International Journal on Advanced Science Engineering Information Technology

Vol.11 (2021) No. 3
ISSN: 2088-5334

Comparison of JSON and XML Data Formats in Document Stored NoSql Database Replication Processes

Rianto[a], Muhamad Arsyad Rifansyah[a], Rohmat Gunawan[a], Irfan Darmawan[b], Alam Rahmatulloh[a,*]

[a] *Department of Informatics, Siliwangi University, Tasikmalaya, Indonesia*
[b] *Department of Information System, Telkom University, Bandung, Indonesia*
Corresponding author:[*]*alam@unsil.ac.id*

***Abstract*— The current trend of solutions in storing large amounts of data is using the NoSQL Database. A document stored is one type of NoSQL database that uses the JavaScript Object Notation (JSON) and eXtensible Markup Language (XML) data formats in data storage. High Availability database is significant to support cloud-based applications and services. Replication is one solution to maintain the consistency of source data and target data. This study aims to determine the performance of JSON and XML data formats in the document stored NoSQL Database replication process. In this study, ArangoDB, RethinkDB, and MongoDB were chosen for use in the trial process of replication from master-server to slave-server with two different data formats, JSON and XML. Data transfer, CPU usage, memory usage, and execution time are measured in each trial. Based on research and experiments that have been carried out, the JSON data format consumes bandwidth with an average value smaller than the XML data format; this occurs in MongoDB, CouchDB, and RethinkDB. In CPU usage, JSON data format, on average, consumes less CPU compared to the XML data format. This is the case with MongoDB. While on CouchDB and RethinkDB, the average CPU usage for XML and JSON data formats does not show a significant difference. The average memory usage for the JSON data format is smaller than the XML data format. The average execution time of the XML data format a little faster than the JSON data format.**

***Keywords*—Data; JSON; NoSQL; replication; XML.**

Manuscript received 4 Apr. 2020; revised 22 Aug. 2020; accepted 29 Nov. 2020. Date of publication 30 Jun. 2021.

I. INTRODUCTION

The traditional relational database management system is continuously being replaced by NoSQL data storage due to the increasing demand for big data applications [1]-[4]. NoSQL databases have become an increasingly popular solution for handling unstructured data[5], provide high performance and scalability [6], can be used to store large amounts of data, and work faster than a relational database [7], [8]. The document stored is one type of NoSQL database that is available at this time [9]-[12]. The increasing use of Web 2.0 applications has driven growth in volume, speed, and variations in data sources beyond the limits of relational database capabilities [13]. The cloud serves NoSQL data storage to overcome this problem and provides a replication mechanism to ensure fault tolerance, high availability, and increased scalability [14], [15].

High Availability (HA) database is significant to support cloud-based applications and services. Replication is one solution that can be used to solve this problem [16]-[19]. Replication is a process of storing data at more than one site or node. Replication can be used to maintain the consistency of source data and target data [20]. Replication must consider modeling data to achieve optimal performance [14]. Java Script Object Notation (JSON) and eXtensible Markup Language (XML) are two types of data formats commonly used in NoSQL document stored [10], [21]. The JSON data format is effective in data size and XML Web Application Programming Interface (API) response times. Still, for some applications that require the delivery of multiple heterogeneous XML formats, it offers better support [22]. Using different data formats can affect application performance [23].

Several experiments related to the comparison of JSON and XML data formats have been carried out in previous studies. Research conducted by [22], try to compare the method of sending data using JSON and XML formats. Data size and response time are the two main parameters measured in the experiments in this study. The results show that the JSON data format is more effective in data size and web API response time than the XML data format. However, for

applications requiring complex heterogeneous data structures, the XML format offers better support.

Research conducted by [23], try to compare code, data models, accessing and extracting in JSON and XML data formats. The results of his study revealed that to transfer documents with many different types and data elements, XML is an ideal choice. JSON is more suitable for dynamic web applications and simple data transmission. JSON performance speed is higher than XML because of its simple structure and easy access to data. JSON will not wholly replace XML in the Web area. XML offers more luxurious features so that it has a place in the transfer and validation of documents. JSON is more suitable for data exchange, whereas XML is more suitable for data transmission between servers and web applications, for example, in Ajax calls.

Previous research [24] has compared JSON and XML data formats in web technology. The criteria measured in the experiments in the study include a form of exchange, validity, ease of data process, readability, efficiency, debugging and troubleshooting, ease of data creation, security. His research shows that JSON data formats produce higher values on the criteria: a form of exchange, comfort of data process, efficiency compared to XML. However, for the requirements: validity, readability, debugging and troubleshooting, ease of data creation, XML security is better than JSON. Each criterion was further expanded to sub-criteria: user CPU utilization, system CPU utilization, and memory utilization. The experimental results show that JSON is best in user CPU utilization than XML, whereas in the CPU utilization system, XML performance is better than JSON. For memory utilization, there is no significant difference between JSON and XML.

Other studies related to NoSQL Database replication have been conducted in previous studies, including benchmarking replication in NoSQL datastores [14], asynchronous NoSQL Database replication performance measurement [25], and analysis of replication mechanisms in the NoSQL Database [26]. The impact of replication on Cassandra and MongoDB NoSQL Database performance has been explored [14]. Evaluate the effect of replication compared to clusters that are not replicated to the same size that are hosted in a private cloud environment. Benchmark experiments are carried out in the process of reading and writing heavy workloads with different access distributions and the level of consistency that can be changed. This study's experimental results indicate that replication must be considered in empirical studies and modeling to achieve an accurate evaluation of document-based NoSQL databases' performance.

Asynchronous replication performance measurements were performed on a document stored NoSQL databases [25]. The study discusses NoSQL databases with MongoDB, CouchDB, and CouchBase types. The parameters tested are at the time of execution of the CRUD operation by calculating the average value. Experiments in that study showed CouchDB had a perfect overall performance time on the insert, update and delete queries, whereas MongoDB read questions were faster than other NoSQL databases.

The replication mechanism in the MongoDB NoSQL Database, including Master-Slave and replica sets, has been analyzed [26]. The writing operation's implementation is run on Master; Slave is configured so that it can send out synchronous data synchronous commands to the Master to update the data. The read operation is only implemented on the Master to provide durable consistency, while the reading operation implementation on Slave gives the ultimate flexibility. A replica collection is a group of servers that run Mongod and keep copies of the same data with automatic failover and automatic recovery of node members. His research results show that MongoDB is one of the best NoSQL databases, document-oriented, schema-free, and supports complex data structures and can hold large amounts of data, rich query support, and scalability, and high performance.

Measuring the performance of JSON and XML data formats in a stored database based on the NoSQL replication process is the main objective of this study. MongoDB, ArangoDB, and RethinkDB were selected for use in experiments, which were installed on Master-Server and Slave-Server. Data entry requests are made from clients connected to the Master-Server. The configuration is done on each machine so that the replication process from Master-Server to Slave-Server occurs. The test scenario that will be carried out is by inputting data repeatedly. The data entered by the bears are tested with different amounts of input. These experiments will then be measured and examined with parameters test memory usage, data transfer, and CPU usage. The test data is stored and presented in tables and graphs, used as material for concluding.

II. Material and Method

There are three main stages carried out in this study, namely: system preparation, implementation, measurement.

A. System Preparation

1) Software Preparation: At this stage, each software used in the experiment will be prepared, including the operating system, NoSQL database, and tools. The software specifications used in the experiments in this study are shown in Table 1.

TABLE I
SOFTWARE SPECIFICATIONS

Item	Master-Server	Slave- Server	Client
Operating System	Linux Ubuntu 16.04	Linux Ubuntu 16.04	Linux Ubuntu 16.04
NoSQL Database	MongoDB 4.0.8 CouchDB 2.3.0 RethinkDB 2.3.6	MongoDB 4.0.8 CouchDB 2.3.0 RethinkDB 2.3.6	-
Programming Tools	Python 3	Python 3	Python 3
System Monitoring Tools	Atop 2.2.6 Netatop 1.0	Atop 2.2.6 Netatop 1.0	-

2) Hardware Preparation: At this stage, an identification of each hardware used in the experiment will be performed. The equipment used in the research consisted of Master Server, Slave Server, and Client computers. The hardware

specifications used in the experiments in this study are shown in Table 2.

TABLE II
HARDWARE SPECIFICATIONS

Item	Master-Server	Slave- Server	Client
CPU	AMD E-450 (Zacate) 1.65 GHz	Intel Celeron 2.0 GHz	Intel Celeron B77 1.4 GHz
Memory	DDR3 4 GB	DDR2 2 GB	DDR3 4 GB

3) Test Data: The data used as an experiment in this study is of type string. Data entities inputted are smartphone data with attributes: name, network, display, CPU, memory card, primary camera, battery, price.

4) System Architecture Preparation: At this stage, the system architecture is needed to support the experiments that will be carried out by involving hardware, software, and network configuration. In general, the system of architecture built is shown in Figure 1.

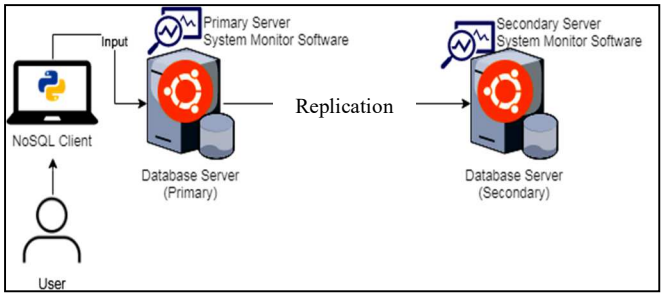

Fig. 1 NoSQL database document stored replication system architecture

Figure 1 shows the general system architecture needed for the NoSQL Database replication process to be performed in an experiment. The NoSQL Client application is installed on users who are connected to a database server that acts as the Master-Server. One-way replication is designed to be able to run from Master-Server to Slave-Server.

B. Implementation

Linux Ubuntu 16.04 is installed on Master-Server, Slave-Server, and Client. NoSQL Database MongoDB 4.0.8, CouchDB 2.3.0, and RethinkDB 2.3.6 are installed on Master-Server and Slave-Server. Atop 2.2.6 and Netatop 1.0 are used to measure data transfer, CPU usage, Memory usage, and execution time installed on Master-Server and Slave-Server. The Python-based NoSQL Client library on the client computer application is used to be able to process query insert data from the client to the Master-Server. The replication process is one-way from Master-Server to Slave-Server. Network transmission media uses UTP CAT 5 cable. Data that has been prepared previously, inputted from the client to the Master-Server. Master-Server and Slaver-Server configuration are done so that the replication process can run. Data input is repeated using different amounts: 2.000, 4.000, 6.000, 8.000, 10.000.

C. Measurement

The results of data transfer measurements, CPU usage, memory usage, and execution time of each trial are recorded in the table.

III. RESULT AND DISCUSSION

A. Replication Configuration

At this stage, a replication configuration for the NoSQL Database will be used. In this experiment,t the Master-Server uses the Internet Protocol Address 192.168.1.1/24, while the Slave-Server uses the Internet Protocol Address 192.168.1.2/24. The replication configuration results in RethinkDB are shown in Figure 2.

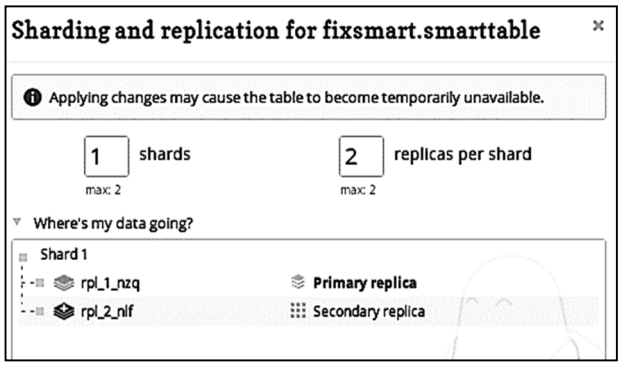

Fig. 2 Configuring replication on RethinkDB

Figure 2 shows the configuration of replication in RethinkDB, which is done through the administration panel.

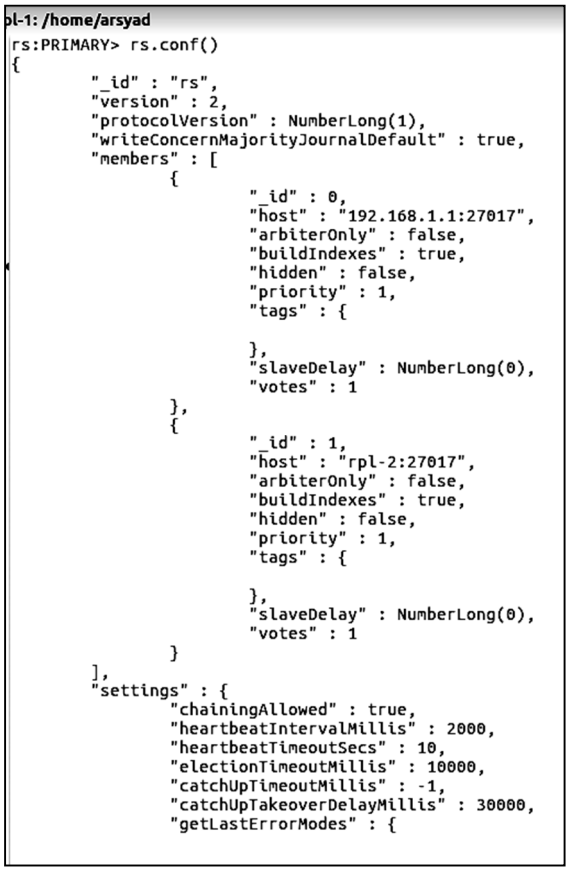

Fig. 3 Configuration of replication in MongoDB

Figure 3 shows the replication configuration in MongoDB that is saved in the rs.conf file. Master-Server is set with _id: 0 and Slave-Server with _id: 1. Port 27017 was chosen for use in the replication process.

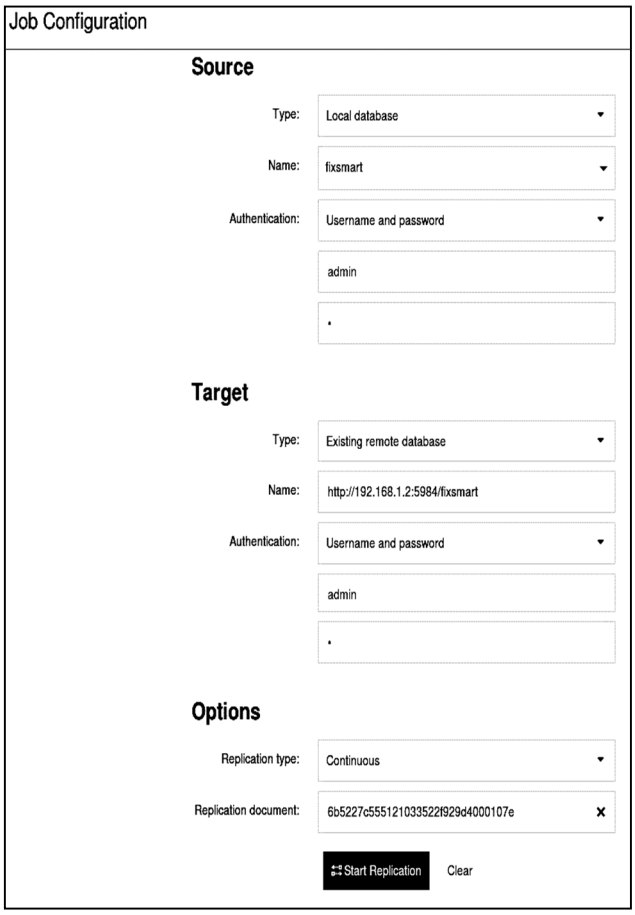

Fig. 4 Configuration of replication in CouchDB

Figure 4 shows the configuration of CouchDB replication performed on the Master-Server. Slave-Server as a replication target uses IP Address 192.168.1.2 with port 5984.

B. Query Preparation

Figure 5 is a display of the insert query on MongoDB on the Master Server with the JSON data format. This stage is the process of preparation and making sure the queries to be used are appropriate.

```
1  import pymongo
2
3  myclient = pymongo.MongoClient("
   mongodb://192.168.1.1:27017/")
4  mydb = myclient["fixsmart"]
5  mycol = mydb["smarttable"]
6  for x in range(10000):
7      mylist = {"_id" : x, "Smartphone": [{"Name" :
       "Asus ZenFone Live (L2)", "Network" : "GSM / HSPA
       / LTE", "Display" : "IPS LCD capacitive
       touchscreen, 16M colors", "CPU" : "Quad-core 1.4
       GHz Cortex-A53", "MemoryCard" : "microSD, up to 1
       TB (dedicated slot)", "MainCamera" : "8 MP or 13
       MP, f/2.0, PDAF", "Battery" : "Non removable
       Li-Ion 3000 mAh battery", "Price" : 4300000}]}
8      x = mycol.insert_one(mylist)
```

Fig. 5 Insert data in MongoDB with the JSON data format

Figure 5 shows the insert process query's source code with the JSON data format applied to MongoDB. 10.000 repetitions are performed using the for-do source code. In Figure 5, lines 7 - 8 show the JSON data format in an array.

```
1  import pymongo
2
3  myclient = pymongo.MongoClient("
   mongodb://192.168.1.1:27017/")
4  mydb = myclient["fixsmart"]
5  mycol = mydb["smarttable"]
6  for x in range(10000):
7      mylist = {"_id": x, "xml" :
       "<Smartphone><Name>Asus ZenFone Live
       (L2)</Name><Network>GSM / HSPA /
       LTE</Network><Display>IPS LCD capacitive
       touchscreen, 16M colors</Display><CPU>Quad-core
       1.4 GHz Cortex-A53</CPU><MemoryCard>microSD, up to
       1 TB (dedicated slot)</MemoryCard><MainCamera>8 MP
       or 13 MP, f/2.0, PDAF</MainCamera><Battery>Non
       removable Li-Ion 3000 mAh
       battery</Battery><Price>4300000</Price></Smartphone>
       "}
8      x = mycol.insert_one(mylist)
```

Fig. 6 Insert data in MongoDB with XML data format

Figure 6 shows the insert process query's source code with the XML data format applied to MongoDB. 10.000 repetitions are performed using the for-do source code. In Figure 6, lines 7 - 8 display the XML data format marked with the opening tag "< >" and the closing tag "< / >" on each data item.

```
1  import couchdb
2  user = "admin"
3  password = "a"
4  couchserver = couchdb.Server("
   http://%s:%s@192.168.1.1:5984/" % (user, password))
5
6  dbname = "fixsmart"
7  db = couchserver[dbname]
8
9  for i in range(10000):
10     db [str(i)] = {"Smartphone": [{"Name" : "Asus
       ZenFone Live (L2)", "Network" : "GSM / HSPA / LTE",
        "Display" : "IPS LCD capacitive touchscreen, 16M
       colors", "CPU" : "Quad-core 1.4 GHz Cortex-A53",
       "MemoryCard" : "microSD, up to 1 TB (dedicated
       slot)", "MainCamera" : "8 MP or 13 MP, f/2.0, PDAF"
       , "Battery" : "Non removable Li-Ion 3000 mAh
       battery", "Price" : 4300000}]}
```

Fig. 7 Insert data in CouchDB with the JSON data format

```
1  import couchdb
2  user = "admin"
3  password = "a"
4  couchserver = couchdb.Server("
   http://%s:%s@192.168.1.1:5984/" % (user, password))
5
6  dbname = "fixsmart"
7  db = couchserver[dbname]
8
9  for i in range(10000):
10     db [str(i)] = {"xml" : "<Smartphone><Name>Asus
       ZenFone Live (L2)</Name><Network>GSM / HSPA /
       LTE</Network><Display>IPS LCD capacitive
       touchscreen, 16M colors</Display><CPU>Quad-core
       1.4 GHz Cortex-A53</CPU><MemoryCard>microSD, up to
       1 TB (dedicated slot)</MemoryCard><MainCamera>8 MP
       or 13 MP, f/2.0, PDAF</MainCamera><Battery>Non
       removable Li-Ion 3000 mAh
       battery</Battery><Price>4300000</Price></Smartphone>
       "}
```

Fig. 8 Insert data in CouchDB with XML data format

Figure 7 shows the insert process query's source code with the JSON data format applied to CouchDB. 10.000 repetitions

are performed using the for-do source code. In figure 7, line 10 display the JSON data format in an array.

Figure 8 shows the syntax of the insert process query with the XML data format applied to CouchDB. 10.000 repetitions are performed using the for-do syntax. In figure 8, line 10 display the XML data format marked with the opening tag "< >" and the closing tag "< / >" on each data item.

```
1  import rethinkdb as rdb
2  r = rdb.RethinkDB()
3  r.connect('192.168.1.1', 28015).repl()
4
5  for x in range(10000):
6      r.db('fixsmart').table('smarttable').insert({'id':
       x, 'Smartphone': [{'Name' : 'Asus ZenFone Live
       (L2)', 'Network' : 'GSM / HSPA / LTE', 'Display' :
       'IPS LCD capacitive touchscreen, 16M colors', 'CPU'
        : 'Quad core 1.4 GHz Cortex-A53', 'MemoryCard' :
       'microSD, up to 1 TB (dedicated slot)',
       'MainCamera' : '8 MP or 13 MP, f/2.0, PDAF',
       'Battery' : 'Non-removable Li-Ion 3000 mAh battery'
       , 'Price' : 4300000 }]}).run()
```

Fig. 9 Insert data in RethinkDB with the JSON data format

Figure 9 shows the syntax of the insert process query with the JSON data format applied to CouchDB. 10.000 repetitions are performed using the for-do syntax. In figure 9, line 6 display the JSON data format in an array.

```
1  import rethinkdb as rdb
2  r = rdb.RethinkDB()
3  r.connect('192.168.1.1', 28015).repl()
4
5  for x in range(10000):
6      r.db('fixsmart').table('smarttable').insert({'id':
       x, 'xml' : '<Smartphone><Name>Asus ZenFone Live
       (L2)</Name><Network>GSM / HSPA /
       LTE</Network><Display>IPS LCD capacitive
       touchscreen, 16M colors</Display><CPU>Quad-core
       1.4 GHz Cortex-A53</CPU><MemoryCard>microSD, up to
       1 TB (dedicated slot)</MemoryCard><MainCamera>8 MP
       or 13 MP, f/2.0, PDAF</MainCamera><Battery>Non
       removable Li-Ion 3000 mAh
       battery</Battery><Price>4300000</Price></Smartphone>
       '}).run()
```

Fig. 10 Insert data in RethinkDB with XML data format

Figure 10 shows the syntax of the insert process query with the XML data format applied to RethinkDB. Ten thousand repetitions are performed using the for-do syntax. In figure 10, line 6 display the XML data format marked with the opening tag "< >" and the closing tag "< / >" on each data item.

C. Query Execution

During the measurement process, each application running has its own process ID so that identification can be made, and measurements are not interrupted by other applications running simultaneously on the machine being used.

Units of Kilobits per second (Kbps) are used to measure data transfer. The data transfer size is obtained based on two parts, namely the Master-Server Output Bandwidth and the Slave-Server Input Bandwidth consumed during the NoSQL database replication process. The data transfer size used when replicating is shown in Figure 11.

Figure 11 is a display in the process of executing the request that is displayed in a column named BANDWI and BWNDWO. BANDWI is the input bandwidth, while BANDWO is the output bandwidth. The CPU used for each replication process experiment is measured in per cent (%). Percentages are obtained based on overall CPU capacity. The rate of CPU used when replicating is shown in Figure 12.

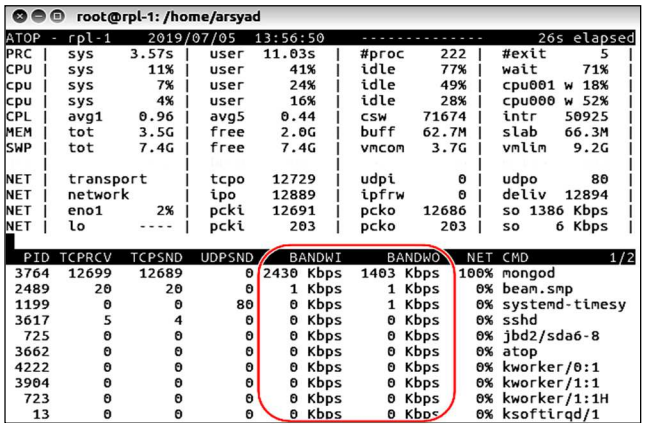

Fig. 11 Measurement of data transfer using ATOP

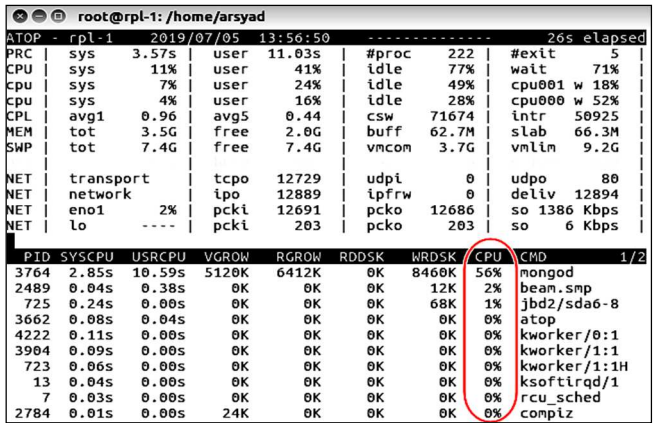

Fig. 12 Measurement of CPU usage using ATOP

In Figure 12, CPUs used are displayed in the CPU column in units of%. Simultaneously, the Memory used during the replication process is measured in Megabytes (MB). Memory measurements are obtained based on the Memory used by the NoSQL database application during the replication process, as shown in Figure 13.

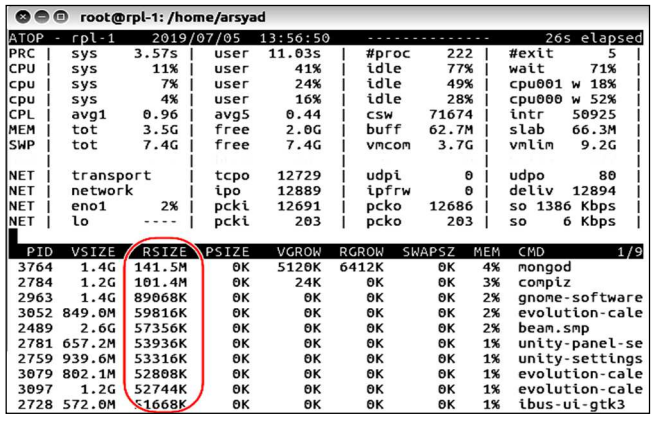

Fig. 13 Measurement of memory usage using ATOP

Figure 13 shows the size of memory usage when the replication process is displayed in the RSIZE column. Each running application has a unique process ID (PID), so it can be identified. Execution time is done by measuring the time

used in the replication process of a document-based NoSQL database server application stored in second (s) units. Execution time is obtained based on the length of time the CPU is used in two parts, namely the CPU System and the User CPU, as shown in Figure 14.

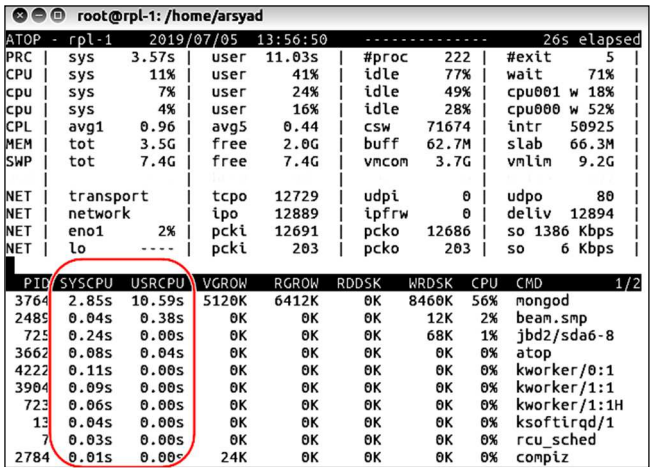

Fig. 14 Measurement of exclusion time using ATOP

Execution time at the replication process in Figure 14 is displayed in the SYSCPU column for System CPU and USRCPU for User CPU.

D. Measurement Results

Experimental data that has been stored in a table is then displayed in graphical form. Data presentation is focused on comparing the use of JSON and XML data formats in the one-way replication process from Master-Server to Slave-server. Measurement of data transfer is divided into two parts, namely the input bandwidth and output bandwidth, shown in Figure 15.

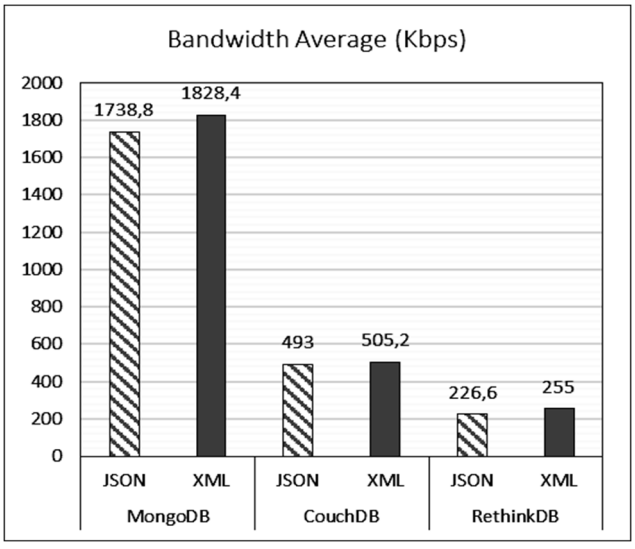

Fig. 15 Average data transfer

Figure 15 shows that the bandwidth consumed for the JSON data format on MongoDB, CouchDB, and RethinkDB with an average of 1.738,8 Kbps, 493 Kbps, and 226,6 Kbps is smaller than the XML data format with an average of 1.828,4 Kbps, 505,2 Kbps and 255 Kbps.

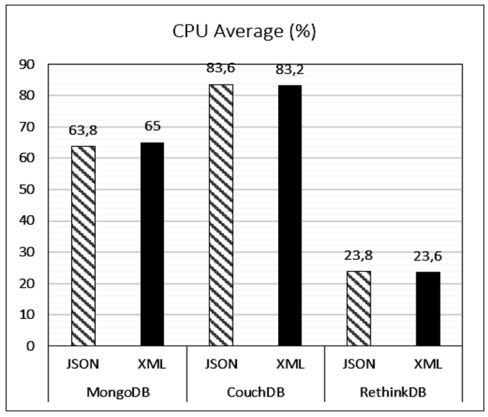

Fig. 16 Average CPU usage

Figure 16 shows that the CPU used for the JSON data format on MongoDB with an average of 63.8% smaller than the JSON data format, with an average of 65%. While on CouchDB and RethinkDB, the average CPU usage for XML and JSON data formats does not show a significant difference.

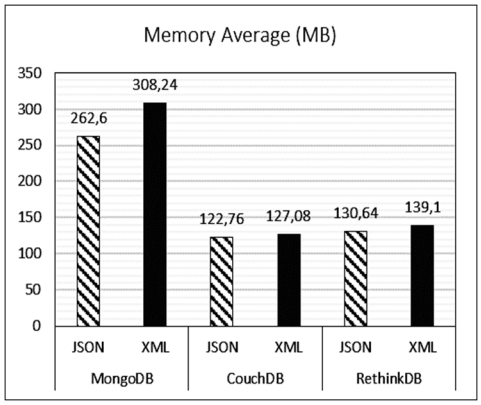

Fig. 17 Average memory usage

Figure 17 shows the average JSON data format memory usage in MongoDB (262,6 MB), CouchDB (122,76 MB) and RethinkDB (130,64 MB) smaller than the XML data format with an average MongoDB (308,24MB), CouchDB (127,08 MB) and RethinkDB (139,1MB).

Figure 18 shows the average execution time of XML data format in MongoDB (10.19 second), CouchDB (475,006 second) and RethinkDB (59,696 second) slightly faster than the JSON data format with MongoDB average (10.324 second), CouchDB (476,966 second) and RethinkDB (60,482 second).

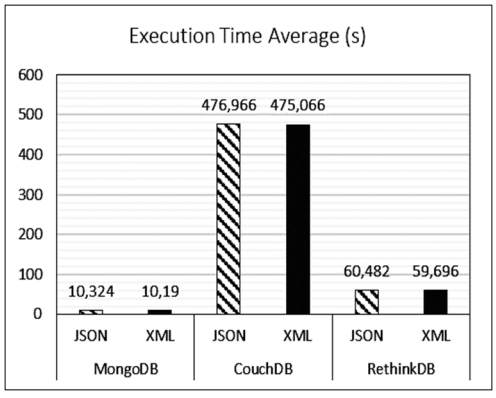

Fig. 18 Average execution time

IV. CONCLUSION

This research results that the JSON data format is known to be able to consume smaller bandwidth when compared to the XML data format, it occurs in all NoSQL databases that have been tested, namely MongoDB, CouchDB, and RethinkDB. For CPU usage, JSON data format, on average, consumes less CPU compared to the XML data format, this is the case with MongoDB. While on CouchDB and RethinkDB, the average CPU usage for XML and JSON data formats does not show a significant difference. The average memory usage for the JSON data format is smaller than the XML data format and for the average execution time of the XML data format a little faster than the JSON data format. Choosing other NoSQL database applications and choosing other replication schemes such as two-way replication are some challenges that can be tried in subsequent studies.

REFERENCES

[1] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaf, "Persisting big-data: The NoSQL landscape," pp. 1–23, 2016, doi: 10.1016/j.is.2016.07.009.

[2] H. Gujral, A. Sharma, and P. Kaur, "Empirical Investigation of Trends in NoSQL-Based Big-Data Solutions in the Last Decade," *2018 11th Int. Conf. Contemp. Comput. IC3 2018*, pp. 1–3, 2018, doi: 10.1109/IC3.2018.8530582.

[3] A. Makris, K. Tserpes, V. Andronikou, and D. Anagnostopoulos, "A Classification of NoSQL Data Stores Based on Key Design Characteristics," *Procedia Comput. Sci.*, vol. 97, pp. 94–103, 2016, doi: 10.1016/j.procs.2016.08.284.

[4] G. Bathla, R. Rani, and H. Aggarwal, "Comparative study of NoSQL databases for big data storage," *Int. J. Eng. Technol.*, vol. 7, no. 2, pp. 83–87, 2018, doi: 10.14419/ijet. v7i2.6.10072.

[5] S. Bjeladinovic, "A fresh approach for hybrid SQL/NoSQL database design based on data structuredness," *Enterp. Inf. Syst.*, vol. 12, no. 8–9, pp. 1202–1220, 2018, doi: 10.1080/17517575.2018.1446102.

[6] C. Gomes, E. Borba, E. Tavares, and M. N. D. O. Junior, "Performability model for assessing NoSQL DBMS consistency," *SysCon 2019 - 13th Annu. IEEE Int. Syst. Conf. Proc.*, pp. 1–6, 2019, doi: 10.1109/SYSCON.2019.8836757.

[7] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," *IEEE Pacific RIM Conf. Commun. Comput. Signal Process. - Proc.*, no. November, pp. 15–19, 2013, doi: 10.1109/PACRIM.2013.6625441.

[8] V. Abramova, J. Bernardino, and P. Furtado, "SQL or NoSQL? Performance and scalability evaluation," *Int. J. Bus. Process Integr. Manag.*, vol. 7, no. 4, pp. 314–321, 2015, doi: 10.1504/IJBPIM.2015.073655.

[9] B. Acharya, A. K. Jena, J. M. Chatterjee, R. Kumar, and D.-N. Le, "NoSQL Database Classification," *Int. J. Knowledge-Based Organ.*, vol. 9, no. 1, pp. 50–65, 2018, doi: 10.4018/ijkbo.2019010105.

[10] A. Davoudian, L. Chen, and M. Liu, "A Survey on NoSQL Stores," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 1–43, 2018, doi: 10.1145/3158661.

[11] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "NoSQL databases: Critical analysis and comparison," *2017 Int. Conf. Comput. Commun. Technol. Smart Nation, IC3TSN 2017*, vol. 2017-Octob, pp. 293–299, 2018, doi: 10.1109/IC3TSN.2017.8284494.

[12] R. Gunawan, A. Rahmatulloh, and I. Darmawan, "Performance Evaluation of Query Response Time in The Document Stored NoSQL Database," in *2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*, 2019, pp. 1–6, doi: 10.1109/QIR.2019.8898035.

[13] M. M. Patil, A. Hanni, C. H. Tejeshwar, and P. Patil, "A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retrieval operations using a web/android application to explore load balancing-Sharding in MongoDB and its advantages," *Proc. Int. Conf. IoT Soc. Mobile, Anal. Cloud, I-SMAC 2017*, pp. 325–330, 2017, doi: 10.1109/I-SMAC.2017.8058365.

[14] G. Haughian, R. Osman, and W. J. Knottenbelt, "Benchmarking replication in Cassandra and MongoDB NoSQL datastores," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9828 LNCS, no. 3, pp. 152–166, 2016, doi: 10.1007/978-3-319-44406-2_12.

[15] W. Hendricks, "Review of NoSQL Data Stores: Using a reactive three-tier application for software developers to achieve a high availability application design architecture," pp. 71–77, 2019.

[16] R. Shrestha, "High Availability and Performance of Database in the Cloud - Traditional Master-slave Replication versus Modern Cluster-based Solutions," no. Closer, pp. 413–420, 2017, doi: 10.5220/0006294604130420.

[17] E. Tang and Y. Fan, "Performance comparison between five NoSQL databases," *Proc. - 2016 7th Int. Conf. Cloud Comput. Big Data, CCBD 2016*, pp. 105–109, 2017, doi: 10.1109/CCBD.2016.030.

[18] K. Tabet, R. Mokadem, and M. R. Laouar, "Towards a new data replication strategy in MongoDB systems," *ACM Int. Conf. Proceeding Ser.*, 2018, doi: 10.1145/3213187.3287609.

[19] K. Tabet, R. Mokadem, and M. R. Laouar, "A data replication strategy for document-oriented NoSQL systems," *Int. J. Grid Util. Comput.*, vol. 10, no. 1, pp. 53–62, 2019, doi: 10.1504/IJGUC.2019.097227.

[20] K. Ma and B. Yang, "Stream-based live data replication approach of in-memory cache," *Concurr. Comput.,* vol. 29, no. 11, 2017, doi: 10.1002/cpe.4052.

[21] H. Hashem and D. Ranc, "Evaluating NoSQL document-oriented data model," *Proc. - 2016 4th Int. Conf. Futur. Internet Things Cloud Work. W-FiCloud 2016*, pp. 51–56, 2016, doi: 10.1109/W-FiCloud.2016.26.

[22] A. R. Breje, R. Gyorödi, C. Gyorödi, D. Zmaranda, and G. Pecherle, "Comparative study of data sending methods for XML and JSON models," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 12, pp. 198–204, 2018, doi: 10.14569/IJACSA.2018.091229.

[23] A. Šimec and M. Magličić, "Comparison of JSON and XML Data Formats," *Cent. Eur. Conf. Inf. Intell. Syst.*, pp. 272–275, 2014.

[24] Z. U. Haq, G. F. Khan, and T. Hussain, "A Comprehensive analysis of XML and JSON web technologies," *New Dev. Circuits, Syst. Signal Process. Commun. Comput.*, pp. 102–109, 2014.

[25] C. O. Truica, F. Radulescu, A. Boicea, and I. Bucur, "Performance evaluation for CRUD operations in asynchronously replicated document-oriented database," *Proc. - 2015 20th Int. Conf. Control Syst. Comput. Sci. CSCS 2015*, pp. 191–196, 2015, doi: 10.1109/CSCS.2015.32.

[26] Y. Gu *et al.*, "Analysis of Data Replication Mechanism in NoSQL Database MongoDB," pp. 66–67, 2015.