JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p. 1120 - 1129
https://publishoa.com
ISSN: 1309-3452

Developing a sequential data migration method for document-based database and key value database

[1]Lim Fung Ji, [2] Nurulhuda Firdaus Mohd Azmi
[1] Tunku Abdul Rahman University College ,[2] Universiti Teknologi Malaysia
[1]limfj@tarc.edu.my,[2] huda@utm.my

ABSTRACT

Data migration between databases is commonly occurred due to reasons such as migration of legacy system to new system, sharing and updating of data between systems and et cetera. In the cases of migration between NoSQL database such as MongoDB and Redis, the migration of data may not as straight forward as directly moving the data to the designated database. One of the reasons is due to the different storage paradigm of the databases. MongoDB stores data into Json document where Redis manages records by applying key-mapping. In addition, the flexibility of MongoDB's storage paradigm allows records within the same collection to be stored in different structure. Furthermore, the flexibility of storage paradigm will lead to the complexity of record structure such as nested documents. Therefore, this affects the storage structure on the targeted database. The research is to examine the storage schema of Redis in handling the data migrated from MongoDB. From the result of research experiment, the predetermined structure is able to represent the targeted record (document) schema from MongoDB

Index Terms—algorithm*, data migration, database schema, document-oriented database, key value databases.*

I. INTRODUCTION

According to Matthes and Schulz [1], data migration refer as a one-time tool-supported procedure that seeks to transfer formatted data from a source structure to a destination data structure where the two structures differ conceptually and/or physically. In the process of data migration, it is not only not only moving existing data to the target database, but also fitting data models and structures to the destination database without compromising data integrity and correctness [2]. The migration of data between NoSQL databases is challenging due the flexibility of storage paradigm in NoSQL databases. Unlike relational database, records are stored in a fixed row-column format where records within the same group (table) are having the same schema structure. Migration of data can be in a more direct way using column to column mapping. However, this is not the case of NoSQL database. The flexibility of NoSQL storage paradigm may not allow a simple migration with direct mapping between the source and target datastore especial for heterogeneous type of NoSQL database. Therefore, our research is to examine the data migration method that could be applied for heterogeneous NoSQL databases. In our research, we propose a sequential record to record data migration. The NoSQL databases applied in our study is MongoDB and Redis.

II. NOSQL DATABASES

The idea of "NoSQL" started in 1998 where it meant of "Not Only SQL" which comprise of both Relational Database Management System (RDMS) and NoSQL technology [3]. Some reasons [4] of using NoSQL databases are such as rise of cloud computing, storing unstructured data and agile development method which needs rapid change of data schema. The characteristics of NoSQL database are summarized in Table. I.

Table I Characteristics of NoSQL database.

Characteristic	Description
Scalability	Scale out. Performance will be improved when machine (data nodes) is added [5].
Data Replication	Consistency of data is achieved eventually. When a machine is down, the data still be able to read from replicas in another machine [5].
Schema-Less	Not required pre-defined data structure.

(Flexible data structure)	Different types of data format are supported [5].
"Shared Nothing" Architecture	Compare to normal storage, NoSQL uses local storage pool allows faster speed by increasing the number of data storage nodes. In addition, cost with be saved since commodity hardware can be used with this architecture [6].
Elasticity	Expand dynamically. Subsets of data will be replicated to newly added data nodes [6].

There are basically four types of NoSQL database, which are, document based, key value, column oriented and graph-based database [7]. Each type of database has their own unique storage paradigm. The features comparison of NoSQL database is depicted in Table. II.
Document oriented NoSQL stores data in JSON or BSON format. Similar to key-value NoSQL, documents are identified by a set of key value. The difference between document based with key value NoSQL is document-based NoSQL allows the enclosing key values in the documents and search can be based on both key and value. Data stored in graph database as nodes and it is connected by edge. The edge shows the relationship between nodes. There is a pointer in the nodes that point the next nodes [8].

Key value database store data in stable byte array, thus, a key is required to retrieve the store value. This is like the concept of using unique key in relational database. In key value database, it uses a key-value hash-table model for the assessment of data, i.e. each data item has a corresponding key that point to the data item. The key can be automatic generated or synthetic [9].

Column base NoSQL stores data in distributed structure. It is a mix of row/column storage compare to pure column of relational database [8]. Another explanation from Google on BigTable is a "distributed, ordered, sparse and multidimensional maps". Within rows, multiple versions of random key pairs is stored to support versioning and reach performance [9]

Table II Comparisons of NoSQL Database.

NoSQL	**Features**
Key-value	• Stored data using hash table [10]. • Data stored in unstructured format [11]. • Unique key assigned for specific data types[10]. • Use pointer to allocate data [10]. • Examples are Riak, Amazon S3 Dynamo[12].
Document based	• Data stored as document format such as BSON, JSON, XML[12]. • Key-value pairs are used for data association [12]. • Data can be accessed either using the key or value[12]. • Consists of features such as documents replication, persistent data model, cross server distribution[11]. • Examples are MongoDB, CouchDB [12]
Column based	• Data stored in cell of column instead of row[10]. • Data are group into set of columns which then group as column-families [13]. • A column family is required when access data [12]. • Unique key assigned to each column. The column family able to store large amount of

	distributed data[11]. • Resistant to failure due to the powerful cache and exabytes scalable systems [11]. • Examples are HBase, Cassandra [12].
Graph based	• Representing data as a graph such as using graph to visualize data in network analysis [12]. • Data stored represented by nodes, edges and relationships in table form [11]. • Able to process huge datasets, flexible schema representation and difficult to model complex structure [11]. • Examples are Neo4J, InfoGrid [12].

III. RELATED WORKS ON DATA MIGRATION

Scherzinger and Sombach [14] propose an approach for both eager and lazy migration of data due to changes in schema attributes and programme codes as a result of the modification of programme applications. The method employs a tool called Datalution to capture the state of the data store, schema modifications, and top-down or bottom-up programme evaluation using the Datalog programming language. The lazy migration of database schema is one of the methods for preventing application pauses. The lazy migration method permits the conversion of old schema fields to new ones on demand. This will prevent a significant number of application outages [15]. This method, however, will burden the application's developers with code management. Another approach, which also employs lazy migration but reduces the burden on developers by isolating the migration-performing components in a database library, allows applications to access data as usual. This method examines the versions of Java codes that evaluate MongoDB in order to comprehend the schema evolution. However, the approaches did not target distinct NoSQL databases, and the on-demand migration may experience delays as the number of concurrent data accesses increases[16]. Data migration may necessitate validation in the new database. Another technique that assists the developer in analysing and comprehending the evolution of the NoSQL database schema by analysing the different versions of an application's source code[17]. Since the approach detects change at the code level, it may not be able to identify a hidden field within a database. Some hidden field may not be seen at code level but at data model.

From the migration works discussed, our research further concentrates on the migration with the following objectives:
Obj1: Examine data migration algorithm between documents-based database and key-value database.
Obj2: Examine the data schema of key value database in storing data with different schema from document-based database.

IV. EXPERIMENT CONFIGURATIONS

The research study the migration of data from MongoDB to Redis. The configuration of the experiment is discussed on the following subsections.

A. Databases

For the research experiment, the MongoDB version 4.2.3 community as local services at port 27017 and Redis version is 3.0.504 and port 6379 is used for access. The research utilised the default command line interface (CLI) to send commands to the Redis server and receive feedback.

B. Data source

The data source applied in the experiment is The dataset is from a curated list in GitHub [18]. The dataset consists of sets of Json collection. Table. III shows the json files in the dataset.

Table III Json file in database.

Physical Name	**Description**
books.json	Consists of information regarding different types of books.
city_inspection.json	Inspection information of some shops in different system.
companies.json	Information regarding different companies.
countries-big.json	Contain abstract information on countries.
countries-small.json	Contain detail information on countries but less records.
covers.json	Book covers information.
grades.json	Student scores in study.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p. 1120 - 1129
https://publishoa.com
ISSN: 1309-3452

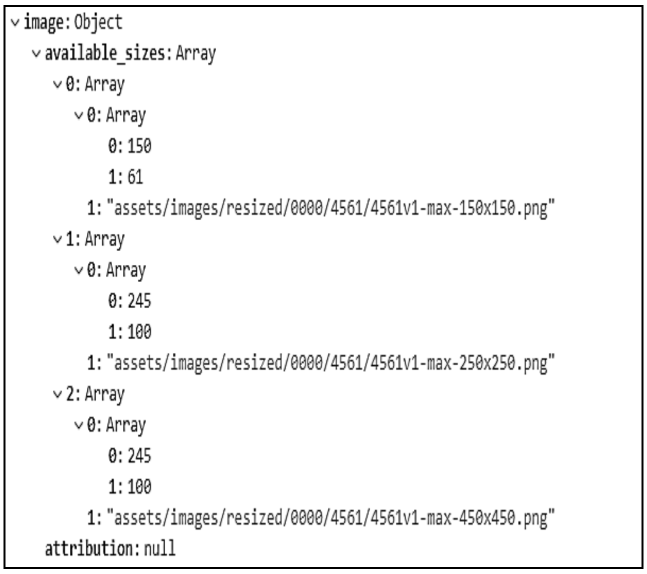
palbum.json	Consists of image lists.
people-bson.zip	People information in zip format.
products.json	Information of different products.
profiles.json	Consists of status of client.
restaurant.json	Restaurant address information.
students.json	Students examination scores.
tweets.zip	Tweeter data in zip format.

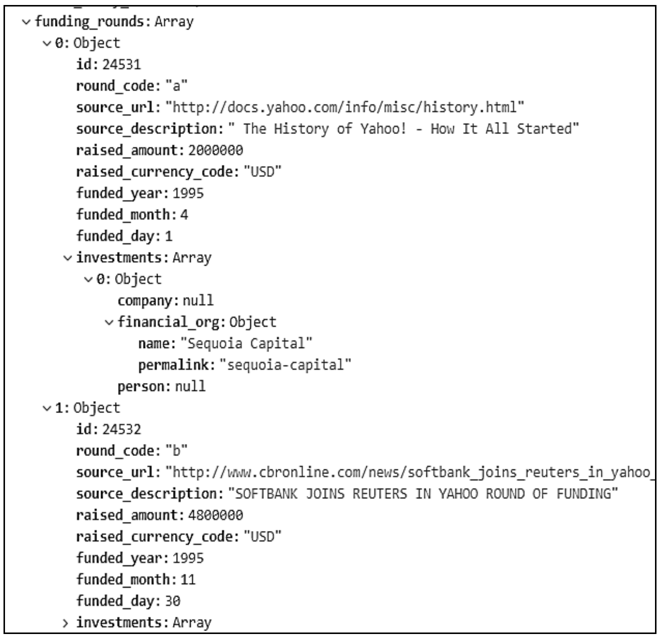
The research is using the *companies.json* for migration to Redis. The *companies.json* consists of 1583 Json documents and these documents are imported into MongoDB database named *Source* and the collection name is *Companies*. Table. IV shows the schema of *Companies* collection which is concluded from the schema analysis using MongoDB Compass.

Table IV Schema of *Companies* collection in MongoDB.

Field	**Type**	**Field**	**Type**	**Field**	**Type**
_id	ObjectId	deadpoolead_year	Mixed	providerships	Array
name	String	tag_list	String	total_money_raised	String
permalink	String	alias_list	Mixed	funding_rounds	Array
crunchbase_url	String	email_address	Mixed	investments	Array
homepage_url	String	phone_number	Mixed	acquisition	Mixed
blog_url	String	description	Mixed	acquisitions	Array
blog_feed_url	String	created_at	Mixed	offices	Array
twitter_username	Mixed	updated_at	String	milestomes	Array
category_code	String	overview	String	video_embeds	Array
number_of_employees	Mixed	image	Mixed	screenshorts	Array
founded_year	Mixed	products	Array	external_links	Array
found_month	Mixed	relationships	Array	partners	Array
founded_day	Mixed	competitions	Array		

From Table. IV, the Mixed type represents complex field type, it could be a document, an array or nested array etc. For example, the *image* field, it is an array which the array item is an array. Another example is the *funding_rounds* field, which is an array of object. Fig. 1 and Fig. 2 show the sample data of *image* and *funding_rounds* fields.


```
v image: Object
  v available_sizes: Array
    v 0: Array
      v 0: Array
          0: 150
          1: 61
        1: "assets/images/resized/0000/4561/4561v1-max-150x150.png"
    v 1: Array
      v 0: Array
          0: 245
          1: 100
        1: "assets/images/resized/0000/4561/4561v1-max-250x250.png"
    v 2: Array
      v 0: Array
          0: 245
          1: 100
        1: "assets/images/resized/0000/4561/4561v1-max-450x450.png"
  attribution: null
```
Figure 1 Sample *image* field.



```
⌄ funding_rounds: Array
  ⌄ 0: Object
      id: 24531
      round_code: "a"
      source_url: "http://docs.yahoo.com/info/misc/history.html"
      source_description: " The History of Yahoo! - How It All Started"
      raised_amount: 2000000
      raised_currency_code: "USD"
      funded_year: 1995
      funded_month: 4
      funded_day: 1
    ⌄ investments: Array
      ⌄ 0: Object
          company: null
        ⌄ financial_org: Object
            name: "Sequoia Capital"
            permalink: "sequoia-capital"
          person: null
  ⌄ 1: Object
      id: 24532
      round_code: "b"
      source_url: "http://www.cbronline.com/news/softbank_joins_reuters_in_yahoo_
      source_description: "SOFTBANK JOINS REUTERS IN YAHOO ROUND OF FUNDING"
      raised_amount: 4800000
      raised_currency_code: "USD"
      funded_year: 1995
      funded_month: 11
      funded_day: 30
    › investments: Array
```

Figure 2 Sample *funding_rounds* field.

The complex field of the schema of the collection indicates that it may not be able to direct mapping into the Redis storage. In addition, the flexibility of schema may lead to different possibilities that affects the target database schema. Therefore, in the transferring process of data to Redis, it needs to consider each single field in a document.

C. Determining Redis schema

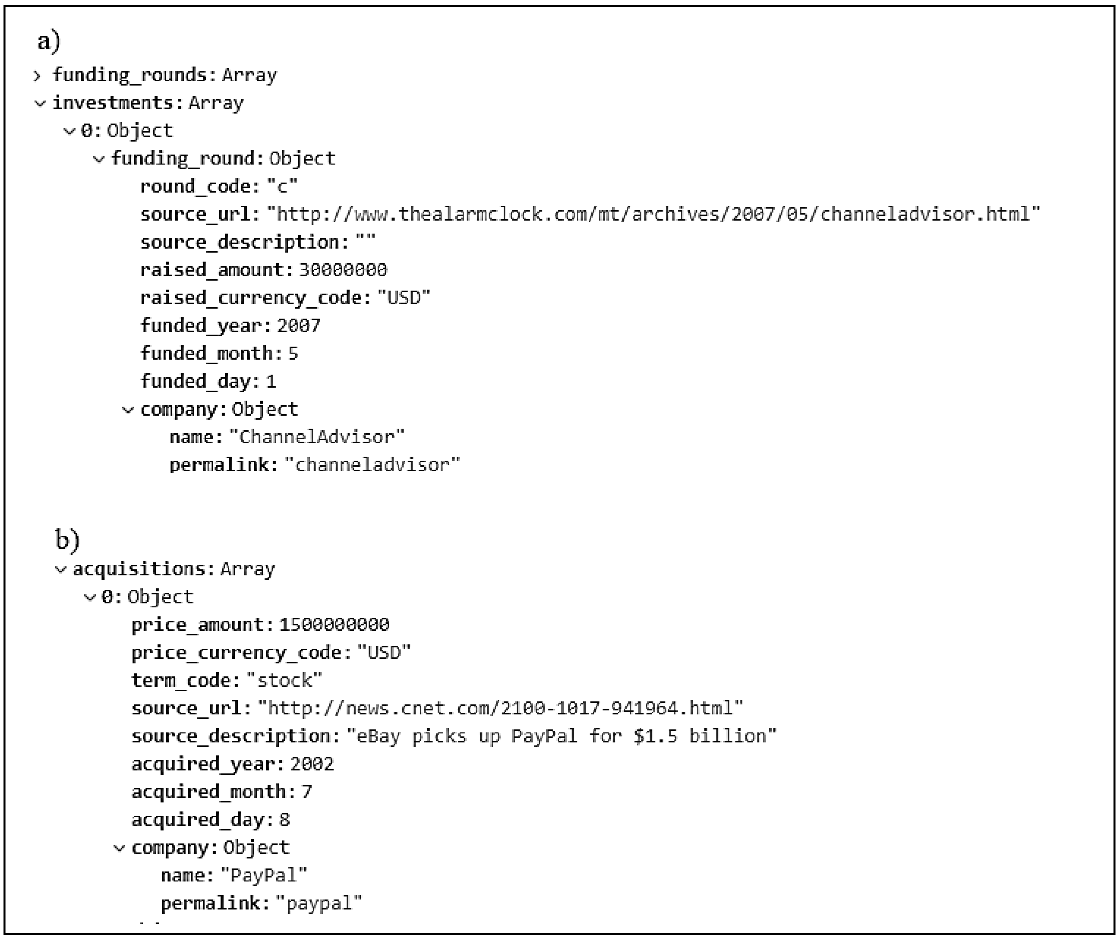
Redis is a key value database where each record is identified by a unique key. Therefore, it is necessary to decide the schema of the Redis key records in representing the MongoDB records that are going to be migrated. The colon (:) symbol is used to separate the parts of the schema, and the dot (.) is used to denote the level of a field in an existing MongoDB document when defining the schema of Redis record keys. For example, *funding_rounds* is an array type field that consists of *investment* field, which is an array field that comprises Object type field named *firm*, as illustrated in Fig. 3. Within the same document, the *company* subfield of the *Companies* collection can be found in the *acquisitions* (in Fig. 3(a)) and *funding_rounds* fields (in Fig. 3(b)). The value of the corporate fields, on the other hand, is different. Furthermore, the *acquisitions* field is an array of objects, each of which is a nested object called firm. As a result, the dot symbol serves as a level indication, indicating the field's current level. When the *company* field is transferred to Redis, it will be displayed as a separate record with the prefix *Companies. acquisitions. company* and *Companies.investments.funding_round.company* respectively.

From the prefix of the Redis key, not only will we know the position level of a record's origin data source, but it will also help to ensure that duplicate field migration uses a unique key. In addition, the usage of colon and dot symbols enables post-migration validation of record values. The research determines the whole structure and format of Redis keys based on the determined prefix of keys below:

a) The key format for a document with no relationship or parent document would be <collection name>:<mongo id>.

b) For sub-documents, array of documents (or objects) that will be separated as a new Redis record with an additional grouping key with format <collection1 name>:<mongo id>:<collection2 name>. This format defines the grouping of documents that are linked to the parent document, with <collection2 name>:<mongo id> serving as the key for each document within the group. For instance, if field *competition* is a sub-document of *Companies* in MongoDB, a separate record will be created in Redis for *competition*, and the key that represents this *Companies-competition* record will be Companies:<Companies Document id>: Companies.competition, with each Redis record of *competition* having a unique key of Companies.competition:<Companies Document id>. For arrays of document field types, however, a number will be added as the key's postfix; this number follows the array's position index.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p. 1120 - 1129
https://publishoa.com
ISSN: 1309-3452



```
a)
> funding_rounds: Array
v investments: Array
  v 0: Object
    v funding_round: Object
        round_code: "c"
        source_url: "http://www.thealarmclock.com/mt/archives/2007/05/channeladvisor.html"
        source_description: ""
        raised_amount: 30000000
        raised_currency_code: "USD"
        funded_year: 2007
        funded_month: 5
        funded_day: 1
      v company: Object
          name: "ChannelAdvisor"
          permalink: "channeladvisor"

b)
v acquisitions: Array
  v 0: Object
      price_amount: 1500000000
      price_currency_code: "USD"
      term_code: "stock"
      source_url: "http://news.cnet.com/2100-1017-941964.html"
      source_description: "eBay picks up PayPal for $1.5 billion"
      acquired_year: 2002
      acquired_month: 7
      acquired_day: 8
    v company: Object
        name: "PayPal"
        permalink: "paypal"
```

Figure 3 Duplicate *company* field in same document

c) For array and other list-type fields, there are numerous considerations. The research examines both the basic array and the complex array. Simple array type in which all array elements are simple type data such as integers and strings; the simple array elements will be added to Redis as "[x,x,x,x]" to represent an array. If the array item of a complex type array is of document type, it will be inserted into another Redis group according to the structure described in b). For items that are arrays (sub-array cases), the array elements will be accessed using the format described in a), b), and c).

V. ALGORITHM

On the basis of the previously stated schema representation of Redis, the migration algorithm is adjusted to conform to the schema structure. The algorithm starts by accessing the first document in the *Companies* collection and pass the document to a function method named *AddParentField*. The function will access each field and identify the type of the field. For native (simple) field type, the field will be added into a newly created Redis record with the key value of, in this case, is Companies:<Mongo Id of the document>. For complex field such as document, the function will call itself and pass the field as argument and repeat the same process. That is, the field and its data will only be added into Redis when it is a simple type field and additional key is created to represent the linkage between the *Companies* record and the separated record that represents the field in Redis. The recursive function call will stop until all field is added into Redis. For Arraylist type, each array element will be access and the similar process is applied to each item. Fig. 4. Depicts the algorithm that detects and add the field data into Redis.

VI. EXPERIMENT EXECUTION AND RESULT

The experiment is executed by selecting the *Companies* collection in MongoDB. The result of migration is observed when the migration process is completed. To check on the migration outcome, records are randomly selected and compare them with the record in Redis. We choose complex field types for matching; the chosen document's id is *52cdef7c4bab8bd675297d93* as shown in Fig. 5. The comparison on the *investment* field which is an array that consists of a document object field named *funding_round* in each of its array item. In addition, *funding_round* embeds the company object field. The similar situation happened on the acquisitions field in which each array item is an object of document.

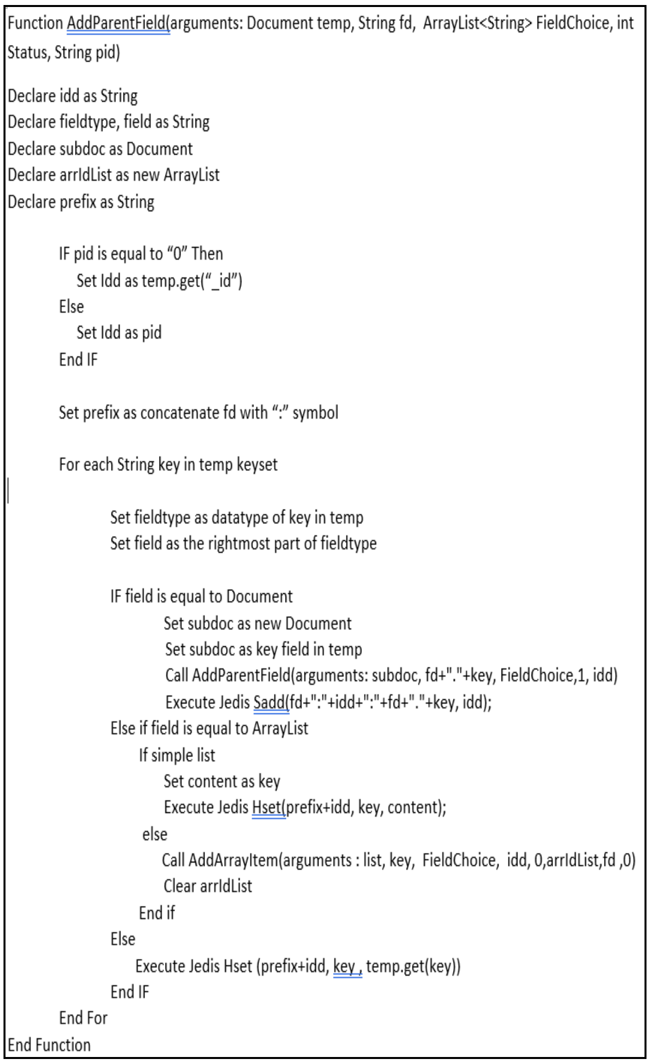
This selection is intended to check the migration of complex field as discussed earlier in the section. According to the predetermined schema of Redis key, the keys for the respective records in Redis should be the following:

- *Companies.investments.funding_round:52cdef7c4bab8bd675297d93-1*
- *Companies.investments.funding_round.company: 52cdef7c4bab8bd675297d93-1*
- *Companies.aquisitions:52cdef7c4bab8bd675297d93-0*
- *Companies.aquisitions.company:52cdef7c4bab8bd675297d93-0*

The first two keys represent the key for migrated *funding_round* field and the *company* field that embedded in the *funding_round* field. We retrieve the value based on the generated keys through the command line interface. The result is shown by Fig. 6. The result shown indicates that the values are matched with the value of *funding_round* in Fig. 5. As for the value of company field, another Redis s key is used to retrieve the data due to company field is object type which is represented by different record in Redis.

Fig. 7 depicts the value of migrated data for *company* field.
The key is *Companies.investments.funding_round.company:52cdef7c4bab8bd675297d93-1* and the values are matched. The “-1” of the key represents the index position of the array field in origin source. Similar steps are performed to validate the values of *company* field which is under *acquisitions* field in the origin MongoDB. Fig. 8 and Fig. 9 depict the retrieved value at Redis for the *acquisitio*ns and *company* fields which is a sub field of *acquisitions* field.

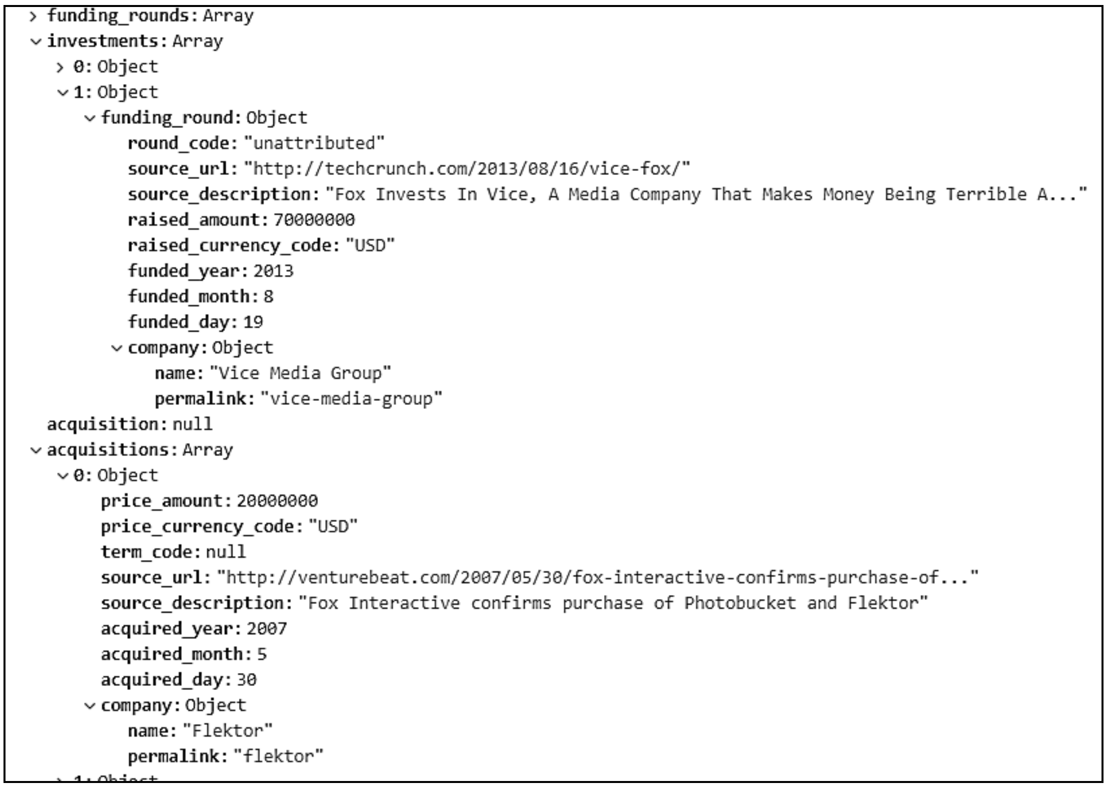
Figure 8 shows an empty list as the result or acquisition field. This is due to the field is an array and only consist of one object type field named company, therefore there is no data retrieved based on the key. However, the company data in Fig. 9 shows it match with the value in Fig. 5. From the manual validation, it proved that the data has been migrated completely and correctly with our algorithm. The experiment shows different way of arranging migrated data.



```
Function AddParentField(arguments: Document temp, String fd,  ArrayList<String> FieldChoice, int
Status, String pid)

Declare idd as String
Declare fieldtype, field as String
Declare subdoc as Document
Declare arrIdList as new ArrayList
Declare prefix as String

    IF pid is equal to “0” Then
        Set Idd as temp.get(“_id”)
    Else
        Set Idd as pid
    End IF

    Set prefix as concatenate fd with “:” symbol

    For each String key in temp keyset

        Set fieldtype as datatype of key in temp
        Set field as the rightmost part of fieldtype

        IF field is equal to Document
            Set subdoc as new Document
            Set subdoc as key field in temp
            Call AddParentField(arguments: subdoc, fd+"."+key, FieldChoice,1, idd)
            Execute Jedis Sadd(fd+":"+idd+":"+fd+"."+key, idd);
        Else if field is equal to ArrayList
            If simple list
                Set content as key
                Execute Jedis Hset(prefix+idd, key, content);
            else
                Call AddArrayItem(arguments : list, key,  FieldChoice,  idd, 0,arrIdList,fd ,0)
                Clear arrIdList
            End if
        Else
            Execute Jedis Hset (prefix+idd, key , temp.get(key))
        End IF
    End For
End Function
```

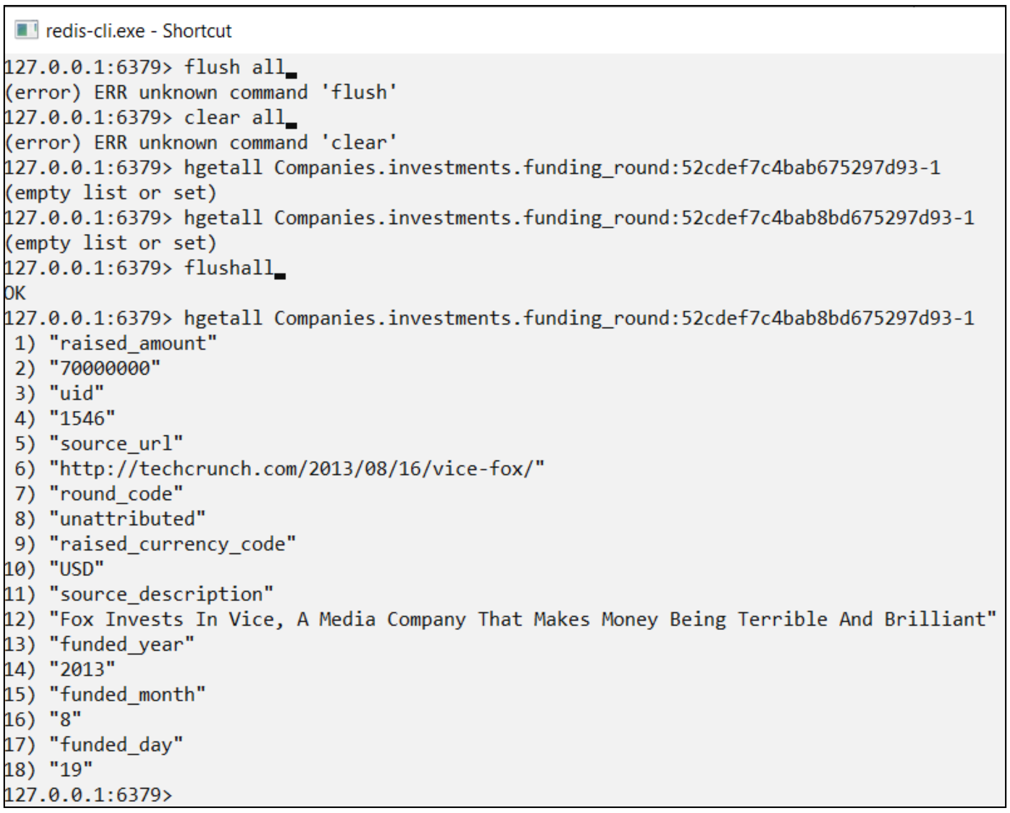
Figure 4 Migration algortih

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p. 1120 - 1129
https://publishoa.com
ISSN: 1309-3452



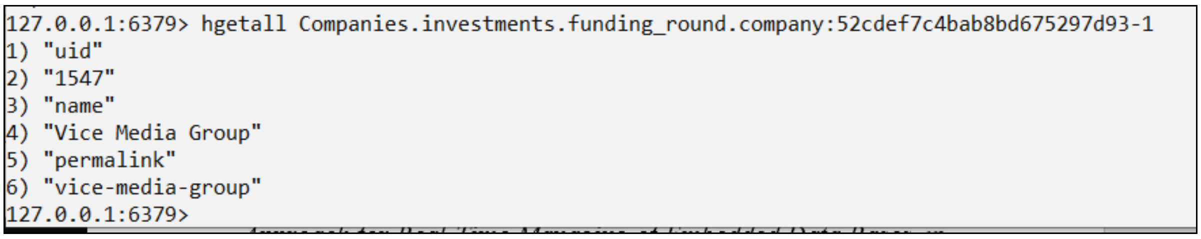
```
> funding_rounds: Array
v investments: Array
  > 0: Object
  v 1: Object
    v funding_round: Object
        round_code: "unattributed"
        source_url: "http://techcrunch.com/2013/08/16/vice-fox/"
        source_description: "Fox Invests In Vice, A Media Company That Makes Money Being Terrible A..."
        raised_amount: 70000000
        raised_currency_code: "USD"
        funded_year: 2013
        funded_month: 8
        funded_day: 19
      v company: Object
          name: "Vice Media Group"
          permalink: "vice-media-group"
  acquisition: null
v acquisitions: Array
  v 0: Object
      price_amount: 20000000
      price_currency_code: "USD"
      term_code: null
      source_url: "http://venturebeat.com/2007/05/30/fox-interactive-confirms-purchase-of..."
      source_description: "Fox Interactive confirms purchase of Photobucket and Flektor"
      acquired_year: 2007
      acquired_month: 5
      acquired_day: 30
    v company: Object
        name: "Flektor"
        permalink: "flektor"
```

Figure 5 Screenshot of complex data fields



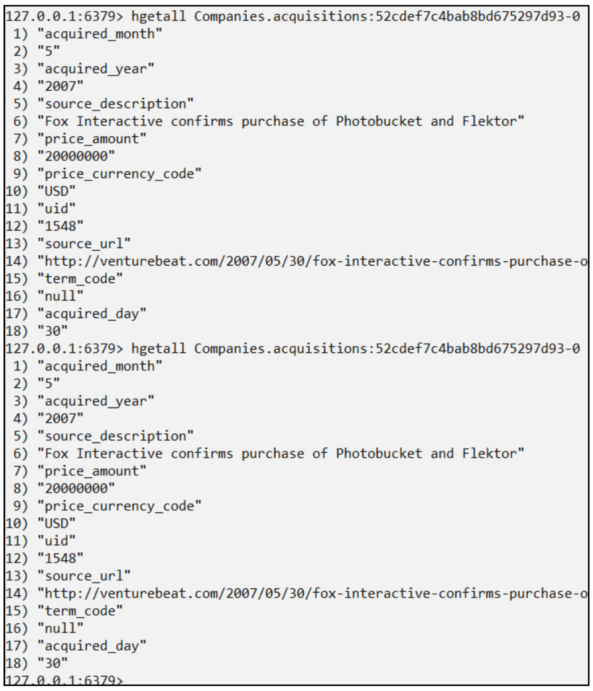
```
127.0.0.1:6379> flush all
(error) ERR unknown command 'flush'
127.0.0.1:6379> clear all
(error) ERR unknown command 'clear'
127.0.0.1:6379> hgetall Companies.investments.funding_round:52cdef7c4bab675297d93-1
(empty list or set)
127.0.0.1:6379> hgetall Companies.investments.funding_round:52cdef7c4bab8bd675297d93-1
(empty list or set)
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> hgetall Companies.investments.funding_round:52cdef7c4bab8bd675297d93-1
 1) "raised_amount"
 2) "70000000"
 3) "uid"
 4) "1546"
 5) "source_url"
 6) "http://techcrunch.com/2013/08/16/vice-fox/"
 7) "round_code"
 8) "unattributed"
 9) "raised_currency_code"
10) "USD"
11) "source_description"
12) "Fox Invests In Vice, A Media Company That Makes Money Being Terrible And Brilliant"
13) "funded_year"
14) "2013"
15) "funded_month"
16) "8"
17) "funded_day"
18) "19"
127.0.0.1:6379>
```

Figure 6 Migrated data for *funding_round* fields



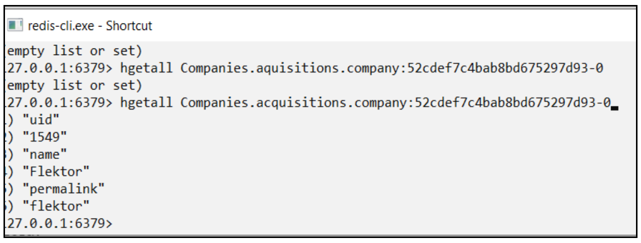
```
127.0.0.1:6379> hgetall Companies.investments.funding_round.company:52cdef7c4bab8bd675297d93-1
1) "uid"
2) "1547"
3) "name"
4) "Vice Media Group"
5) "permalink"
6) "vice-media-group"
127.0.0.1:6379>
```

Figure 7 Migrate data of *company* field.



```
127.0.0.1:6379> hgetall Companies.acquisitions:52cdef7c4bab8bd675297d93-0
 1) "acquired_month"
 2) "5"
 3) "acquired_year"
 4) "2007"
 5) "source_description"
 6) "Fox Interactive confirms purchase of Photobucket and Flektor"
 7) "price_amount"
 8) "20000000"
 9) "price_currency_code"
10) "USD"
11) "uid"
12) "1548"
13) "source_url"
14) "http://venturebeat.com/2007/05/30/fox-interactive-confirms-purchase-o
15) "term_code"
16) "null"
17) "acquired_day"
18) "30"
127.0.0.1:6379> hgetall Companies.acquisitions:52cdef7c4bab8bd675297d93-0
 1) "acquired_month"
 2) "5"
 3) "acquired_year"
 4) "2007"
 5) "source_description"
 6) "Fox Interactive confirms purchase of Photobucket and Flektor"
 7) "price_amount"
 8) "20000000"
 9) "price_currency_code"
10) "USD"
11) "uid"
12) "1548"
13) "source_url"
14) "http://venturebeat.com/2007/05/30/fox-interactive-confirms-purchase-o
15) "term_code"
16) "null"
17) "acquired_day"
18) "30"
127.0.0.1:6379>
```

Figure 8   Retrieved value for *acquisitions* field.



```
redis-cli.exe - Shortcut
(empty list or set)
27.0.0.1:6379> hgetall Companies.aquisitions.company:52cdef7c4bab8bd675297d93-0
(empty list or set)
27.0.0.1:6379> hgetall Companies.acquisitions.company:52cdef7c4bab8bd675297d93-0
) "uid"
) "1549"
) "name"
) "Flektor"
) "permalink"
) "flektor"
27.0.0.1:6379>
```

Figure 9   Data of *Companies.acquisitions.company*

VII. FUTURE WORKS

With the random selection of matching, it indicates the algorithm migrates the data as according to the pre-determined format in Redis. However, the random check does not guarantee correctness of migrated data for bigger data samples. In addition, with this format of key will increase depending on the level of fields in the document field hierarchy. Therefore, there is unavoidable of long key value of record in Redis if the level of field is huge. In addition, the recursive call of function may be a complex solution for the migration. On future work, our research will proceed with migrating more complex field structure to other NoSQLs with validation process and vice-versa.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p. 1120 - 1129
https://publishoa.com
ISSN: 1309-3452

REFERENCES

[1] Matthes, F. and C. Schulz, *Towards an integrated data migration process model-State of the art & literature overview*. Technische Universität München, Garching bei München, Germany, Tech. Rep, 2011.
[2] Scavuzzo, M., E.D. Nitto, and S. Ceri. *Interoperable Data Migration between NoSQL Columnar Databases*. in *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*. 2014.
[3] Gueidi, A., H. Gharsellaoui, and S.B. Ahmed. *A NoSQL-based Approach for Real-Time Managing of Embedded Data Bases*. in *2016 World Symposium on Computer Applications & Research (WSCAR)*. 2016.
[4] *Document Databases*. 2017 [cited 2017 9/9/2017]; Available from: https://www.mongodb.com/document-databases.
[5] Tsuyuzaki, K. and M. Onizuka *NoSQL Database Characteristics and Benchmark System*. 2012. **10**, 5.
[6] Ganesh Chandra, D., *BASE analysis of NoSQL database.* Future Generation Computer Systems, 2015. **52**: p. 13-21.
[7] Zafar, R., et al. *Big Data: The NoSQL and RDBMS review*. in *2016 International Conference on Information and Communication Technology (ICICTM)*. 2016.
[8] Swaroop, P., et al., *NoSQL Paradigm and Performance Evaluation.* SSARSC International Journal of Geo Science and Geo Informatics, 2016. **3**(1).
[9] Makris, A., et al., *A Classification of NoSQL Data Stores Based on Key Design Characteristics*. Vol. 97. 2016. 94-103.
[10] Schulz, W.L., et al., *Evaluation of relational and NoSQL database architectures to manage genomic annotations*. Journal of Biomedical Informatics, 2016. **64**: p. 288-295.
[11] Mathew, A.B. and S.D.M. Kumar. *Analysis of data management and query handling in social networks using NoSQL databases*. in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2015.
[12] Kumar, K.B.S., Srividya, and S. Mohanavalli. *A performance comparison of document oriented NoSQL databases*. in *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*. 2017.
[13] Ho, L.Y., et al. *Data Partition Optimization for Column-Family NoSQL Databases*. in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. 2015.
[14] Scherzinger, S., et al., *Datalution: a tool for continuous schema evolution in NoSQL-backed web applications*, in *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*. 2016, ACM: Saarbr&#252;cken, Germany. p. 38-39.
[15] Klettke, M., et al. *NoSQL schema evolution and big data migration at scale*. in *2016 IEEE International Conference on Big Data (Big Data)*. 2016.
[16] Saur, K., T. Dumitraş, and M. Hicks. *Evolving NoSQL Databases without Downtime*. in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016.
[17] Meurice, L. and A. Cleve. *Supporting schema evolution in schema-less NoSQL data stores*. in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2017.
[18] Ozler, H. *A curated list of JSON/BSON datasets from the web in order to practice/use in MongoDB*. 2019 5/7/2019 [cited 2019 13/7/2019]; Available from: http://www.github.com/ozlerhakan/mongodb-json-files.