**International Journal of Electrical and Computer Engineering (IJECE)**
Vol. 12, No. 6, December 2022, pp. 6373~6386
ISSN: 2088-8708, DOI: 10.11591/ijece.v12i6.pp6373-6386 

# An optimized cost-based data allocation model for heterogeneous distributed computing systems

**Sashi Tarun[1], Mithilesh Kumar Dubey[1], Ranbir Singh Batth[1], Sukhpreet Kaur[2]**
[1]School of Computer Science Engineering, Lovely Professional University, Phagwara, India
[2]Department of Computer Science Engineering, Chandigarh Engineering College, Chandigarh, India

**Article Info**

***Article history:***

Received Jun 25, 2021
Revised Jun 11, 2022
Accepted Jul 7, 2022

***Keywords:***

Communication cost
Computation cost
Data allocation
Execution time
Network cost
Total cost

**ABSTRACT**

Continuous attempts have been made to improve the flexibility and effectiveness of distributed computing systems. Extensive effort in the fields of connectivity technologies, network programs, high processing components, and storage helps to improvise results. However, concerns such as slowness in response, long execution time, and long completion time have been identified as stumbling blocks that hinder performance and require additional attention. These defects increased the total system cost and made the data allocation procedure for a geographically dispersed setup difficult. The load-based architectural model has been strengthened to improve data allocation performance. To do this, an abstract job model is employed, and a data query file containing input data is processed on a directed acyclic graph. The jobs are executed on the processing engine with the lowest execution cost, and the system's total cost is calculated. The total cost is computed by summing the costs of communication, computation, and network. The total cost of the system will be reduced using a Swarm intelligence algorithm. In heterogeneous distributed computing systems, the suggested approach attempts to reduce the system's total cost and improve data distribution. According to simulation results, the technique efficiently lowers total system cost and optimizes partitioned data allocation.



***Corresponding Author:***

Sashi Tarun
School of Computer Science Engineering, Lovely Professional University
Phagwara, India
Email: sashitarun79@gmail.com

## 1. INTRODUCTION

The success of a distributed system is determined by how data is fragmented and distributed across several geographical locations [1], [2]. One of the most difficult aspects of distributed architecture is overcoming the growing workforce load and cutting down data allocation costs. Load implies a longer query completion time, which has an impact on operational costs and increases the overall system execution cost. To keep the overall system execution cost low, data fragments must be broken down into little sub-tasks and planned to be processed in parallel. The objective of this type of heuristic division is to reduce the time it takes to complete all jobs.

The number of processing engines and tasks used in the execution is determined by the query file and the number of processing engines. Each engine has its own computing cost for calculating task execution costs. The execution cost is calculated by adding the lowest execution cost to the communication and network costs incurred during the processing engine's execution.

To parallel execute all jobs, a directed acyclic graph (DAG) with vertices and edges as shown in Figure 1 was utilized. Each job is represented as a vertex or node in a DAG, and the edges between them

reflect the communication cost and connection between the tasks. The cost of communication is incurred when data is transferred between nodes. DAG is responsible for three sorts of costs: communication, network, and computation. Each expense factored into the total cost of system execution. Only the calculation cost with the shortest execution time is considered for total cost computation.

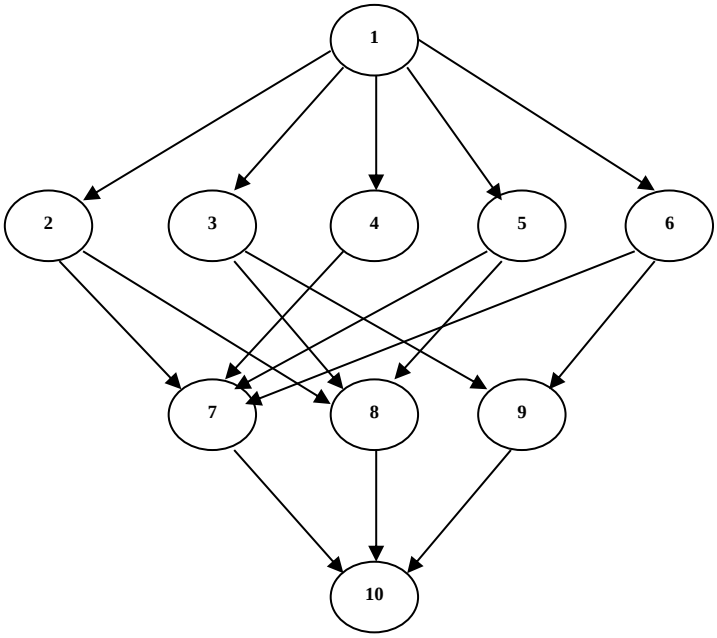


Figure 1. Directed acyclic graph

We consider a distributed system with heterogeneous processors and system tasks in this paper. $P=P_1, P_2, ..., P_n$ of heterogeneous processors connected by communications links, and $T=t_1, t_2, ...., t_m$ of system tasks that collectively express a purpose, are represented by a distributed system. The execution cost matrix (ECM), which is an asymmetrical matrix of order *m*m*, represents the cost of execution of tasks on different processors, while the network cost matrix (NCM), which is an asymmetrical matrix of order m*m, represents the cost of communication between multiple tasks, as shown in Tables 1 and 2. By using both, the suggested artificial bee colony (ABC) algorithm may compute the best overall cost.

Table 1. Execution cost matrix

| Query/Tasks | Processors | | |
|---|---|---|---|
| | P1 | P2 | P3 |
| 1 | 0.81 | 0.16 | 0.66 |
| 2 | 0.91 | 0.97 | 0.04 |
| 3 | 0.13 | 0.96 | 0.85 |
| 4 | 0.91 | 0.49 | 0.93 |
| 5 | 0.63 | 0.80 | 0.68 |
| 6 | 0.10 | 0.14 | 0.76 |
| 7 | 0.28 | 0.42 | 0.74 |
| 8 | 0.55 | 0.92 | 0.39 |
| 9 | 0.96 | 0.79 | 0.66 |
| 10 | 0.96 | 0.96 | 0.17 |

Table 2. Network cost matrix

| Query/Tasks | Processors | | |
|---|---|---|---|
| | P1 | P2 | P3 |
| 1 | 0.71 | 0.44 | 0.28 |
| 2 | 0.03 | 0.38 | 0.68 |
| 3 | 0.28 | 0.77 | 0.66 |
| 4 | 0.05 | 0.80 | 0.16 |
| 5 | 0.10 | 0.19 | 0.12 |
| 6 | 0.82 | 0.49 | 0.50 |
| 7 | 0.69 | 0.45 | 0.96 |
| 8 | 0.32 | 0.65 | 0.34 |
| 9 | 0.95 | 0.71 | 0.59 |
| 10 | 0.03 | 0.75 | 0.22 |

Delay in response, high execution time, and high completion time are the issues with data allocation in a distributed system [3]. It progressively boosts system costs and impacts workforce progress. The use of several processors increased network and communication costs during task execution, which had an impact on the overall system cost. Earlier techniques used by the researchers excluded network costs from overall system cost calculations. In this situation, the previously produced study results appear to be erroneous and useless in different experiments in a distributed environment. The proposed study focuses on a swarm intelligence-based artificial bee colony method useful for addressing and resolving current difficulties and

rapidly working for handling previous flaws [4]–[9] discusses previous issues. The proposed method is based on the learning and adaptive behavior of bees, which might be beneficial in resolving performance difficulties. It controls bee-degradation loss that arises as a result of the high expense of collecting the bees from the location. To accomplish optimization, such costs are subtracted from the overall system cost to balance loss. It aids in the betterment of data allocation in a distributed computing system.

The following is a list of the topics covered in this paper. The second section goes through important research and findings. The third section introduces the technique we suggest. The simulation results of our suggested ABC method are shown in section 4. Section 5 compares and contrasts the proposed algorithm with existing algorithms. Finally, section 6 presents the paper's conclusion.

## 2. RELATED WORK

An energy-efficient dynamic loading and resource scheduling method includes reducing energy usage and decreasing application times. The method also successfully decreases energy efficiency by modifying the central processing unit (CPU) clock frequency of smart mobile devices to the optimum in local computing and adjusting the communication energy of wireless channels in cloud computing [10]. For the placement of virtual machines in cloud computing an energy-efficient order exchange and migration ant colony system (OEMACS) algorithm was created. The intended virtual machine placement was achieved with the fewest number of active machines and by turning off idle nodes. According to experimental investigations, OEMACS aimed to minimize the number of active servers, increase resource use, balance diverse resources, and reduce power consumption [11]. To offer energy and service-sensitive performance in the placement and consolidation of virtual machines, a multi-target colony optimization technique was presented. The results demonstrate that this technique outperforms the other ways in terms of energy consumption, limiting CPU waste, lowering energy communications costs caused by traffic sharing across virtual machines, and reducing the number of virtual migrations to system and service level agreement (SLA) violations [12]. To minimize all of cloud data center power consumption a platform for virtual machine placement was introduced. The adaptability and scalability of the platform proposed resulted in exceptional success in virtual machine deployment and relocation processes [13].

An evaluation of current reliability and energy management strategies and their effect on cloud computing was discussed. There were debates on the classification of resource loss, failure tolerance mechanisms, and mechanisms for energy conservation in cloud systems. Different problems and study gaps have been established in the balance between energy reliability and quality [14]. A strong immune clonal optimization method based on the dynamic load balance approach and immune clonal selection theory in green cloud computing has solved the problem of high energy consumption and reduced cloud utilization. In terms of solution efficiency and processing costs, the experimental findings show that the method outperforms clonal selection techniques [15]. The need for energy management is demonstrated when addressing the dual position of cloud computing as a significant contributor to rising energy use and reducing energy waste. The research provided an in-depth analysis of current energy management methods in cloud computing. It also supplies taxonomies for the assessment of current work in the area of science [16]. Consolidation of tasks as an efficient way to maximize resource usage and minimize energy use was addressed. The research focused on two energy-conscious energy consolidation heuristics intended to optimize the use of resources and take both active and idle energy use directly into account. The heuristics suggested assigning each job to the resource, which minimizes the energy needs explicitly or indirectly, without degradation in performance [17]. An energy-efficient cloud computing architectural structure and concepts were suggested. The study identified open analysis problems, the provision of infrastructure as well as algorithms to handle cloud computing environments effectively. The conclusions indicate that the suggested model of cloud storage makes substantial cost savings and has a high capacity in complex workload environments for energy efficiency improvements [18].

An algorithm for allocating the total energy consumption was introduced. The effects of simulations were also viewed on a state-of-the-art platform. The suggested solution results in tangible energy conservation, showing energy efficiency dominance in comparison with well-known and widely accepted allocation methods [19]. The research was conducted on maximizing physical and virtual machines' capacity and energy consumption in a cloud computing system. Findings offered a good understanding of how power and energy usage were affected by various workloads. The tools and structure presented can be used for research and improving energy efficiency in any cloud environment and of any scale [20]. A problem of energy optimization has been modeled whereas the task dependence, transfer of data, and some constraints such as response time, and cost have been considered and solved by genetic algorithms. A series of simulation trials have been carried out to assess the algorithm efficiency and the findings suggest that the proposal is more effective than the benchmark method [21]. To decrease energy usage in cloud data centers, an optimal paradigm for work schedules has been proposed. The proposed solution was designed as an

integer programming problem to reduce the energy consumption of a cloud-based data center by organizing activities for a small number of servers and adhering to task response time constraints. As a realistic program, the authors have developed the most effective initial task-programming method for the server to decrease energy expenses. A data center planning system with diverse tasks is modeled and simulated. The study findings reveal that the recommended work scheduling strategy reduces server power consumption by more than 70 times on average when compared to a random job scheduling system [22]. A power-aware scheduling approach for a heterogeneous cloud network was suggested to solve the issue of high energy consumption. The results show that the average power consumption in this system is 23.9-6.6% lower than in modern technology [23]. An abstract model was proposed that uses piecewise linear functions to handle data analytics workload in a distributed cluster architecture. This is responsible to reduce the makespan time to handle cost issues [24]. A hybrid heuristic genetic algorithm and the steepest descent methods were used to achieve optimal task allocation with the reduction in hardware policies to reduce system cost [25]. A latency-aware max-min algorithm (LAM) has been developed for resource allocation in cloud infrastructures. The suggested method was developed to handle resource allocation challenges such as changing user requirements and on-demand access to infinite resources. It may allocate resources in a cloud-based environment to enhance infrastructure performance and increase revenue [26].

## 3. PROPOSED METHOD

The data allocation method is based upon the processing engine. The data allocation process consumes a higher network cost if the cost parameter is not controlled correctly. The cost of doing activities on multiple processors varies in this case. When two dependent jobs are run on the same processor, the computing cost is the same, and communication between them is regarded as zero. The planned activities are repeated on the specific processing engines to compute execution costs. To calculate the task execution cost, the lowest cost of query execution is applied to the communication cost. The following stages are carried out to carry out the planned task, as illustrated in Figure 2.

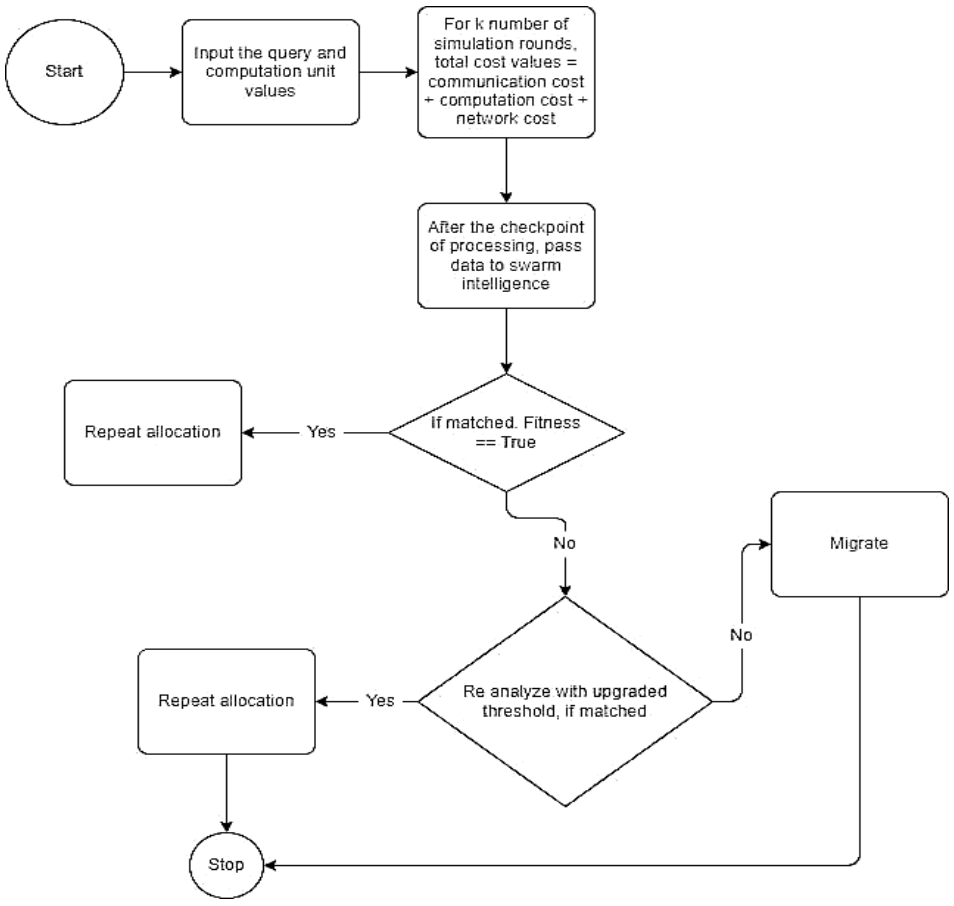


Figure 2. Proposed method flow

### 3.1. Load the query data file

The communication and processing costs of the tasks in a flow are utilized to compute the query cost. It is a flow diagram that shows the execution of queries and tasks at different levels. The lowest cost processing engine (PE) is chosen to determine the calculation cost. The query model flow is used to illustrate the details of all jobs, including their flow and associated communication and computing costs, as shown in Table 3.

Table 3. Query model flow (*query_data*)

| Query/Tasks flow | | Communication Cost | Computation Cost | | |
|---|---|---|---|---|---|
| T1 | T2 | | PE1 | PE2 | PE3 |
| 1 | 2 | 12 | 12 | 10 | 14 |
| 1 | 3 | 15 | 8 | 9 | 15 |
| 1 | 4 | 17 | 14 | 18 | 15 |
| 1 | 5 | 14 | 11 | 14 | 9 |
| 1 | 6 | 16 | 12 | 14 | 17 |
| 2 | 7 | 14 | 11 | 15 | 17 |
| 2 | 8 | 15 | 12 | 17 | 19 |
| 3 | 8 | 18 | 12 | 17 | 19 |
| 3 | 9 | 17 | 17 | 18 | 13 |
| 4 | 7 | 18 | 11 | 15 | 17 |
| 5 | 7 | 17 | 11 | 15 | 17 |
| 5 | 8 | 25 | 12 | 17 | 19 |
| 6 | 9 | 10 | 17 | 18 | 13 |
| 6 | 7 | 14 | 11 | 15 | 17 |
| 7 | 10 | 12 | 14 | 15 | 13 |
| 8 | 10 | 13 | 14 | 15 | 13 |
| 9 | 10 | 14 | 14 | 15 | 13 |

### 3.2. Total cost estimation

The total cost is defined as the total energy incurred during the execution of all tasks. At the node level, a level-wise cumulative computation is used to calculate overall costs. At each level, the load is measured by summing all communication, computation, and network access expenses for all jobs. The overall cost spent after selecting processors for task execution is represented by all three factors. All processors have their specifications and perform tasks without violating the priority limitations set by the operating system. It indicates it will not let you break the workflow sequence. Sub-tasks are executed after each of its parents has completed their execution. Instead of focusing on task prioritization at each node, preference is given to processors with lower execution costs for a task assessment. Tasks are data fragments that are placed on each node and vary in size. To compute the total cost at level 1 (root-level) following steps are executed as shown in pseudocode 1. It is responsible to compute total cost by SumUp least execution cost of PE with other parameters cost incurred at each level.

Pseudocode 1. Pseudocode to calculate execution pattern of level 1 tasks

```
Input: entry_query, query_comp_cost, myminvalue, myminpos
Output: total_engine_value
1  execution_pattern=[ ];
2  total_engine_value=zeros(1,3);
3  [myminvalue,myminpos]=min(query_comp_cost(1,:));
4  execution_pattern(1,1)=entry_query;
5  execution_pattern(1,2)=0;
6  execution_pattern(1,3)=myminvalue;
7  execution_pattern(1,4)=myminpos;
8  total_engine_value(1,myminpos)=total_engine_value(1,myminpos)+myminvalue;
```

Here, to calculate the execution pattern at level 2, it is required to count number of queries comes under this level, find connectivity between parent and current nodes and compute their communication cost. The communication cost become same in the case if tasks are executed on the same processor else not. To calculate the total cost incurred during the execution of individual task/queries at current level communication cost, and least execution cost of machines are involved. Pseudocode 2 is responsible to evaluate all cost parameters to compute the overall cost of level 2.

Pseudocode 2. Pseudocode to calculate execution pattern of level 2 tasks

```
Input: last_engine, total_query_count, lvcount2, current, parent, query_engine
Output: comm_costt, total_cost, total_engine_value
1  last_engine=myminpos;
```

```
2  total_cost=[ ];
3  total_query_count=2;
4  for i=1:lvcount2          % return level-2 queries count
5  current=level(2,i);
6  parent=entry_query;
7  for j=1:query_engines
8  comp_current=query_comp_cost(current,j);
9  if j~=last_engine
10 comm_costt=0;
11 sd=find(query_comm_cost(:,1)==entry_query );  % return total no. of queries related to 1
12 for k=1:numel(sd)
13 kp=query_comm_cost(sd(k),2);
14 if kp==current
15 comm_costt=query_comm_cost(k,3);
16 end
17 end
18 total_cost(j)=comm_costt+comp_current+total_engine_value(1,j);
19 else
20 total_cost(j)=comp_current+total_engine_value(1,j);
21 end
22 end
```

The pseudocode 3 estimate the level 3 tasks total execution cost. Here, all tasks are evaluated level-wise in sequence to compute the total cost pattern. During computation cost, communication cost and network cost values are summed-up.

Pseudocode 3. Pseudocode to calculate execution pattern of level 3 tasks

```
Input: lvcounter, query_data, execution_pattern
Output: parentcurrent, parentfinishtime, totalcost
1  for i=1:lvcounter            % Tally execution pattern of level 3
2  current=lv3jobs(i);
3  parentcurrent=[ ];
4  parentfinishtime=[ ];
5  counter=0;
6  [dp,pos]=find(query_data(:,2)==current);
7  for j=1:numel(dp)
8  parentcurrent(j)=query_data(dp(j),1);
9  currentparent=parentcurrent(j);
10 m=find(execution_pattern(:,1)==currentparent);
11 currentparentfinishtime=execution_pattern(m,3);
12 parentfinishtime(j)=currentparentfinishtime;
13 parentprocessor(j)=execution_pattern(m,4);
14 end
15 [maxval,maxpos]=max(parentfinishtime);
16 minstarttime=maxval;
17 parentp=parentprocessor(maxpos);
18 totalcost=[ ];
19 for j=1:3
20 [p,k]=find(execution_pattern(:,4)==j);
21 lasttime=execution_pattern(p(numel(p)),3);
22 if lasttime<minstarttime
23 lasttime=minstarttime;
24 end
25 totalcost(j)=lasttime;
26 end
```

The queries at level 4 are conducted once the parent tasks at level 3 have been completed. The size of the level is calculated at this level, and each task is assessed row-by-row. There is one task marked as current in this. According to *query_data*, the current task or query has three parents that are each represented as *parentcurrent*. Each *parentcurrent's* execution cost is calculated separately. As illustrated in pseudocode 4, network costs and PEs with the lowest execution costs are added to compute the overall cost during evaluation of tasks and queries at level 4. At the end, total execution cost of processing engines (PEs) is computed and the smallest execution cost of PE is picked.

Pseudocode 4. Pseudocode to calculate execution pattern of level 4 tasks

```
Input: lvcounter, current, parentcurrent, parentfinishtime, query_data, execution_pattern
Output: totalcost
1. for i=1:lvcounter
2. current=lv4jobs(i);
3. parentcurrent=[ ];
```

```
parentfinishtime=[ ];
counter=0;
[dp,pos]=find(query_data(:,2)==current);
for j=1:numel(dp)
parentcurrent(j)=query_data(dp(j),1);
currentparent=parentcurrent(j);
m=find(execution_pattern(:,1)==currentparent);
currentparentfinishtime=execution_pattern(m,3);
parentfinishtime(j)=currentparentfinishtime;
parentprocessor(j)=execution_pattern(m,4);
end
[maxval,maxpos]=max(parentfinishtime);
minstarttime=maxval;
parentp=parentprocessor(maxpos);
totalcost=[ ];
for j=1:3
[p,k]=find(execution_pattern(:,4)==j);
lasttime=execution_pattern(p(numel(p)),3);
if lasttime<minstarttime
lasttime=minstarttime;
end
totalcost(j)=lasttime;
end
```

### 3.2.1. Calculation of total system cost

The system's total cost is calculated by adding the least execution cost of all tasks using (3). The execution cost of the processing engine for each task is estimated using (1). The least execution cost is estimated using (2). The practical calculation of implementation for each processing engine is shown in Table 4.

Table 4. Execution cost of each processing engine

| Query/tasks | 1 | | | 2 | | | 3 | | | 4 | | | 5 | | | 6 | | | 7 | | | 8 | | | 9 | | | 10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Computation cost of tasks | 9 | 8 | 7 | 12 | 10 | 14 | 8 | 9 | 15 | 14 | 18 | 15 | 11 | 14 | 9 | 12 | 14 | 17 | 11 | 15 | 17 | 12 | 17 | 19 | 17 | 18 | 13 | 14 | 15 | 13 |
| Execution cost of each PEs | 0 | 0 | 7 | 24 | 22 | 21 | 23 | 24 | 43 | 54 | 35 | 43 | 48 | 63 | 37 | 51 | 65 | 82 | 53 | 57 | 60 | 53 | 45 | 37 | 63 | 60 | 61 | 64 | 72 | 70 |
| Least exec. cost | 7 | | | 21 | | | 23 | | | 35 | | | 37 | | | 51 | | | 53 | | | 45 | | | 60 | | | 64 | | |
| Total system cost | 396 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

$$PEs\ Execution\ Cost = communication\ cost + computation\ cost + network\ cost \quad (1)$$

$$Least\ Execution\ Cost = min(execution\ cost\ of\ each\ PEs) \quad (2)$$

$$Total\ Execution\ Cost = sum\ of\ least\ execution\ cost\ of\ all\ tasks \quad (3)$$

### 3.2.2. Calculation of execution pattern

Execution cost matrix (ECM) and network cost matrix (NCM) results as shown in Tables 1 and 2 respectively are used to compute execution patterns for each processing task and indicated processing engine task wise. In this, jobs at various levels are processed in topological order, and their associated communication, network, and computing costs from the communicating node are calculated and added. This method is used for all jobs and lies on different levels (0-$n^{th}$ levels) depending on directed acyclic graph (DAG) size.

The results computed at each level are added to determine the overall execution cost. Information on the tasks that are carried out one after the other starting at the root level is provided by pseudocode 5. From the root level, the execution cost pattern is calculated individually for each job from 1 to 10 by noting the starting and ending consumption units and the processing engines involved in each task. The overall cost is then calculated task-by-task as shown in Table 5. Every time, the difference between the starting and ending consumption units is used to calculate the execution cost. For each task, the energy pattern from the ECM is chosen based on the corresponding processing engine ID.

Pseudocode 5. Pseudocode to calculate execution cost pattern for each task with PEs ID

```
Input: execution_pattern, network_cost, currentp, total_query_count, ex_pt
Output: ecost
```

```
ex_pt=[ ];
ex_pt{1,1}='Query No';
ex_pt{1,2}='Starting Consumption Unit';
ex_pt{1,3}='Ending Consumption Unit';
ex_pt{1,4}='DB Engine ID';
total_query_count=10;
for i=1:total_query_count
ex_pt{i+1,1}=execution_pattern(i,1);
ex_pt{i+1,2}=execution_pattern(i,2);
ex_pt{i+1,3}=execution_pattern(i,3);
ex_pt{i+1,4}=execution_pattern(i,4);
end
for i=1:total_query_count
currentdiff=execution_pattern(i,3)-execution_pattern(i,2);
currentp=execution_pattern(i,4);
ecost=energypattern(i,currentp);
ecost=ecost+networkcost(i,currentp);
execution_pattern(i,5)=ecost;
end
```

Table 5. Execution cost pattern for each processing task

| Tasks | Starting consumption unit | Ending consumption unit | Processing engines ID | Execution cost pattern (mJ) |
|---|---|---|---|---|
| 1 | 0 | 7 | 3 | 6.5224 |
| 2 | 7 | 21 | 3 | 0.7154 |
| 3 | 0 | 23 | 1 | 0.4039 |
| 4 | 0 | 35 | 2 | 1.2806 |
| 5 | 28 | 37 | 3 | 0.7977 |
| 6 | 23 | 51 | 1 | 0.9210 |
| 7 | 51 | 53 | 1 | 0.9733 |
| 8 | 37 | 45 | 2 | 1.5620 |
| 9 | 51 | 60 | 2 | 1.5016 |
| 10 | 60 | 64 | 1 | 0.9993 |

### 3.3. Proposed artificial bee colony

A swarm intelligence algorithm is a step toward dealing with issues that cannot be handled by the traditional numerical methods. The honey bees represent a quick social collective behavior having the ability to adapt, learn, and update themselves. It inspired most researchers to apply it for the optimization of results. This algorithm is based on bee colony behavior. Here bees are of three types; employed bees (those responsible for food collection), onlooker bees (those responsible for food monitoring), and scout bees (those are in rest). ABC algorithm works here to optimize the total execution cost as shown in pseudocode 6.

Pseudocode 6. Artificial bee colony (ABC)

```
Input: Food Source
Output: [scout,beedegradation] =beefitness(employed_bee, energypattern, networkcost,
timemodel, currentprocessor, taskname)
scout=0;
beedegradation=0;
restprocessors=[ ];
rc=1;
for i=1:3
if i~=currentprocessor
restprocessors(rc)=i;
rc=rc+1;
end
end
for i=1:numel(restprocessors)
onlooker_bee_value(i)=timemodel*(energypattern(taskname, restprocessors(i)),
networkcost(taskname, restprocessors(i)));
end
selected_food_source=min(onlooker_bee_value);
onlooker_bee_selection=selected_food_source;
employed_bee=employed_bee*timemodel;
natural_change_onlooker=rand;
natural_change_employed=rand;
if onlooker_bee_selection*natural_change_onlooker > employed_bee*natural_change_employed
scout=0;
```

```
21. beedegradation=0;
22. else
23. scout=1;
24. beedegradation=((employed_bee*natural_change_employed) -
    (onlooker_bee_selection*natural_change_onlooker)) /timemodel;
25. end
26. end
```

### 3.4. Fitness function checking

Fitness function aids in validating the overall execution cost in distributed systems to accomplish data allocation at a minimal cost. It compares and verifies the outcomes, as well as cover the bee degradation part to reach to the optimal state. The code lines 19-25 of proposed ABC pseudocode 6 depicts that if the results after multiplication of *onlooker_bee_selection*natural_change_onlooker* is greater than the results of *employed_bee*natural_change_employed*, in such case there is no occurrence of *scout* and *beedegradation* loss otherwise *scout* and *beedegradation* occurs. This can be accomplished by subtracting the cost of *beedegradation* from the total system cost. This is the expense of collecting nectar from various sites, including the time spent traveling back and forth.

### 3.5. Re-analyze with upgraded threshold to get optimized result

Table 6 illustrates the results of the ABC algorithm's calculations. It shows the measurements between different variables involves to achieve optimal results. Finally, it calculates optimal results in mJ, which show the system's total execution cost in pseudocode 7. This can be achieved by setting threshold value in which optimal results is further subtracted by time spent by bees during the collection of nectar from flowers in different location indicated as *beedegradation*. The measurement in Table 6 shows the calculations carried out to justify the effectiveness of ABC algorithm. It shows total system cost estimated after reanalyzing the cost discovered. Here, all variables are interpreted in view to achieve the results.

Pseudocode 7. Pseudocode to re-analyze with upgraded threshold

```
Input: scout, beed, timemodal
Output: Optimal
1. if scout>0
2. optimal(i,5)=abs(optimal(i,5)-beed/timemodel);
3. end
```

Table 6. Fitness value and re-analyze process for optimal result

| Tasks | Current Processor | Rest Processor | 0nlooker_ bee value | onlooker _ bee_ selection (A) | natural_ change_ onlooker (B) | employed_ bee (C) | natural_ change _ employed (D) | If A*B > C*D | Scout | beed | Optimal Result After re-analyze (total system cost in mJ) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 1 | 10.6454 | 4.1745 | 0.7513 | 45.6565 | 0.2551 | False | 1 | 1.21 | 6.3487 |
| | | 2 | 4.1745 | | | | | | | 58 | |
| 2 | 3 | 1 | 13.1267 | 13.1267 | 0.5060 | 140.2212 | 0.6991 | False | 1 | 6.52 | 9.5496 |
| | | 2 | 18.9301 | | | | | | | 74 | |
| 3 | 1 | 2 | 39.6217 | 34.5972 | 0.8909 | 213.6683 | 0.9593 | False | 1 | 7.5716 | 8.9607 |
| | | 3 | 34.5972 | | | | | | | | |
| 4 | 2 | 1 | 33.5842 | 38.3812 | 0.5472 | 44.8201 | 0.1386 | False | 1 | 5.6881 | 44.6576 |
| | | 3 | 38.3812 | | | | | | | | |
| 5 | 3 | 1 | 6.5654 | 6.5654 | 0.1493 | 64.6164 | 0.2575 | False | 1 | 1.7399 | 6.9863 |
| | | 2 | 8.8844 | | | | | | | | |
| 6 | 1 | 2 | 17.6862 | 17.6862 | 0.8407 | 722.0626 | 0.2543 | False | 1 | 6.0264 | 25.5727 |
| | | 3 | 35.1709 | | | | | | | | |
| 7 | 1 | 2 | 1.7347 | 1.7347 | 0.8143 | 3.8933 | 0.2435 | True | 0 | 0 | 1.9467 |
| | | 3 | 3.4058 | | | | | | | | |
| 8 | 2 | 1 | 6.9118 | 5.8609 | 0.9293 | 99.9711 | 0.3500 | False | 1 | 3.6927 | 12.0348 |
| | | 3 | 5.8609 | | | | | | | | |
| 9 | 2 | 1 | 17.1696 | 11.1667 | 0.1966 | 121.6273 | 0.2511 | False | 1 | 3.1493 | 13.1642 |
| | | 3 | 11.1667 | | | | | | | | |
| 10 | 1 | 2 | 6.8567 | 1.5800 | 0.6160 | 15.9894 | 0.4733 | False | 1 | 1.6486 | 3.5852 |
| | | 3 | 1.5800 | | | | | | | | |

## 4. SIMULATION RESULTS AND DISCUSSION

The ABC method is implemented in MATLAB and executed on an Intel Core i3 processor 11 generations with a clock speed of 3.00 GHz and 4 GB of RAM. Megajoules (mJ) are units of measurement for the amount of energy consumed. In Table 7, the proposed work is compared to existing approaches. It was discovered that previously suggested approaches did not account for network costs in their calculations,

and anomalies were discovered that impacted overall system costs. To obtain the optimal least cost, all parameters such as communication cost, computation cost, network cost, as well as bee degradation are applied in the suggested work.

All activities are executed in parallel on processing engines, which improves performance and strengthens distributed task allocation. Processing units with low execution costs can complete jobs in large numbers and in a short amount of time. As indicated in Table 8, the total incurred cost on task execution is utilized to determine system cost. The suggested work is compared to other current techniques with concerns, and it is discovered that the ABC algorithm helps to obtain optimal system cost to construct a resilient distributed environment, as shown in Table 9.

Table 10 compares the before and after results of total energy consumption during task execution. It shows that total cost is reduced by the proposed ABC algorithm. The simulation results of before and after optimization are shown graphically in Figure 3.

Table 7. Comparative study of projected work with existing approaches

| Existing Approaches | References | Purpose | Assumption | Drawbacks |
|---|---|---|---|---|
| Artificial intelligence | [4] | This used list-based heterogeneous earliest finish time (HEFT) algorithm to reduce cost by minimizing energy consumption rate. | Assume to reduce system cost | Network cost is not used during the computation of cost |
| Communication link sum (CLS) | [19] | To reduce the inter-processor communication to minimize the system cost for task allocation in distributed computing systems | Assume to reduce system cost | Network cost is not used during the computation of system cost. On the other side, this approach follows a static task allocation policy |
| Enhanced PSO | [27] | Proposed a load balancing mutation particle swarm optimization (LBMPSO) to allocate the best resources to tasks for maintaining execution time, transmission cost, and makespan. | Assume to improve efficiency by allocating data with all resources at a low cost | network cost parameter is avoided here in the data allocation perspective |
| Proposed Approach | | Work to reduce overall system cost using artificial bee colony approach for DAG | Assume to reduce system cost by learning, adapting, and updating behavior to achieve performance in distributed computing | does not consider fault tolerance part to adjust the load |

Table 8. Optimal task allocation

| Optimal allocation | | Total execution cost | System cost |
|---|---|---|---|
| Tasks | Processing engines | | |
| $t_3$, $t_6$, $t_7$, $t_{10}$ | PE1 | 191 | 396 |
| $t_4$, $t_8$, $t_9$ | PE2 | 140 | |
| $t_1$, $t_2$, $t_5$ | PE3 | 65 | |

Table 9. Assessment of system cost parameter with other methods

| Proposed algorithm | | System cost | Hamed algorithm [28] | | System cost | Yadav algorithm [27] | | System cost |
|---|---|---|---|---|---|---|---|---|
| Tasks | Processing engines | | Tasks | Processing engines | | Tasks | Processing engines | |
| $t_3$, $t_6$, $t_7$, $t_{10}$ | PE1 | 396 | $t_4$, $t_7$ | PE1 | 459 | $t_5$,$t_7$ | PE1 | 528 |
| $t_4$, $t_8$, $t_9$ | PE2 | | $t_2$, $t_3$, $t_8$, $t_9$ | PE2 | | $t_2$, $t_3$, $t_8$, $t_9$ | PE2 | |
| $t_1$, $t_2$, $t_5$ | PE3 | | $t_1$, $t_5$, $t_6$ | PE3 | | $t_1$, $t_4$, $t_6$ | PE3 | |

Table 10. Earlier and subsequent total cost results in mJ

| Query/Tasks | Total Cost Before Optimization in mJ | Total Cost After Optimization in mJ |
|---|---|---|
| 1 | 6.5224 | 6.3487 |
| 2 | 10.0158 | 9.5496 |
| 3 | 9.2899 | 8.9607 |
| 4 | 44.8201 | 44.6576 |
| 5 | 7.1796 | 6.9863 |
| 6 | 25.7880 | 25.5727 |
| 7 | 1.9467 | 1.9467 |
| 8 | 12.4964 | 12.0348 |
| 9 | 13.5141 | 13.1642 |
| 10 | 3.9973 | 3.5852 |
| Average Cost of Tasks | 13.56 | 13.28 |

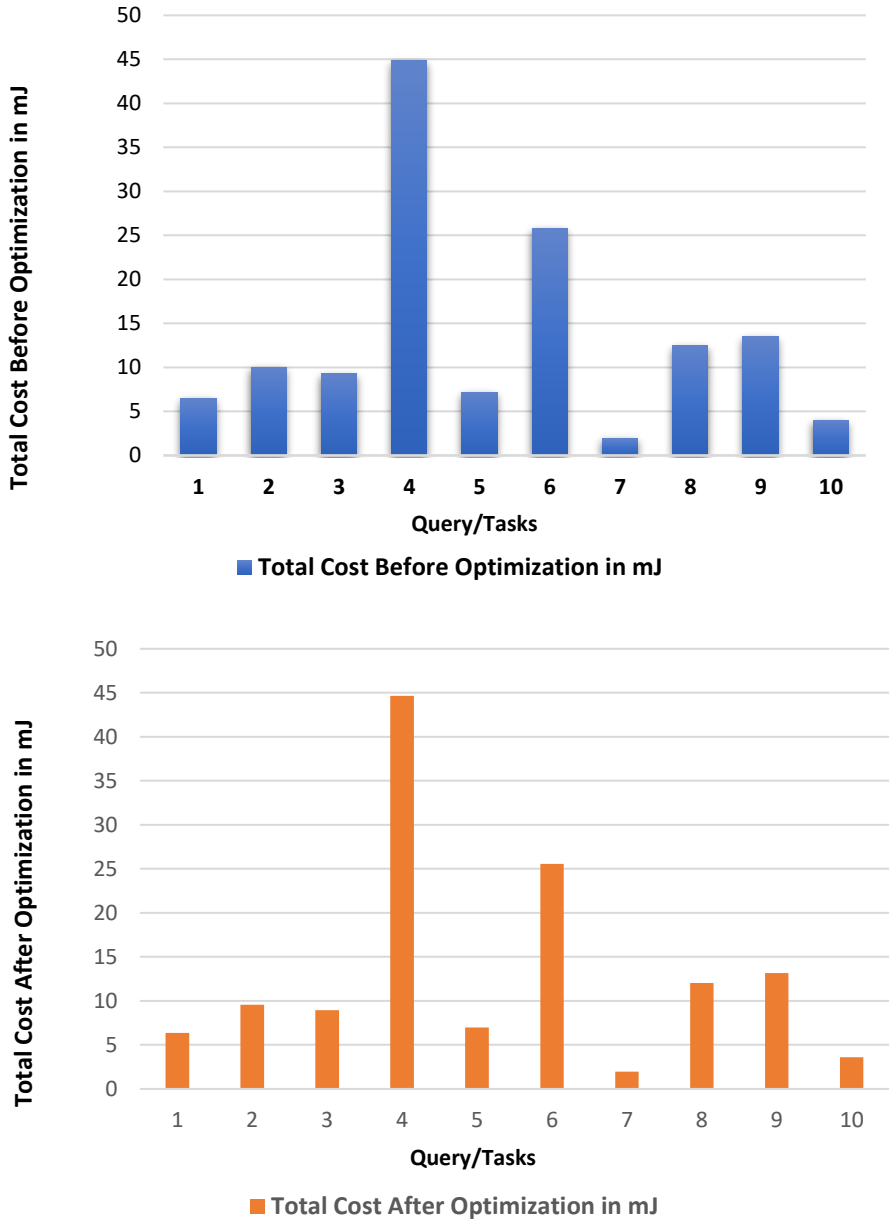


Figure 3. Energy consumption with and without optimization

In Table 11, the results are compared with a list-based task scheduling algorithm that employs artificial intelligence [4], and a task allocation model for system cost analysis that employs communication link sum (CLS) [27]. These techniques consider all indicators except network cost. In comparison to prior techniques, the proposed work saves 13.28 in overall costs, as shown in Figure 4. This method can easily allocate large data fragments and perform them fast and inexpensively. With this method, there are no extended waits, delays, or completion times, which lowers the performance of the distributed system.

Table 11. Comparison with existing techniques

| Research technique used | Reduced total execution cost (%age) |
|---|---|
| Proposed work using artificial bee colony (ABC) | 13.28 |
| Existing work using AI [4] | 60.6 |
| Existing work using communication link sum (CLS) [27] | 24 |

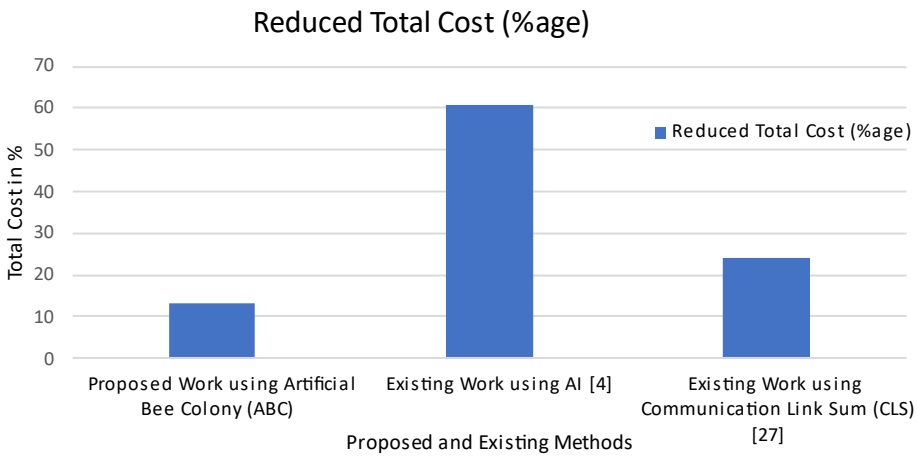


Figure 4. Comparative representation of techniques

## 5. CONCLUSION

We present a swarm intelligence-based artificial bee colony (ABC) method to reduce system execution costs and enhance data allocation in distributed systems in this study. It also makes it easier to trace the degradation loss of bees by subtracting equivalent cost units from the overall cost. When compared to previous approaches, the ABC algorithm was found to considerably lower total execution costs and improve system efficiency. Network expenses are not utilized to calculate system costs, according to previous studies. As a result, past results used to perform tests are inaccurate. This cost allocation model takes into account all expenses incurred during data processing in a distributed system. In the future, attempts might be made to bring new approaches to enhance data allocation by focusing more on fault tolerance in distributed computing systems.

## REFERENCES

[1] S. Tarun, R. S. Batth, and S. Kaur, "A novel fragmentation scheme for textual data using similarity-based threshold segmentation method in distributed network environment," *International Journal of Computer Networks and Applications*, vol. 7, no. 6, 231, Dec. 2020, doi: 10.22247/ijcna/2020/205322.

[2] S. Tarun, R. S. Batth, and S. Kaur, "A review on fragmentation, allocation and replication in distributed database systems," in *International Conference on Computational Intelligence and Knowledge Economy*, Dec. 2019, pp. 538–544., doi: 10.1109/ICCIKE47802.2019.9004233.

[3] A. Osman, A. Sagahyroon, R. Aburukba, and F. Aloul, "Optimization of energy consumption in cloud computing datacenters," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 1, pp. 686–698, Feb. 2021, doi: 10.11591/ijece.v11i1.pp686-698.

[4] Akanksha, "List-based task scheduling algorithm for distributed computing system using artificial intelligence," in *Advances in Intelligent Systems and Computing*, vol. 941, Springer International Publishing, 2020, pp. 1006–1014, doi: 10.1007/978-3-030-16660-1_98.

[5] A. Gandomi, A. Movaghar, M. Reshadi, and A. Khademzadeh, "Designing a MapReduce performance model in distributed heterogeneous platforms based on benchmarking approach," *The Journal of Supercomputing*, vol. 76, no. 9, pp. 7177–7203, Sep. 2020, doi: 10.1007/s11227-020-03162-9.

[6] N. Lotfi, "Data allocation in distributed database systems: a novel hybrid method based on differential evolution and variable neighborhood search," *SN Applied Sciences*, vol. 1, no. 12, Dec. 2019, doi: 10.1007/s42452-019-1787-3.

[7] R. Tariq, F. Aadil, M. F. Malik, S. Ejaz, M. U. Khan, and M. F. Khan, "Directed acyclic graph based task scheduling algorithm for heterogeneous systems," in *Advances in Intelligent Systems and Computing*, vol. 869, Springer International Publishing, 2019, pp. 936–947., doi: 10.1007/978-3-030-01057-7_69.

[8] S. Sandokji and F. Eassa, "Communication and computation aware task scheduling framework toward exascale computing," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 7, pp. 119–128, 2019, doi: 10.14569/IJACSA.2019.0100718.

[9] I. O. Hababeh and N. Bowring, "A method for fragment allocation design in the distributed database systems," *The Sixth Annual UAE*, pp. 4–12, 2005

[10] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 319–333, Feb. 2019, doi: 10.1109/TMC.2018.2831230.

[11] X.-F. Liu, Z.-H. Zhan, J. D. Deng, Y. Li, T. Gu, and J. Zhang, "An energy efficient ant colony system for virtual machine placement in cloud computing," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 113–128, Feb. 2018, doi: 10.1109/TEVC.2016.2623803.

[12] M.-H. Malekloo, N. Kara, and M. El Barachi, "An energy efficient and SLA compliant approach for resource allocation and consolidation in cloud computing environments," *Sustainable Computing: Informatics and Systems*, vol. 17, pp. 9–24, Mar. 2018, doi: 10.1016/j.suscom.2018.02.001.

[13] S. Vakilinia, B. Heidarpour, and M. Cheriet, "Energy efficient resource allocation in cloud computing environments," *IEEE Access*, vol. 4, pp. 8544–8557, 2016, doi: 10.1109/ACCESS.2016.2633558.
[14] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: survey and taxonomy," *Journal of Network and Computer Applications*, vol. 74, pp. 66–85, Oct. 2016, doi: 10.1016/j.jnca.2016.08.010.
[15] Z. Long and W. Ji, "Power-efficient immune clonal optimization and dynamic load balancing for low energy consumption and high efficiency in green cloud computing," *Journal of Communications*, vol. 11, no. 6, pp. 558–563, 2016, doi: 10.12720/jcm.11.6.558-563.
[16] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: a survey and taxonomy," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–46, Nov. 2015, doi: 10.1145/2742488.
[17] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, May 2012, doi: 10.1007/s11227-010-0421-3.
[18] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, May 2012, doi: 10.1016/j.future.2011.04.017.
[19] A. Scionti, K. Goga, F. Lubrano, and O. Terzo, "Towards energy efficient orchestration of cloud computing infrastructure," in *Advances in Intelligent Systems and Computing*, vol. 772, Springer International Publishing, 2019, pp. 172–183, doi: 10.1007/978-3-319-93659-8_15.
[20] N. Khan and R. Shrestha, "Optimizing power and energy efficiency in cloud computing," in *9th International Conference on Cloud Computing and Services Science*, 2019, pp. 380–387., doi: 10.5220/0007723503800387.
[21] C. Tang, S. Xiao, X. Wei, M. Hao, and W. Chen, "Energy efficient and deadline satisfied task scheduling in mobile cloud computing," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jan. 2018, pp. 198–205, doi: 10.1109/BigComp.2018.00037.
[22] N. Liu, Z. Dong, and R. Rojas-Cessa, "Task scheduling and server provisioning for energy-efficient cloud-computing data centers," in *IEEE 33rd International Conference on Distributed Computing Systems Workshops*, Jul. 2013, pp. 226–231, doi: 10.1109/ICDCSW.2013.68.
[23] H. Zhao, G. Qi, Q. Wang, J. Wang, P. Yang, and L. Qiao, "Energy-efficient task scheduling for heterogeneous cloud computing systems," in *IEEE 21st International Conference on High Performance Computing and Communications*, Aug. 2019, pp. 952–959, doi: 10.1109/HPCC/SmartCity/DSS.2019.00137.
[24] R. Li, N. Mi, M. Riedewald, Y. Sun, and Y. Yao, "A case for abstract cost models for distributed execution of analytics operators," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10440, Springer International Publishing, 2017, pp. 149–163., doi: 10.1007/978-3-319-64283-3_11.
[25] C.-C. Hsieh and Y.-C. Hsieh, "Reliability and cost optimization in distributed computing systems," *Computers and Operations Research*, vol. 30, no. 8, pp. 1103–1119, Jul. 2003, doi: 10.1016/S0305-0548(02)00058-8.
[26] K. A. Shakil, M. Alam, and S. Khan, "A latency-aware max-min algorithm for resource allocation in cloud," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 1, pp. 671–685, Feb. 2021, doi: 10.11591/ijece.v11i1.pp671-685.
[27] P. K. Yadav, M. P. Singh, and K. Sharma, "An optimal task allocation model for system cost analysis in heterogeneous distributed computing systems: a heuristic approach," *International Journal of Computer Applications*, vol. 28, no. 4, pp. 30–37, Aug. 2011, doi: 10.5120/3374-4664.
[28] A. Y. Hamed, "Task allocation for maximizing reliability of distributed computing systems using genetic algorithms," *International Journal of Computer Networks and Wireless Communications (IJCNWC)*, vol. 2, no. 5, pp. 560–569, 2012.

## BIOGRAPHIES OF AUTHORS

**Sashi Tarun** is a Ph.D. Research Scholar in the School of Computer Science and Engineering at Lovely Professional University, Punjab, India. He has completed M.Tech. Computer Science from Jamia Hamdard University, New Delhi. His research interests are distributed systems, cloud systems, database systems, computer networks, artificial intelligence, and machine learning. He has several papers on his credit. He has 7 years of teaching experience as an Assistant Professor. He can be contacted by email: sashitarun79@gmail.com.

**Mithilesh Kumar Dubey** is working as a Professor in the School of Computer Application of Lovely Professional University Jalandhar Punjab India. He has handsome experience in the software industry as well as in Research development. He has published many articles at international level in reputed journals. He can be contacted by email: mithilesh.21436@lpu.co.in

**Ranbir Singh Batth** is working as an associate professor in the School of Computer Science and Engineering and also serves as a coordinator for international relations at Lovely Professional University, Punjab, India. In 2018, he received his Ph.D. in computer science and engineering from Punjab Technical University, India. His research interests include wireless sensor networks, cloud computing, network security, ad-hoc networks, machine learning, deep learning, wireless communications, and mobile computing. He is a senior member of IEEE and faculty coordinator of the ACM research chapter. He can be contacted by email: ranbir.21123@lpu.co.in.

**Sukhpreet Kaur** is working as Associate Professor in the CSE department at Chandigarh Engineering College, Landran, Mohali. She has in total 15 years of vast experience in teaching and research. She has done a Ph.D. in CSE from I.K Gujral Punjab Technical University, Jalandhar, and has done her Masters in Technology in CSE from GNDEC, Ludhiana. The various research areas in which she worked include image processing, artificial intelligence, and computer vision. She can be contacted by email: sukhpreet.4479@cgc.edu.in.