Multi-Objective Query Optimization for Mobile-Cloud Database Environment

Seyyed Kamal Vaezpour, Mohammadreza Mollahoseini Ardakani*, Kamal Mirzaie**
Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran
Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran
Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran

Abstract:
Mobile-cloud computing is one of the most prominent infrastructures of mobile technologies of the future, as it accumulates the advantages of both mobile computing and cloud computing, providing optimized services to users. In a database system distributed in the cloud, connections necessary for a query plan may be stored on multiple sites, which exponentially increases the number of possible equivalent plans in the search for an optimum query execution plan. However, a thorough search of all possible plans is not computationally logical in such a large search space. In this study, we aim to identify a cost model including a multi-objective function with diverse (and possibly contradictory) QoS parameters to solve the query optimization problem in heterogeneous (in terms of pricing models) and mobile cloud databases. Then, we propose a novel strategy to optimize queries in such environments using the Teaching-Learning-Based Optimization (TLBO) algorithm. Finally, the obtained results are evaluated in the CloudSim environment and compared with genetic optimization and Ant Colony Optimization (ACO).

Keywords: Cloud Computing, Distributed Database, Teaching-Learning-Based Optimization, Query Optimization.
DOI: 10.24297/j.cims.2023.4.6

1. Introduction

India mobile cloud computing (mcc)

Mobile devices (e.g., smartphones and tablets) are increasingly becoming an integral part of people's lives as the most effective and convenient time- and place-unbound communication tools. Mobile computing's rapid advancement has become a strong trend in developing information technology, business, and industry. Mobile devices, on the other hand, face

numerous challenges in terms of resources (e.g., battery life, storage, and bandwidth) and communications (e.g., mobility and security)[1][2]. The lack of resources makes it difficult to improve service quality.

With the growing popularity of mobile applications and the increasing number of user requests, resource constraints in such devices have become more prominent. Some of these challenges have been addressed by combining mobile devices and cloud computing. Cloud computing opens up new possibilities for mobile app development by allowing mobile devices to keep a very thin layer for user applications while offloading computational and processing overhead to the cloud. Mcc is one of the most important areas of future mobile technology because it takes advantage of both mobile processing and cloud computing to provide users with the best possible services.

In its most basic form, mcc refers to the infrastructure that stores and processes data outside of the mobile device (mcc association). Mobile cloud applications move computing and storage power away from mobile subscriptions and into the cloud. Mcc is also described as a new mobile app model that moves data processing and storage from mobile devices to cloud-based, powerful, and centralized computing platforms (aepona). These centralized applications are then accessed via a thin client or web browser via a wireless connection from mobile devices. On the other hand, mcc is a combination of mobile web and cloud computing, and it is the most widely used tool for giving mobile users access to internet applications and services. In a nutshell, mcc is a cloud-based data processing and storage service for mobile users. Since all complex computing modules can be processed in the cloud, mobile devices do not require robust configuration (e.g., processor speed and storage capacity)[3][4].

Mcc architecture

According to the definition provided for mcc, the general mcc architecture can be introduced, as illustrated in fig. 1. Mobile devices connect to mobile networks through base stations, as shown in fig. 2 (e.g., base transceiver station, access point, or satellite). These stations create and control network and mobile device connections (aerial links). Mobile users' requests and information (e.g., id and location) are sent to central processors connected to servers that provide mobile network services[5][6]. Mobile network operators can provide services like authentication, licensing, and auditing to mobile users based on subscription data stored in databases. Requests for subscriptions are then sent over the internet to a cloud. Cloud controllers handle

requests in the cloud to provide mobile users with cloud services. Utility computing, virtualization, and service-oriented architecture (soa) (e.g., web, application, and database servers) are all used to create these services[7].

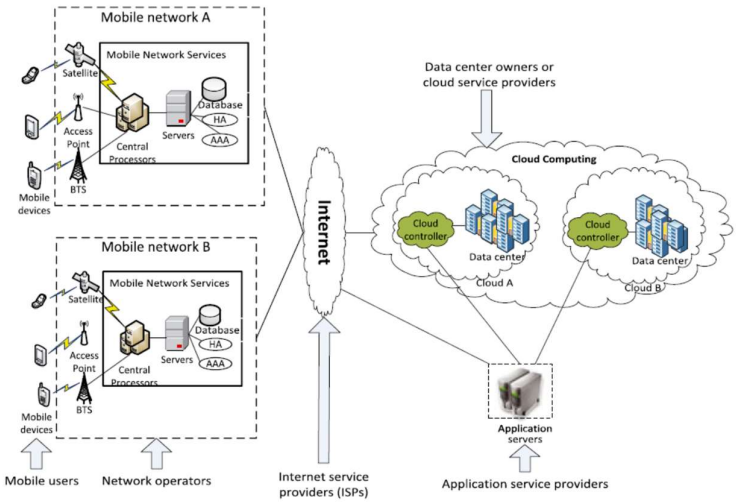

Fig. 1: mcc architecture [8]

Query optimization in a mobile-cloud database environment

Queries are sent from mobile devices to retrieve data stored in the cloud or possibly in mobile devices in a mobile-cloud database environment[9][10]. In this environment, the process of determining an optimal query execution plan is critical in many ways. Organizations attempt to adjust query execution monetary costs to fit their budget in an application scenario where many queries are executed on a daily basis. They are also interested in reducing query execution time to meet customer query response time requirements and maximize employee productivity. Furthermore, users tend to reduce power consumption when queries are executed on mobile devices[4]. This optimization process includes a vast array of contradictory statements when taking different cloud pricing models into account[11].

Current decision-making strategies are primarily concerned with a single objective: execution time. Fig. 2 depicts three query execution plans: monetary costs (m), execution time (t), and energy consumption (e). Because these qeps tend to cost the least in one of these three objectives, focusing on a single objective always leads to deciding whether to execute qep1 or qep2 plans. Qep3 is never chosen because it lacks the lowest cost in any of the three objectives listed above, even though it could be a competitive choice if all three objectives are given equal

weight. Thus, making a comprehensive decision necessitates implementing a strategy that encompasses all objectives simultaneously[5], [12][13][14][15][16].

QEP1: {M= $0.080; T= 0.5s; E= 0.012 mA}
QEP2: {M= $0.050; T= 3.0s; E= 0.300 mA}
QEP3: {M= $0.055; T= 0.6s; E= 0.013 mA}

Fig. 2: an example of plan execution costs [3]

According to the definitions, topics, and challenges discussed in the previous sections, this study initially identifies a cost model comprising a multi-objective function with various and possibly contradictory qos parameters to solve the query optimization problem in heterogeneous (in terms of pricing models) and mobile-cloud database environments, and then present it using a strategic teaching-learning algorithm for query optimization in such environments.

2. Research method

The purpose of this study is to optimize distributed queries in mobile cloud environments. For this purpose, a cost function should be used to consider the parameters of query execution time and energy consumption in mobile cloud environments and minimize them according to user preferences. Moreover, the teaching-learning based optimization (tlbo) method is applied to yield the desired results.

Figure 3 illustrates a simplified schematic of the research method.

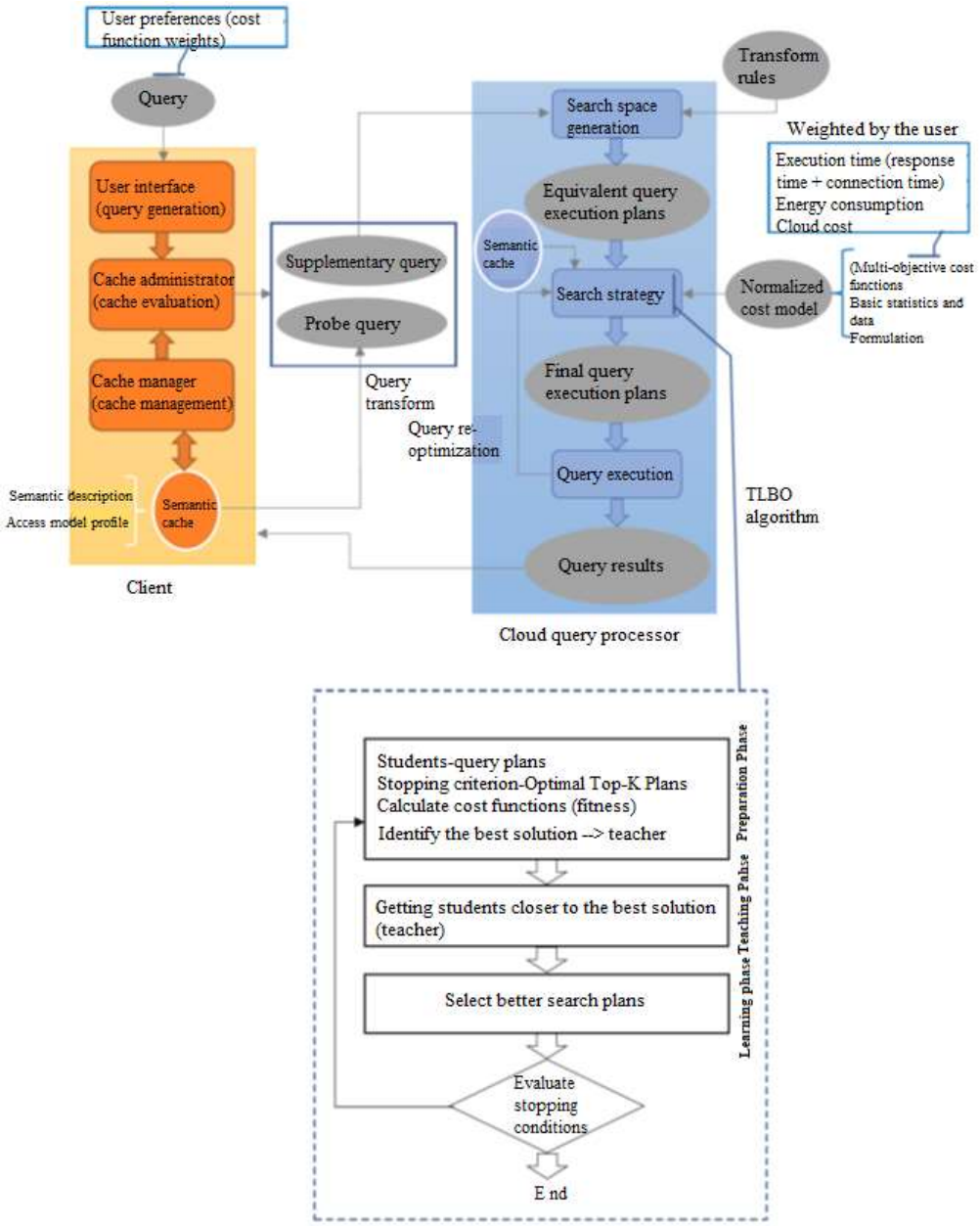

figure 3: research method

Each client (e.g., tablet, cellphone, etc.) Consists of the following three parts:

1) A user interface through which users can export their queries.
2) A query administrator to evaluate queries
3) A cache manager to manage semantic cache (e.g., a client database)

First, the query administrator analyzes the query issued to determine if it can be locally answered. If all the required data items are available in the local cache, the query becomes a

local query called fully responsive. Conversely, if only part of the query can be locally evaluated, it becomes a probe or supplementary query called partially responsive. A probe query refers a local query that allows retrieve data items from the client cache, while a supplementary query is sent to a user-friendly cloud query processor for evaluation based on query preferences. Supplementary query exactly associates missing data items. The process by which a given query is converted to its own probe or supplementary counterparts is called the query transform process. As soon as the result of a supplementary query is received by the client, the cache manager will save it in the local cache. Final query result is obtained by adding probe or supplementary queries[14], [17]–[20].

Figure 4 shows the structure and strategy of the semantic cache optimizer.

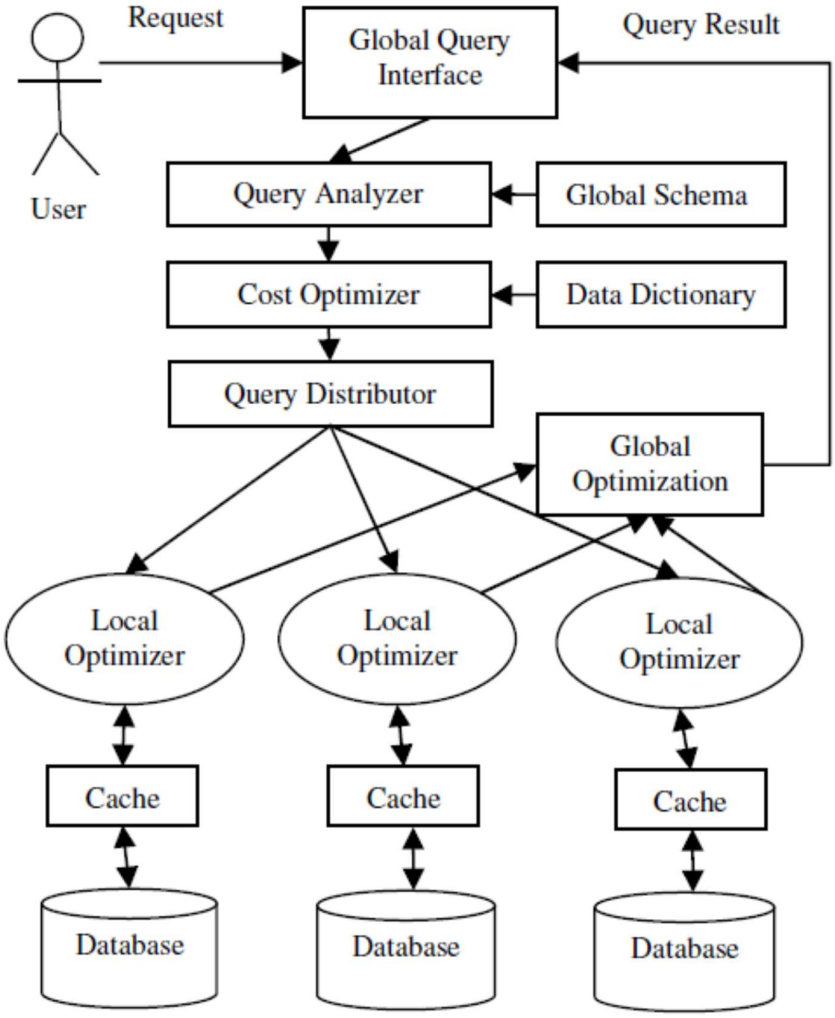

Figure 4: cache-based query optimization [21]

Cache-based query optimization in a mobile cloud distributed database environment is as follows:

Step 1: get a query obtained using one of the search methods.
Step 2: calculate the cost function for the query using the following method.
The following values are calculated for each server base i in which the main query subquery is copied:

1) **Current queue length q (li):** the number of running processes to be executed on the server
2) **Server distance sd (i):** server distance refers to the geographical distance between server and client. That is, the closer the server is, the less time it takes for the data to be fetched from the server.
3) **Server capacity sc (i):** the number of executable processes without normal server operations being disrupted
4) **Server loading:** the ratio of actual processes running on the server to server capacity

The next step is to prioritize databases where the original query subqueries are copied (p).
Pr = pr + (q (li) -q (lj))/10
Pr = pr + (sd (i) -sd (j))/100
L (i) = ql (i)/sc (i)
L (j) = ql (j)/sc (j)
Pr = pr + (l (i) -l (j))
Pr = high-priority server
Qli = length of server i queue
Qlj = length of server j queue
Sdi = distance between server i to the responding server
Sdj = distance between server j to the responding server
Li = server i fetch

This algorithm initially prioritizes all databases according to the parameters above. It then sends the subquery, for example, to the i database, and explores the local cache of the i database. If the data is available in the database cache i, it will fetch the results from the local cache; otherwise, it attempts to fetch the results from the database, update the local cache, and finally, fetch the results.

Eno	Ename	Title
e1	J.doe	Elect
E2	M.smith	Syst.anal
E3	A.lee	Mech.eng
E4	J.miller	Programmer
E5	B.casey	Syst.anal

A cloud query processor first attempts to create a search space using the conversion rules introduced in classical texts. Search space refers to a set of equivalent and alternative execution plans for input queries. While these schemes are considered equivalent in terms of yielding similar results, they differ in terms of operator implementation order, operator implementation method, and consequently, efficiency.

In the next step, a cost model is developed, including cost functions, basic statistics and data, and formulation. The purpose of the cost model is to estimate the costs of each search space execution plan. In order to ensure proper performance, the cost model needs to be armed with the profound knowledge of the execution environment. In a query optimization problem for a heterogeneous (in terms of pricing models) cloud and mobile database environment, the cost model function is a multi-objective function with varying and potentially contradictory qos parameters. Qos parameters are considered execution time (a traditional parameter, response time + connection time), monetary costs (cloud computing environments with heterogeneous pricing models provide on-demand base services, where users have to pay the query execution cost), and energy consumption mobile devices (it is vital to consider energy constraint in query-issuing mobile devices).

This research uses the query below to achieve a cost model: Query: name of employees working on the “cad/cam” project.

Figure 5: query tables [22]

The above sql query commands will be as follows:

Select emp.ename

From emp, asg, proj

Where emp.eno = asg.eno and pname = "cad/cam"

Relational algebra commands equivalent to the above-stated commands will be as follows:
Π ename (empeno ∞ (σpname = "cad/cam" (proj)))))

The above query can be processed using the join and semijoin techniques.
The semijoin method is preferred to the join method in distributed database environments because less data volume will be exchanged between various sites to evaluate a given query. This is critical because the purpose of this research is to reduce query execution time[6], [23], [24].

Evaluating the above query using the semijoin method:
Let's assume that the database is available as a distribution on the following four sites:
Emp (eno, ename, title), asg (eno, pno, resp, dur), proj (pno, pname, budget, loc), pay (title, sal)

The above query can be processed using the semijoin technique as follows:
Step 1: restrict rproj (pname = "cad/cam")
Project pname from restriect rproj
Step 2: transmit the result r1 (from step 1) to asg site
Step 3: join result r1 (step 1) and rasg (r2)
Step 4: transmit result r2 (step 3) to emp site
Step 5: join result r2 (step 3) with restricted remp at emp site

Cost semijoin strategy:
Processing cost at step 2: n1 * tcost-a
Processing cost at step 3: n1 * n2 * ccomp-tuple + n3 * ccn-tuple
Processing cost at step 4: n3 * 5 * tcost-a
Processing cost at step 5: n1 * n3 * ccom-tuple + n3 * ccn-tuple

Total cost function is derived from total processing cost in the above steps as follows:
Total processing cost: (n1 + 5 * n3) tcost-a + 2 * n3 * ccn-tuple + (n1 * n2 + n1 * n3) * ccomp-tuple
N1 =number of relation tuples r1
N2 = number of tuples in relation r2

N3 = number of tuples in relation r3

Tcost-a =transfer cost per attribute

Ccomp-tuple = cost per row comparison

Ccn-tuple = cost per row connection

The next important issue in distributed query optimization in mobile cloud environments is the amount of energy consumed by mobile devices. To this aim, the following formulation is used:

Eij = pj * tij if xij = 1

Eij = 0 if xij = 0

Pj = pjbusy - pjidle

Tij = li fj^{-1} if xij = 1

Tij = 0 if xij = 0

Li = (n1 + 5 * n3) tcost-a + 2 * n3 * ccn-tuple + (n1 * n2 + n1 * n3) * ccomp-tuple

Parameter explanation:

Eij = amount of energy consumed by the mobile device i on which the query j is executed.

Tij: query execution time j on mobile device i

Pj = amount of energy consumed by mobile device j

Xij: it determines whether query i is available on mobile device j.

Pjidle = amount of energy consumed by mobile device j in idle mode

Pjbusy = amount of energy consumed by mobile device j in busy mode

Fj^{-1} = computing capacity of mobile device j

The purpose of this study is distributed query optimization in mobile cloud environments. For this purpose, a cost function should be used that considers and minimizes the parameters of query execution time, energy consumption, and monetary costs in mobile cloud environments. The proposed cost function, meeting the above conditions, will be as follows:

$F=\min(\sum_{i,j=1}^{n}(wj(Eij)+Wj(Tij)+Wj(Cj))$

Parameter explanation:

Tij= query execution time

Wj = user-defined weight based on their preferences

Cj = monetary costs for each site in a distributed environment

The above cost function includes a parameter called w, which is determined by the user based on his preferences, prioritizing which of the above three parameters (ie, execution time, energy consumption, and monetary costs). Moreover, the tlbo method was employed to optimize the cost function. The steps taken to implement this algorithm will be described below.

3. Teaching-learning based optimization method:

Teaching-learning based optimization (tlbo) is a novel optimization method introduced by r.v. Rao et al.[25]. This method is based on the influence of the teacher on students. Similar to all other nature-inspired algorithms, tlbo is a population-based method that utilizes a population of solutions to reach a global solution[26][27]. Population refers to a group of learners or students in a class. A teacher tries to improve knowledge level in the classroom by teaching learners. This way, each student can achieve an acceptable score/grade according to his/her ability. Indeed, a good teacher tries to improve the knowledge level of his/her students. A teacher can be defined as a knowledgeable community member who shares his/her knowledge with his/her students so that the best solution (best member in the entire population) functions as a teacher in the same iteration. The tlbo method consists of two phases: 1) teacher phase, including the learning process from the teacher and 2) learner phase, in which students learn through interaction with other learners[11]. Herein, a set of queries generated using the above cost function is considered as a population of learners or students. Then, a query is randomly selected as the teacher. Teacher phase is obtained using the following formula:

Xnew, d = xold, d + r (xteacher, d-tfmd)

Where d is the number of queries (problem variable), xoldd is the old member, who still has to learn from the teacher to increase his/her level of knowledge, including a 1 * d vector, containing the results of each topic or course. Besides, r is a random number in the [0, 1] range, xteacher, d is the best member in the entire population in this iteration, which tries to shift the mean result of the class (population) to its position. Moreover, tf is the teaching factor, and md is a 1 * d vector, containing the mean values of the class results for each topic. Tf can take 1 or 2, an innovative step with equal probability of chance. A new xnewd member is accepted if it proves better than the old member.

Learner phase: learners try to increase their level of knowledge in two ways, one through teacher and the other through interaction. One learner interacts with other learners randomly

through group discussion, presentation, etc. A learner will learn new things from other students on condition that others are equipped with a higher knowledge level than his/her's:

Xnew, i = xold, i + ri (xj - xk)

Where the index i varies from 1 to the total number of members, xold, i is an old member that has not yet learned anything from interaction with other students, ri is a random number in the [0, 1] range. Besides, xj and xk represent two randomly selected interactions (student) on condition that j ≠ k and the objective function xj are better than xk. A new member of xnew, i is accepted if it proves better than the old member, xold, i.

Figure 6 shows the flowchart of the tlbo algorithm.

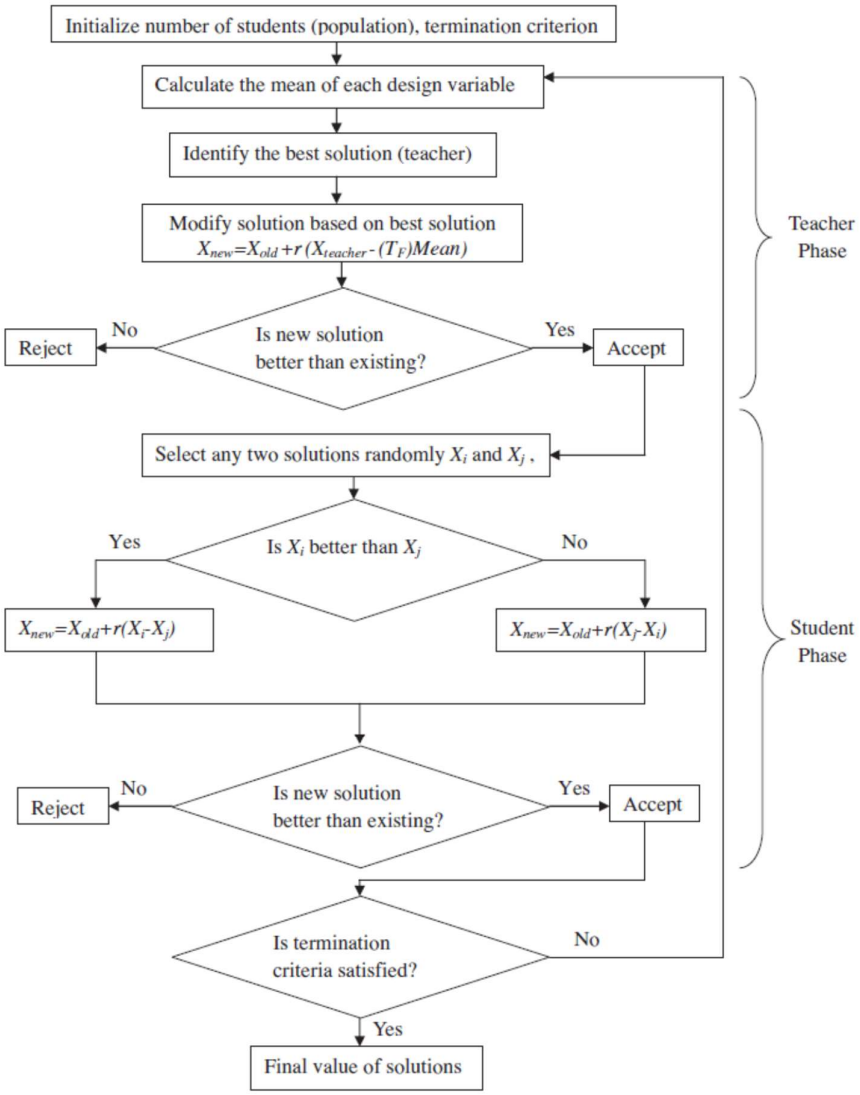

Figure 6: flowchart of the tlbo algorithm [28]

4. Simulation results:

In this simulation, three algorithms are applied, namely tlbo algorithm, aco algorithm, and ga [29] to optimize the execution time and the amount of energy Consumed by distributed queries in mobile cloud environments. for simulation, the cloudsim toolkit is used to simulate the cloud environment and the Hierarchical network topology for the data center. The number of virtual machines (vms) is equal to or greater than the number of hosts. Vms were also executed in the Same number of different tasks. For this purpose, a number of tasks were changed from 50 to 1500, and each time the experiment was repeated more than 10 times. The Simulation results are presented in figures 7 and 8.

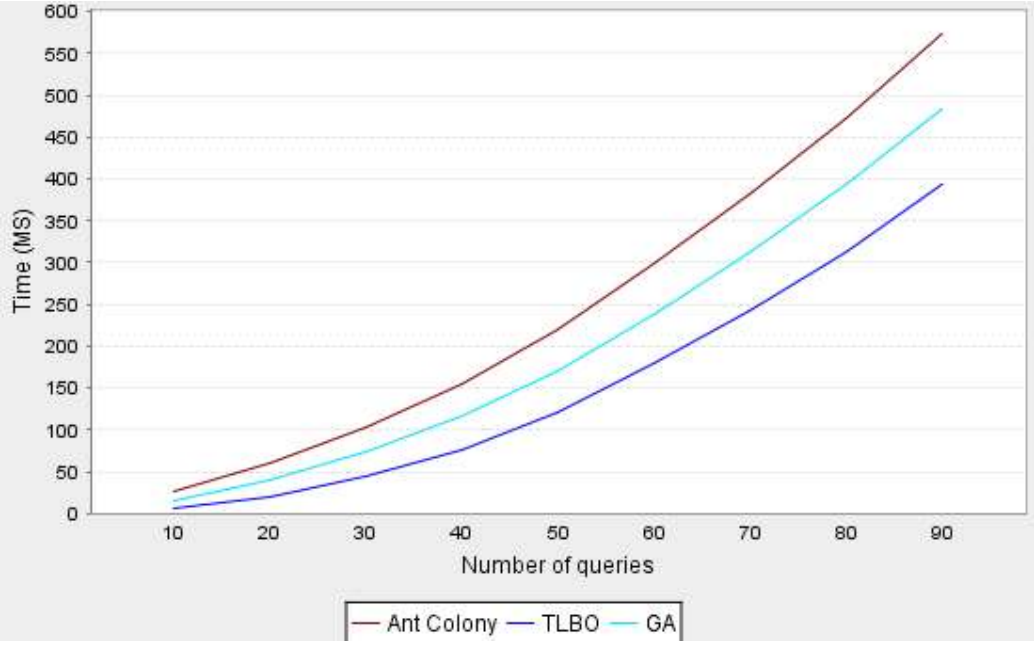

Figure 7: query execution time

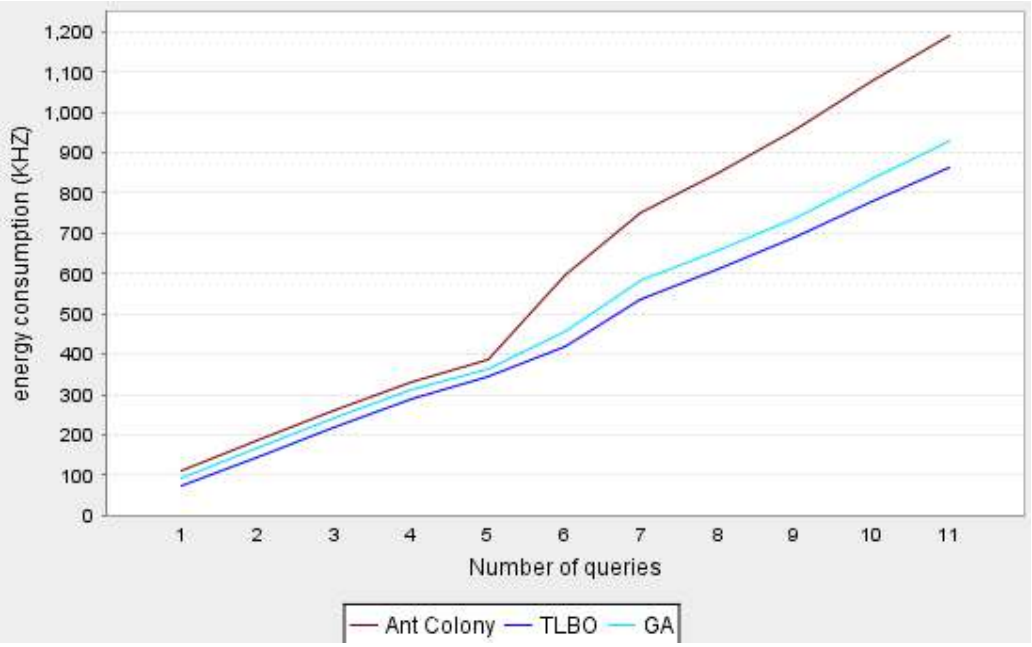

Figure 8: energy consumption

5. Conclusions:

The simulation results indicated that with an increase in the number of queries, the query execution time increases while the increasing slope of query execution time in the teaching-learning method is less than aco and ga.

Moreover, as the number of queries increases, the amount of energy consumed in cloud mobile environments in all three algorithms increases, while the increasing slope of energy consumption in the teaching-learning method is less than in aco and ga. As the number of queries increases, the tlbo algorithm will outperform aco and ga.

In conclusion, the proposed cost function, applied techniques, and tlbo algorithm yielded better results than aco and ga in solving query execution time optimization problem and energy consumption in mobile cloud environments.

References

1. c. Nartey *et al.*, "blockchain-iot peer device storage optimization using an advanced time-variant multi-objective particle swarm optimization algorithm," doi: 10.1186/s13638-021-02074-3.
2. c. Wang, l. Gruenwald, l. D' orazio, and e. Leal, "cloud query processing with

reinforcement learning-based multi-objective re-optimization," *lect. Notes comput. Sci. (including subser. Lect. Notes artif. Intell. Lect. Notes bioinformatics)*, vol. 12732 lncs, pp. 141–155, 2021, doi: 10.1007/978-3-030-78428-7_12.

3. i. Conference and c. C. Technology, "2012 ieee 4th international conference on cloud computing technology and science 2012 ieee 4th international conference on cloud computing technology and science," *proc. Ieee cloudcom 2012*, pp. 578–580, 2012, doi: 10.1109/cloud.2011.92.
4. f. Helff and l. Orazio, "weighted sum model for multi-objective query optimization for mobile-cloud database environments," 2016.
5. c. Wang *et al.*, "cloud query processing with reinforcement learning-based multi-objective re-optimization to cite this version : hal id : hal-03522314 cloud query processing with reinforcement," 2022.
6. x. Li, h. Yu, l. Yuan, and x. Qin, "query optimization for distributed spatio-temporal sensing data processing," pp. 1–21, 2022.
7. i. Foster, y. Zhao, i. Raicu, and s. Lu, "cloud computing and grid computing 360-degree compared," 2008, doi: 10.1109/gce.2008.4738445.
8. h. T. Dinh, c. Lee, d. Niyato, and p. Wang, "a survey of mobile cloud computing: architecture, applications, and approaches," *wirel. Commun. Mob. Comput.*, 2013, doi: 10.1002/wcm.1203.
9. a. A. Alabbadi and m. F. Abulkhair, "multi-objective task scheduling optimization in spatial crowdsourcing," *algorithms*, vol. 14, no. 3, 2021, doi: 10.3390/a14030077.
10. "time-,energy-,and monetary cost-aware cache design for nobile-cloud data base system," 2015.
11. v. Mishra and v. Singh, "generating optimal query plans for distributed query processing using teacher-learner based optimization," *procedia comput. Sci.*, vol. 54, pp. 281–290, 2015, doi: 10.1016/j.procs.2015.06.033.
12. m. Satyanarayanan, "mobile computing," in *proceedings of the 1st acm workshop on mobile cloud computing & services social networks and beyond - mcs ′ 10*, 2010, pp. 1–6, doi: 10.1145/1810931.1810936.
13. m. Satyanarayanan, "fundamental challenges in mobile computing," 1996, doi: 10.1145/248052.248053.
14. t. Wang, x. Wei, c. Tang, and j. Fan, "efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints," *peer-to-peer netw. Appl.*, vol. 11, no. 4, pp. 793–807, 2018, doi: 10.1007/s12083-017-0561-9.

15. e. Azhir, n. Jafari navimipour, m. Hosseinzadeh, a. Sharifi, and a. Darwesh, "deterministic and non-deterministic query optimization techniques in the cloud computing," *concurr. Comput.*, 2019, doi: 10.1002/cpe.5240.
16. w. T. Tsai, x. Sun, and j. Balasooriya, "service-oriented cloud computing architecture," 2010, doi: 10.1109/itng.2010.214.
17. m. Kumar, n. Batra, and h. Aggarwal, "cache based query optimization approach in distributed database," *int. J. Comput. Sci. Issues*, vol. 9, no. 6, pp. 389–395, 2012.
18. q. Ren and m. H. Dunham, "using semantic caching to manage location dependent data in mobile computing," *proc. Annu. Int. Conf. Mob. Comput. Networking, mobicom*, pp. 210–221, 2000, doi: 10.1145/345910.345948.
19. b. Chidlovskii and u. M. Borghoff, "semantic caching of web queries," *vldb j.*, vol. 9, no. 1, pp. 2–17, 2000, doi: 10.1007/s007780050080.
20. s. Dar, m. Franklin, b. Jonsson, d. Srivastava, and m. Tan, "semantic data caching and replacement," 1996.
21. q. Ren, m. H. Dunham, and v. Kumar, "semantic caching and query processing," *ieee trans. Knowl. Data eng.*, vol. 15, no. 1, pp. 192–210, 2003, doi: 10.1109/tkde.2003.1161590.
22. dra. An fauzia rozani, *principles of distributed database systems*. 2017.
23. m. T. Özsu and p. Valduriez, "principles of distributed database systems," *principles of distributed database systems*. 2020, doi: 10.1007/978-3-030-26253-2.
24. a. Thakkar, k. Chaudhari, and m. Shah, "a comprehensive survey on energy-efficient power management techniques," *procedia comput. Sci.*, vol. 167, no. 2019, pp. 1189–1199, 2020, doi: 10.1016/j.procs.2020.03.432.
25. r. V rao, v. J. Savsani, and d. P. Vakharia, "teaching – learning-based optimization : an optimization method for continuous non-linear large scale problems," *inf. Sci. (ny).*, vol. 183, no. 1, pp. 1–15, 2012, doi: 10.1016/j.ins.2011.08.006.
26. p. Tiwari and s. V. Chande, "optimal ant and join cardinality for distributed query optimization using ant colony optimization algorithm," in *advances in intelligent systems and computing*, 2019, vol. 841, pp. 385–392, doi: 10.1007/978-981-13-2285-3_45.
27. w. Yeh, "a simple heuristic algorithm for generating all minimal paths," no. October 2007, 2017, doi: 10.1109/tr.2007.903290.
28. r. V. Rao and v. Patel, "international journal of industrial engineering computations," vol. 3, pp. 535–560, 2012, doi: 10.5267/j.ijiec.2012.03.007.

29. h. Kadkhodaei and f. Mahmoudi, "a combination method for join ordering problem in relational databases using genetic algorithm and ant colony a combination method for join ordering problem in relational databases using genetic algorithm and ant colony," no. November, 2011, doi: 10.1109/grc.2011.6122614.