containerized web applications in distributed clouds Cloud Failover Techniques :**, *Future Generation Computer Systems*, vol. 101, pp. 1081–1095, 2019, doi: 10.1016/j.future.2019.07.032.
23. M. Tortonesi, M. Govoni, A. Morelli, G. Riberto, C. Stefanelli, and N. Suri. **Taming the IoT data deluge: An innovative information-centric service model for fog computing applications**, *Future Generation Computer Systems*, vol. 93, pp. 888–902, 2019, doi: 10.1016/j.future.2018.06.009.
24. J. Baker *et al.* **Megastore: Providing scalable, highly available storage for interactive services**, *CIDR 2011 - 5th Biennial Conference on Innovative Data Systems Research, Conference Proceedings*, pp. 223–234, 2011.