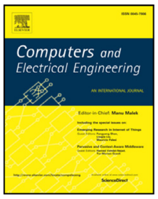
Computers and Electrical Engineering 000 (2018) 1–15

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng



Data allocation optimization for query processing in graph databases using Lucene☆

Anita Brigit Mathew

Department of Computer Science and Engineering, NIT, Calicut, India

A R T I C L E I N F O

Article history:
Received 25 August 2017
Revised 16 January 2018
Accepted 17 January 2018
Available online xxx

Keywords:
Big Data
Query retrieval
Graph NoSQL databases
Data allocation
Best Fit Decreasing
Ant Colony Optimization

A B S T R A C T

Methodological handling of queries is a crucial requirement in social networks connected to a graph NoSQL database that incorporates massive amounts of data. The massive data need to be partitioned across numerous nodes so that the queries when executed can be retrieved from a parallel structure. A novel storage mechanism for effective query processing must to be established in graph databases for minimizing time overhead. This paper proposes a metaheuristic algorithm for partitioning of graph database across nodes by placement of all related information on same or adjacent nodes. The graph database allocation problem is proved to be NP-Hard. A metaheuristic algorithm comprising of Best Fit Decreasing with Ant Colony Optimization is proposed for data allocation in a distributed architecture of graph NoSQL databases. Lucene index is applied on proposed allocation for faster query processing. The proposed algorithm with Lucene is evaluated based on simulation results obtained from different heuristics available in literature.

1. Introduction

In today's big data era, storage of massive data sets is a serious issue to be considered [1]. NoSQL graph databases, provide a pathway for unstructured data to be stored in structured form offering significant advantage in processing. It meets the needs of data holders based on the growing demand of storage. This data is stored in a distributed setup in graph NoSQL database. The distributed data in graph NoSQL databases allows load balancing based on their capacities. The data distributed across nodes is accessed by the controller at the time of query processing. Query process invokes nodes based on the query request made by the user. Hence there is a requirement that the data locality of the nodes to be geographically near so that query retrieval can be made faster. The data stored in each of these nodes represent relationships because the data dealt here is from social sites. Hence there is a need for efficient storage of related and replicated data for effective query processing.

In relational data model, the theory of sharing has a long way to go, and different procedures have been examined for partitioning tabular data into shards and assign these shards to other nodes [2]. Many other NoSQL databases like key-value store, column family, and document databases use range-based fragmentation for distribution of data [3]. Apart from conventional methodologies of query answering, a flexible query processing provides mechanisms for the intellectual answering of user queries. This provides user to retrieve any type of queries whether it is range or related. Sometimes the data to be searched may be present in database but due to lack of efficient related storage, the query retrieval process fails to retrieve

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. J. D. Peter.
E-mail address: anita_p120024cs@nitc.ac.in

https://doi.org/10.1016/j.compeleceng.2018.01.022
0045-7906/© 2018 Elsevier Ltd. All rights reserved.

the data. Internally a flexible query processing system revise query process at time of failure and return results in an informative manner compared to empty results. Efficient placement of all related terms with an index structure leads to flexible query processing without any failure in retrieval.

The rest of the paper is structured as follows, Section 2 Background, gives a review on related works of existing data storage techniques available in the literature. Section 3 gives a sketch of graph database architecture. Section 4 illustrates the graph database allocation problem with replication is a variation to Bin Packing Problem (BPP) which is a NP-Hard problem. Graph database allocation problem is solved using 0 1 Integer Linear Programming(ILP). To solve the hard allocation strategy by taking replication and relation factors into consideration a metaheuristic based Ant Colony Optimization BFD integration is modeled, further an index named Lucene is also implemented and computational results are analyzed in Section 5. Section 6 presents the experimental setup and implementation details. Section 7 concludes the work with the conclusion and future scope.

2. Background

Many researchers have studied the problems of query retrieval over relational databases. Ning et al. [1] develop data allocation scheme based on the prior knowledge of type of related data and grouping this data by encapsulation schemes available. This form of data allocation try to store all related or shared data to a single node. If a raw data could not be inserted into a single node, data is compressed and stored on the same node. This strategy used can cause loss of data during data retrieval process. Sun et al. [4] discuss a data allocation strategy by balancing the data across various nodes. They use a next fit technique of data distribution. But this technique failed to allocate duplicate data in a distributed environment and relations were also not considered. How to query for keywords in a distributed network which is connected to data sources are covered by Kavitha et al. [5] The data allocation scheme pre-owned by them is based on load balancing technique across all available nodes. Mathew et al. [6] suggest a new allocation scheme for social data in NoSQL database. This technique reflects on the depreciation of search procedure thereby increasing query performance. This procedure deeply constrained to keywords associated to user file data like names of files, other data including record and field information are unable to search. Here each search dimension file keyword moves across nodes in a distributed peer to peer model. Li et al. [7] proves in their paper distributed NoSQL systems, object, and graph NoSQL databases are lightweight and they support both low latency and high throughput for fast data allocation. Distributed computing in cloud environment provides platform services to users to allocate the Big Data using their own algorithms in a pay-as-you-go structure [8]. The adaptation from relational to NoSQL databases in cloud environment resulted in significant profits in Big Data allocation and cost savings with respect to node usage but lack of data retrieval techniques forced for a research in query processing techniques [9]. Maheswari et al. [2] discuss about the resource allocation in mobile cloud environment is a tedious task hence various techniques to resolve these issues are studied by them. Among the different techniques, FFD strategy was used in order by them to resolve the issue of allocation in a distributed environment of cloud. Zhang et al. [10] survey how the recent advancements in NoSQL databases support for Big Data. They also focus on the challenges neglected and results obtained for the development of data computing applications across homogenously distributed nodes. David Gil et al. [11] is a framework for data load optimization of Big Data storage and computation in nodes. In this approach amount of data transferred between the nodes, structure of communication, memory requirements, I/O activity, hard disk, affinity of relationships, etc. are all considered. These characteristics are analyzed and efficient techniques are implemented. The main drawback seen here is replication and relationships are not considered. Gondhi et al. [12] discuss how feature selection is done in text categorization. To upgrade the performance of text classification, they present a new algorithm depended on Ant Colony Optimization. This algorithm is based on the real ants movement in search for shortest path to reach the source of food. Here the movement of ants in search of food through shortest path indicate the relation between blocks stored in different nodes but replication of blocks is not considered.

Social big data allocation analytics consists of terabytes or petabytes of data. To store these data, many nodes are required. So for efficiently storing this input data based on relations between them and with replication, social data input needs more than one node which means it needs a distributed environment to store this big data. Execution time of query processing depends on the optimal store of this social data in the distributed environment. Since here we are using graph NoSQL databases integrated with social network sites like Twitter and Facebook for data allocation, the motivation is that the data allocated should be optimal in distributed architecture by minimizing the query processing overhead based on replications and relations.

3. Architecture of graph NoSQL databases

Graph NoSQL databases architecture consists of numerous nodes in a distributed environment. Each node constitutes a different configuration. These nodes are set with distributed configuration setup to support graph social data allocation. The replicated data are placed on different nodes and data related to each other are placed on the same node based on the proposed method to achieve efficient query processing. Social data storage in the era of big data is a high requirement for effective query processing from NoSQL graph databases. Cypher Query Language (CQL) is used to fetch queries from graph NoSQL databases architecture after storing across numerous adjacent node clients.

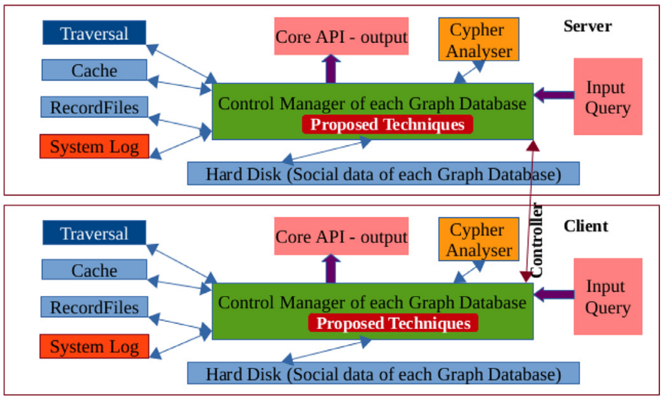


Fig. 1. Server-client allocation framework of graph databases with controller.

Control manager of graph databases controls the store and fetch of data requests through CQL. Each control manager is associated with a controller. The primary role of a controller is data processing, tracking availability of the query term based on which node have the term. When a social network user data is taken for storage using CQL, the control manager invokes cypher parser and translates the query, followed by which the control manager calls record file and creates record level sequential index [13,14] for the newly stored data to hard disk, after an invocation, file system cache is activated. Then when a query comes to processing the traversal phase traverse from one node to another with the help of record level sequential index of each node through control manager controlled by controller. All clients send a heartbeat message to the server via controller every five seconds to indicate they are alive with the data stored. If the server does not receive a clock tick from a precise client for 5 seconds, then it acknowledges that client node to be inactive or out of service and then it initiates the controller to connect to some other adjacent client node, and move a copy of dead client from the replicated data stored in server or any other client to the new alive client node.

Once when a query comes to processing, control manager of server translate query using cypher query parser. It checks the object cache and file system cache whether the query comes from the last 61 committed queries (after every 61 queries the old query gets overwritten). If so it takes the data from system cache else it transmits the call to traversal API. The traversal process is illustrated in detail by the same authors in [13]. If the data could not be obtained from server the controller of server contacts the controller of the adjacent client and checks for the query terms. The same process of server continues for other clients also. The process terminates once the data terms searched is obtained or once all the clients are traversed. The above-stated process is graphically shown in Fig. 1.

The solution defines query Q to be a set of tuples r record files with f fields, from database D data is retrieved after JOIN CQL command. The tuples are translated using CQL in Q. Therefore when it comes to m nodes the overall time will be

$$O((f_{ijk})_{k=1}^{\gamma(i,j)}U(r_{ij})_{j=1}^{\beta}U(F_i)_{i=1}^{\alpha})^{\kappa}$$

, were $F_i)_{i=1}^{\alpha}$ set of files, $(r_{ij})_{j=1}^{\beta}$ set of records for each file and $(f_{ijk})_{k=1}^{\gamma(i,j)}$ set of fields for each of the records. κ denotes the growth in time for processing the files. This needs to be reduced because there was no optimum allocation of records and fields of data to take replication and relation into account.

Fig. 1. discuss the process how social data allocation is done in graph NoSQL databases. The blocks of data obtained from social networks are taken care of by control manager of server. Server node initiates the controller and transfers the data based on BFDACO to other clients such that the number of clients used is less. The similarity between bin packing and graph NoSQL databases allocation is further being illustrated in next section.

4. Graph NoSQL databases allocation

Consider any social network big data to be stored in graph NoSQL database D, were

$$D = (F_1, F_2 \ldots F_\alpha) \tag{1}$$

Data in graph NoSQL database D is stored in the form of files which can be represented as $U_{i=1}^{\alpha}F_i$. Each of these files is a collection of records

$$F_i = (r_{i1}, r_{i2} \ldots r_{i\beta(i)}) \tag{2}$$

denoted as $U_{j=1}^{\beta}(r_{ij})$ and each of the records contain fields

$$r_{ij} = (f_{ij1}, f_{ij2} \ldots\ldots f_{ij\gamma(i,j)}) \tag{3}$$

indicated as $U_{k=1}^{\gamma(i,j)}(f_{ijk})$. Whenever a data arrives or data driven from dataset, in both the cases data has to be stored based on file-record-field structure in graph NoSQL database . This file-record-field structure data is horizontally partitioned into blocks irrespective of size. The fragmented data need to be allocated based on intention to perform effective query retrieval with less aloft in time. For this, the relation existing between blocks and replicas of blocks should also be considered during data allocation. Therefore, our requirement to perform query retrieval Q using CQL (shown in Eq. (4)) were,

$$Q \in U_{i=1}^{\alpha} F_i U U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)}(f_{i2k}) U U_{i=1}^{\alpha} U_{k'=1}^{\gamma(i,3)}(f_{i3k'}) \tag{4}$$

The parameters given in (4) are: $U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)}(f_{i2k})$ indicate the ith file 2nd record any field k is same as $U_{i=1}^{\alpha} U_{k'=1}^{\gamma(i,3)}(f_{i3k'})$ indicate the ith file 3rd record any field k'. The query problem is to find match of Q in D and retrieve Q from D provided, query execution time is reduced without compromising the quality of the query search.

To achieve the above-mentioned fault tolerant, reliable and high available query retrieval of data in a distributed graph NoSQL database, replication of data is required across different nodes. Whenever one thread fails, the other client gets loaded with the distributed architecture. Which means if one of the database nodes fails, a data replica can be retrieved from one of the other nodes. This is very much necessary so that it would be a helping hand for efficient query search. The data need to be replicated to different nodes (same block not in same).

This allocation problem with replication is a formal description of the task of distribution of copies of data blocks across a set of nodes in a distributed graph database system. This allocation problem is basically a BPP. The problem of allocating data optimally and reducing the number of nodes utilized is formally defined in the following subsections. The hardness is also proved.

4.1. Proof of hardness for graph NoSQL databases allocation problem

Graph NoSQL databases allocation problem is a data allocation problem for efficient packing of data in nodes to support query processing. Here we are given a set of n blocks of data $\{ft_j\}$, were $1 \le j \le n$ with replication $\{r_j\}$ and weight $\{w_j\}$. Our objective is to allocate the blocks to nodes such that the sum of the weights of the blocks allocated to a node does not exceed the capacity of the node, minimum possible number of nodes are used and fragment $\{ft_j\}$ is replicated in at least $r_j + 1$ distinct nodes.

The mathematical formulation of the above problem yields the following $0 - 1$ Integer Linear Programming (ILP):

Given $(n, k, \{w_j\}, \{r_j\}, \{C_i\})$, were n, k are positive integers, $\{w_j\}, \{r_j\}, \{C_i\}$ are ordered sets of rational numbers and $1 \le j \le n$, $1 \le i \le k$.

The problem is to minimize

$$\Sigma_{i=1}^{k} y_i \tag{5}$$

subject to constraints:

$$\Sigma_{j=1}^{n} x_{ij} w_j \le C_i y_i \ \forall i \in \{1 \ldots k\} \tag{6}$$

$$\Sigma_{i=1}^{k} x_{ij} \ge r_j + 1 \ \forall j \in \{1 \ldots n\} \tag{7}$$

$$x_{ij} \in \{0, 1\}; \quad y_i \in \{0, 1\} \tag{8}$$

Here, $x_{ij} = 1$ if block j is allocated to node i, $x_{ij} = 0$ otherwise;

Also, $y_i = 1$ if node i is used, $y_i = 0$ otherwise;

In the above formulation if $r_j = 1 \ \forall 1 \le j \le k$ then the rightside of (7) becomes unity. Next section proves the NP-Hardness of the problem by reduction from the Bin Packing Problem.

4.2. Bin packing problem to graph NoSQL databases allocation problem reduction

This section shows that Bin Packing Problem is reducible to graph NoSQL databases Allocation Problem. It is known from literature that Bin Packing Problem is NP-Hard [6]. Consequently, graph NoSQL databases Allocation Problem is also NP-Hard.

Definition 1: Bin Packing Problem (BPP)

Given a set of n items with weights $\{w_j\}$, where $1 \le j \le n$. The aim is to pack these items into k bins with capacities $\{C_i\}$ associated with each bin, where $1 \le i \le k$. The problem is to pack these items into bins subject to the constraint that the sum of the weights of the items packed into a bin does not exceed the overall capacity of the bin. The objective function is to minimize the entire number of bins used. The mathematical formulation of the above problem grants the following $0 - 1$ ILP:

Given $(n, k, \{w_j\}, \{C_i\})$, where n, k are positive integers, $\{w_j\}, \{C_i\}$ are ordered sets of rational numbers and $1 \le j \le n$, $1 \le i \le k$.

The problem is to

Minimize

$$\Sigma_{i=1}^{k} y_i \tag{9}$$

subject to constraints:

$$\Sigma_{j=1}^{n} x_{ij} w_j \le C_i y_i \;\; \forall i \in \{1 \ldots k\} \tag{10}$$

$$x_{ij} \in \{0, 1\}; \;\; y_i \in \{0, 1\} \;\; \forall \Sigma_{i=1}^{k} x_{ij} \ge 1 \tag{11}$$

Here, $x_{ij} = 1$ if fragment j is allocated to ith node, $x_{ij} = 0$ otherwise;

Also, $y_i = 1$ if node i is used, $y_i = 0$ otherwise;

4.2.1. Reduction

It is observed that an instance $(n, k, \{w_j\}, \{C_i\})$ of BPP, rewritten as $(n, k, \{w_j\}, 0, \{C_i\})$ is an equivalent instance of the NHADA problem. Hence, the following conclusions are made:

Corollary 1: BPP $\le_m{}^p$ Graph NoSQL Databases Allocation Problem.

Corollary 2: Graph NoSQL databases allocation problem is NP-Hard.

The next section presents a metaheuristic based algorithm BFDACO for solving the NHADA problem for the case r_1, r_2 ... $r_n = 0$.

4.3. How to solve graph NoSQL databases allocation problem?

In the previous subsection graph NoSQL databases allocation is proved to be a NP-Hard problem. Graph NoSQL databases allocation is a data allocation hard problem and computationally infeasible for huge social big data allocation. This problem is to find a feasible set of nodes to optimally allocate social data across them. Heuristic algorithms like FF, BF, FFD, and BFD is an option for BPP []. These heuristic techniques obtain rather good solutions, which may be optimal but are not suited when replication and relation exist among social big data. Metaheuristic based ACO algorithm has been successfully used to solve hard selection problems like vertex cover, maximum clique, edge weight cardinality tree, multiple knapsacks, constraint satisfaction, boolean satisfiability and so on [15]. BFD BPP heuristic was proved to be a linear constant space by Mathew et al. [6]. Hence BFD can be chosen for exhibiting the blocks into groups based on the similarity of data. For large instances, Brugger et al. [16] demonstrated that performance of ACO is superior to genetic approaches. Compared to simulated annealing, genetic algorithms and tabu search in bin selection problems Gondhi et al. [12] suggest ACO outperforms them. The next section talks about the metaheuristic based BFDACO algorithm for the selection of nodes for optimal social data allocation with replication and relation constraints.

5. Graph NoSQL databases allocation using metaheuristic BFDACO Lucene

As broached in Section 2, most of the previous works consider data allocation and replication as segregate problems. In data-intensive computing applications, both the replication and allocation of data affect the query processing time. The nodes assigned to allocate the data should be either the server or client choose as server as shown in Fig. 1. Here multiple nodes are considered for data allocation where one acts as server others act as clients. The server has a controller to invoke the clients during data allocation. The control manager of server performs the BFDACO metaheuristic algorithm to allocate social data efficiently by considering relation and without allocating replicated data on the same node. Each of the nodes has enough storage capacity.

Given a set of nodes and social data as blocks to be allocated, the subset of nodes are to be selected such that the sum of weights of blocks of social data allocated to each node does not overshoot the overall capacity of node. The number of nodes utilized for storage should be less and an optimal packing for improving query efficiency is procured. The previous section proved that this problem of social data allocation with replication in graph NoSQL databases is NP-Hard. This section proposes an algorithm BFDACO Lucene to select a set of adjacent nodes to perform optimum allocation with few nodes and improve query retrieval efficiency. ACO is a metaheuristic algorithm [6] adapted for solving the relation between blocks and replicated data placement problem in graph NoSQL database after integration with BFD. In graph NoSQL databases allocation problem, the selection of nodes for allocation depends on the execution of heuristic BFD taking weights of blocks into consideration. To obtain optimum solution based on relation and replication placement of blocks for social big data, ACO metaheuristic is integrated with BFD.

The pheromone components are stored in *kk* matrix, where *k* is the number of nodes. Based on the pheromone values computed a set of ants try building solutions examining replication and relation based on the pheromone values and an optimum solution is obtained. Paths with more pheromone values of relation and weight have greater probability towards selection. At the end of each solution building phase, the pheromone values are updated (parameter solution indicates this). The solution is searched based on a fitness function computed and checks whether the new solution obtained exceeds the capacity of node. Then the solution is checked whether it is formed by valid pair of grouping blocks for this valid pair function is called and the *out relation* of each data block is compared to the *in relation* of the other. This is illustrated in 3 to 9 lines of the algorithm valid pair. The parameters used in the Algorithms 1–3 are:

Algorithm 1: BFDACO metaheuristic.

```
Input: Set of nodes $k$, blocks $\{ft_j\}$, relation associated with blocks $\{r_j\}$,
weights $\{w_j\}$) for each of the blocks, $\{C_i\}$) capacity of the node
Output: Solution($k$)
1. Sol$\longleftarrow$ $k$ using BFD /*Compute Initial Solution using BFD, randomly choose node*/
2. for$a \longleftarrow$ 1 to m do
3. for$i \longleftarrow$ 1 to $k$ do
4. for$j \longleftarrow$ 1 to $k$ do
5. if$ft_j \in J_a(ft_i)$ and $ft_i \cap ft_j = \phi$
/*Probability of path chosen by ant(block) in
node i to go to node j based on pheromone relation trail*/
6. $p_a(ft_i, ft_j) \longleftarrow \frac{\zeta(ft_i, ft_j)\eta(ft_i, ft_j)^{\vartheta}}{\sum_{ff \in J_a(ft_i)} \zeta(ft_i, f_{ff})\eta(ft_i, f_{ff})^{\vartheta}}$
7. if$(ft_i, ft_j) \in$ tour of ant a and $ft_i \cap ft_j = \phi$
/*Pheromone gets updated based on the tour made by ants*/
8. $\Delta\zeta_a(ft_i, ft_j) \longleftarrow 1/L_a$
9. /*Choosing a Heuristic*/ $\zeta(ft_i, ft_j) \longleftarrow \rho\zeta(ft_i, ft_j)+\sum_{a=1}^{m} \Delta\zeta_a(ft_i, ft_j)$
10. $\eta(ft_i) \approx U_{i=1}^{\alpha} U_{j=1}^{\gamma(i,2)}(f_{i2j}) = U_{i=1}^{\alpha} U_{j'=1}^{\gamma(i,3)}(f_{i3k'})$
11. /*Getting a new sol'*/ Eliminate $j^{th}$ node setting sol'$\longleftarrow$ sol/$M_j$
12. $p_a(s, k, ft_j) \longleftarrow \frac{\zeta_k(ft_j)\eta(ft_j)^{\vartheta}}{\sum_{ff \in J_a(s,k)} \zeta_k(g)\eta(g)^{\vartheta}}$
/*Building of a Solution*/ $\zeta_k(ft_j) = \frac{\sum_{i \in k} \zeta(ft_i, ft_j)}{|k|}$
13. Place the blocks from $j^{th}$ node into any other $M_k$-1 nodes were $\Sigma_{r=1}^{n} r_n = 1$
14. if sol' is feasible then
15. sol$\longleftarrow$ sol'
16. else do /*Pheromone gets trail updated*/
17. $\zeta(ft_i, ft_j) \longleftarrow \rho\zeta(ft_i, ft_j)+t(ft_i, ft_j)fun(sol^{best})$
18. $\varsigma_{min} \longleftarrow \frac{1/(1-\rho)(1-\sqrt[fn]{pb})}{(average-1)\sqrt[fn]{pb}}$
/*Apply local search to the fitness function and check if the solution is feasible*/
19. solution$\longleftarrow$ search(sol')
20. ifsolution is feasible then
21. sol$\longleftarrow$ solution
22. return sol
```

5.1. Description of the Algorithm

Ant Colony Optimization(ACO), a meta-heuristic based technique used to energize the competence of ants to search for the minimum distance between food source and nest. The first ACO algorithm in literature called Ant System(AS), which was an answer to the transportation problem, developed by Dorigo in 1992 [17] became a preferred work after its publication and many researches have been developed as an improvement to the real algorithm. AS was also applied, integrated to a collection of various other problems. The key to AS minimum distance is a pheromone trail they leave behind while walking. This smell of pheromone trail left behind helps the ants to follow the same path. Initially, each ant chooses one path randomly, from the presented sets of possible paths resulting in a solution where almost 50% of ants going over each path. However, it is plain that the minimum distance used ants reached back faster compared to the others. Hence, after they come back there will be high pheromone on the minimum distance path, impacting others to accompany the same path. This consequence the ants to follow minimum distance among the whole colony (Table 1).

AS is depended on this biological metaphor that links the pheromone amount $\zeta(ft_i, ft_j)$ with the relation between two blocks ft_i and ft_j. The ant is randomly placed on a node. A solution is build by the movement of ant from node to node until blocks are all inserted. The probability that an ant a in node i chooses to go to node j is given by (12).

$$p_a(ft_i, ft_j) = \frac{\zeta(ft_i, ft_j)\eta(ft_i, ft_j)^{\vartheta}}{\sum_{ff \in J_a(ft_i)} \zeta(ft_i, f_{ff})\eta(ft_i, f_{ff})^{\vartheta}} \tag{12}$$

$$if\; ft_j \in J_a(ft_i)\; and\; ft_i \cap ft_j = \phi \qquad p_a(ft_i, ft_j) = 0\; otherwise$$

Algorithm 2: Search for optimum solution.

```
Input: Set of nodes $k$, blocks $\{ft_j\}$, relation associated with blocks $\{r_j\}$,
weights $\{w_j\}$) for each of the blocks, $\{C_k\}$ capacity of the node
Output: New improved solution($k$)
Function search(sol')
1. Set limprove$\longleftarrow$ true
2. while sol' is infeasible and limprove then
3. limprove$\longleftarrow$ false
/*Computation of Fitness function*/
4. for j= $M_{sol'}$ to $M_2$ then
5. for i=1... (j-1) then
6. $fitness(sol) = \sum((M_i(f_n)/C_i)^{stress}/k$
/*Check whether the new solution weight is greater than capacity of node*/
7. if $w_sol'(ft_i) > C_i$then
8. if no limprove then
9. if valid(sol',$ft_k$, $ft_j$) then
10. Obtain neighbor $sol'_{ft_k,ft_j}$ by applying the differencing
method to the $k$th and $j$th bins of solution sol'
11. if $(w_{sol'_{ft_k,ft_j}}(ft_k) - w_{sol'_{ft_k,ft_j}}(ft_j))<(w_{sol'}(ft_k) - w_{sol'}(ft_j))$ then
12. sol'$\longleftarrow$ $sol'_{ft_k,ft_j}$
13. limprove$\longleftarrow$ true
14. return sol'
```

Algorithm 3: Valid allocation based on relation.

```
Function valid(soll,$ft_k$, $ft_j$)
1. vpair$\longleftarrow$false
2. for l=1 to n then
/*Comparing the heuristic solutions based on block relations of the data to be allocated*/
3. for i $\longleftarrow$ 1 to $\alpha$ then
4. for k $\longleftarrow$ 1 to $\gamma$ then
5. $\eta(ft_l) \longleftarrow (U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)}(f_{i2k}) \equiv U_{i=1}^{\alpha} U_{k'=1}^{\gamma(i,3)}(f_{i3k'}))$
6. compare$\longleftarrow$compare$(\eta(ft_l), \eta(ft_{l'}))$
7. if compare$>$ $M_{ft_l,ft_{l'}}$ then
8. if $w_{sol'}(ft_l) \neq w_{sol'}(ft_{l'})$ then
9. vpair$\longleftarrow$true
10. return vpair
```

Table 1
Parameters used in the Algorithms.

Parameters	Explanation
$\{F_i\}_{i=1}^{\alpha}$	Set of files
$\{r_{ij}\}_{j=1}^{\beta}$	Set of records for each file
$\{f_{ijk}\}_{k=1}^{\gamma(i,j)}$	Set of fields for each of the records
k	Number of Machines taken initially
ft_i ft_j	Two blocks constituting set of files of records of data
$\{r_j\}$	Replication-associated with blocks
$\{w_j\}$	Weights associated with one of the block
$\{C_i\}$	Capacity of one node
$zeta(ft_i, ft_j)$	Pheromone trail intensity between two blocks
$\eta(ft_i, ft_j)$	Static value between blocks obtained from the inverse of relation cost between blocks
L_a	Length of complete tour created by ant
$\Delta\zeta_a(ft_i, ft_j)$	Tour of ant visiting all blocks
ρ	Speed of evaporation of pheromone value
$J_a\{ft_i\}$	Set of blocks not visited by ant a
ϑ	Maximum heuristic value compared to pheromone
ς_{min}	Based on partially best solution set, used for constructing solution
solution, *sol* and *sol′*	Parameters used for optimal solution computation
$w(sol'\{ft_i\})$	Weights of blocks selected as solution
vpair	Valid pairing of blocks based on relation

Algorithm 4: BFDACO Lucene query processing.

```
Procedure 1: Query Retrieval
1. $q \longleftarrow CypherQuery$
2. Function TransactionManagement()
3. Function CypherParser()
4. Function TransactionLog(), ObjectCache(), FileSystemCache()
5. if CypherQuery found
6. Retrieve query
7. else Function Search() in file-record-field
Procedure 2: Search in Graph Database Storage
1. while($node_{value} \neq$file-record-field)
2. while($((f_{ijk})_{k=1}^{\gamma(i,j)})_{index}.ptr \neq \emptyset$)
3. Function Index()
4. if $((f_{ijk})_{k=1}^{\gamma(i,j)})_{index}$ = $node_{value}$)
5. then get $node_{(f_{ijk})_{k=1}^{\gamma(i,j)}\ index}\ node_{property}$,get $relation_{(f_{ijk})_{k=1}^{\gamma(i,j)}\ index}\ relation_{property}$,$node_{(f_{ijk})_{k=1}^{\gamma(i,j)}\ index}$, $relation_{(f_{ijk})_{k=1}^{\gamma(i,j)}\ index}$
6. else $(f_{ijk})_{k=1}^{\gamma(i,j)}\ index$
7. end while
8. end while
9. if $node_{value}$ = file-record-field
10. then goto step4
Procedure 3: Lucene
1. Function Assign($nsr_{no}$ n, file-record-field)
2. RL $\longleftarrow$ new AssociativeList
3. for all term qt $\in$ file-record-field do
4. RLqt $\longleftarrow$ RLqt + 1
5. for all term qt $\in$ RL do /* Assignment */
6. Emit(term qt, assign n, $RL_{qt}$)
7. Function Shrink(term qt, assign[(n1,f1)... ])
8. L new List
9. L.Append(n,f)
10. L.Sort()
11. Emit(term t, assign L)
```

The heuristic to guide the ant is given in (12) denoted by $\eta(ft_i, ft_j)$. This heuristic value is the result obtained by taking the inverse of computation cost of the relation between f_i and f_j. So the choice of ant a in node i to go to node j is partially interpreted by the strength of pheromone relation between ft_i and ft_j, and relatively by the heuristic favorable of choosing ft_j after ft_i. Parameter ϑ defines the role of the heuristic information compared to the pheromone information. $J_a(ft_i)$ is the set of nodes that have not still been traversed by ant a in node i (Algorithm 4).

Once all ants have constructed a visit, the pheromone gets changed as

$$\zeta(ft_i, ft_j) = \rho\zeta(ft_i, ft_j) + \sum_{a=1}^{m} \Delta\zeta_a(ft_i, ft_j) \tag{13}$$

$$\Delta\zeta_a(ft_i, ft_j) = 1/L_a \tag{14}$$

$$if\ (ft_i, ft_j) \in\ tour\ of\ ant\ a\ and\ ft_i \cap ft_j = \phi \qquad \Delta\zeta_a(ft_i, ft_j) = \ 0\ otherwise$$

Eq. (13) left part produces the pheromone decay on all paths as the ant moves away. The speed of this decay is termed as ρ, also called as the evaporation parameter. The right part rises the pheromone value visited by ants on all the available paths. The rate of pheromone sedimented by an ant on a path is defined by L_a, the length by which the ant creates its tour. In this way, the rise of pheromone value for a path controls on the ants number that utilizes this path, and based on the quality of results found by these ants, this is given in (14).

Pheromone trail is defined as the ordering of nodes which means encoding the benefits of visiting a certain node j after another node i.

A good heuristic is chosen in line 9 of Algorithm 1 to integrate with the pheromone data used for building results. The most effective and simplest heuristic technique for solving a BPP is FFD strategy where the data are sorted in order of size and then, starting with the biggest size data, gets placed into the first bin that they fit in, this follows until all data are placed in bins.

However, since the filling of data in nodes is one by one, there is a need to reformulate FFD slightly for our case. Starting with a group of nodes, fill each node in turn by repeatedly placing the biggest data from those waiting, that will still be appropriate for the bin. If no data left are small enough to fill the node, a new node is started.

This method also gives the result similar to FFD solution but is more useful for our purpose because it shows that the heuristic convenient for a data is precisely related to its size. In other words, choosing of the next data to pack rely upon the node currently chosen, first large data should be pulled, then the relation of large data should be given more preference compared to smaller blocks. This is obtained by positioning the heuristic function in (15) for a block being favored to be equal to its size and then the relation.

$$\eta(ft_i) = w_i \; and \; \eta(ft_i) \approx U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)}(f_{i2k}) = U_{i=1}^{\alpha} U_{k'=1}^{\gamma(i,3)}(f_{i3k'}) \tag{15}$$

The heuristic information and the pheromone trail defined above is utilized by ants to construct solutions(Line 12 of Algorithm 1). Each ant begins with a group of blocks to be stored in an unfilled node. All blocks are placed one after the other to each of the nodes until none of the blocks remained are sufficiently light to fit in the node. Then no further allocation takes place in the node, and a new node is taken. The probability of ant choosing a block ft_j as the next block for the present node i, in partial solution s is given as

$$p_a(s, i, ft_j) = \frac{\zeta_i(ft_j)\eta(ft_j)^{\vartheta}}{\sum_{ff \in J_a(s,i)} \zeta_i(ff)\eta(ff)^{\vartheta}} \tag{16}$$

$$if \; ft_j \in J_a(s,i) and J_a(s,i) \cap ft_j = \phi \qquad p_a(s, i, ft_j) = 0 \; otherwise$$

In (16), $J_a(s.i)$ is the group of blocks that certify for admittance in the present node. There are blocks that are still remaining after partial allocation result s is formed, and are sufficiently light to fit in node i. w_j is the weight of block ft_j. The value of pheromone $\zeta_i(ft_j)$ is given in (17) for block ft_j in a node i. It is the total pheromone values sum between block ft_j and the block ft_i that are already in node i(that is if

$$U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)}(f_{i2k}) \equiv U_{i=1}^{\alpha} U_{k'=1}^{\gamma(i,3)}(f_{i3k'})$$

), divided by the block number in i. If i is vacant, $\zeta_k(ft_j)$ is set to 1.

$$\zeta_k(ft_j) = \frac{\sum_{i \in k} \zeta(ft_i, ft_j)}{|k|} \; if \; (k \neq \phi) \quad \zeta_k(ft_j) = 0 \; otherwise \tag{17}$$

Fragments ft_i and ft_j are related, and they go together different times in various nodes to form the best solution. If we can increase $\zeta(ft_i, ft_j)$ each time ft_i and ft_j relations are grouped. Hence finally, we can obtain (18).

$$\zeta(ft_i, ft_j) = \rho\zeta(ft_i, ft_j) + t(ft_i, ft_j) fun(sol^{best}) \tag{18}$$

In (18), $t(ft_i, ft_j)$ indicates how many times ft_i and ft_j relation go together is the best solution sol^{best}. Eq. (18) is obtained from (12).

The utilization of only the best feasible ant for updation upgrades the search more assertive. This best feasible solution gets a lot of exploration by the combination of nodes used. Therefore it is possible to take from Ning et al. [18] to stabilize exploration versus exploitation. The first primary choice is to choose between the solutions by using the iteration-feasible ant sol^{ik} and the global-feasible sol^{gb}. Usage of sol^{gb} marks strong exploitation if the value gets alternated with sol^{ik}. A parameter ν indicates the number of updations we analyze before sol^{gb} used again.

Another method of intensifying the exploration is acquired by defining an upper and lower bound ς_{max} and ς_{min} for the pheromone values similar. The MAX-MIN problem stated by Stützle and Hoos [19] explain the value for the upper and lower bound for ACO based gene optimization. In our approach, usage of an upper bound is not feasible. This is because, counting on the number of times block size and relations go together in the good solutions, pheromone data get reinforced more, and they progress to different values. Different maximum values depending on pheromone entries have to be used in our case. The range of maximum value selected lies between 0.9 and 1.6.

We do use the lower bound ς_{min}. Stiltzle and Hoos compute the value for $MIN(\varsigma_{min})$, in their paper depending on best (pb in our case), the anticipation of assembling the good feasible result found when all the pheromone values have intersected to either ς_{max} or ς_{min}. An ant builds up the best feasible result found if it accumulates at every point during result construction the block with the highest pheromone value. Beginning from this, Sttitzle and Hoos find the following formula for ς_{min} in (19).

$$\varsigma_{min} = \frac{\varsigma_{max}(1 - \sqrt[ft_n]{pb})}{(average - 1)\sqrt[ft_n]{pb}} \tag{19}$$

Variable ft_n in (19) indicate the block, and *average* indicates the average figure of blocks to be chosen at every conclusion point while constructing a result, defined as blocks/2 (ft_n/2). Here the formulae of Stützle and Hoos [19] methodology is adopted but ς_{max} is replaced by $1/(1-\rho)$. This is, in fact, an approximation of ς_{max} as calculated by Stiitzle and Hoos for values of the sum of bins m as 0 or 1 and finally substituting the fitness of the best feasible result as 1. Fitness optimal solution can be found in (20).

$$\varsigma_{min} = \frac{1/(1-\rho)(1-\sqrt[ft_n]{pb})}{(average-1)\sqrt[ft_n]{pb}} \tag{20}$$

Several combinations of blocks are possible based on the interference with relations quoted in (20) obtained. The cost of *pb* should be considered as a crude approximation for constructing the best solution.

To model the proposed algorithm towards best result, it is very important to assess the quality of the results pb obtained. Therefore it requires a fitness function. Falkenauer et al. in their paper [20] define a genetic algorithm based fitness function for BPP as two to be the perfect result, because the pieces are grouped into less number of bins possible.

$$fitness(sol) = \sum_{i=1}^{K}((M_i(f_n)/C_i)^{stress}/K \tag{21}$$

Eq. (21) calculates the fitness solution where, k is the nodes number, $M_i(f_n)$ the sum of blocks of node i, and C_i the maximum capacity of node i. The parameter *stress* defines how much stress is filled on the nominator indicating the allocation of data in nodes as opposed to the denominator which indicates the sum of nodes. Setting variable *stress* to 1 makes us take the inverse of the number of nodes. If the value of *stress* can be increased a better fitness to feasible results that contain a combination of well-occupied and less well-occupied nodes can be obtained, rather than equally filled nodes. Burke et al. report in their work that a value of 2 seems to be optimal. This fitness optimum value is again proved by Dosa in [21] and by Falkenauer et al. in [20]. Falkenauer et al. also suggests values higher than 2 can lead to incomplete convergence, as the fitness of suboptimal results can come too near to the fitness of optimal results. They prove algebraically using the genetic algorithm set problem of crossover and mutation that if k(*stress* in our case) is given a value larger than 2, the best result of N + 1 clustering of nodes with NF full nodes would get a fitness greater than a result with N equally filled nodes.

5.2. Computational cost

The number of nodes used in the algorithm is M_k and the ants is m. The number of times the nodes get traversed to allocate blocks is $M_k{*}M_k$. Lucene is used for initial allocation of blocks to nodes hence the complexity becomes $ft_n log(ft_n)$. Lines 2–21 takes $m{*}M_k{*}M_k$ times in ACOLucene. The algorithm search(sol') runs with a cost of $sol'{*}M_k{*}M_k$ times and the algorithm to compute valid(sol'$,ft_k$, ft_j) takes a cost of $n\alpha\beta$. The algorithm search(sol') and valid(sol'$,ft_k$, ft_j) are a part of ACOLucene. Hence the overall computational cost for the algorithm of ACOLucene is,

$$O(ft_n log(ft_n) + m * M_k * M_k + sol' * M_k * M_k + n\alpha\beta)$$

From MathML [22] view the computational cost can be written as $M_k >> m$, which is $O(sol'{*}M_k{*}M_k)$. Hence the overall time complexity of the algorithm is $O(sol'{*}M_k{}^2$, therefore the computational cost depends on the number of nodes used in the scenario of social data allocation in graph NoSQL databases.

5.3. Lucene index

An indexing service named Lucene[23] is built after BFDACO to create a full-text search index. Lucene helps to open each file $U_{i=1}^{\alpha}F_i$ in SetOpenMode function and insertion of records $U_{j=1}^{\beta}(r_{ij})$ are done using IndexWriter function. IndexDoc function creates the index after each IndexWriter function called. Lucene uses the Cypher analyser to analyze the type of input data. Fig. 2 explains how server-client Lucene allocation is done in graph databases. Each data block of file records NodeStoreRecord(NSRs) are subjected to assign phase where the file is opened and the records are assigned. If there exist similar records the group key function groups the records of each node using a unique group key. This helps during further processing of records. The sort function sorts the records in a structured form to help query processing. Combine/Merge function combines the similar records from nearby nodes without affecting the BFDACO allocation. Filter and IndexWriter function filters are redistributed the grouped terms and create an index term such that all other similar terms will remain to indexed term in Shrink function. Hence the index is ready for any execution of the query to be processed.

6. Experimental results of metaheuristic BFDACO on graph NoSQL databases

The proposed system is implemented on graph NoSQL databases architecture. The proposed algorithms described in this paper are developed in Java with graph databases. Choose any database among the five graph NoSQL databases and login to any social network account example Twitter, Facebook or LinkedIn. In this case, Facebook and Twitter accounts are considered for login. For experimental setup Fujitsu System 3100 Q4 with Intel Xeon E3-1220 processor, 16GB RAM and 1 TB hard disk, Dell Precision Tower 5810 with Intel Xeon E5-1620V3, 3.5 GHz, 16GB and 1 TB hard disk, and five Dell OptiPlex 7010, Core i3, 3220, 3.3 GHz with 2GB RAM and 250GB hard disk. All these nodes should be close to each other in order to avoid queuing delay and minimize the cost of transmission delay. In many cases, there could be processing delay at server

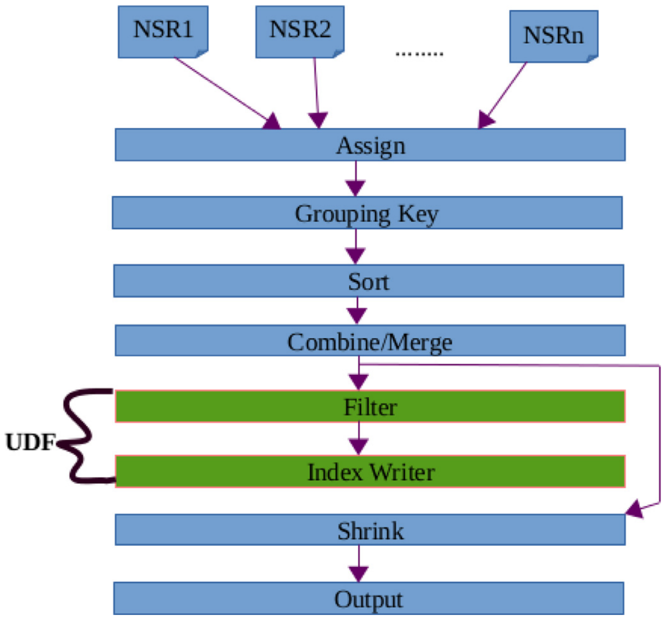


Fig. 2. Server-Client Lucene Allocation Framework of Graph Databases.

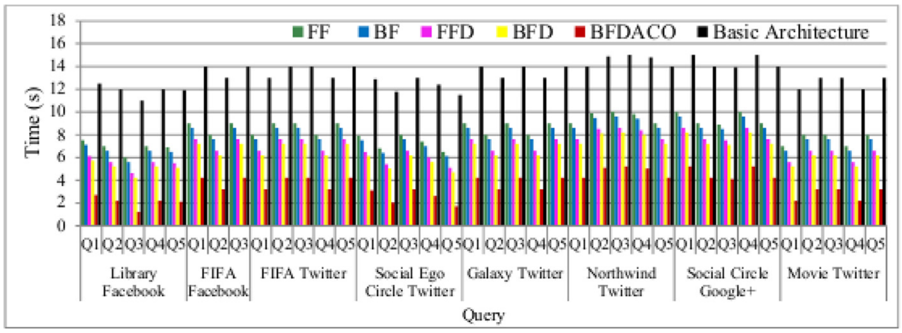


Fig. 3. Neo4jHA graph data allocation with heuristics available in literature.

side as the server need to analyze the type of query and match the query terms with the data in nodes. The server (Fujitsu) receive heart beats from clients Dell Precision and five Dell OptiPlex 7010 every five seconds.

Seven datasets namely FIFA 2015, Galaxy, Ego Network, Northwind, Social Circle, Movie, and Library were tested on graph NoSQL databases. Each dataset with different sizes was separately stored on each of the nodes based on the BFDACO run by the server using the controller to invoke the clients during data allocation. The control manager of server allocates social data efficiently by considering relation and without allocating replicated data on the same node using the proposed BFDACO procedure.

The graph NoSQL databases allocation is an NP-Hard problem and we need to optimize the results such that less number of nodes are used for storage to avoid delay in latency. The optimization is solved using the fitness function suggested by Dosa [21] and E.Falkenauer et al. in [20] for bin packing where the global best solution is fixed and global minimum is iterated and found. ACO intensify the search using global minimum solution using repeated values of ς_{min} for fixed number of times and setting a ς_{max} with an approximation factor of 2 to get the optimal solution.

All the seven datasets are tested and the results obtained are shown from Figs. 3 to 7 for graph data processing. This graph data processing is done on graph databases like Neo4jHA, AllegroGraph, FlockDB, OrientDB, Infogrid with heuristics like FF, BF, FFD, BFD available in literature and the proposed BFDACO. Figs. 8–12 indicate the graph data processing with BFDACO (replication 1 and 3) and Lucene index for the stored seven datasets on five graph databases used in social networks. These results show that the data allocation using BFDACO and Lucene index is optimum in processing time compared to other heuristic approaches from literature in data allocation.

The limitation of the experimental setup was the processing delay to analyze the data present in which node.

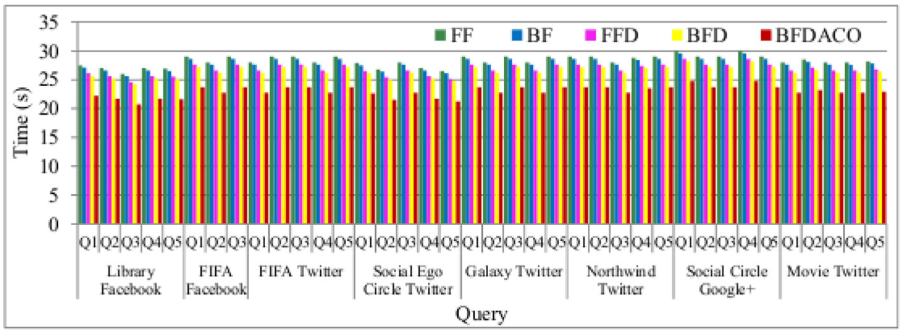


Fig. 4. Allegrograph graph data allocation with heuristics available in literature.

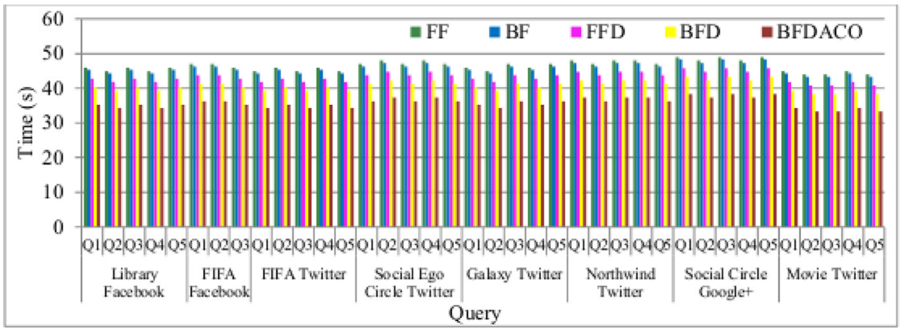


Fig. 5. Infogrid graph data allocation with heuristics available in literature.

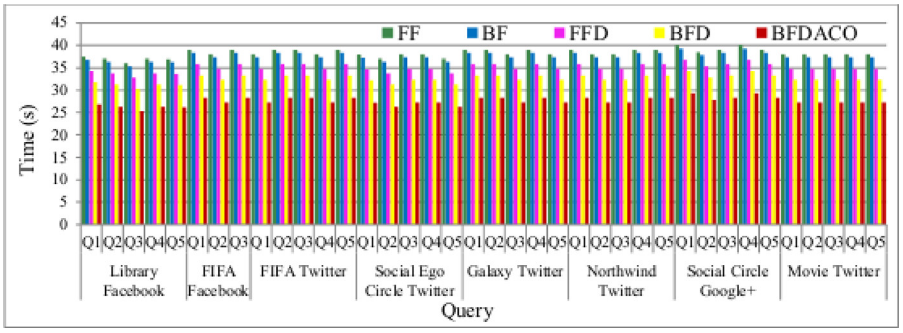


Fig. 6. OrientDB graph data allocation with heuristics available in literature.

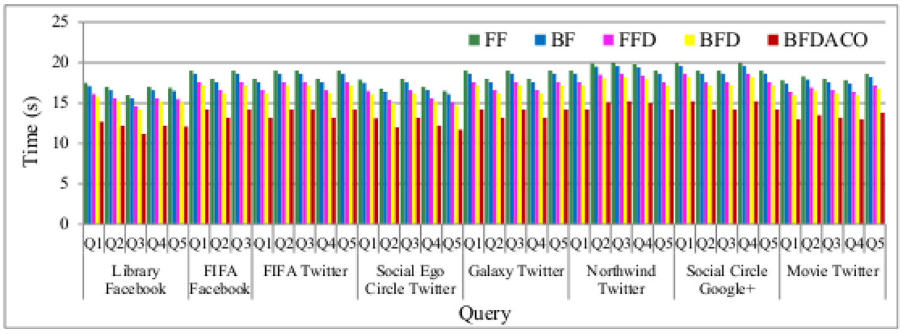


Fig. 7. FlockDB graph data allocation with heuristics available in literature.

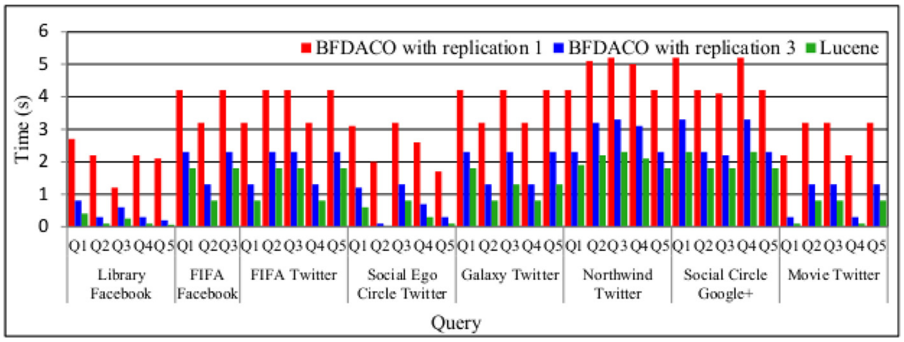


Fig. 8. Neo4jHA graph data allocation with BFDACO and Lucene.

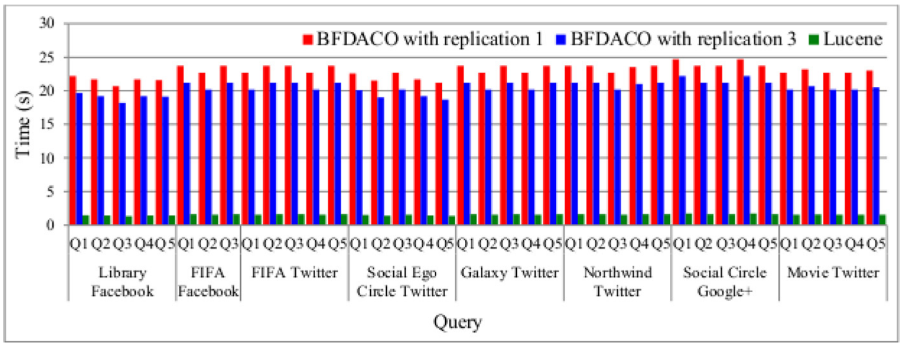


Fig. 9. Allegrograph graph data allocation with BFDACO and Lucene.

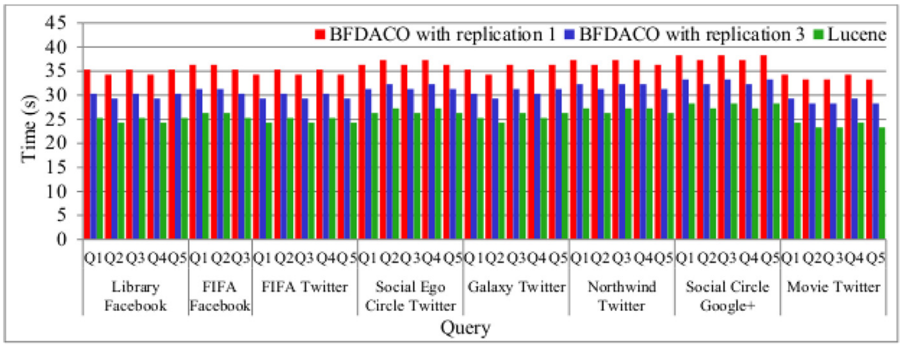


Fig. 10. Infogrid graph data allocation with BFDACO and Lucene.

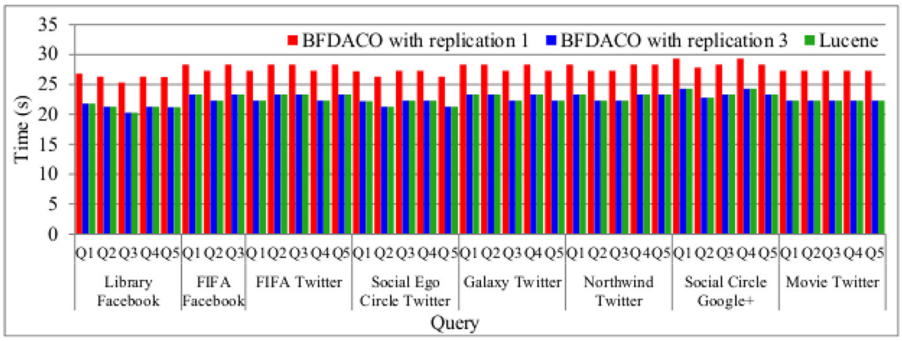


Fig. 11. OrientDB graph data allocation with BFDACO and Lucene.

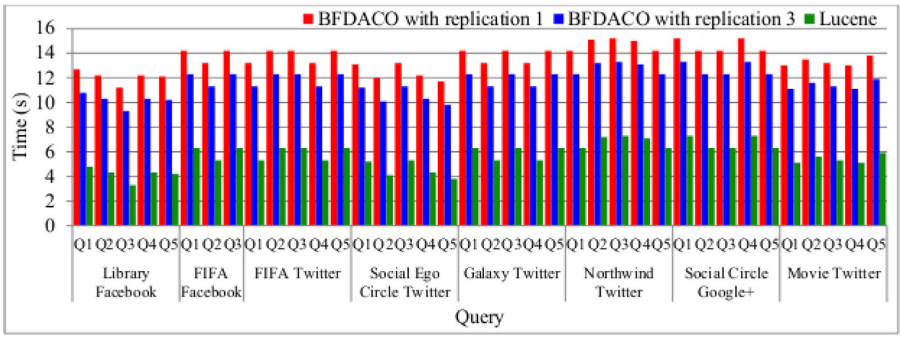


Fig. 12. FlockDB graph data allocation with BFDACO and Lucene.

7. Conclusion and future scope

Social data allocation in graph NoSQL databases integrated with a social network helps for query processing using cypher query language. The main goal of this work is to obtain optimal allocation of data considering replication and relation with less number of nodes such that the total allocation space for blocks of data does not exceed the node capacity. It will result in less overhead and efficient query retrieval process. A metaheuristic algorithm, Ant Colony Optimization integrated with BFD is used for selecting the nodes that satisfy this objective. The replicated blocks of data are stored to the nodes using BFD in such way that no two blocks go the same node and do not exceed the overall node capacity. The relations between blocks of data are considered as the pheromones the ant produces when blocks of data are transferred from one node to another. Lucene index is applied to index the data residing in each node. This could support a faster query processing path. Simulation results demonstrate how BFDACO with Lucene in graph NoSQL databases data allocation reduces the amount of time during query execution compared to the other approaches were FF, BF, FFD and BFD are only considered.

In this paper, since our main objective is to optimize query processing time the replication factor for blocks of data is considered as one and three, as part of as future work replication factor can be contemplated as ‘n’. The limitation of processing delay can be overcome by the introduction of efficient index mechanism in the near future.

References

[1] Ning Z, Guo L, Peng Y, Wang X. Joint scheduling and routing algorithm with load balancing in wireless mesh network. Comput Electr. Eng. 2017;38(3):533–50.
[2] Maheswari S, Karpagam G. Performance evaluation of semantic based service selection methods. Comput Electr Eng 2017.
[3] Brigit MA, Madhu KS. Novel research framework on SN's NoSQL databases for efficient query processing. Int J Reason-based Intell Syst 2015;7(3–4):330–8.
[4] Sun G, Liu Y, Li H, Wang A, Liang S, Zhang Y. A novel connectivity and coverage algorithm based on shortest path for wireless sensor networks. Comput Electr Eng 2017.
[5] Kavitha N, Malathi P. Analysis of congestion control based on Engset loss formula-inspired queue model in wireless networks. Comput Electr Eng 2017.
[6] Mathew AB, Kumar SM, Krishnan KM, Salam SM. Efficient query retrieval in Neo4jhA using metaheuristic social data allocation scheme. Comput Electr Eng 2017.
[7] Li T, Zhou X, Wang K, Zhao D, Sadooghi I, Zhang Z, et al. A convergence of key-value storage systems from clouds to supercomputers. Concurr Comput 2016;28(1):44–69.
[8] Braun TD, Siegel HJ, Beck N, Bölöni LL, Maheswaran M, Reuther AI, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 2001;61(6):810–37.
[9] Assunção MD, Calheiros RN, Bianchi S, Netto MA, Buyya R. Big data computing and clouds: trends and future directions. J Parallel Distrib Comput 2015;79:3–15.
[10] Zhang Y, Chen M, Mao S, Hu L, Leung V. Cap: community activity prediction based on big data analysis. IEEE Netw 2014;28(4):52–7.
[11] Gil D, Song I-Y. Modeling and management of big data: challenges and opportunities. 2016.
[12] Gondhi NK, Sharma A. Local search based ant colony optimization for scheduling in cloud computing. In: Advances in computing and communication engineering (ICACCE), 2015 second international conference on. IEEE; 2015. p. 432–6.
[13] Mathew AB, Kumar SM. An efficient index based query handling model for Neo4j. IJCST 2014;3(2):12–18.
[14] Mathew AB, Pattnaik P, Madhu Kumar S. Efficient information retrieval using Lucene, Lindex and Hindex in Hadoop. In: Computer systems and applications (AICCSA), 2015 IEEE/ACS 11th international conference on. IEEE; 2015. p. 333–40.
[15] Stützle T. Ant colony optimization. In: International conference on evolutionary multi-criterion optimization. Springer; 2009. p. 2.
[16] Coffman Jr EG, Csirik J, Johnson DS, Woeginger GJ. An introduction to bin packing. Bibliographie Siehe www.inf.u-szeged.hu/~csirik 2004;.
[17] Maniezzo A. Distributed optimization by ant colonies. In: Toward a practice of autonomous systems: proceedings of the first European conference on artificial life. Mit Press; 1992. p. 134.
[18] Ning X, Lam K-C, Lam MC-K. Dynamic construction site layout planning using max-min ant system. Autom Constr 2010;19(1):55–65.
[19] Stützle T, Hoos HH. Max–min ant system. FuturGenerComputSystems 2000;16(8):889–914.
[20] Falkenauer E, Delchambre A. A genetic algorithm for bin packing and line balancing. In: Robotics and automation, 1992. proceedings., 1992 IEEE international conference on. IEEE; 1992. p. 1186–92.
[21] Dósa G, Li R, Han X, Tuza Z. Tight absolute bound for first fit decreasing bin-packing: Ffd (l) 11/9opt (l)+ 6/9. Theor Comput Sci 2013;510:13–61.
[22] Miner R. The importance of MathML to mathematics communication. Not AMS 2005;52(5):532–8.
[23] Musto C, Semeraro G, de Gemmis M, Lops P. A hybrid recommendation framework exploiting linked open data and graph-based features. In: Proceedings of the 25th conference on user modeling, adaptation and personalization. ACM; 2017. p. 375–6.

Anita Brigit Mathew is a Research Scholar in the Department of Computer Science and Engineering, National Institute of Technology Calicut. She received her M.E. in Computer Science and Engineering from Karunya Institute of Technology. Her research interests include Social Big Data Analytics and Management.